

# DATA BASE MANAGEMENT SYSTEM



**Prof. Samiran Chattopadhyay**

**Prof. Partha Pratim Das**  
**Computer Science & Engineering**  
Indian Institute of Technology, Kharagpur



# INDEX

<b>S.No</b>	<b>Topic</b>	<b>Page No.</b>
	<b><i>Week 1</i></b>	
1	Course Overview	01
2	Introduction to DBMS/1	18
3	Introduction to DBMS/2	38
4	Introduction to Relational Model/1	59
5	Introduction to Relational Model/2	80
	<b><i>Week 2</i></b>	
6	Introduction to SQL/1	100
7	Introduction to SQL/2	119
8	Introduction to SQL/3	143
9	Intermediate SQL/1	166
10	Intermediate SQL/2	194
	<b><i>Week 3</i></b>	
11	Advanced SQL	218
12	Formal Relational Query Languages	249
13	Entity-Relationship Model/1	277
14	Entity-Relationship Model/2	297
15	Entity-Relationship Model/3	317
	<b><i>Week 4</i></b>	
16	Relational Database Design	343
17	Relational Database Design (Contd.)	366
18	Relational Database Design /3	387
19	Relational Database Design (Contd.)	412
20	Relational Database Design /5	444
	<b><i>Week 5</i></b>	
21	Application Design and Development/1	469
22	Application Design and Development/2	495
23	Application Design and Development/3	518
24	Storage and File Structure/1: Storage	543
25	Storage and File Structure/2: File Structure	568
	<b><i>Week 6</i></b>	
26	Indexing and Hashing/1 : Indexing/1	586

27	Indexing and Hashing/2 : Indexing/2	603
28	Indexing and Hashing/3 : Indexing/3	626
29	Indexing and Hashing/4 : Hashing	650
30	Indexing and Hashing/5 : Index Design	678

***Week 7***

31	Transactions/1	698
32	Transactions/2 : Serializability	716
33	Transactions/3 : Recoverability	735
34	Concurrency Control/1	759
35	Concurrency Control/2	783

***Week 8***

36	Recovery/1	798
37	Recovery/2	825
38	Query Processing and Optimization/1 : Processing	848
39	Query Processing and Optimization/2 : Optimization	876
40	Course Summarization	897

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 01**  
**Course Overview**

(Refer Slide Time: 00:45)

**Module Objectives**

- To understand the importance of database management systems in modern day applications
- To Know Your Course

SWAYAM: NPTEL-NOCO MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018

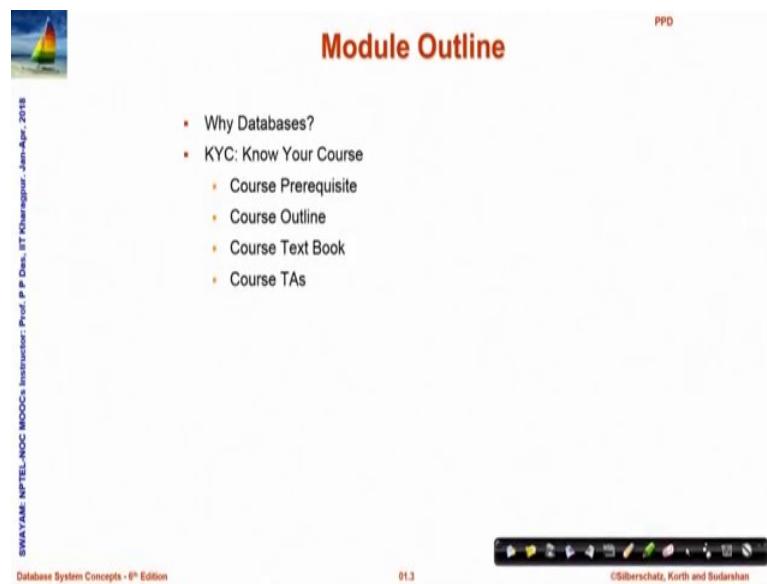
Database System Concepts - 8<sup>th</sup> Edition

01.2

©Silberschatz, Korth and Sudarshan

Welcome to database management systems. In this course, we will have 40 modules; each module would be of about half an hour. So, this is the first module, where we would talk about the overview of the course. So, we will discuss the importance of database management systems in modern day applications, and we will familiarize you with different aspects of the course.

(Refer Slide Time: 01:06)



The slide is titled "Module Outline" in red at the top right. In the top left corner, there is a small logo of a sailboat on water. The title "Module Outline" is centered above a bulleted list of course topics. The footer contains copyright information and navigation icons.

PPD

## Module Outline

- Why Databases?
- KYC: Know Your Course
  - Course Prerequisite
  - Course Outline
  - Course Text Book
  - Course TAs

SWAYAM: NPTEL-NOOCs Instructor: Prof. P. P. Den, IIT Kharagpur - Jam-Apr-2018

Database System Concepts - 8<sup>th</sup> Edition

01.3

©Silberschatz, Korth and Sudarshan

So, this will be the outline. First, we will try to explain why we need databases, and then we will run through a KYC on the course prerequisites, course outline, the textbook and the TAs who will help us in this course.

(Refer Slide Time: 01:28)



The slide features a large red title "WHY DATABASES?" in the center. In the top right corner, there is a small logo of a sailboat on water. To the right of the title, there is a bulleted list of two items. The footer contains copyright information and navigation icons.

PPD

# WHY DATABASES?

- Why Databases?
- KYC: Know Your Course

SWAYAM: NPTEL-NOOCs Instructor: Prof. P. P. Den, IIT Kharagpur - Jam-Apr-2018

Database System Concepts - 8<sup>th</sup> Edition

01.4

©Silberschatz, Korth and Sudarshan

So, first why do we need databases?

(Refer Slide Time: 01:32)

The slide has a title 'Database Management System (DBMS)' in red at the top right. To its left is a small sailboat icon. On the far left, vertical text reads 'SWAYAM-NPTEL-NOOCs Instructor: Prof. P. P. Doss, IIT Kharagpur' and 'Date: June-Apr., 2018'. At the bottom left is the text 'Database System Concepts - 8<sup>th</sup> Edition'. The main content is a bulleted list:

- DBMS contains information about a particular **enterprise**
  - Collection of interrelated **data**
  - Set of **programs** to access the data
  - An **environment** that is both *convenient* and *efficient* to use
- Database **Applications**:
  - Banking: transactions
  - Airlines: reservations, schedules
  - Universities: registration, grades
  - Sales: customers, products, purchases
  - Online retailers: order tracking, customized recommendations
  - Manufacturing: production, inventory, orders, supply chain
  - Human resources: employee records, salaries, tax deductions
  - ...
- Databases can be very **large**
- Databases touch **all aspects** of our lives

At the bottom right is a small toolbar with icons for search, refresh, etc.

A database management system contains information about a particular enterprise. So, it deals with collection of interrelated data. We have a set of programs which access and manipulate that data; and together a DBMS presents an environment for convenient as well as efficient use of data of the enterprise under consideration. So, there could be different database application.

Actually, if we look around in the world that we are living today, every aspect of our life has been touched by certain database application. Some of the very common and wide applications will include banking; we are doing net banking very regularly. Net banking is highly distributed database application that allow us to do different kinds of retail as well as corporate banking.

We perform different kind of booking reservation, railway reservation, airline reservation, hotel reservations all of these are now possible over internet as well these are all big database applications. When we are attending colleges, universities, the students, courses, teachers, course enrolments, performance of the students, and different courses, examination all are parts of huge university database applications.

There are different sales applications, online retailing applications like I am sure all of you have been using some of the Amazon, Flipkart, E-bay, Snapdeal, all these are online retail database applications which allow us to order to select items to make payments and to track the deliveries of different items that we have ordered. There are database

applications in terms of manufacturing, production systems, inventory management, any big factory of manufacturing need to use a huge database application to manage the supply chain, the inventory, the orders everything. There are database applications in human resources applications like LinkedIn are huge human resource application of database, the social media applications all involve different kinds of database applications.

So, in this database management system course, we are going to understand how such applications can be designed developed and managed over a period of time. Database applications are typically characterized by the fact that they are very large. If we have a relatively small set of data then possibly we will be able to manage it in terms of using an excel sheet or a couple of excel sheets, but soon when it goes beyond a certain size we need a database application. So, the fact of being large is a critical factor of any DBMS. So, with all this we as we observe the database says cover all aspects of our life and hence understanding DBMSs is a critical requirement for any computer science information technology student.

(Refer Slide Time: 05:54)

**University Database Example**

- Application program examples
  - Add new students, instructors, and courses
  - Register students for courses, and generate class rosters
  - Assign grades to students, compute grade point averages (GPA) and generate transcripts
- In the early days, database applications were built directly on top of file systems

SWAYAM: NPTEL-MOOCs Instructor: Prof. P. P. Chou, IIT Kharagpur - Date: April 2018

Database System Concepts - 8<sup>th</sup> Edition

01.6

Gilbert

So, if we talk about a specific database example say university database, then we will have several application programs which will allow us to do several required applications like when new students join, we would need to add the students. We will need to add new courses when courses are floated; we will need to add new instructors

when faculty join; we will need to do allotments; we will need to do registration of students for courses. We will need to conduct examinations, we will need to assign grades to students when they are graded for different courses and compute their GPA and so on. So, all the activities that a university has to deal with are application programs of varied kinds that the university database need to work with.

In the earlier days, before the days of databases, typically such applications used to be managed through file systems. We all know that all our system has a file system, where different files text as well as data files can be stored, and information can be written in these files in a certain order. And just to quickly recall file systems typically have a large number of sequential files which can be written and read in a certain order and some random access files where you can reach a particular point in the file to do certain access operation certain manipulation operations. So, in the earlier days, it was a collection of file systems which managed large enterprise data as is required.

(Refer Slide Time: 07:53)

SWAYAM NPTEL-NOC MOOCs Instructor: Prof. P. K. Khuriya Date: 17-Apr-2018

### Drawbacks of using file systems to store data

- Data **redundancy** and **inconsistency**
  - Multiple file formats, duplication of information in different files
- Difficulty in **accessing data**
  - Need to write a new program to carry out each new task
- Data **isolation**
  - Multiple files and formats
- **Integrity** problems
  - Integrity constraints (e.g., account balance > 0) become "buried" in program code rather than being stated explicitly
  - Hard to add new constraints or change existing ones

Database System Concepts - 6<sup>th</sup> Edition 01.7

But over time it was observed that the file systems to store data to manage data has a lot of drawbacks. For example, if you look at in a file system, there is often a lot of data redundancy and inconsistency. These are terms which we will loosely define here and as we go across along the course, we will understand these terms better. But redundancy just to explain redundancy is a concept where the same data is written at multiple places in different forms and that may give rise to several forms of inconsistencies because if

you write the same data in multiple files, because you need to deal with many of these aspects.

So, there is a file for students, there is a file for teachers, there is a file for particular courses, there is a file for enrolment and so on several data items may be redundantly copied in multiple files. And once that is done then we can it is very easy to get inconsistent in terms of the data because you may update the data in one file, and may forget to update the data in another file, so that is one of the first problems with using the file systems.

Then this difficulty in accessing the data because as I said the data is the files often are sequential in nature, even if they are random access, then every task might need to use data from multiple files. And opening those files and reaching to the appropriate point of access is a non-trivial task. Then there is issues of data isolation, because there are multiple types of files, there are multiple formats used therein. Very importantly there are a lot of integrity problems, the database any database application need to have a lot of integrity.

For example, if you want to withdraw money from an account - bank account, then certainly the balance needs to be positive, you can withdraw only that much amount up to that much amount, which exists in the account. So, any application will need to check for this. So, if you use a file system based application to store the data then at every point wherever you are updating the balance, you will need to make such checks which make the application quite complicated, and often creates the possibility that certain integrity checks may be missed out. So, it is hard to code these new constraints over a period of time.

(Refer Slide Time: 10:40)

**Drawbacks of using file systems to store data (Cont.)**

- **Atomicity** of updates
  - Failures may leave database in an inconsistent state with partial updates carried out
  - Example: Transfer of funds from one account to another should either complete or not happen at all
- **Concurrent access** by multiple users
  - Concurrent access needed for performance
  - Uncontrolled concurrent accesses can lead to inconsistencies
    - Example: Two people reading a balance (say 100) and updating it by withdrawing money (say 50 each) at the same time
- **Security** problems
  - Hard to provide user access to some, but not all, data

**Database systems offer solutions to all the above problems**

SWAYAM: NPTEL-NOC's Instructional PPT: Dr. Jitendra Patel, IIT Kanpur © 2018

Database System Concepts - 8<sup>th</sup> Edition 01.8 ©Silberschatz, Korth and Sudarshan

Then there are issues of atomicity of update. What atomicity means is the ability to do certain operations in a as a single unit. So, what you want is either that operation happens or the and if it happens then it happens in full in totality, whereas otherwise it may not happen at all. For example, consider that there is a funds being transformed from one account to another, so this means the account from which the funds are being transferred needs to be debited certain amount, and that same amount has to get credited to the accounts to which it is being paid.

Now, if for some reason of failure or because of the fact that there was link issues or something, if you are not able to make this whole transfer, then it is possible that you have already debited the account, but you have not been able to credit that account. Now, this will be a major cause of inconsistency in the database. So, what you want is if the transfer can happen then it must happen in totality that is what that debit and the credit must happen together or nothing should happen at all. So, there are several examples of requirement of automaticity for a update which is critical for maintaining consistency of the data.

The other aspect which has become very, very deeply required in every aspect of database is concurrency of access. If there is a database then certainly there is not a single user, there are multiple hundreds of users you think about net banking, you think about railway reservation, multiple users are trying to make bookings in multiple trains

from varied stations to vary stations in different classes and so on. So, all of this must go on at the same time that is what is called the concurrency of update.

So, it is quite possible that while you are trying to update, you check berth availability on a certain train on a certain date that you intend to travel. At the same time, someone else may be checking for the berth availability on the same train on the same date, and there could be conflict of concurrency because there may be one berth available and you are trying to book that you have seen that one berth is available.

So, you go ahead and book it try to book it, and there is another user who also saw that one berth is available and that user makes payments and tries to book that. So, concurrency needs to make sure that both of these users should not be allowed to make the booking to the same berth because then that will be a disaster. So, uncontrolled concurrency can add to several inconsistencies in the application.

Then certainly there are you all would be very familiar that today we are living under a whole lot of security threats. So, there has to be proper security that it should be possible to access the data by a user to the extent the user is allowed to do that. So, as a user you should be able to access certain parts of the data, the manager of the system, the administrator should be able to access a bigger part of the data possibly. So, security is hard to provide in terms of a file system based applications to store data. So, all these with we conclude the data based systems had those which provide solution to take care against all those above problems, and we are trying to learn how to do such things.

(Refer Slide Time: 14:37)

The slide features a sailboat icon in the top left corner. In the top right, there are two bullet points: '• Why Databases?' and '• KYC: Know Your Course'. The title 'KNOW YOUR COURSE' is centered in large red capital letters. At the bottom, there is a navigation bar with icons for back, forward, search, and other presentation controls. The footer contains the text 'SWAYAM: NPTEL-NOC MOOCs' and 'Prod: P. P. Des., IIT Kharagpur - Jain-Agr., 2018' on the left, and 'Database System Concepts - 8<sup>th</sup> Edition' and '©Silberschatz, Korth and Sudarshan' on the right.

So, moving on I would quickly take you through familiarizing with you with the overall plan of the course.

(Refer Slide Time: 14:47)

The slide features a sailboat icon in the top left corner. The title 'Course Prerequisites' is centered in large red capital letters. Below it, under the heading '• Essential', is a list of topics: 'Set Theory' (with sub-points: Definition of a Set, Intentional Definition, Extensional Definition, Set-builder Notation), 'Membership, Subset, Superset, Power Set, Universal Set', 'Operations on sets' (with sub-points: Union, Intersection, Complement, Difference, Cartesian Product), 'De Morgan's Law', and 'MOOCs: Discrete Mathematics' (with a link: [https://npTEL.ac.in/noc/individual\\_course.php?id=noc16-ma01](https://npTEL.ac.in/noc/individual_course.php?id=noc16-ma01)). At the bottom, there is a navigation bar with icons for back, forward, search, and other presentation controls. The footer contains the text 'SWAYAM: NPTEL-NOC MOOCs' and 'Prod: P. P. Des., IIT Kharagpur - Jain-Agr., 2018' on the left, and 'Database System Concepts - 8<sup>th</sup> Edition' and '©Silberschatz, Korth and Sudarshan' on the right.

So, what we first talk about are the course prerequisites. These prerequisites are kind of certain elementary level knowledge in computer science and related discrete mathematics that you should have. You should be that would make it easier for you to understand and follow the course; otherwise if you find at any stage that you are finding any of this aspect difficult to follow in the course then I would advise that you go back to

some of the background material and try to study them. So, I have tried to list down these prerequisite topics, one certainly is the set theory because that is the basic premise on which databases are designed on.

So, starting from definition of the set membership, concepts of subset, superset, power set, different operations of union, intersection, complementation, difference Cartesian product De Morgans law, all these basic set theory you should be very familiar and conversant with. If you are not I have mentioned one MOOC's course this is a past course, but you can access the videos and the contents which has a very nice discussion on these aspects of set theory which you may refer to.

(Refer Slide Time: 16:13)

Course Prerequisites

PPD

- \* Essential
  - Relations and Functions
    - › Definition of Relations
    - › Ordered Pairs and Binary Relations
      - Domain and Range
      - Image, Preimage, Inverse
      - Properties – Reflexive, Symmetric, Antisymmetric, Transitive, Total
    - › Definition of Functions
    - › Properties of Functions – Injective, Surjective, Bijective
    - › Composition of Functions
    - › Inverse of a Function
  - MOOCs: Discrete Mathematics:  
[https://nptel.ac.in/noc/individual\\_course.php?id=noc16-ma01](https://nptel.ac.in/noc/individual_course.php?id=noc16-ma01)

SWAYAM NPTEL-MOOCs Instructor: Prof. P. P. Doshi, IIT Kharagpur - 2016

Database System Concepts - 8<sup>th</sup> Edition

01.11

©Silberschatz, Korth and Sudarshan

Moving on the next, which goes on top of the sets are the concept of relations and functions. We all know that a relation is a subset of a set. So, if I have a set A, then I can define a binary relation over that set A, which is basically the pair of elements from set A or in other words the relation binary relation is a subset of the cross product of A with itself where the domain and the range are related together. So, there are concepts of the image of a domain the pre-image of the range the inverse relation, and several basic properties of relations like the relation being reflexive, symmetric, anti-symmetric, transitive, total relations and so on.

You should be familiar with these; otherwise, you will find difficult to follow major concepts in the database systems because this tribal system primarily the one that we are

going to take you through in this course is relational in model. So, it is heavily based on relations and functions. And you should understand one specifically a relation becomes a function, and when what is meant by functions being injective, subjective, bijective, what is meant by composition, and inverse of functions and so on. Again if you need, you can refer to this MOOC's course on discrete mathematics to brush up your knowledge about relations and functions.

(Refer Slide Time: 17:52)

The screenshot shows a presentation slide titled "Course Prerequisites". At the top right is a small "PPD" icon. On the left is a decorative image of a sailboat on water. The slide content is organized into sections:

- Essential**
  - Propositional Logic**
    - Truth Values & Truth Tables
    - Operators: conjunction (and), disjunction (or), negation (not), implication, equivalence
    - Closure under Operations
  - MOOCs: Discrete Mathematics:**  
[https://nptel.ac.in/noc/individual\\_course.php?id=noc16-ma01](https://nptel.ac.in/noc/individual_course.php?id=noc16-ma01)
  - Predicate Logic**
    - Predicates
    - Quantification – Existential, Universal
  - MOOCs: Discrete Mathematics:**  
[https://nptel.ac.in/noc/individual\\_course.php?id=noc16-ma01](https://nptel.ac.in/noc/individual_course.php?id=noc16-ma01)

At the bottom of the slide, there is footer text: "SWAYAM-NPTEL-MOOCs Instructor: Prof. A.P.Datta, IIT Kharagpur - Jan-Apr-2015", "Database System Concepts - 6<sup>th</sup> Edition", "01.12", and "©Silberschatz, Korth and Sudarshan".

We also need you to have a basic understanding of the propositional logic which is truth values true and false; and the different operations of conjunction, disjunction, negation that is and or not, what is meant by implication, what is meant by equivalence. You know that given two variables, which have or two propositions which can take a value true or false the conjunction of them can be represented in terms of a truth table where we say that only when both these propositions are true, then the resultant conjunctive proposition becomes true; otherwise, a conjunctive proposition is false. So, you should be familiar with these concepts; if you are not, please brush up your ideas about propositional logic.

We need a little bit of predicate logic as well a predicate logic in contrast to propositional logic deals with quantification that the knowledge of existential and universal quantifier. When you say that whether certain proposition predicates hold for all values in the domain or for some value in the domain whether there exists some value for which it

holds or whether for all values it holds. And based on that predicate logic is build up we do not need very advanced concept here just basic level familiarization will help and the same MOOC's course on discrete mathematics would be of your help in case you need to brush it up further.

(Refer Slide Time: 19:33)

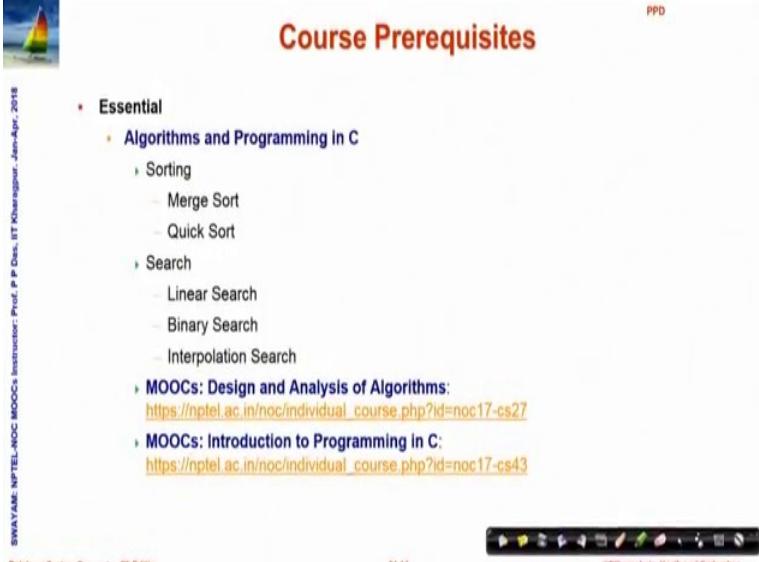
The screenshot shows a presentation slide titled "Course Prerequisites" in red at the top right. On the left, there is a small logo of a sailboat on water. The main content is a bulleted list under the heading "Essential".

- \* Essential
  - Data Structures
    - Array
    - List
    - Binary Search Tree
      - Balanced Tree
    - B-Tree
    - Hash Table / Map
  - MOOCs: Design and Analysis of Algorithms:  
[https://nptel.ac.in/noc/individual\\_course.php?id=noc17-cs27](https://nptel.ac.in/noc/individual_course.php?id=noc17-cs27)
  - MOOCs: Fundamental Algorithms – Design and Analysis:  
[https://nptel.ac.in/noc/individual\\_course.php?id=noc16-cs24](https://nptel.ac.in/noc/individual_course.php?id=noc16-cs24)

On aspects of computer science, certainly you need a good familiarity with the data structures array list particularly binary search tree, what is called a binary search tree, what is meant by a height of a binary search tree, when we say that a binary search tree is balanced. What are the ways and conditions of balancing is particularly the B-trees for organizing good search trees hash tables, what is hashing we need you to be familiar with this concept, because the databases will be heavily designed based on the concepts of B trees and hash tables and so on.

There are courses on design and analysis of algorithms, fundamental of algorithms have mentioned two excellent courses in MOOC's from which you can brush up if you need to.

(Refer Slide Time: 20:31)



The slide is titled "Course Prerequisites" in red at the top right. It features a small sailboat icon in the top left corner. The background is white with a thin vertical border on the left side. The text is organized into sections with bullet points.

**Essential**

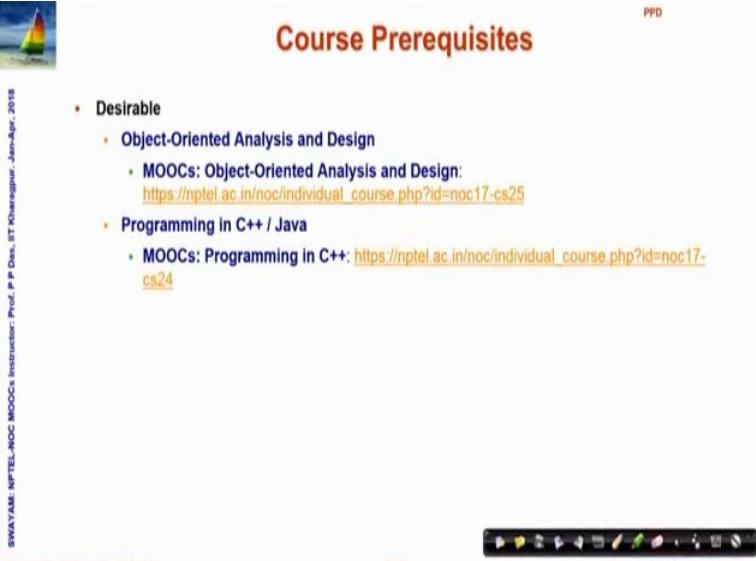
- Algorithms and Programming in C
  - Sorting
    - Merge Sort
    - Quick Sort
  - Search
    - Linear Search
    - Binary Search
    - Interpolation Search
- MOOCs: Design and Analysis of Algorithms:  
[https://nptel.ac.in/noc/individual\\_course.php?id=noc17-cs27](https://nptel.ac.in/noc/individual_course.php?id=noc17-cs27)
- MOOCs: Introduction to Programming in C:  
[https://nptel.ac.in/noc/individual\\_course.php?id=noc17-cs43](https://nptel.ac.in/noc/individual_course.php?id=noc17-cs43)

SWAYAM-NPTEL-MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - 2018  
PPD

Database System Concepts - 8<sup>th</sup> Edition 01.14 ©Silberschatz, Korth and Sudarshan

Certainly you need certain familiarity with common algorithms particularly I had mentioned sorting and searching algorithms, because these are critical for database applications. And again the same MOOC's courses would be of your help. And it will be good to have familiarity with programming in C, because several of the applications need some application high level application programming to be performed. And we would assume that those aspects we described in C because that is a fundamental and most commonly known language.

(Refer Slide Time: 21:11)



The slide is titled "Course Prerequisites" in red at the top right. It features a small sailboat icon in the top left corner. The background is white with a thin vertical border on the left side. The text is organized into sections with bullet points.

**Desirable**

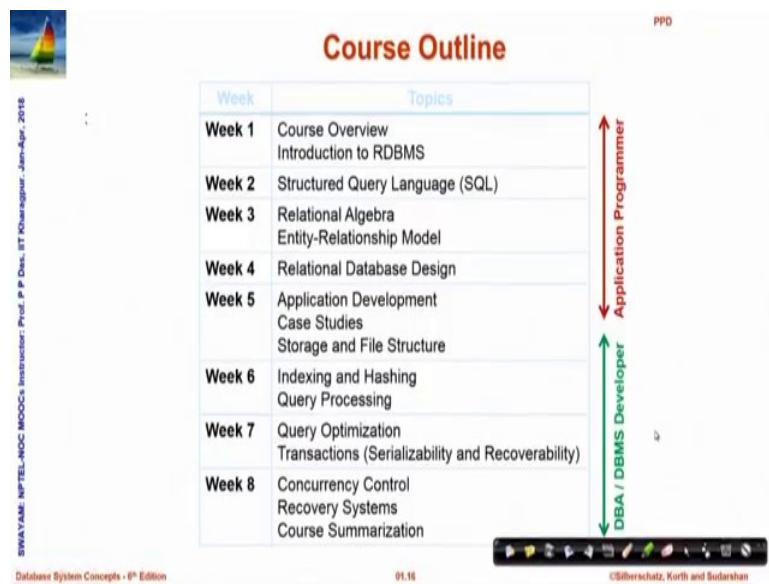
- Object-Oriented Analysis and Design
  - MOOCs: Object-Oriented Analysis and Design:  
[https://nptel.ac.in/noc/individual\\_course.php?id=noc17-cs25](https://nptel.ac.in/noc/individual_course.php?id=noc17-cs25)
- Programming in C++ / Java
  - MOOCs: Programming in C++:  
[https://nptel.ac.in/noc/individual\\_course.php?id=noc17-cs24](https://nptel.ac.in/noc/individual_course.php?id=noc17-cs24)

SWAYAM-NPTEL-MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - 2018  
PPD

Database System Concepts - 8<sup>th</sup> Edition 01.15 ©Silberschatz, Korth and Sudarshan

Besides these prerequisites which I have marked as essential, because they will certainly be required for the major parts of the course, it will be good if you have some familiarity with object oriented analysis; and design and some language which is more heavily object oriented object aligned in nature like C++ or java. So, again some MOOCs courses the related MOOCs courses are mentioned here in case you need to brush up.

(Refer Slide Time: 21:46)



**Course Outline**

Week	Topics
Week 1	Course Overview Introduction to RDBMS
Week 2	Structured Query Language (SQL)
Week 3	Relational Algebra Entity-Relationship Model
Week 4	Relational Database Design
Week 5	Application Development Case Studies Storage and File Structure
Week 6	Indexing and Hashing Query Processing
Week 7	Query Optimization Transactions (Serializability and Recoverability)
Week 8	Concurrency Control Recovery Systems Course Summarization

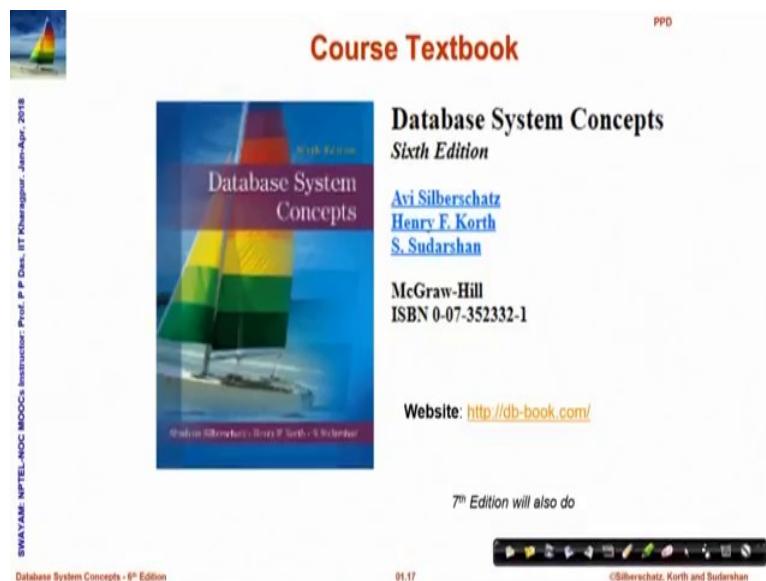
Moving on this is your course outline. So, as I said the course comprise 40 modules. So, it is divided into 8 weeks. So, this plan is given based on what we do in different weeks from week 1 to week 8. So, as the course unfolds, you will be able to we will take you through these modules on these topics. And on the right, you can find that I have marked that the initial part of the course which we cover from week one to first half of week 5 is primarily meant for application programming which it means that the database system has already been designed and basic premises the schemas and constraints have been set up.

But now you want to write different data query different data manipulation applications and that is where the large volume of database engineers work. So, they are called application programmers, so that is I should say the first level in terms of a database understanding. And you must target to become a master of application programming to get started with.

The other half of the course which start from the middle of week five with the storage and file structure and goes on till the next is meant for the analysts who are responsible either for designing a particular database which the application programmer can use tune that for performance index it properly design queries to be efficient. So, these kind of analysts will be involved the more with the understanding of the second part of the course.

And the second part of the course would also be useful for the database a DBMS designers. If you want to really become an advanced programmer we want to work on database engineering in terms of creating database management systems not merely creating databases or database applications, then you need to have a good initial command over the second half of the course. So, while you prepare for the course where you go through the modules please keep this in mind that your familiarity with the application programming must be at the highest level. And in the later parts will be relatively little advanced, but they are required for good design and good development of consistent efficient system in future.

(Refer Slide Time: 24:39)



We will follow a textbook. This is as you can see this is a sixth edition that I am following in this course. It is called Database System Concept by Silberschatz, Korth and Sudarshan. This is kind of a classical book in database systems; current version actually is the seventh edition. So, if you get access to the seventh edition, you can use that as

well, but whatever we are following in this course sixth edition is good enough. So, I advise the, that you try to get a copy of this book to yourself.

(Refer Slide Time: 25:24)

The screenshot shows a presentation slide with the following details:

- Title:** Course TAs
- Content:**
  - Srijoni Majumdar, [majumdarsrijoni@gmail.com](mailto:majumdarsrijoni@gmail.com), 9674474267
  - Himadri B G S Bhuyan, [himadribhuyan@gmail.com](mailto:himadribhuyan@gmail.com), 9438911655
  - Gurunath Reddy M, [mgurunathreddy@gmail.com](mailto:mgurunathreddy@gmail.com), 9434137638
- Navigation:** A standard presentation navigation bar with icons for back, forward, search, and other controls.
- Page Number:** 01/18
- Source:** ©Silberschatz, Korth and Sudarshan
- Page Footer:** SWAYAM - NPTEL-NOC MOOCs Instructor: Prof. P P Chakrabarti - Jatin-Agarwal

So, moving on we have different three TAs we will help you in this course Srijoni Majumdar, Himadri Bhushan Bhuyan and Gurunath Reddy. So, these are their emails; I have also put their mobile numbers. However, I would advise that unless you are really stuck avoid calling them on the mobile, because they are research students as well, busy with their own work as well, but they can certainly put all your questions on the forum, which will be promptly responded by some of these TAs or by myself. And in case you would have very specific follow ups to do, you can write email to one or all of these TAs. So, this is about the course overview.

(Refer Slide Time: 26:17)

The slide is titled "Module Summary" in red at the top right. It features a small sailboat icon in the top left corner. On the left edge, there is vertical text: "SWAYAM NPTEL-NOOC Instructor: Prof. P.P. Doshi, IIT Kanpur Date: 2016". At the bottom left, it says "Database System Concepts - 8<sup>th</sup> Edition". The bottom right shows the copyright notice "©Silberschatz, Korth and Sudarshan". The center contains two bullet points under the heading "Module Summary":

- Elucidates the importance of database management systems in modern day applications
- Introduced various aspects of the Course

So, in this module we have discussed about the importance about database management systems in the modern day applications. And we have tried to familiarize you with different aspects of the course. And I reiterate that please give due consideration to the prerequisites as mentioned, we have floated in an assignment called assignment zero where there are questions on different aspects of these prerequisites, please try to solve that assignment and see your performance. This assignment I would like to mention that this assignment will not go in the final evaluation of the performance of the course; this is just to give you an idea for self assessment of your preparedness for this course.

So, if you find that on questions on certain topics say on relation function or on data structure, if you have not been able to answer the questions well then it will be good to check back and go through the prerequisites once more. But please keep in mind that database management system is a course which depends on these background knowledge quite heavily. So, if you have gaps in understanding those topics, then all through the course you will get into several difficulties in understanding and problem solving. So, that is about our first module. So, from the next module, we will start introducing the database management system. Enjoy the course, and try to learn, try to become a good database engineer.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 02**  
**Introduction to DBMS/1**

(Refer Slide Time: 00:42)

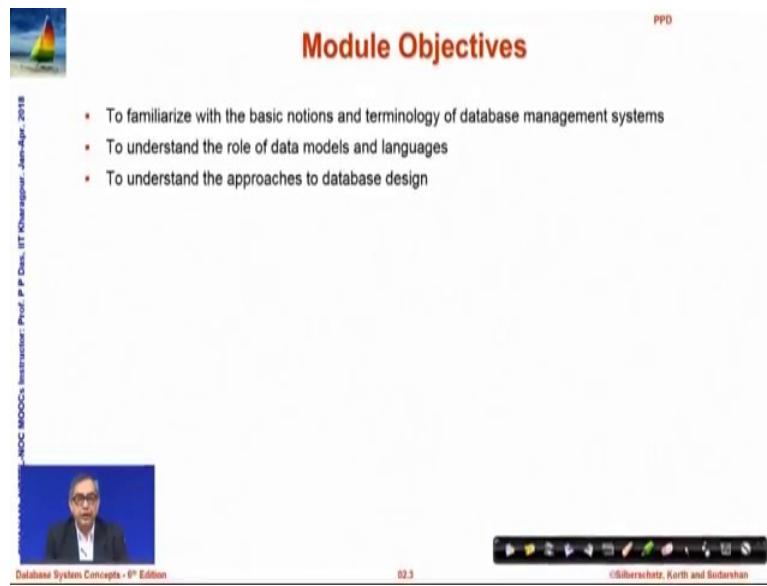
The slide is titled "Module Recap" in red at the top right. It features a small sailboat icon on the left and a "PPD" watermark in the top right corner. The main content is a bulleted list of course topics:

- Why Databases?
- KYC: Know Your Course
  - Course Prerequisite
  - Course Outline
  - Course Text Book
  - Course TAs

At the bottom, there is footer text: "SWAYAM: NPTEL-MOOCs Initiatives Prof. P. P. Das, IIT Kharagpur - June-Aug. 2018", "Database System Concepts - 8<sup>th</sup> Edition", "02.2", and "©Silberschatz, Korth and Sudarshan".

Welcome to module two of database management systems. In this module, we will talk primarily about the introduction to the DBMS. This discussion will span two modules that is the current one module 2 and the next one that is module 3. Just to quickly recap in the last module we have discussed about why we need databases, and we have introduced you to different aspects of the course.

(Refer Slide Time: 00:54)

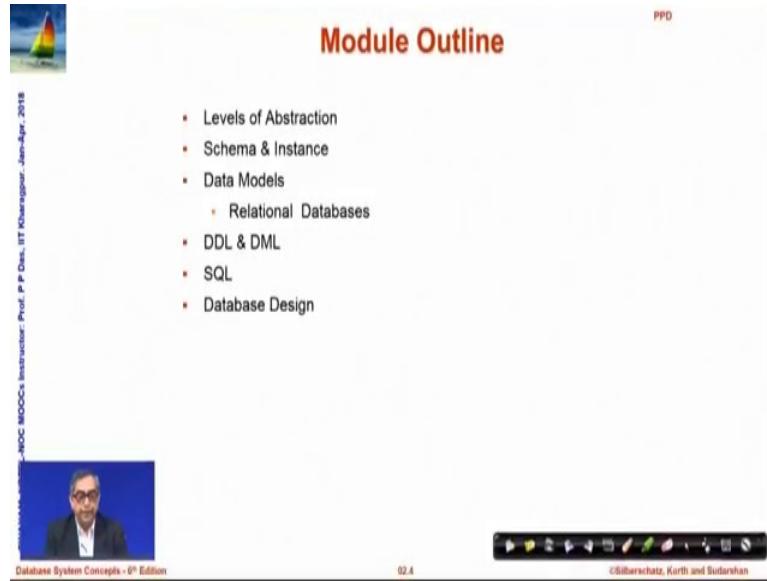


This slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. On the far left, there is vertical text: "APU MOOCs", "APU MOOCs Instructor", "Prof. P. P. Das, IIT Kharagpur", and "2018". At the bottom left is a video frame showing a man in a suit. The bottom right contains the text "Database System Concepts - 8<sup>th</sup> Edition", "02.3", and "©Silberschatz, Korth and Sudarshan". A navigation bar is at the bottom.

- To familiarize with the basic notions and terminology of database management systems
- To understand the role of data models and languages
- To understand the approaches to database design

In view of this, in this module, we would familiarize you with basic notions and terminology of a database management system just at an introductory level. We will try to understand the role of data models and specific languages for database systems. And we would also outline the approach to database design.

(Refer Slide Time: 01:21)



This slide is titled "Module Outline" in red at the top right. It features a small sailboat icon in the top left corner. On the far left, there is vertical text: "APU MOOCs", "APU MOOCs Instructor", "Prof. P. P. Das, IIT Kharagpur", and "2018". At the bottom left is a video frame showing a man in a suit. The bottom right contains the text "Database System Concepts - 8<sup>th</sup> Edition", "02.4", and "©Silberschatz, Korth and Sudarshan". A navigation bar is at the bottom.

- Levels of Abstraction
- Schema & Instance
- Data Models
  - Relational Databases
- DDL & DML
- SQL
- Database Design

So, the module outline would be like this. And as we go along you will be able to follow which particular aspect of this outline we are discussing about.

(Refer Slide Time: 01:35)

The slide features a small sailboat icon in the top left corner. On the right side, there is a vertical list of topics under the heading 'PPD': 'Levels of Abstraction', 'Schema & Instance', 'Data Models', 'DDL & DML', 'SQL', and 'Database Design'. In the center, the title 'LEVELS OF ABSTRACTION' is displayed in large red capital letters. Below the title is a small video frame showing a person's face. At the bottom, there is a navigation bar with icons and the text 'Database System Concepts - 8<sup>th</sup> Edition' and '©Silberschatz, Korth and Sudarshan'.

So, to start with we talk about levels of abstraction.

(Refer Slide Time: 01:40)

The slide features a small sailboat icon in the top left corner. The main content is titled 'Levels of Abstraction' in red. Below the title, there is a bulleted list: 'Physical level: describes how a record (e.g., instructor) is stored' and 'Logical level: describes data stored in database, and the relationships among the data'. Underneath the logical level bullet point is a code snippet for a Pascal-like language:

```
type instructor = record
    ID : string;
    name : string;
    dept_name : string;
    salary : integer;
end;
```

Following the code, there is another bulleted list: 'View level: application programs hide details of data types' and 'Views can also hide information (such as an employee's salary) for security purposes'. At the bottom, there is a navigation bar with icons and the text 'Database System Concepts - 8<sup>th</sup> Edition' and '©Silberschatz, Korth and Sudarshan'.

A database system like any other system is conceptualized in terms of three levels of abstraction. At the lower most level is the, what we say is a physical data level or the physical level which describes how a record is actually stored, so that is about the physical storage in the system. At the next level, we talk about it we say it is a logical level which describes the data stored in databases and its relationship amongst the data. So, you can any data that is stored you can think about it as a record. So, if we here we

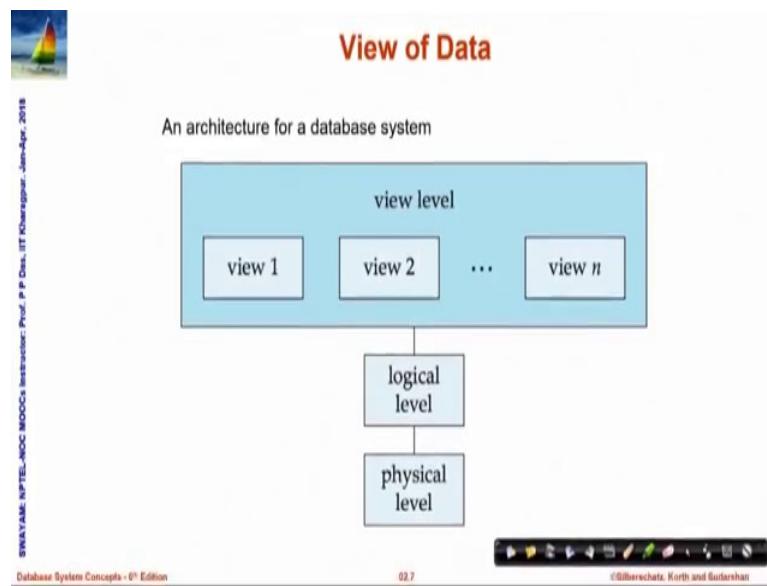
are illustrating the record of an instructor who teaches a course. So, as you know the record is a collection of multiple fields of different types. So, here we have field to describe the ID identifying number or shrink of an instructor, we have the name of the instructor, the name of the department, the salary and so on.

So, this logically says this is the entity this is a record or this is the structure of a record that I want at a logical way. So, this in contrast to the physical level, logical level is not concerned particularly with how that these data, how this string the number and all that will be actually stored, and how these multiples of hundreds and thousands of records will actually be stored so that they can be efficiently used. But we are just concerned with the logical view that I should be able to deal with records as they are.

At the third level which is it can say the topmost level is called the view level where the application program tries to view the data. And when the application program tries to view the data, it deals with the details that it needs to; and rest of the details are usually hidden from this view. For example, if here we talked about the university database in the last module, so if you are talking about the university database, and then you are a student. And you are when you access the database you should be able to see what all courses you are enrolled in or where is that course being held who is the instructor of that course and so on. But you should not be able to access or see the view of what is the instructors salary or for that matter, what are the grades that are obtained to by different students in different courses and so on.

Whereas, an instructor may be able to view the performance of the students in multiple different courses particularly the ones that he or she is involved in evaluation, but she again may not be allowed to check the salary of other instructors. So, view level is a high level where of abstraction where you try to provide the information about the data in terms of what the application needs, what the users of that application need, but you do not actually deal with the details of all the records that the logical level has.

(Refer Slide Time: 05:29)



So, these are three levels form the basic structure of a database system architecture of a database system. As you can see the physical level using that a logical level of records are formed. Physical level typically is in terms of database files is binary in nature, the organization of those files. The logical level deals with the records and the different fields of the records the schema of the database and so on.

And the view level is something which is constructed from the logical level in terms of different views that the different applications need. I am sure at this stage you may not understand the whole of these levels and their implications, but this is just to give you an idea of the existence of three levels, and the need to deal with the three levels. And as we go along, we will see that we will refer to these levels more and more and discuss about the specific aspects of those.

(Refer Slide Time: 06:27)

The slide has a header with a sailboat icon and the text 'PPD'. A vertical sidebar on the left contains the text 'SWAYAM: NPTEL-NOC MOOCs Initiator: Prof. P.P. Das, IIT Kharagpur; Admin-Help: Jahn-Bert - 2018'. The main title 'SCHEMA AND INSTANCE' is in red. Below it is a video player showing a man speaking. The footer includes 'Database System Concepts - 8<sup>th</sup> Edition', '02.8', and '©Gillberschattz, Korth and Sudarshan'.

Next, let us talk about schema and instance.

(Refer Slide Time: 06:34)

The slide has a header with a sailboat icon and the text 'PPD'. A vertical sidebar on the left contains the text 'SWAYAM: NPTEL-NOC MOOCs Initiator: Prof. P.P. Das, IIT Kharagpur; Admin-Help: Jahn-Bert - 2018'. The main title 'Schemas and Instances' is in red. The content lists components of a schema: 'Similar to types and variables in programming languages', 'Schema' (with 'Logical Schema' defined as the overall logical structure of the database, analogous to type information in a program, with examples of customers and accounts in a bank), 'Customer Schema', 'Account Schema', and 'Physical Schema' (the overall physical structure of the database). The footer includes 'Database System Concepts - 8<sup>th</sup> Edition', '02.8', and '©Gillberschattz, Korth and Sudarshan'.

We will very regularly keep on referring about schemas and instance. The schema is in a very simple terms say if we talk about first a logical schema, it is a way a certain data needs to be organized, it is a plan for organizing data. So, if you can draw a parallel then say when a building is constructed, a plan is prepared. And according to that plan several buildings in a say residential complex may be constructed. So, there is a difference between the plan which gives you the layout of where different rooms should be where

there is a staircase where is the courtier and so on and the actual building when or the group of buildings which are constructed. So, the schema primarily talks about what is the plan to organize the data.

So, if we talk about a customer schema, it has multiple different fields, it should talk about the name of the customer, ID of the customer, it is account possibly the other ID the mobile number and so on. So, the fact that these the fields need to be present for describing a customer, forms a customer schema. Whereas, when we talk about a specific schema of account that the customer holds with a bank, then we need the account number, account type, interest rate, minimum balance, the current balance and so on. These are the fields of information that we need; and we need to know what is the type of every such field, and all those and those kind of information from the schema information.

And again in line with the abstractions of physical logical and view as we did, schema also can be at a logical schema which is corresponding to the logical level of abstraction. And we may also have a physical schema which tells actually how the data is physically organized in the database, what are the different database files, how they are indexed and so on.

So, all these information which we can say is kind of a metadata information. This is not actually that it is not the customer schema is not saying who the customers are, the account schema is not saying, what are the accounts, what is their balance. But it is saying that if a customer needs to be defined; then what is the information that you need to know, what is the information that you need to manage. If an account need to be described then what is the different fields that are important. So, this schematic or this metadata is called the schema of a database.

(Refer Slide Time: 09:25)

The slide has a header 'Schemas and Instances' and a footer with course details: SYNA1101: Database System Concepts - IIT Kharagpur, Prof. P.P. Chakrabarti, Prof. A.M. Korattiri, and Prof. S. Sudarshan. The footer also includes 'Database System Concepts - 8<sup>th</sup> Edition', '02.10', and '©Silberschatz, Korth and Sudarshan'.

**Instance**

- The actual content of the database at a particular point in time
- Analogous to the value of a variable
- Customer Instance

Name	Customer ID	Account #	Aadhaar ID	Mobile #
Pavan Laha	6728	917322	182719289372	9830100291
Lata Kala	8912	827183	918291204829	7189203928
Nand Prabhu	6617	372912	127837291021	8892021892

Account #	Account Type	Interest Rate	Min. Bal.	Balance
917322	Savings	4.0%	5000	7812
372912	Current	0.0%	0	291820
827183	Term Deposit	6.75%	10000	100000

Now, based on this schema specific instances of databases happen, instances when you actually have populated different records according to the schema. Now, naturally once the schema is fixed, your records will need to have values in all of those fields that the schema has; and every value must be of the type that the particular field is specified with. So, I have just shown here certain sample instance of customer schema, where you can see three customers with their name, customer ID, account number, other ID, and mobile number, these are all fictitious data of course, but this is just to give you an idea of how this customer instance would look like.

Similarly, we have shown what is a accounts instance, so you can see that there is some kind of a relationship that you can see between these instances. For example, the first customer ID on the customer instance can be seen as a first entry in the account instance and so on. So, when we actually populate the schema with different records and this is what keeps on changing.

So, certainly when we do operations on the database, then certainly very regularly new records will get added, some records might get deleted from this instance, and fields of certain records may keep on changing. For example, in an accounts instance very regularly whenever a transaction is done, the account balance will get changed; maybe at a certain time the bank might decide to change the interest rate for a certain type of

account then the interest rate field will get changed, new customers may come into the customer instance and so on. So, instances keep on regularly being updated manipulated added deleted updated, but the schema remains unchanged.

So, change of the schema is very rare in a database and needs to be done only when the database is designed or when it is being upgraded. Because once you change the schema, it changes the way you look at the whole world, you look at the whole database scenario. So, if you are changing the schema at a logical level, then naturally the your view will also get affected, because you are using these schemas to decide how you would like to present a transaction application to the user or for a balance check application to the user and so on.

(Refer Slide Time: 12:22)

The slide has a title 'Schemas and Instances' in red. To the left is a small logo of a sailboat on water. On the right is a list of bullet points under the heading 'Physical Data Independence'. At the bottom, there is footer text: 'SWAYAM-NPTEL-ANOC-MOOCs Initiator: Prof. P. P. Desai, IIT Kharagpur', 'Database System Concepts - 8<sup>th</sup> Edition', '02.11', and 'Übersicht, Korth und Sudarshan'.

- **Physical Data Independence** – the ability to modify the physical schema without changing the logical schema
  - Analogous to independence of 'Interface' and 'Implementation' in Object-Oriented Systems
  - Applications depend on the logical schema
  - In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.

But, of course, what do we would want is between the physical schema and the logical schema we normally would want certain independence. What it means is the logical schema is what you need to deal with, because it is linked with the view that you have at a higher level of abstraction. On the other end, the logical schema is based on the physical schema; physical schema is how you are actually organizing the information in terms of the binary files the database files.

Now, certainly you want that logical schema not to change because if it changes then at a view level all your applications will have to change. But it is quite possible that your physical schema the way you have organizing files and so on might need a change,

because maybe it is just that you had designed the database for 10,000 records and you already have 9000 records and you would like to expand it to maybe 1,00,000 records.

And the physical this system needs to be different you may need to reorganize the files and so on, you may need to index it in a different way and all this, but you would like to do that change in the physical level without requiring any change at a logical schema. So, this property of a database schema is very required which we say is a physical data independence or the ability to change the physical schema without actually affecting the logical schema or the view level. So, that will be a critical factor that will have to keep in mind.

(Refer Slide Time: 14:16)

PPD

- Levels of Abstraction
- Schema & Instance
- Data Models
- DDL & DML
- SQL
- Database Design

DATA MODELS

SWATANTRUPTEL-NOC: Instrument Prof. P. P. Das, IIT Kharagpur - 2018

Database System Concepts - 8<sup>th</sup> Edition

02.12

©Silberschatz, Korth and Sudarshan

So, next is data models.

(Refer Slide Time: 14:20)

The slide has a header 'Data Models' in red. On the left, there is a small logo of a sailboat on water. The main content is a bulleted list of items related to data models:

- A collection of tools for describing
  - Data
  - Data relationships
  - Data semantics
  - Data constraints
- **Relational model** (we focus in this course)
- Entity-Relationship data model (mainly for database design)
- Object-based data models (Object-oriented and Object-relational)
- Semi-structured data model (XML)
- Other older models:
  - Network model
  - Hierarchical model

At the bottom, there is footer text: 'SYNTHOME: INFTEL-NOC: IIMODC's Infrastructure', 'Prof. P.P. Doshi, IIT Kharagpur', 'Database System Concepts - 8<sup>th</sup> Edition', '02.13', and '©Giberschattz, Korth and Sudarshan'.

Data models are a collection of tools that describe the data because we are talking about a database system. So, certainly the main focus here is to be able to model that it to be able to represent the data, so that talks about relationships between data, it talks about the meaning of the data the data semantics, it talks about data constraints. For example, it is an account balance, we just refer to account balance in the account schema in the instance, now, the question is will can the account balance be negative, the answer is yes or no.

If the bank mandates that I can only withdraw as much money up to which I have deposited, then account balance cannot be negative, but if the bank is giving me the facility to overdraft then I may be able to draw more money than I have actually deposited. So, my account balance might note negative. In some banks it could be that the bank says that well there is a minimum balance. So, minimum balance is 5,000. So, which means that my account balance cannot go below five thousand rupees the bank will not allow those transactions. So, these are examples of different constraints that might exist in the real world, and therefore, will be required in terms of the data model representation.

So, there are several data models that exist today. The most widely used the most popular and most powerful in terms of a certain section of database applications which we are commonly interested in is a relational model and that is what we focus in this course. We

will have a major discourse in terms of the relational model and lot of things will be developed based on that. But it is not easy to directly design a database in terms of the relational model, because you first need to understand the real world in which the design is happening.

So, we normally start with a less formal model known as the entity-relationship data model or an ER model, ER diagram, these are commonly called. So, if you recall your knowledge about object oriented systems, and if you happen to know UML, you already know about ER models and corresponding class diagrams that result. So, we will talk briefly about your model and show you how to do modelling on the real world in terms of the ER diagrams. But, then they are not actually models which the database systems directly used they are subsequently converted to some relational or other model and which the database systems will use.

Next is a object-based data models. You all would be knowing familiar with the fact that objects give a better power to represent the system which objects are not like simple strings or numbers or characters like that, they are encapsulated concept of an entity which can be manipulated in a certain way. So, like in the real world, you have several objects, it would have been nice to have similar objects in the databases. So, quite a few database system have been designed used which are object-based database systems. So, there are models for those. However, we will do little of that in this course, because it is little bit advanced in notion.

The other model, which has become very popular is called the semi-structured data model. It is primarily in terms of XML. I am sure all of you would be familiar with the basics of XML, which is extensible mark up language in which you can create use tags and use different mark ups to describe the data. You can say this is the field and this is the value kind of. And this is become a very nice way to represent the data. And XML or the semi-structured models are particularly useful today to exchange data between different systems.

I may be using a my SQL kind of database system, my friend may be using an oracle system, and we need to exchange data tables between these two, these two systems will represent the data in the physical schema which are not may not be compatible. So, we can represent both of them in terms of XML models convert the data. So, I convert the

data into XML, give it to my friend and my friend can import from that XML into the database. So, it is a XML is a data model which is frequently used in terms of data exchanges.

Then there are several other models like the network model, hierarchical model which used to be very popular in the early days of database systems before relational model came into force. They still exist in terms of the some of the databases. And some of the newer emerging big data databases actually we have started using this old concept in a new way again. So, this is a overall set of data model.

(Refer Slide Time: 19:58)

**Relational Model**

- All the data is stored in various tables
- Example of tabular data in the relational model

ID	name	dept_name	salary
2222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58853	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

So, here I am just showing an example of a relational model data which is simply looks like a table. So, you can see that on top row in the blue are the names of the different fields which describe the schema. It says that it has an ID, it has a name, it has a departments name, it has salary. They are trying to describe a particular instructor, and then a whole lot of records rows in that table, which are every row individually is a particular data entry or a record. So, columns are attributes, and rows are records that the relational model described.

(Refer Slide Time: 20:40)

The slide features a title 'A Sample Relational Database' at the top right. On the left, there is a small sailboat icon and a vertical watermark reading 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Doshi, IIT Kharagpur - June-Aug., 2015'. In the center, there are two tables. Table (a) 'The instructor table' has columns ID, name, dept\_name, and salary, with data from rows 22222 to 76543. Table (b) 'The department table' has columns dept\_name, building, and budget, with data from rows Comp. Sci. to Physics. At the bottom, there is a navigation bar with icons and the text 'Database System Concepts - 8<sup>th</sup> Edition'.

**(a) The instructor table**

ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Sald	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58883	Califeri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

**(b) The department table**

dept_name	building	budget
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

Some more of that the instructor table along with the department table. So, the table below describes details of a department, so that has its own schema and the individual records. We have of course seen similar instances already in terms of the customer and accounts instance that we have just discussed a couple of while ago.

(Refer Slide Time: 21:03)

The slide features a title 'DDL AND DML' at the top right. On the left, there is a small sailboat icon and a vertical watermark reading 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Doshi, IIT Kharagpur - June-Aug., 2015'. In the center, there is a bulleted list of database concepts: Levels of Abstraction, Schema & Instance, Data Models, DDL & DML, SQL, and Database Design. At the bottom, there is a navigation bar with icons and the text 'Database System Concepts - 8<sup>th</sup> Edition'.

- Levels of Abstraction
- Schema & Instance
- Data Models
- DDL & DML
- SQL
- Database Design

Let me introduce these two terms DDL and DML.

(Refer Slide Time: 21:10)

**Data Definition Language (DDL)**

- Specification notation for defining the database schema
  - Example: `create table instructor (`  
    `ID           char(5),`  
    `name        varchar(20),`  
    `dept_name  varchar(20),`  
    `salary       numeric(8,2))`
- DDL compiler generates a set of table templates stored in a **data dictionary**
- Data dictionary contains metadata (i.e., data about data)
  - Database schema
  - Integrity constraints
    - ↳ Primary key (ID uniquely identifies instructors)
  - Authorization
    - ↳ Who can access what

Syntax: INTEL-DOC Bookmarks  
Prof. Dr. Jitendra K. Joshi  
Database System Concepts - 8<sup>th</sup> Edition

02.17 ©Silberschatz, Korth and Sudarshan

DDL talks about data definition language. So, what the concept wise what we are trying to say is certainly we have a schema and we have instance. So, we need certain language constructs to be able to define a schema and certain other language constructs to be able to manipulate the data in that schema or they are basically manipulate the instances. So, DDL is the language or part of the language which is used to define and manipulate the schema of a database that is why schema is a way to define a database. So, it is called a data definition language.

So, you can define that I will am going to have a table called `instructor` which will have four different fields, each having certain types of data. So, it says that the `ID` will be a five character data; the `name` would we would have a variable length, because you cannot say that the `name` will be of a fixed length, but it will be a variable length that is what `varchar` is, but the length will not exceed 20. And similarly, `salary` will be a numeric data with up going up to 8 figures, and having a decimal part having two parts.

So, this way of defining the schema in terms of the different attributes and their types or columns in the table or trying to define the structure of that table is a main issue of the data definition language. So, the data definition language compiler who generates a set of tables in the data dictionary, where the data dictionary basically contains metadata as I said the schema is nothing but a metadata about the database tables. So, which will have the database schema, it will have different integrity constraints, it could say that well the

account balance cannot be negative or account balance cannot be less than the minimum balance. So, these are different integrity constraints. It could say that this is the primary key, we will talk more in more depth in terms of what is key. And it could also specify the authorization as to who is allowed to access which part of the data and so on, so that is these all are part of the schema definition and forms the DDL of the language.

(Refer Slide Time: 23:36)

The slide has a title 'Data Manipulation Language (DML)' in red at the top right. To the left of the title is a small logo of a sailboat on water. The main content is a bulleted list under the heading '• Language for accessing and manipulating the data organized by the appropriate data model'. This list includes: '• DML also known as query language', '• Two classes of languages', '• Pure – used for proving properties about computational power and for optimization' (with sub-points: '» Relational Algebra (we focus in this course)', '» Tuple relational calculus', '» Domain relational calculus'), and '• Commercial – used in commercial systems' (with sub-point: '» SQL is the most widely used commercial language'). At the bottom left is a video thumbnail showing a person speaking. The bottom right shows a navigation bar with icons for back, forward, search, and other presentation controls. The footer contains the text 'SWAYAM MOOCs Instructional Prod. P. P. Desai, ST Khemchand Patel', 'Database System Concepts - 3<sup>rd</sup> Edition', '02.18', and '©Baburao, Karth and Sudarshan'.

In contrast, the data manipulation language is a language for accessing and manipulating the data organized. So, it is for access, update, addition of new records, deletion of existing records and so on. And very commonly we will refer to the data manipulation language as a query language, because this is what you want to know what exists in the database. So, the query language will be designed, they are designed primarily in one of the two ways. One group of languages is known as a pure language, they are more mathematical in nature. They have a formal basis that can you can prove that whatever do you do in these languages are correct, and will give you the correct result.

So, they are different languages based on the relational model, they are called relational algebra, tuple relational calculus, domain relational calculus and so on. Of these three, we will in this course deal only with relational algebra. There are mathematical proof which show that whatever you can do in relational algebra you can do it in tuple relational calculus and vice versa. Similarly, whatever you can do in relational algebra, you can do in domain relational calculations and so on. In one sense that these languages

are equally powerful; the same thing can be done in any one of them, but we will just take the simplest of them and study here in terms of the relational algebra, but these are more mathematical representations are not easy to write as a program. So, normally we will use certain commercial query language which is called SQL for most of our applications and we will do the coding in that.

(Refer Slide Time: 25:19)

The screenshot shows a presentation slide with a title 'SQL' in red at the top right. To the left of the title is a small icon of a sailboat on water. Below the title is a bulleted list of facts about SQL:

- The most widely used commercial language
- SQL is NOT a Turing machine equivalent language
- To be able to compute complex functions SQL is usually embedded in some higher-level language
- Application programs generally access databases through one of
  - Language extensions to allow embedded SQL
  - Application program interface (e.g., ODBC/JDBC) which allow SQL queries to be sent to a database

On the left side of the slide, there is vertical text that reads: 'SWAYAM-NIPTEL-MOOCs Initiator: Prof. P. P. Doshi, IIT Kharagpur - June-Aug. 2018'. At the bottom left is a video player showing a person speaking, with the text 'Database System Concepts - 8th Edition' above it. At the bottom right is a navigation bar with icons for back, forward, search, and other presentation controls.

So, SQL which is a most widely used commercial language and mind you this is not a Turing equivalent language which means that everything that can be that need to be computed cannot be computed in SQL, there are certain computations which SQL cannot do. It is a limitation; it is a restricted language. So, often SQL is used in conjunction with some common high-level programming language like C or C++ and so on. So, whatever is there can be done in SQL in terms of data manipulation will be done in terms of the relational model, but there could be additional logic that needs to be built in, in terms of the high level language. So, application programs are typically written through them. So, we can do this through a process of embedding that is put in SQL as a part of a C program or use certain libraries which can actually take a query from C, and fire it on the SQL database.

(Refer Slide Time: 26:26)

The slide features a title 'DATABASE DESIGN' in large red capital letters. To the right of the title is a bulleted list of topics: '• Levels of Abstraction', '• Schema & Instance', '• Data Models', '• DDL & DML', '• SQL', and '• Database Design'. In the top right corner, the text 'PPD' is visible. On the left side, there is a vertical column of text: 'SYNTHAINE, INFLTEL-NOC, MOOCs Institute', 'Prof. P.P. Drs. IIT Kharagpur - June-Apr. - 2018', and 'Database System Concepts - 8<sup>th</sup> Edition'. At the bottom left is a small video thumbnail showing a man speaking. The bottom right contains a navigation bar with icons and the text '©Güberachitz, Korth and Sudarshan'.

So, we will see how to do this in the course of time.aspect.

(Refer Slide Time: 26:32)

The slide has a title 'Database Design' in red. Below the title is a definition: 'The process of designing the general structure of the database:'. A bulleted list follows, divided into two main categories: '• Logical Design – Deciding on the database schema.' and '• Physical Design – Deciding on the physical layout of the database.' The first category includes sub-points: 'Business decision' (with 'What attributes should we record in the database?'), 'Computer Science decision' (with 'What relation schemas should we have and how should the attributes be distributed among the various relation schemas?'). The second category includes a single point: 'Physical Design – Deciding on the physical layout of the database'. The slide's layout is identical to the previous one, with the same sidebar text and footer elements.

Coming to the database design this is a process through which the databases need to be designed. And certainly the first part of the design is the logical design where you want you need to identify what are the schemas and you know what are the constraints that apply, what is authorization required. And first set of decisions those are related to the business as we say. Business means it is basically comes from the domain. So, it is if I am doing a university database, the business decisions will come in terms of you know I

have courses, students, instructors, and the instructor teach courses, can an instructor teach multiple courses, can a course be taught by multiple instructors these kind of business decisions are critical for the database design.

And then there is a whole set of computer science decision or the data based decisions to decide as to if this is the kind of business information that you want to keep in the database, then what is the kind of relation, what is the kind of schemas that we should use what are should be the attributes, which attribute should be of what type what should be strain, what should be numbers and so on. So, these are formed the basis of the physical logical design. And of course, we then need a physical design which decides on the physical layout of the data, what are the different database files, how they should be indexed and so on.

(Refer Slide Time: 27:53)

**Database Design (Cont.)**

- Is there any problem with this relation?

ID	name	salary	dept_name	building	budget
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

So, here we are showing an example table. So, it has multiple fields. It shows the instructors expanded form of the instructor table you saw earlier. It is expanded with the departments name and the building in which it is housed. So, if you look carefully that this certainly comes from the business decision that you need to know the department to which an instructor belongs and certainly you need to know the building in which that department exists. So, knowing the department of the instructor and the building of where that department is are critical, but the question is this a good design, is this so we

will discuss as to when why this is a good, this may not be a good design to represent the data.

(Refer Slide Time: 28:41)

The slide is titled "Module Summary" in red at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list of learning objectives:

- Familiarized with the basic notions and terminology of database management systems
- Introduced the role of data models and languages
- Introduced the approaches to database design

At the bottom left, there is a small video thumbnail showing a man speaking. The text "Database System Concepts - 8<sup>th</sup> Edition" is visible below the thumbnail. The bottom right corner shows a navigation bar with icons for back, forward, search, and other presentation controls. The overall background is white with a thin vertical border on the right side.

So, in this module, we have taken you through the basic notions and terminology of database management systems, highlighting primarily the levels of abstraction, the schema an instance, the basic data models the languages that you need DDL, DML and the commercial SQL language. And we have also tried to give you a glimpse of the approach that is required in terms of the database design. We will elaborate on this more in the second part of our introduction to DBMS which will be taken up in module 3.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 03**  
**Introduction to DBMS/2**

Welcome to module 3 of Database Management Systems course. We started discussions introducing the database management systems in module 2. This is the second and concluding part of that discussion.

(Refer Slide Time: 00:37)

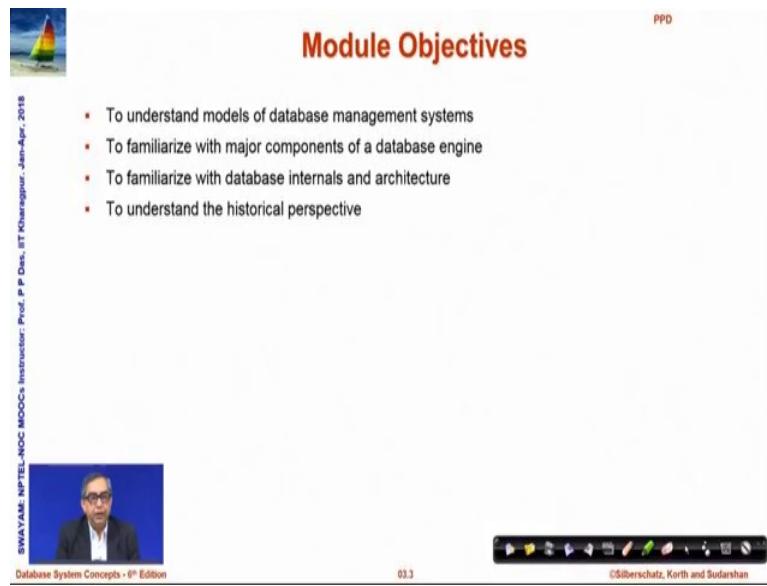
**Module Recap**

- Levels of Abstraction
- Schema & Instance
- Data Models
  - Relational Databases
- DDL & DML
- SQL
- Database Design

SWAYAM: NPTEL-NOC  
Prof. P P Das, IIT Kharagpur - Instructor  
Database System Concepts - 8<sup>th</sup> Edition  
©Bilberschatz, Korth and Sudarshan

So, this is what, these are the aspects that we are discussed earlier starting from level of abstraction to the outline of database design.

(Refer Slide Time: 00:47)



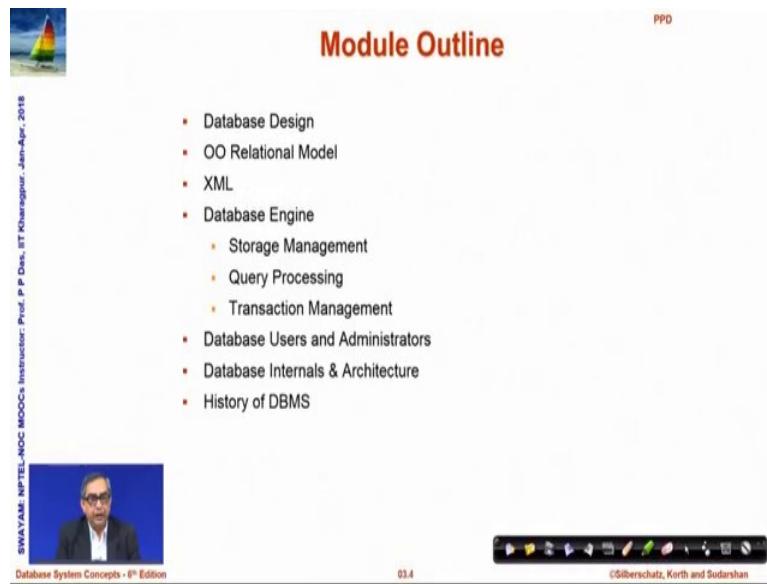
This slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. On the far right, there is some very small, illegible text. The main content is a bulleted list of objectives:

- To understand models of database management systems
- To familiarize with major components of a database engine
- To familiarize with database internals and architecture
- To understand the historical perspective

At the bottom, there is a video frame showing a person speaking, the text "Database System Concepts - 8<sup>th</sup> Edition", the number "03.3", and the copyright notice "©Silberschatz, Korth and Sudarshan".

In the current module, we would like to understand the modules of database management systems little bit more and we will try to familiarize with the concept of major components of database engine, will elaborate on those and will familiarize. So, with the basic architecture of a database management system, some of the internal components and we will present a brief historical perspective of the DBMS.

(Refer Slide Time: 01:21)



This slide is titled "Module Outline" in red at the top right. It features a small sailboat icon in the top left corner. On the far right, there is some very small, illegible text. The main content is a bulleted list of outline items:

- Database Design
- OO Relational Model
- XML
- Database Engine
  - Storage Management
  - Query Processing
  - Transaction Management
- Database Users and Administrators
- Database Internals & Architecture
- History of DBMS

At the bottom, there is a video frame showing a person speaking, the text "Database System Concepts - 8<sup>th</sup> Edition", the number "03.4", and the copyright notice "©Silberschatz, Korth and Sudarshan".

So, this is the outline that we will follow.

(Refer Slide Time: 01:25)

The slide features a title 'DATABASE DESIGN' in large red capital letters at the top center. To the right of the title is a vertical list of topics:

- Database Design
- OO Relational Model
- XML
- Database Engine
- Database Users and Administrators
- Database Internals & Architecture
- History of DBMS

At the bottom left, there is a small video player window showing a man speaking. The video player has a progress bar labeled '03.5'. At the very bottom of the slide, it says 'Database System Concepts - 8<sup>th</sup> Edition'.

(Refer Slide Time: 01:34)

The slide features a title 'Database Design' in large red capital letters at the top center. Below the title, a definition is provided: 'The process of designing the general structure of the database:'. The slide is divided into two main sections:

- **Logical Design**
  - Deciding on the database schema. Database design requires that we find a "good" collection of relation schemas.
  - Business decision
    - What attributes should we record in the database?
  - Computer Science decision
    - What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
- **Physical Design**
  - Deciding on the physical layout of the database

At the bottom left, there is a small video player window showing a man speaking. The video player has a progress bar labeled '03.6'. At the very bottom of the slide, it says 'Database System Concepts - 8<sup>th</sup> Edition'.

(Refer Slide Time: 01:46)

So, we have already discussed about the database design, I would like to raise a few further issues about that. So, we have seen that, there is a logical design which is driven by the business decisions and refined by the computer science decisions, there is a physical design as well; and based on that we had presented this particular table asking, whether, this database is a good design or not.

So, let us have a little look into this for example, we have introduced the department name and the building in which the department is housed. So, if we look at there are multiple instructors say, let us say Professor Einstein, who teaches in the Physics department, that is housed in the Watson building and if we look through there is a Professor Gold, who also teaches in the Physics department and naturally that is housed in the Watson building.

Now, the question is; so, Physics department, if it is housed in the Watson building then, all the instructors in this table, all the instructors who are part of the Physics department, would have their department housed in the Watson building. So, there is a certain issue of redundancy in these two, that is, the same information is given more than once, which is not a very desirable thing.

The consequence of this could be suppose, tomorrow the university decides to move this Physics department from Watson to the Taylor building. This will mean that once this is moved, then this Watson will have to be changed to Taylor, also this Watson will also have to be changed to Taylor. All instances of Watson that corresponded to the Physics

department in this table, will have to be changed to Taylor, and that is not a good scenario. So, it is not only that we have redundancy, we have potential for anomaly; that is program the application programmer might forget to update the building say, for this entry then, we will be in an inconsistent database. So, to put it in simple terms that this is not a good design and there are several issues to consider, in terms of whether some design is good or some design needs refinement.

(Refer Slide Time: 04:22)

The slide has a title 'Design Approaches' in red at the top right. To its left is a small icon of a sailboat on water. On the far left, there is vertical text: 'SWAYAM: NPTEL-MOOCs Instructor: Prof. P. P. Doshi, IIT Kharagpur - Jan-Apr - 2015'. At the bottom, there is footer text: 'Database System Concepts - 8<sup>th</sup> Edition', '03.8', and '©Silberschatz, Korth and Sudarshan'.

**Design Approaches**

- Need to come up with a methodology to ensure that each of the relations in the database is "good"
- Two ways of doing so:
  - Entity Relationship Model (Chapter 7)
    - Models an enterprise as a collection of *entities* and *relationships*
    - Represented diagrammatically by an *entity-relationship diagram*:
  - Normalization Theory (Chapter 8)
    - Formalize what designs are bad, and test for them

So, we need to come up with a methodology to ensure that, each of the relations in the database is good. So, we primarily follow two approaches in doing this. One is, using the entity relationship model, which models the enterprise as a collection of entities or concepts or if you are familiar with the object orientation classes and the relationships that hold between these entities.

So, in a university database the entities are students, courses, teachers and the relationships are a teacher teaches a set of courses, a student attends a set of courses and so on, the teachers supervise a set of students for projects and so on and then represent them diagrammatically in terms of a ER diagram entity relationship diagram and once that has been done then, we try to follow a certain normalization theory.

This normalization theory tries to capture that what are the properties that must hold in this database design, that must be satisfied on this database design, in terms of what is known as database dependencies, there are, varied forms of dependencies functional

dependencies, multi value dependencies, joint dependencies and so on and try to formalize and evaluate whether a design is good or it is bad, test them for quality and then normalize to make them better; make them the best possible that can happen.

So, this is something that is starting from the entity relationship model, which captures the real world to the actual database schema, there is a process of representation and then capturing of ground truths, that should hold in the database system and then normalize the database is a basic requirement of the design approach.

(Refer Slide Time: 06:34)

The slide features a small sailboat icon in the top left corner. In the top right corner, the letters 'PPD' are displayed above a list of topics: Database Design, OO Relational Model, XML, Database Engine, Database Users and Administrators, Database Internals & Architecture, and History of DBMS. The main title 'OBJECT-RELATIONAL DATA MODELS' is centered in large red capital letters. At the bottom, there is footer text: 'SWAYAM: NPTEL-NOOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jam-Apr- 2015', 'Database System Concepts - 8<sup>th</sup> Edition', '03.9', and '©Silberschatz, Korth and Sudarshan'.

(Refer Slide Time: 06:39)

The slide features a small sailboat icon in the top left corner. The main title 'Object-Relational Data Models' is centered in large red capital letters. Below the title is a bulleted list of points: Relational model: flat, "atomic" values; Object Relational Data Models; Extend the relational data model by including object orientation and constructs to deal with added data types; Allow attributes of tuples to have complex types, including non-atomic values such as nested relations; Preserve relational foundations, in particular the declarative access to data, while extending modeling power; Provide upward compatibility with existing relational languages. At the bottom, there is footer text: 'SWAYAM: NPTEL-NOOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jam-Apr- 2015', 'Database System Concepts - 8<sup>th</sup> Edition', '03.10', and '©Silberschatz, Korth and Sudarshan'.

We have talked about object relational data models for a few more points about that, that in a relational model everything is flat, every value is atomic, in the sense that everything if you, look back and think in terms of C then, every field is a value which can be a simple, you know, built in type like integer, like fixed length string, variable length string, a floating point number like that, but I cannot have a composite you know, object kind of fields.

But in a relational data model we extend in the object relational data model, we extend the relational model by including the object orientation and the constituent constructs to deal with added data types, higher data types where attributes are allowed to have complex types, non atomic values that may allow things like nested relation; that is a value could itself be a relation, could itself be a table but, we try to preserve the relational foundation and we will see what those foundations mean and provide upward compatibility to existing relational databases. So, this is what the basic concept of object relational data models are and as I said that, we will just glimpse through it, but this is not the primary objective that we will try to cover.

(Refer Slide Time: 08:04)

The slide has a white background with a small sailboat icon in the top-left corner. On the right side, there is a vertical list of topics:

- Database Design
- OO Relational Model
- XML
- Database Engine
- Database Users and Administrators
- Database Internals & Architecture
- History of DBMS

At the bottom, there is a decorative footer bar with icons for navigation, search, and other presentation functions. The footer also contains the text "Database System Concepts - 8<sup>th</sup> Edition", "03.11", and "©Silberschatz, Korth and Sudarshan".

(Refer Slide Time: 08:07)

The slide has a header 'XML: Extensible Markup Language' with a sailboat icon. The main content is a bulleted list of facts about XML. At the bottom left is a video player showing a man speaking, and at the bottom right is a navigation bar.

• Defined by the WWW Consortium (W3C)  
• Originally intended as a document markup language not a database language  
• The ability to specify new tags, and to create nested tag structures made XML a great way to exchange **data**, not just documents  
• XML has become the basis for all new generation data interchange formats  
• A wide variety of tools is available for parsing, browsing and querying XML documents/data

In contrast, the XML extensible markup language was defined by W3C, and it was originally intended for marking up document languages. It was not designed as a database language, it was designed for marking up. So, it is kind of saying that this particular element should be put in capital, this should be in blue colour, this means a verb, this means a paragraph, there should be a page break here, those kind of markups, But subsequently, it turned out that the way XML deals with different components in terms of tags, and the ability to create nested tags, makes a great language for exchange of data, As I explained in the last module also.

So, has become the basis for all kinds of new generation data interchange format. So, as I explained, that any database should be able to convert the data instances of the tables in terms of corresponding XML format and then you take it to some other database, in which you are intending to interchange the data and that target database should be able to import from that XML structure, and it has become widely available that you have different tools for parsing, browsing and querying XML content document data and so on. So, if you are familiar with C programming, I hope so you are! You can look up certain XML parsing and try out, there are great tools to learn.

(Refer Slide Time: 09:53)

The slide features a title 'DATABASE ENGINE' in large red capital letters at the top center. To the left of the title is a small video window showing a man speaking. On the right side, there is a vertical list of topics under the heading 'PPD': Database Design, OO Relational Model, XML, Database Engine, Database Users and Administrators, Database Internals & Architecture, and History of DBMS. The bottom of the slide includes a navigation bar with icons and the text 'Database System Concepts - 8<sup>th</sup> Edition', '03.13', and '©Silberschatz, Korth and Sudarshan'.

Moving on, let us briefly look at what is the core of a database management system, the database engine.

(Refer Slide Time: 10:03)

The slide has a title 'Database Engine' in red at the top center. Below the title is a bulleted list of three components: Storage manager, Query processing, and Transaction manager. To the left of the list is a video window showing a man speaking. The bottom of the slide includes a navigation bar with icons and the text 'Database System Concepts - 8<sup>th</sup> Edition', '03.14', and '©Silberschatz, Korth and Sudarshan'.

The database engine, primarily contains 3 major components; the storage manager, the query processing engine, sub engine and the transaction manager.

(Refer Slide Time: 10:14)

The slide has a header 'Storage Management' with a sailboat icon. On the left, there is vertical text: 'SWAYAM, NPTEL-NOOC, IIT-Kharagpur', 'Prof. P. P. Desai', and 'Jain-Apr.-2018'. The main content is a bulleted list:

- **Storage manager** is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible to the following tasks:
  - Interaction with the OS file manager
  - Efficient storing, retrieving and updating of data
- Issues:
  - Storage access
  - File organization
  - Indexing and hashing

Below the list is a video player showing a person speaking. The video player includes controls like play, stop, and volume, and shows 'Database System Concepts - 8<sup>th</sup> Edition' and '03.15'. At the bottom right is a copyright notice: '©Siberschatz, Korth and Sudarshan'.

The storage manager, is a module or a collection of modules in a database management system, that provide the interface between the low level data and the application program. So, we have looked at the storage manager is the one, which is a bridge between the physical level of abstraction and the logical level of abstraction, then finally, to the view level of abstraction. So, the storage manager has to deal with interactions with the operating system on which the DBMS is kept, the file manager of the operating system, it is responsible for efficient storage, retrieval update of the data, it is responsible to make sure that if there are certain problems in the file system, then the data is not corrupted and so on.

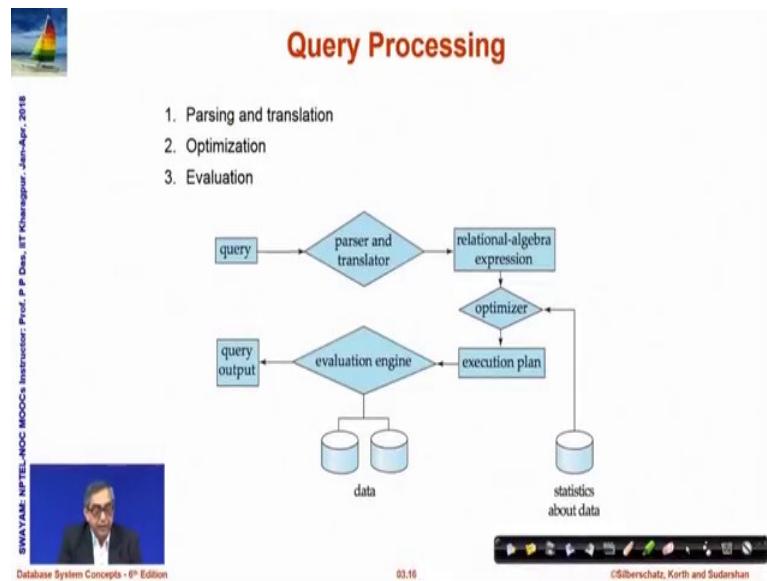
So, the issues certainly that involve are :- the access to the storage, the organization of the files and very importantly indexing and hashing and we will talk about the concept of indexing later in the course. It primarily says that, if I want to, for example, you can simply understand that if you have a large chunk of data that you want to organize for efficient search then, you can use the binary search tree in simple algorithm terms. The binary search tree needs to be organized in terms of one data component.

We say that, well there is one value based on which you can say that, comparison is done. So, that at every node if that value is smaller, you go to the left sub tree, if that value is larger, you go to the right sub tree and so on. So, if we want to organize the

records of a database system in terms of such a binary search tree, then the question certainly is, which field do I use for the search tree comparison.

Now, whatever field I used for search tree comparison, on that field the searching would be very efficient, but if I want to search on a value for a different field, the searching would not remain that efficient. So, indexing is a mechanism by which, you can actually create auxiliary search trees on multiple fields. So that, the search on multiple fields can be made efficient and we will talk about this later when, that particular module comes up, but the storage manager has to deal with such issues.

(Refer Slide Time: 12:43)



Moving on, the query processing is, if we have already talked about the language; the DDL, the DML, the query language. So, it is some kind of, like the C program, it is some kind of a text based programming code. So, naturally that code needs to be parsed and translated, as we typically do in a C compiler. So, there needs to be a query compiler. So, it parses and analyses the code, but translated unlike the C program, which translates the C program into an intermediate code and then into the binary instructions of the machine, the assembly binary instructions of the machine.

The query translator, translates the query into relational algebra expressions. I said that, there are two kinds of languages :- the commercial query language and the pure language. So, it translated in terms of a program in the pure language, which could be a

relational algebra language and then it tries to optimize. So, that is a critical term to be noted that there is an optimizer.

So, this optimizer is a critical component, which tries to make sure that the query, when it is run on your data will run with the most you know, least amount of time in an effective manner. So, then an execution plan needs to be decided, we will be able to understand this when we go to the actual relational algebra execution plan, basically says that if, there are multiple operations in that query to be performed, then how those operations, in which order they should be performed and where should temporary tables be used, where they should be skipped and so on and then, once that has been done then it passes on to an evaluation engine which actually runs that query on the data that you have, the instances of the data that you have, and that brings out the resultant query output which is another table of results that we get. So, this query processing is a core part of a database engine, which actually allows us to write text based queries and relative data efficiently, change-update data efficiently, insert data efficiently, and so on.

(Refer Slide Time: 15:12)

The slide features a title 'Query Processing (Cont.)' in red at the top right. To the left is a small image of a sailboat on water. The main content is a bulleted list of points:

- Alternative ways of evaluating a given query
  - Equivalent expressions
  - Different algorithms for each operation
- Cost difference between a good and a bad way of evaluating a query can be enormous
- Need to estimate the cost of operations
  - Depends critically on statistical information about relations which the database must maintain
  - Need to estimate statistics for intermediate results to compute cost of complex expressions

At the bottom left is a video thumbnail showing a person speaking. The bottom right contains navigation icons and the text 'Übersicht, Korth und Sudarshan'.

So, when we do this, we need to look at alternative ways of evaluating a query. There could be different ways to write the same thing, these are called equivalence expression, equivalent expressions and what are the good algorithms for doing each and every operation, there is a cost between good and bad way of evaluating. So, this has to be understood that, the same thing you can compute in a, you have seen this similar

concepts in normal programming language also, I mean we have seen for example, for sorting the several ways to sort and some are better some are not as efficient.

So, if the similar things in terms of a query need to be evaluated and the cost between good and bad ways need to be figured out. So, then we need to estimate the cost of every operation. It depends on the information of what has happened in the past, the statistical information and need to estimate those statistics for intermediate results. These are a couple of things that the query processing sub engine in a database will do.

(Refer Slide Time: 16:36)

The slide has a title 'Transaction Management' in red. To the left of the title is a small sailboat icon. On the right side of the slide, there is a vertical column of text that reads:  
SWATAMI, INFLTEL-NOC, MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jain-Agarwal  
Database System Concepts - 8<sup>th</sup> Edition  
03.18 ©Silberschatz, Korth and Sudarshan

- What if the system fails?
- What if more than one user is concurrently updating the same data?
- A **transaction** is a collection of operations that performs a single logical function in a database application
- **Transaction-management component** ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.
- **Concurrency-control manager** controls the interaction among the concurrent transactions, to ensure the consistency of the database.

At the bottom of the slide is a video player interface showing a video of a person speaking. The video player includes controls like play, pause, and volume, along with the text 'Database System Concepts - 8<sup>th</sup> Edition' and '03.18 ©Silberschatz, Korth and Sudarshan'.

So, beyond the storage manager and the query processor we have a transaction management system, which is very critical and core of the database system. It primarily has to deal with two fundamental issues of a database. One, what if a system fails; see, database systems are unlike the programs that you have written so far. A program starts execution ends, the program always deals with transient data, the data did not exist before your program started, it ceases to exist after your program ends. So, a program; however, complicated; however, important has a limited lifetime.

A database in contrast, has a much longer lifetime which deals with persistent data, that is very important to understand ,that is the each application whether I am doing a bank fund transfer, whether I am making a credit card payment, to whether I am checking the balance or I am booking a railway ticket, whether, I am purchasing a book from Amazon, each one of the applications are like the normal program, it has a fixed lifetime.

If I start it, I do certain operations, I am done with it, but the data that is behind it the data of my accounts, my account balance, my transactions, my different bank charges, all that need to stay on and on and on and beyond every particular operation that I have done on the database. So, which means that if this database system fails, at some stage for some reason, then we have an enormous impact of that and that is not something that we can absorb that something that we can accept.

So, a database system has to come with the concept of recovery. It must be possible, if the system fails, it must be possible to recover it, to a certain earlier point where it is consistent. So, transaction management system is responsible to guarantee this kind of recover ability of databases. Then, the other question that we have discussed about earlier also, is a multiple users are you accessing the same database, the same set of data, at the same time. So, what how to make sure that more than one user can concurrently use an update without the data getting inconsistent, that is as I had mentioned, there is only one seat available, one berth available on a particular train, on a particular date and two users at the same time has initiated a booking.

It should not happen, that both of them get the booking. So, one should get, one should not get and that needs to be the complexity is high, for this kind of you know, decisions because in the databases, as applications are significantly distributed, Indian railways have no idea of who is going to do what booking, of which berth, from where at, which point of time. So, transaction management system is, as the name suggests defines something called, a transaction which always keeps the database consistent and operable. So, a transaction is a collection of operation, that performs a single logical function in a database application. This is very critical. It is a collection of operations and performs a single logical function.

So, it does not do anything and everything, it just does a single particular logical operation and that is what forms the transaction. So, a transaction management system is a component, that ensures that with all these transactions happening, hundreds and thousands of transactions happening every second in a database system, in a typical database system; the data should still remain consistent, in a consistent state, in spite of failures, in spite of concurrences, it must make sure that at no point of time it should happen that, an amount has been debited from an account and has not been credited to account or an amount has been credited to an account and has not been debited to the

corresponding account or the same seat same berth is booked by two persons at the same time and so on, so forth. So, this also includes concurrency control manager, which controls the interaction among different concurrent transactions, which ensures the consistency in the database and provides the total safety.

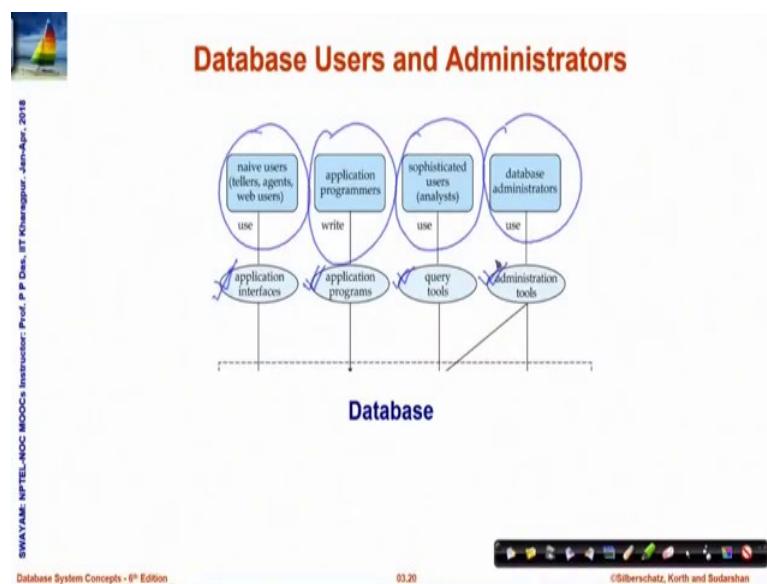
(Refer Slide Time: 21:32)



The slide features a small sailboat icon in the top left corner. On the right side, there is a vertical list of topics under the heading 'PPD': Database Design, OO Relational Model, XML, Database Engine, Database Users and Administrators, Database Internals & Architecture, and History of DBMS. The main title 'DATABASE USERS AND ADMINISTRATOR' is centered in large red capital letters. Below the title is a navigation bar with icons for back, forward, search, and other presentation controls. The footer contains the text 'Database System Concepts - 8<sup>th</sup> Edition', the date '03.19', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, in total, we have seen the different components of the database engine comprising the storage manager, comprising the query processor, and the transaction manager. Now, we will just have a quick look in terms of who are the typical users of a database system.

(Refer Slide Time: 21:57)



So, if we see grossly, the users of a database system can be grouped into, I mean you can group it in multiple different ways, but this is a typical way to group that, you have the naive users, those like the secretarial staff, who sits at the teller of the bank. Now, that person does not know database management system, but that person just needs to know the particular application. He knows a few set of screens; graphical screens, what needs to be filled up, where which button needs to be clicked and so on and can use this database through an application interface.

So, this is a lowest level of user. Then, you have the set of application programmers, about whom I talked about in my course overview presentation, that application programming is a big chunk of you know, IT services that databases need, who actually write the application programs, while the naive user is similarly using it application programmers are responsible for writing coding this application program. So, they need to understand the database designs they need to understand how to write the query language, how to fit with the application data, input, output all the systems.

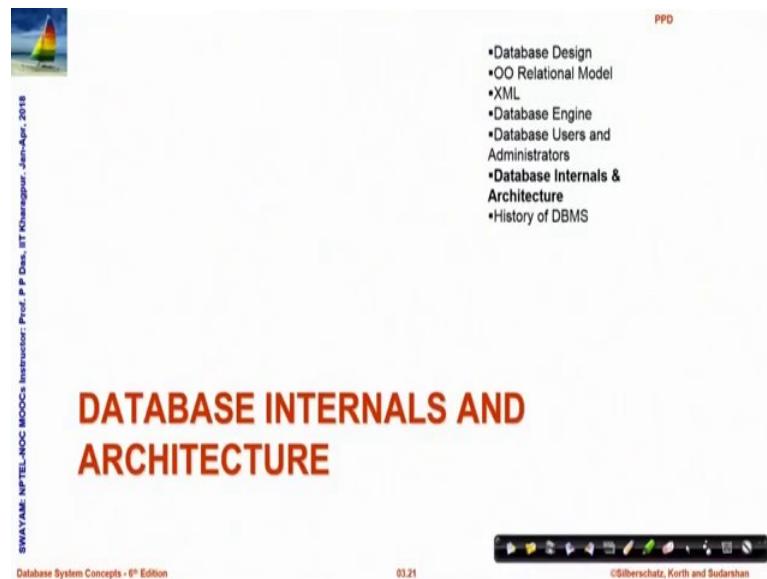
The next levels are the analysts, who are called the sophisticated users. So, they design different kinds of query tools, they are responsible for the design of the database, that is a schema the different constraints, the authorizations and so on and manages that over a period of time, when the application requirements change they might need to redesign the schema, migrate the data from an old schema to a new schema. So, analysts are higher level of programmers, they have far more solid understanding of the database management system to be able to design different kind of query tools that, the application programmer will use and at the end there are database administrators.

So, database administrators are people with specialized rights, who can do a lot of privileged operation on the database. For example, taking backups of databases, for example, creating different users. For example, if there has been a failure then how to do the failure recovery scripts, recovering the database and so on. So, they do all kinds of administration tasks, but not the usual day to day data maintenance and you know, query processing and so on.

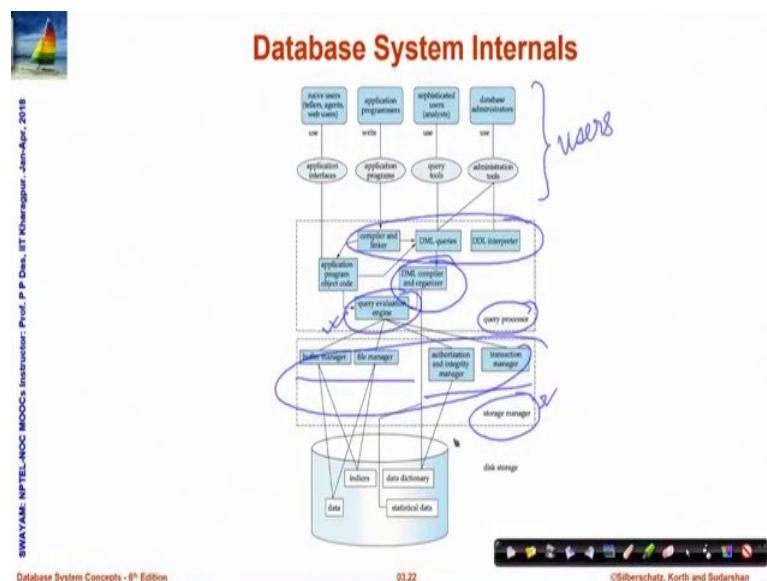
So, if you would like to know your positions then I would say that by this course, you are going to position yourself amongst the application programmers and the analysts and as I mentioned that the first half of the course is focused on application programming aspect

and the second half would be more focused on the analysis and some of the administrative will do little bit of administration, but not nearly serious administration tasks.

(Refer Slide Time: 25:33)



(Refer Slide Time: 25:35)

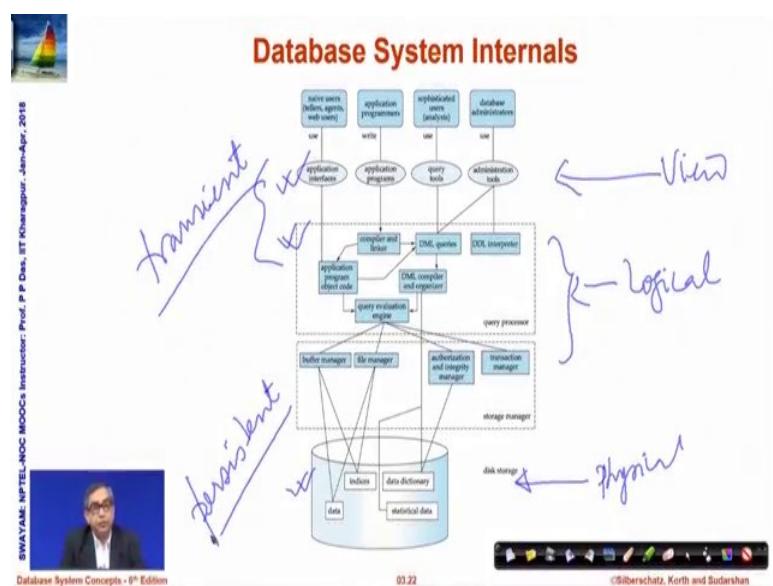


Now, we will take a quick look into the database internals and architecture. So, I will take you to this diagram. I am sorry this diagram is little small, in terms of the script size. So, please refer to the actual presentation. So, if you look at the top here is the users. So, it is trying to show what different users use. This is a query processor, that we have

known. So, the query processor gets a query and so that naturally, this query comes from the application program. So, the compiler link these are the basic processing, then the compiler organization, the evaluation engine which actually takes care of the processing of the whole query and then it goes to the storage manager which is now taking this whatever the evaluation engine needs to do, has to go through different modules in the storage manager, which we will talk about these modules, when we do have a discussion module on the storage management.

Later in this course, and we will then talk about what is a file manager and what is authorization and so on, but these are the sub components of the storage management needs to do and then the storage manager is the only one who deals with the actual data, the actual disk storage the different files and so on.

(Refer Slide Time: 27:05)



So, as you can see that, the whole system is kind of layered in terms of so, this is your basic physical layer that you have, and this is your final view layer that you have and in between this is a logical layer, that you deal with. So, you can see that the abstractions, as we had talked about are also mapped in terms of them, way the actual database system architecture is managed and finally, the data stays in the disk storage which ensures that whatever data I have is actually persistent in nature. It does not go away. Any data that stays within here or within the application interfaces transient.

So, these data are transient, they will disappear as soon as the application is over, but these data are persistent, they exist beyond this and architectures supports that whole gamut from of all applications or transiency of the applications based on the persistence of the data at the storage. So, that is a basic architectural view of a typical database systems. The actual architecture we form more complex, but we just want to take a schematic view. So, that we can understand it better.

(Refer Slide Time: 28:24)

The slide features a title 'History of Database Systems' in red font, accompanied by a small sailboat icon. Below the title is a bulleted list of historical milestones. On the left edge of the slide, there is vertical text indicating the source: 'SWAYAM: NPTEL-MOOCs Instructor: Prof. P. P. Desai, Jai Narayan Kharatmalkar - Jan-Apr - 2018'. At the bottom, there is footer text: 'Database System Concepts - 8<sup>th</sup> Edition', '03.25', and '©Silberschatz, Korth and Sudarshan'.

- 1950s and early 1960s:
  - Data processing using magnetic tapes for storage
    - Tapes provided only sequential access
  - Punched cards for input
- Late 1960s and 1970s:
  - Hard disks allowed direct access to data
  - Network and hierarchical data models in widespread use
  - Ted Codd defines the relational data model
    - Would win the ACM Turing Award for this work
  - IBM Research begins System R prototype
  - UC Berkeley begins Ingres prototype
  - High-performance (for the era) transaction processing

So, the actual architecture may again vary, based on the computer system that you are using. It could be centralized, we will talk about some of these at later modules. Centralized in the sense that there could be one database server or you know, a group of database servers at the same physical location connected together to which, all applications, all users will connect to, it could be in terms of a client server model which is a very typical client server model, that the programming systems are. So, that typically for example, any of the net based internet based database applications we are looking at are necessarily client server in nature.

What you are doing in the browser is a client and there is a server back far back there the multiple tiers in between them. Databases could be single processes or for performance they could be in terms of multiprocessor parallel databases, the data sets themselves could be so large that, it may not be possible to search on them through a single

processor, in a reasonable time, they could be distributed the data itself could be distributed, tables could get so large that I may not be able to keep them at a single point.

So, these are different aspects of you know, complex real life database system that exist and we will deal with some of those aspects over the course of time, but you have to keep in mind that a final architecture of a database and its associated application will have some of these factors that you will need to know and maybe decide on in terms of history well and I will not go through each and every point here. This is more for completeness and to give you an essence of how things have been going. So, database system started in the 50's and early 60's, then major developments of these relational models and all that started happening in the 70's.

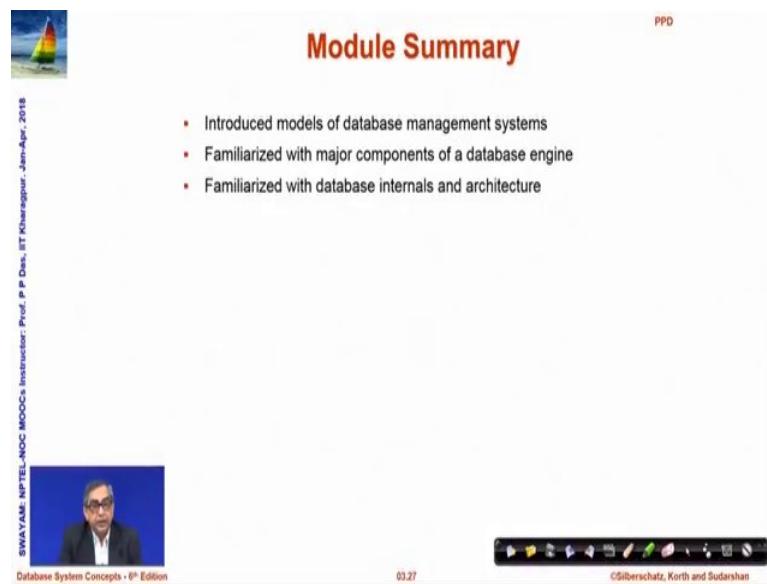
(Refer Slide Time: 30:36)

The slide is titled "History (cont.)" in red. It features a vertical timeline on the left with a sailboat icon at the top. The timeline is divided into four main periods: "1980s:", "1990s:", "Early 2000s:", and "Later 2000s:". Each period has a list of developments. At the bottom, there is a navigation bar with icons for back, forward, search, and other presentation controls.

- 1980s:
  - Research relational prototypes evolve into commercial systems
    - SQL becomes industrial standard
  - Parallel and distributed database systems
  - Object-oriented database systems
- 1990s:
  - Large decision support and data-mining applications
  - Large multi-terabyte data warehouses
  - Emergence of Web commerce
- Early 2000s:
  - XML and XQuery standards
  - Automated database administration
- Later 2000s:
  - Giant data storage systems
    - Google BigTable, Yahoo PNuts, Amazon, ..

And in 80's, really it proliferated large in terms of prototypes and commercial systems, parallel distributed systems, object based systems started happening in 80's 90's. It really exploded in terms of large decision support, data mining applications, you know applications widespread use of internet based data applications and systems emergence of Google and all that. From 20's, early in early 2,000 it has been XML and automated database administration and now what we are facing in at the big data, which are certainly aspects of other courses, but at the back of back of back at the last layer then often is a strong relational system that exists.

(Refer Slide Time: 31:28)

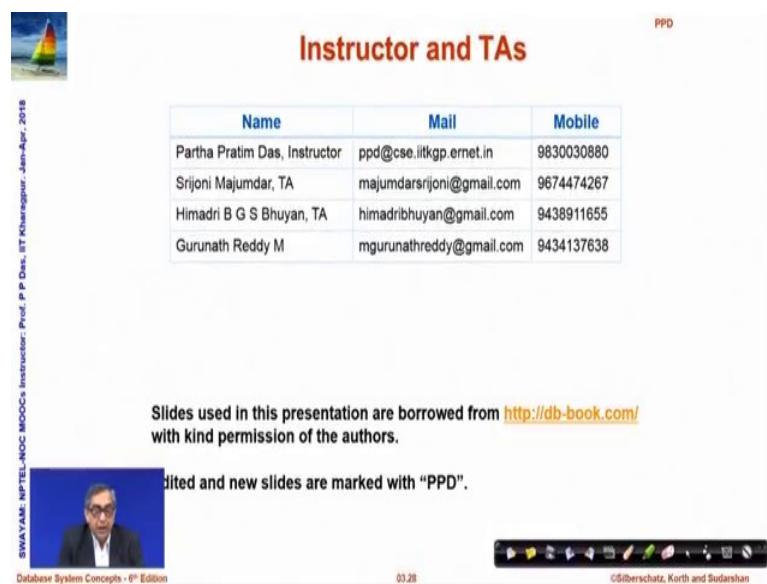


This slide is titled "Module Summary" in red at the top right. It features a small sailboat icon in the top left corner. A vertical sidebar on the left contains the text "SWAYAM: NPTEL-NOOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jam-Apri- 2018". The main content area lists three bullet points under the heading "PPD":

- Introduced models of database management systems
- Familiarized with major components of a database engine
- Familiarized with database internals and architecture

The bottom of the slide includes a video player interface showing a thumbnail of the speaker, the text "Database System Concepts - 8<sup>th</sup> Edition", a timestamp of "03:27", and the copyright notice "©Silberschatz, Korth and Sudarshan".

(Refer Slide Time: 31:41)



This slide is titled "Instructor and TAs" in red at the top right. It features a small sailboat icon in the top left corner. A vertical sidebar on the left contains the text "SWAYAM: NPTEL-NOOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jam-Apri- 2018". The main content area contains a table with three columns: Name, Mail, and Mobile.

Name	Mail	Mobile
Partha Pratim Das, Instructor	ppd@cse.iitkgp.ernet.in	9830030880
Srijoni Majumdar, TA	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, TA	himadribhuyan@gmail.com	9438911655
Gurunath Reddy M	mgurunathreddy@gmail.com	9434137638

Below the table, a note states: "Slides used in this presentation are borrowed from <http://db-book.com/> with kind permission of the authors." Another note says: "Edited and new slides are marked with "PPD". The bottom of the slide includes a video player interface showing a thumbnail of the speaker, the text "Database System Concepts - 8<sup>th</sup> Edition", a timestamp of "03:28", and the copyright notice "©Silberschatz, Korth and Sudarshan".

So, to summarize, in this module, we have introduced the models of database management system, the major components of a database engine and discussed about the internals and architecture, and this will conclude our discussion on the internals of database management systems, and next we will move on to exposing more on the relational model.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 04**  
**Introduction to Relational Model/1**

Welcome to module 4 of database management systems, in the last two modules, we have introduced the basic notions of DBMS in this module and the next; we make an introduction to the relational model, which we said is a major data model that we are going to use.

(Refer Slide Time: 00:46)

**Module Recap**

PPD

- Database Design
- OO Relational Model
- XML
- Database Engine
  - Storage Management
  - Query Processing
  - Transaction Management
- Database Users and Administrators
- Database Internals & Architecture
- History of DBMS

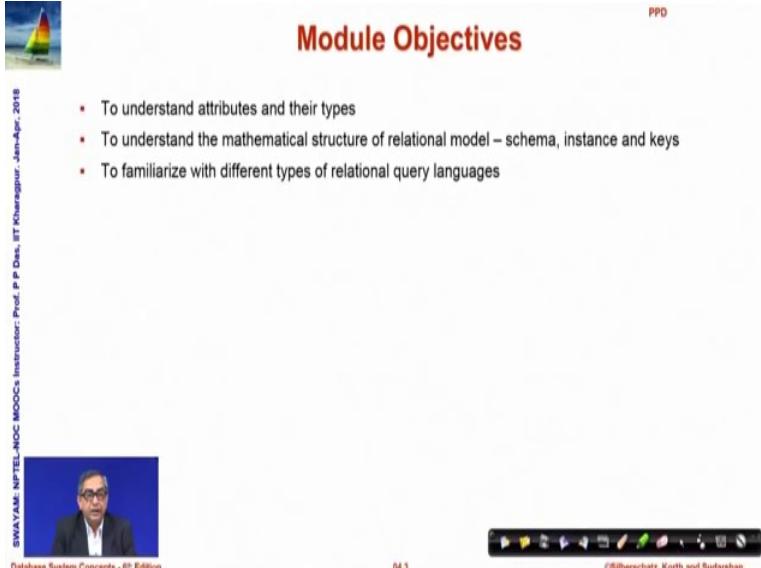
SWAYAM: IITTELMOOCs Instructor: Prof. P. P. Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

04.2 ©Silberschatz, Korth and Sudarshan

So, this is what we did in the last module.

(Refer Slide Time: 00:52)



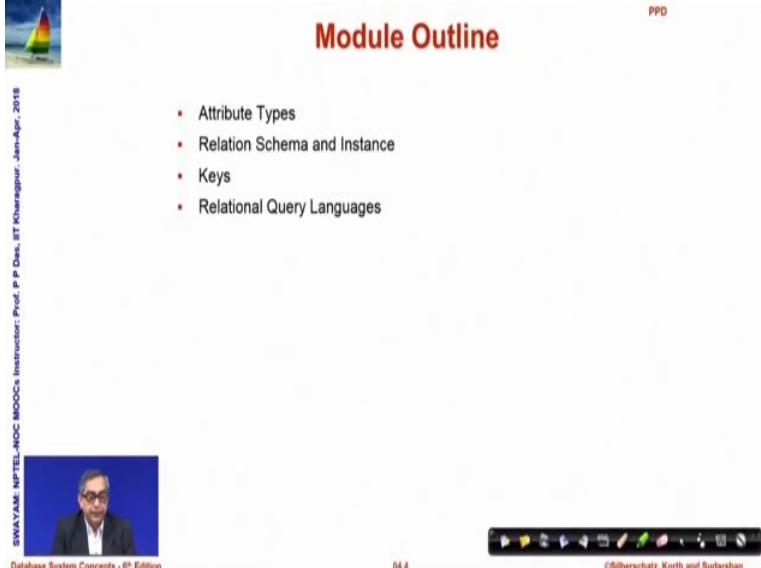
This slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. On the left side, there is vertical text: "SWAYAM: NPTEL-MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jam-Apri- 2018". At the bottom left is a video frame showing a man in a suit. The bottom right contains the text "Database System Concepts - 8<sup>th</sup> Edition" and "©Silberschatz, Korth and Sudarshan". The slide includes a bulleted list of objectives:

- To understand attributes and their types
- To understand the mathematical structure of relational model – schema, instance and keys
- To familiarize with different types of relational query languages

The slide has a "PPD" watermark in the top right corner.

And in the current one, our objective will be to understand key concepts of relational model that is attributes and their types, the basic mathematical structure of instance schema and what is known as keys and to familiarize with different types of relational query languages. This is a module outline that we will follow.

(Refer Slide Time: 01:17)



This slide is titled "Module Outline" in red at the top right. It features a small sailboat icon in the top left corner. On the left side, there is vertical text: "SWAYAM: NPTEL-MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jam-Apri- 2018". At the bottom left is a video frame showing a man in a suit. The bottom right contains the text "Database System Concepts - 8<sup>th</sup> Edition" and "©Silberschatz, Korth and Sudarshan". The slide includes a bulleted list of outline items:

- Attribute Types
- Relation Schema and Instance
- Keys
- Relational Query Languages

The slide has a "PPD" watermark in the top right corner.

(Refer Slide Time: 01:22)



## Example of a Relation

SWAYAM-NPTEL-MOOCs Instructor: Prof. P. P. Doshi, IIT Kharagpur - Jan-Apr - 2018



Database System Concepts - 8<sup>th</sup> Edition

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

attributes  
(or columns)

tuples  
(or rows)



64.5

©Bücherschätz, Korth and Sudarshan

So, we again repeat this example from past, this is an example of instructors, in a university a table of instructors given by attributes or columns ID name, department name and salary. These are the four columns, the four IDs and multiple rows, which are specific rows or we often refer to them as tuples. So, you can say that since there are 4 attributes, that is every row has 4 columns.

So, this is a 4 tuple that we have and such a table is called a relation is as simple as that. So, this is it, whenever we talk about a relation, we have a number of fields number of attributes number of columns, whatever way, we said of a table and that table according to those columns, it has multiple 0 1 or any number of rows of values, filled in and that is what is a relation.

(Refer Slide Time: 02:34)

The slide features a small sailboat icon in the top left corner. In the top right, it says 'PPD'. Below that is a bulleted list under the heading 'Attribute Types':

- Relation Schema and Instance
- Keys
- Relational Query Languages

At the bottom, there's a navigation bar with icons and the text 'Database System Concepts - 8<sup>th</sup> Edition' and '04.6 ©Silberschatz, Korth and Sudarshan'.

Now; so, let us look at attributes more specifically.

(Refer Slide Time: 02:37)

The slide has a sailboat icon and 'PPD' in the top right. The title 'Attribute Types' is centered below it. A bulleted list follows:

- The set of allowed values for each attribute is called the **domain** of the attribute
  - Roll #: Alphanumeric string
  - First Name, Last Name: Alpha String
  - DoB: Date
  - Passport #: String (Letter followed by 7 digits) – nullable
  - Aadhaar #: 12-digit number
  - Department: Alpha String
- Attribute values are (normally) required to be **atomic**; that is, indivisible
- The special value **null** is a member of every domain. Indicated that the value is "unknown"

Below the list is a table with columns: Roll #, First Name, Last Name, DoB, Passport #, Aadhaar #, and Department. Data rows are shown for two students. The 'Passport #' column for the second student contains the value 'null', which is circled in blue. Red arrows point from the list items to the corresponding table columns.

At the bottom, it says 'Database System Concepts - 8<sup>th</sup> Edition' and '04.7 ©Silberschatz, Korth and Sudarshan'.

So, attributes each column is an attribute as you said, this every attribute has a domain. The domain is a set of possible values that attribute can take. So, if you just look into the example here, so, I am trying to define a table having different students. So, there is a roll number for a student, there is a first name last name, the date of birth; DOB, the passport number, the Aadhaar card number, the department to which the student belongs and so on. So, let us say these 1 2 3 4 5 6 7 are the different attributes.

Now, if we look into every attribute, then every attribute has a set of possible values of which some value is entered in a particular row. For example, the roll number is an alphanumeric string, as you can see, it has numeric as well as it has letters whereas, the first name or the last name are simple alpha strings. In fact, we can also say that the roll number actually is not only alphanumeric, it has a fixed length, here is it has length of 9. So, you can say alphanumeric strings of length 9 are eligible for being values of this domain.

There could be more restrictions, but that the domain will be certain collection of values which are possible as values of that attribute, when you talk about D o B that certainly has to be a date. So, it is written in the form of d d m m m y y y y that is two digit date, three letter month codes and four digit year, the passport number is a string, a letter followed by seven digits. The other number is a twelve digit number, the department is alpha string and so on. So, the domain is a set corresponding to an attribute, which define that all possible values that attribute can take ok.

Now, these attribute values if you look at they are atomic in nature that is you cannot divide them into smaller parts. So, what I mean is say when we are talking about date of birth the whole date of birth type the date type is one atomic value. For example, if you were to code this in C what you could do you could possibly create a structure with three fields; one is date, one is a month, one is a year and we will say that this composite record composite structure is actually my date.

They can do a ls type def, you could do if you are working in C++, you will define a class called date, which has these components and as well as operations with them, but that kind of types are not allowed in a relational database, it has to be an atomic type. So, in a relational database will give you atomic type called date, but all of these are pre specified and has to be taken as one unit, other atomic types are integer like we do not have an integer field here. There are strings; there are numerical values, which are kind of floating point values and so on.

Now, some attribute may have a special value called the null value, which is the member. It is domain; actually every attribute of any domain can have this special value. The null value is not actually a value; it is actually an absence of a value. So, it says that this value is not known. So, if you look into the example above, then you will see that for

passport we have said that the passport is a string letter followed by seven digits and it is nullable, which means that in the passport field, I may have a value, may have this null value which means that it is not that, the passport is null, what it is saying is this passport number for this particular student, the row number 2 is not known, is unknown.

Now, all fields may or may not be nullable. For example, will not allow D O B to be nullable, date of birth has to be there, will not allow roll number to be nullable, will not allow first name to be nullable, but we may allow last name to be nullable. It is been a style, let not to use your last name, many people just use one name. So, you could allow that, it is not known, it is not there whereas, department may not be nullable, it must be there. So, null is a very critical concept and what it actually does? It actually creates a lot of issues and complications in terms of defining many operations. So, understanding null as a value in terms of an attribute is a critical requirement for the design.

(Refer Slide Time: 08:03)

The slide is titled "SCHEMA AND INSTANCE" in large red capital letters. In the top right corner, there is a small "PDF" icon. To the left of the title, there is a vertical sidebar with the following text:  
SWAYAM NPTEL MOOC Instructor: Prof. P. P. Datta, IIT Kharagpur - Jam-Apr-2018  
Database System Concepts - 8<sup>th</sup> Edition  
04.8  
©Bücherschutz, Korth und Söderström  
The main content area contains a bulleted list of topics:

- Attribute Types
- Relation Schema and Instance
- Keys
- Relational Query Languages

Now, coming to the schema and instance, we have discussed about the basic understanding of schema and instance. So, understanding them formally now, we say that if we have a schema. So, it is like a table having multiple columns say, there are n columns, having names A 1 to A n, then this A 1 to A n are the attributes.

(Refer Slide Time: 08:26)

The slide has a header 'Relation Schema and Instance' and a bullet point: '•  $A_1, A_2, \dots, A_n$  are attributes' with a handwritten checkmark. To the right is a hand-drawn diagram of a table with four columns and three rows, with the last cell in the third row crossed out with a large 'X'. On the left, there's a vertical footer with text: 'SWAYAM: NPTEL-NOOCs Instruction: Prof. P. P. Desai, IIT Kharagpur - Jam-Apri-2018'. At the bottom, it says 'Database System Concepts - 8<sup>th</sup> Edition' and shows a progress bar at 04.9.

So, these are the different attributes. So, if I have this, then it basically means that I have a table, where these are the columns A 1 A 2 A n like this.

(Refer Slide Time: 08:50)

The slide has a header 'Relation Schema and Instance' and two bullet points: '•  $A_1, A_2, \dots, A_n$  are attributes' and '•  $R = (A_1, A_2, \dots, A_n)$  is a relation schema'. Below this is a section 'Example:' with the definition 'instructor = ( $ID, name, dept\_name, salary$ )'. Another bullet point states: '• Formally, given sets  $D_1, D_2, \dots, D_n$  a relation  $r$  is a subset of  $D_1 \times D_2 \times \dots \times D_n$ '. Below this is the text 'Thus, a relation is a set of  $n$ -tuples  $(a_1, a_2, \dots, a_n)$  where each  $a_i \in D_i$ '. Handwritten notes include ' $A_i \rightarrow D_i$ ' next to the first bullet point. On the left, there's a vertical footer with text: 'SWAYAM: NPTEL-NOOCs Instruction: Prof. P. P. Desai, IIT Kharagpur - Jam-Apri-2018'. At the bottom, it says 'Database System Concepts - 8<sup>th</sup> Edition' and shows a progress bar at 04.8.

So, then a relational schema is a collection of these attributes. So, it is a collection of all these attributes. So, we said R is a relational schema, which has attributes A 1 to A n. Now, every attribute A i has a domain D i. So, for every attribute, I have a set of values that are possible. So, if you, if you recall then here we had different, these are the different attributes and these are their different domain. So, D o B is an attribute and the

domain is date. So, any possible date, other is an attribute and this is the domain, which is A. So, all attributes, each attribute will need to have certain domain and those are marked by the D Sets. So, we will say that a particular relation a particular relation R.

(Refer Slide Time: 10:01)

**Relation Schema and Instance**

- $A_1, A_2, \dots, A_n$  are attributes
- $R = (A_1, A_2, \dots, A_n)$  is a relation schema
- Example:  $\text{instructor} = (ID, name, dept\_name, salary)$
- Formally, given sets  $D_1, D_2, \dots, D_n$  a relation  $R$  is a subset of  $D_1 \times D_2 \times \dots \times D_n$ .  
Thus, a relation is a set of  $n$ -tuples  $(a_1, a_2, \dots, a_n)$  where each  $a_i \in D_i$

$$R = \left\{ (a_1, a_2, \dots, a_n) \in D_1 \times D_2 \times \dots \times D_n \right\}$$

SWAYAM, NIFTEL-NOC, MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018  
Database System Concepts - 10<sup>th</sup> Edition  
04.8  
©Silberschatz, Korth and Sudarshan

So, R is a schema. So, a particular relation R is a subset of  $D_1 \times D_2 \times \dots \times D_n$ . So, recall the mathematical notion of relation which says that a relation is basically a subset of a set. So, these are the possible values. So, the first attribute can take values from D 1, second attribute can take values from D 2 and so on and the nth attribute can take values from D n. So, any specific row, any specific record is a set of values for A 1 A 2 A n and therefore, is a member of this Cartesian product and the relation is a subset of that. So, this is a D value is an n tuple, which is a subset of this  $(a_1, a_2, \dots, a_n)$ .

This particular record is an element of this Cartesian product set and R necessarily is a set of such tuples that is a mathematical view of the schema and the instance. So, this is the schema and this is the instance corresponding to that schema based on the different domains of the different attributes and this is the notion that we will continue using. So, please try to follow this carefully.

(Refer Slide Time: 11:46)

The slide features a small sailboat icon in the top left corner. The title 'Relation Schema and Instance' is centered at the top in a red font. Below the title is a bulleted list of definitions and examples:

- $A_1, A_2, \dots, A_n$  are **attributes**
- $R = (A_1, A_2, \dots, A_n)$  is a **relation schema**

Example:

- $\text{instructor} = (ID, name, dept\_name, salary)$
- Formally, given sets  $D_1, D_2, \dots, D_n$  a **relation r** is a subset of  $D_1 \times D_2 \times \dots \times D_n$ . Thus, a relation is a set of  $n$ -tuples  $(a_1, a_2, \dots, a_n)$  where each  $a_i \in D_i$ .
- The current values (**relation instance**) of a relation are specified by a table
- An element  $t$  of  $r$  is a **tuple**, represented by a **row** in a table

At the bottom of the slide, there is a video player interface showing a video of a professor and some navigation icons.

Now, whenever we have an instance, we mark that as a table and every such table.

(Refer Slide Time: 11:56)

The slide features a small sailboat icon in the top left corner. The title 'Relations are Unordered' is centered at the top in a red font. Below the title is a bulleted list of points:

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- Example: *instructor* relation with unordered tuples

Below the list is a table with handwritten annotations. The table has columns labeled  $ID$ ,  $name$ ,  $dept\_name$ , and  $salary$ . Handwritten labels  $a_1, a_2, a_3, a_4$  are placed next to the first four rows. The table data is as follows:

ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

At the bottom of the slide, there is a video player interface showing a video of a professor and some navigation icons.

So, here you have now understood it very well. So, these are my attributes. So, this is A 1, this is A 2, this is A 3, this is A 4 and any one i name is at the different values a 2 a 3 a 1 a 2 a 3 a 4 98345 is a 1 Kim is a 2 and so on. Now, naturally this, it is not visible from the instance, because we are taking an instance view, we are not being able to see, what that domain is that will be visible? If we look at the corresponding D D 1, the definition language description of the schema, which must have specified I D as a numeric value,

the name as a string value the department name as another string value whereas, salary as a numeric value and so on. Now, what is important to note here is a relation necessarily is a set as we said is a set, which is the as the relation R is a set, this is a set, which is a subset of this set.

(Refer Slide Time: 13:04)

The screenshot shows a presentation slide with the following elements:

- PPD** logo in the top right corner.
- A small sailboat icon in the top left corner.
- A vertical sidebar on the left with the text: "SWAYAM\_NIPTEL-NOC MOOCs Instructor: Prof. P. P. Dand, IIT Kharagpur - Jan-Apr. 2016".
- KEYS** is the main title in large red capital letters.
- A video player window showing a man speaking, with the text "Database System Concepts - 8th Edition" below it.
- A navigation bar at the bottom with icons for back, forward, search, and other controls.
- Page number 04.11 and copyright information "©Bilberschatz, Korth and Sudarshan" at the bottom right.

So, we know the elements in a set are do not have any ordering, they are unordered. So, a relation is necessarily unordered. So, it does not really matter that in terms of this collection of rows, which row is at what position, if I reorder them, the relation does not change it is just that they are a collection of this set of rows. So, that lack of ordering is critical information that we will have to remember in mind next concept is key.

(Refer Slide Time: 13:49)

The slide has a header 'PPD' in red at the top right. The title 'Keys' is in red at the top center. On the left, there is a small logo of a sailboat on water. The main content is a bulleted list:

- Let  $K \subseteq R = (A_1, A_2, \dots, A_n)$
- $K$  is a **superkey** of  $R$  if values for  $K$  are sufficient to identify a unique tuple of each possible relation  $r(R)$
- Example: {ID} and {ID, name} are both superkeys of *instructor*

On the left side of the slide, there is vertical text: 'SWAYAM', 'NPTEL-NOOC', 'MOOCs', 'Instructor', 'Prof. P. P. Desai, IIT Kharagpur', 'Jain-Apr.-2018'. At the bottom left, it says 'Database System Concepts - 8<sup>th</sup> Edition'. In the bottom right corner of the slide area, there is a timestamp '04.12' and a copyright notice '©Silberschatz, Korth and Sudarshan'.

So,  $R$  as we have seen is a relational schema, which is a collection of attributes **A 1, A 2 .....A n**. Now,  $K$  let  $K$  is a subset of  $R$ . So, it is one or more attributes, it has to be a non-empty subset. Now, we will say that  $K$  is a super key of  $R$ , if we consider the values of different tuples in the attributes of  $K$  and we find that there cannot be two tuples, which are different, but match on these attributes, which mean that the values of the attributes of  $K$ , uniquely identify each row of the relation, then we will say that  $K$  is a super key of  $R$ .

So, the instructor table that we have seen, ID is a super key similarly. So,  $K$  can be taken as a singleton set of attribute ID or  $K$  can be thought of as the set comprising ID and name both of them are super keys of instructor.

(Refer Slide Time: 15:32)

PPD

## Keys

- Let  $K \subseteq R$
- $K$  is a **superkey** of  $R$  if values for  $K$  are sufficient to identify a unique tuple of each possible relation  $r(R)$ 
  - Example:  $\{ID\}$  and  $\{ID, name\}$  are both superkeys of *instructor*
- Superkey  $K$  is a **candidate key** if  $K$  is minimal
  - Example:  $\{ID\}$  is a candidate key for *Instructor*
- One of the candidate keys is selected to be the **primary key**
  - Which one?
- A **surrogate key** (or synthetic key) in a database is a unique identifier for either an *entity* in the modeled world or an *object* in the database
  - The surrogate key is *not* derived from application data, unlike a *natural* (or *business*) key which is derived from application data

Now, we say a super key  $K$  is a candidate key, if  $K$  is minimal. So, the idea is like this that this is a key, super key, this is also a super key, but certainly this is a subset of this. This is smaller than this. So, we will say this is a candidate key, but this is not a candidate key, because it does not satisfy the minimality condition.

There could be multiple candidate key in a relation, if there are multiple candidates key then we select one to be the primary key. Now; obviously, there is a question of which one we select, but anyone can be selected as a primary key, which is the key of the relation and we will see that in some cases, there is concept of surrogate keys.

So, if I have a relation where there is no attribute, whose value can uniquely identify each and every row of the table then I might synthetically generate a value for example, like a serial number, I can generate a serial number and say that this is my value. So, that serial number or that computer generated field value has no business implication, the real world did not have this value, it is not like a Aadhaar card number or like a passport number, but it is a value which is purely generated to identify every row uniquely. So, such keys are known as surrogate keys or synthetic keys.

(Refer Slide Time: 17:40)

The slide has a header 'PPD' in red at the top right. On the left, there is a small logo of a sailboat and some vertical text: 'SWAYAM-NPTEL-NCOC', 'Instructor: Prof. P. P. Desai', 'Date: 2018-04-10', and 'Time: 17:40'. The main title 'Keys' is in red at the top center. Below it is a bulleted list of key types:

- **Super Key:** Roll #, (Roll #, DoB)
- **Candidate Keys:** Roll #, (First Name, Last Name), Passport #, Aadhaar #
  - Passport # cannot be a key. Why?
- **Primary Key:** Roll #
- **Secondary / Alternate Key:** (First Name, Last Name), Aadhaar #
- **Simple key:** Consists of a *single attribute*
- **Composite Key:** (First Name, Last Name)
  - Consists of more than one attribute to uniquely identify an entity occurrence
  - One or more of the attributes, which make up the key, are not simple keys in their own right

Below the list is a table with the following data:

Roll #	First Name	Last Name	DoB	Passport #	Aadhaar #	Department
15CS10026	Lalit	Dubey	27-Mar-1997	L4032464	1728-6174-9239	Computer
16EE30029	Jatin	Chopra	17-Nov-1996	null	3917-1836-3816	Electrical
C10016	Smriti	Mongra	23-Dec-1996	G5432849	2045-9271-0914	Electronics
E10038	Dipti	Dutta	02-Feb-1997	null	5719-1948-2918	Civil
S30021	Ramdin	Minz	10-Jan-1997	X8811623	4928-4027-6024	Computer

At the bottom of the slide, there is a footer with the text 'Database System Concepts - 8<sup>th</sup> Edition', '04.13', and '©Silberschatz, Korth and Sudarshan'.

Now, let us look at some examples, this is again the same student database, I just shown a while ago the same set of columns, but I have added few more rows. Now, if we look at what could be a super key there are several candidates, but I have just written a few roll number is certainly a key, because I am assuming that the university assigns roll numbers to uniquely identify every student.

So, there cannot be two rows in this table, which match in the value of the roll number and does not match in the values of the other fields. So, roll number can uniquely identify, if it can then any super set of attributes, which continual number will also be a super key. So, roll number and date of birth together is a super key that can also unique to identify every row trivial. What are the candidate keys? Now, there are of course, that could be several other super keys that has to be kept in mind, the candidate keys are roll number is a candidate key, the first name last name together, we can say is a candidate key.

So, we are saying that not only the first name, but if we take this pair, you remember the key, the set of attributes forming a super key is a set. It is not an individual field. So, I say the first name last name together, from say, key well. This does make some assumption, because if I say the first name last name together from say key; that means, that there cannot be two records in this student table, where the first name and last name match, but the records are different. So, which mean that no, two students having the

same first name and last name can be enrolled in the university. This is a restrictive assumption right, but I am just making that assumption to illustrate the different possibilities; then what is the other possibility passport number? Everybody has a unique passport number. So, passport number could also be a key, could be a candidate key. Aadhaar number; everybody has a unique Aadhaar number. So, that can be a key and so on.

So, these are called the candidate keys. Now of course, we can observe that given the data it is clear and it was also mentioned when the schema was designed this passport number cannot be a key. Why can it not be a key? Can two students have same passport number? Of course, not every student has a unique passport number, but it is possible that some student does not have a passport. So, if some student does not have a passport then the passport number field of that student is a null, the passport number is a nullable field, if the passport number is null then it is possible that multiple students may not have passports.

So, as we can see here that this student Jatin Chopra does not have a passport. So, similarly, Dipti Dutta does not have a passport either. So, certainly if this were to be the key then for all records, for which passport number is nil, this value would not be able to distinguish them in terms of the rows of the table. So, we have to say that passport number cannot be a key or in other words, we can say that no key can be a nullable field.

No key attribute or a participant to a key attribute could be a nullable field right. So, this is one observation here. And, so that clearly also implies that, if we say that Aadhaar number is a valid candidate key that will mean that for admission to that university having Aadhaar number, would be mandatory, if somebody does not have a Aadhaar number that will have to be null, which is not allowed.

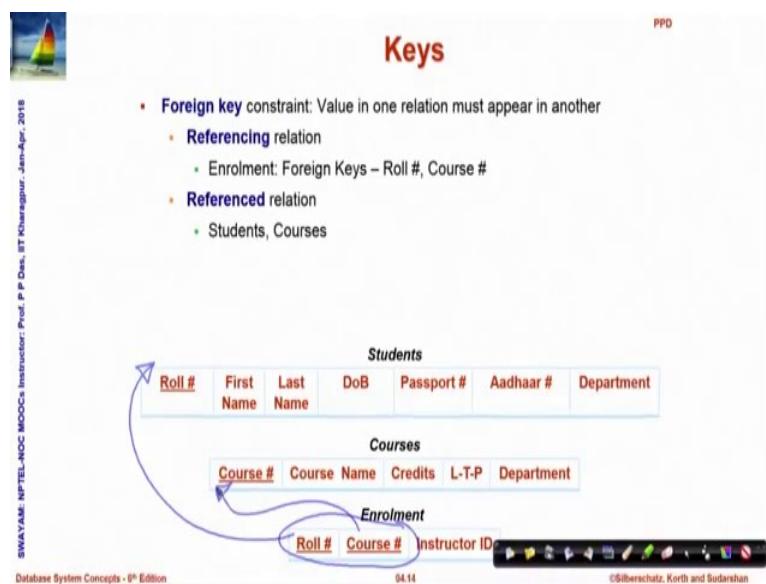
So, let us move on. So, one of these candidate keys have to be made the primary keys. Let us say; we make roll number, the primary key and since, we make roll number the primary key in the schema. We underlined the roll number attribute; this would be a common way to show that roll number is a primary key. So, the others that are not taken as a primary key are called the secondary or alternate key. So, first name last name pair

could be an alternate key Aadhaar number could be an alternate key and so on. A key is said to be simple, if it consists of a single attribute.

So, roll number is a simple key, Aadhaar numbered is a simple key, if it were taken to be primary, but first name last name pair, if we take that to be a primary that will not be considered as simple key, because it has more than one attribute naturally the other, if you have a simple key.

They have other side is a composite key is one, which has more than one field such that none of those fields individually can act as a key, but together they can act as a key. So, first name itself cannot be a key last name itself cannot be a key, but together. They can be a key of course, under the assumption that no two students with the same first name last name are given admission. So, these are the different types of keys that can happen.

(Refer Slide Time: 23:32)



Let us have some more views with the keys, we extend the schema and besides a student I introduce two more schema; one is called the courses, which is given by course number, course name credits L T P. L T P is number of hours of lectures tutorials and practical's and the department. So, these are the different fields and from the convention already stated you can figure out that course number is the key primary key of this relation. I use another schema, which is enrolment, which describes which student is attending which course. So, it has a roll number and the course number.

So, roll number of the student attending the particular course number and it also has an instructor ID as to who is teaching that course given this. You can see that in the enrolment relationship, I have this pair roll number and course number, which will certainly be the key for enrolment, because if I have two rows in enrolment. How they will be distinguished, they cannot be distinguished by roll number, because a particular student may take multiple courses.

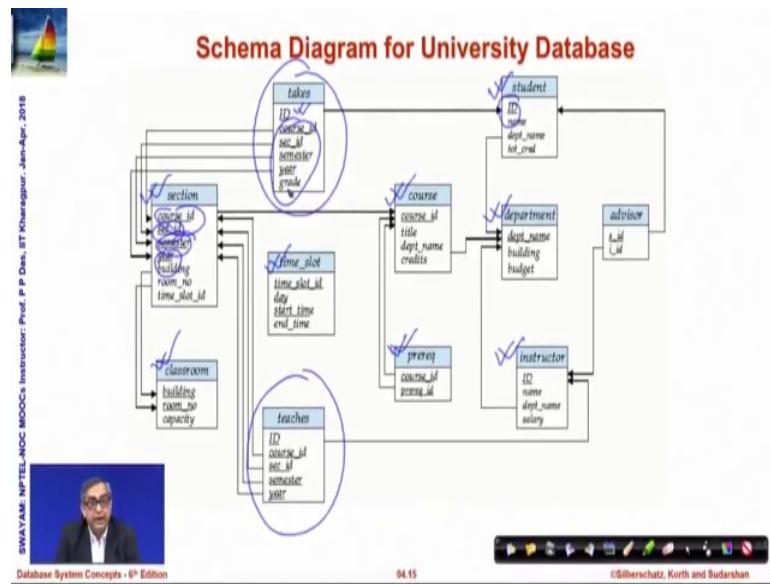
So, there will be multiple records having the same roll number, but different course number, the course number by itself cannot be the key, because every course will have multiple students. So, there will be multiple rows having the same course number, but all different roll numbers, but if we take this together, roll number and course number together then that forms a key.

Now, such a key such a key having roll number the roll number itself is a key of another relation the course number itself is a key of another relation. So, when we take the keys of other relations to form the key of a relation then we say that these are foreign keys. So, roll number and course number are foreign keys in student and course and since from enrolment the student and courses are being referenced are being referred.

So, we say enrolment is a referencing relation and students and courses other reference relation and we will often like to also mention as to what is a foreign key of a relational schema, because that will help us understand how the different schemas are interrelated and we will see that, this will come out directly from the notion of entities and relationships of a year model of a year diagram, a key is called, to be said to be compound, if it consists of more than one attribute to uniquely identify an entity occurrence.

So, each attribute which makes up the key is a simple key, in its own right, mind you there is a subtle, it sounds very similar. We talked about composite key; earlier, we talked up; we are talking about compound key here, the subtlety of the difference is in a composite key, every component attribute is not a simple key by itself, but the components come from the same table in a compound key. The components are simple key in their own right, in some other table and are put together as a compound key in the given table. So, the roll number, course number in the enrolment table is a compound key.

(Refer Slide Time: 27:31)



So, with this I would request you to spend some time with this relatively elaborated schema compared to what we have done already of the university database. So, every rectangular box shows a relational schema on top of each in blue, is written the name of that relation relational schema. So, it has a relational schema like courses, the students the instructors, the departments, the prerequisites, the time slots, the classrooms and so on.

The sections and the relationships between them for example, the relationship is takes is a relationship, which relates students with different sections, with courses, teachers is another relationship, which relates to instructors with sections. So, it is showing you directly as to how the keys of this, what are the attributes? What are the key attributes primary key attributes and also what are the foreign keys that we have in this for example, intakes this is a foreign key, which is featured here, course I D section, I D semester here are the foreign key part of the takes that exists here. So, please study this schema. We will keep on regularly referring to this schema in future as well. So, this is what we have here.

(Refer Slide Time: 29:29)

The slide features a small sailboat icon at the top left. On the right side, there is a vertical list of topics: Attribute Types, Relation Schema and Instance, Keys, and Relational Query Languages. The main title 'RELATIONAL QUERY LANGUAGES' is centered in large red capital letters. Below the title, the text 'Database System Concepts - 8<sup>th</sup> Edition' is visible. At the bottom right, there is a navigation bar with icons and the text '©Silberschatz, Korth and Sudarshan'. The footer contains the text 'SWAYAM: NPTEL-MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jam-Apr.- 2016'.

Now, we move on to the relational query language, we briefly talk about the relational query language.

(Refer Slide Time: 29:33)

The slide has a small sailboat icon at the top left. The main title 'Relational Query Languages' is centered in red capital letters. Below the title, the text 'Procedural vs. Non-procedural or Declarative Paradigms' is displayed. A bulleted list follows, comparing procedural and declarative paradigms. The list includes: Procedural programming (requires telling the computer what to do), Declarative programming (requires a descriptive style), and an example for square root of n. The example for square root of n is divided into Procedural and Declarative sections. The footer contains the text 'SWAYAM: NPTEL-MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jam-Apr.- 2016', a video thumbnail of a professor, 'Database System Concepts - 8<sup>th</sup> Edition', the time '04:17', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

Now, we will have to in this the key thing that we need to understand is the relational query language is somewhat different from the programming languages that you have studied so far which are procedural in nature, in contrast the relational query language is non-procedural or declarative in nature, a procedural programming language requires that the programmer tell the computer how to get the output given, the input a pro

program is about finding output, for a given input and you write a procedure, the sequence of steps that need to be done. So, that given the input, you can compute the output.

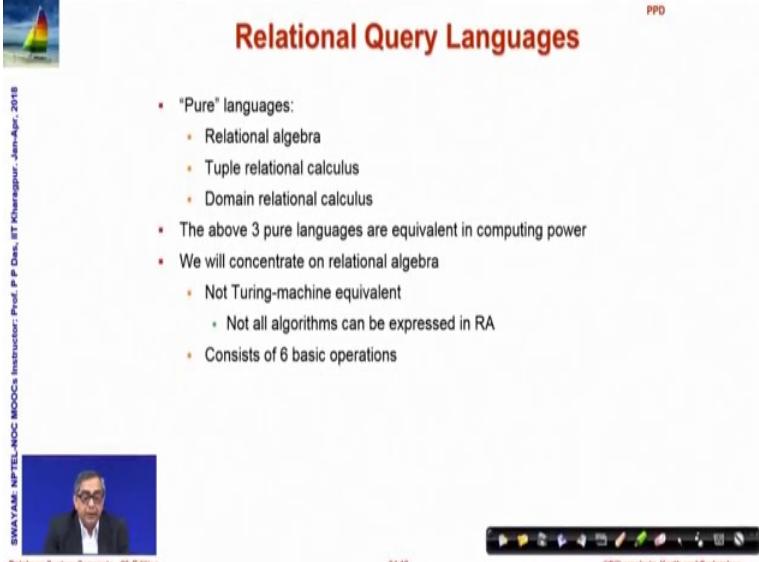
So, you say how that computation has to happen and the programmer must know that algorithm in contrast, in declarative programming, you say what you want? You will do not say how that needs to be computed? How that will be computed? You may not even know that, you may not even know a single algorithm to compute the output, but you specify what output you need. So, this distinction between how and what of programming differentiates procedural and declarative programming.

So, all that you have studied so far in terms of C C++, java python and all that are procedural programming, where you necessarily have to specify, how you will have necessarily; have to specify what the algorithm is, but in declarative you just say what you need. So, just a simple you know pathological example to understand this difference. Suppose, we were interested in computing the square root of a number  $n$  assuming  $n$  is a positive number, the procedural step would be something like; this is an algorithm that you guess a  $X$  or, which is a square root, which is close to the root of  $n$ .

I mean some guess you make and then you repeatedly refine this estimate by taking the arithmetic mean of the estimate and the quotient of the division of  $n$  by this estimate. So, you take an arithmetic mean and find the new estimate and repeat the steps, I mean as long as the difference between the two conservative estimates is more than a certain value  $\delta$ , this is a procedural algorithm, you are giving an algorithm. So, given and following this algorithm, we will find the square root declaratively, you can just say that what is the result? I want to result  $m$  such that  $m^2 = n$ .

So, you are again asking for the same feel, you are expecting the same output, but the way you are saying is not an algorithm, you are rather specifying a predicate, which must be true in your output. You are saying that the predicate is  $m$  square must be  $n$ . So, whatever  $m$  is that square of it must equal  $n$ . So, this style is known as declarative whereas, the earlier style is known as procedure.

(Refer Slide Time: 32:38)



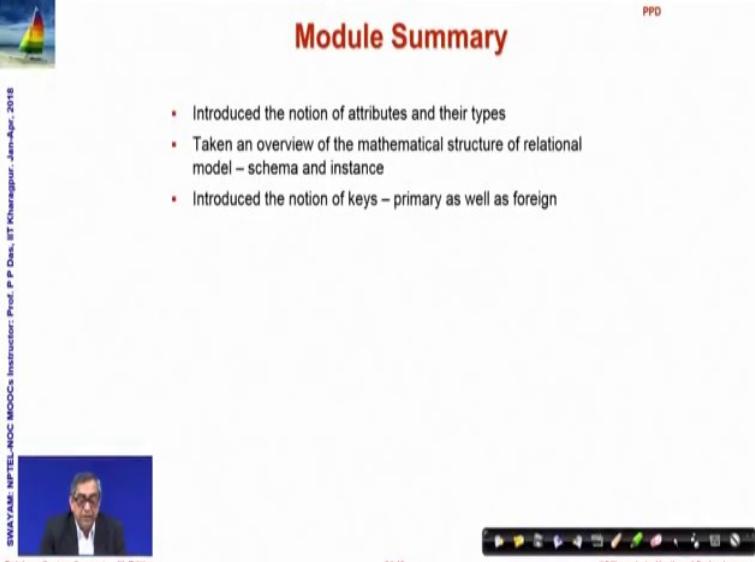
The slide is titled "Relational Query Languages" in red at the top right. It features a small sailboat icon in the top left corner. On the left edge, there is vertical text: "SWAYAM\_NPTEL-MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jam-Apri- 2018". At the bottom left, it says "Database System Concepts - 8<sup>th</sup> Edition". The main content is a bulleted list:

- "Pure" languages:
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
- The above 3 pure languages are equivalent in computing power
- We will concentrate on relational algebra
  - Not Turing-machine equivalent
    - Not all algorithms can be expressed in RA
  - Consists of 6 basic operations

At the bottom right, it says "©Silberschatz, Korth and Sudarshan". A navigation bar is at the very bottom.

All query languages, relational query languages are declarative in nature. We have talked about the pure languages, are they are all equivalent? We mentioned that earlier also and also again to remember that none of them are actually Turing equivalent; that means, that not all algorithms can be expressed in them or specifically relational algebra, which we will look at in more depth and the relational algebra will consist of six basic operations, which we will discuss in the next module.

(Refer Slide Time: 33:11)



The slide is titled "Module Summary" in red at the top right. It features a small sailboat icon in the top left corner. On the left edge, there is vertical text: "SWAYAM\_NPTEL-MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jam-Apri- 2018". At the bottom left, it says "Database System Concepts - 8<sup>th</sup> Edition". The main content is a bulleted list:

- Introduced the notion of attributes and their types
- Taken an overview of the mathematical structure of relational model – schema and instance
- Introduced the notion of keys – primary as well as foreign

At the bottom right, it says "©Silberschatz, Korth and Sudarshan". A navigation bar is at the very bottom.

So, to sum up we have introduced the notion of attributes and their types, we have taken an overview of the mathematical structure of the relational model schema and instance, we would say mathematically they are relations, mathematically they made a mapping and we have introduced the very important concept of keys and in that very specifically, what is a primary key as well as what is a foreign key? In the next module, we will discuss about the different operations of relational model relational logic.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture -04**  
**Introduction to Relational Model/2**

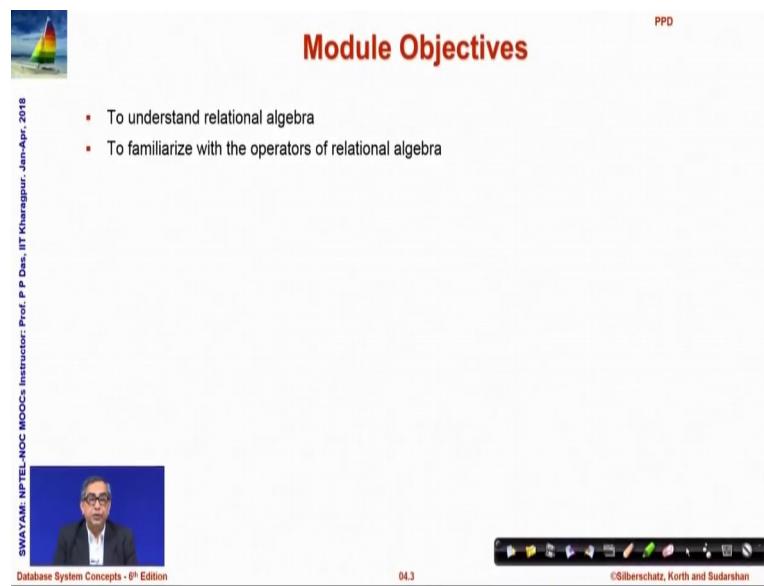
Welcome to module 5 of database management systems. In the previous module, we started discussions on introducing relational model we will conclude that in this module.

(Refer Slide Time: 00:36)

The slide has a header 'Module Recap' in red. On the left is a small image of a sailboat. A vertical sidebar on the left contains the text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr. 2018'. The main content area lists four bullet points: 'Attribute Types', 'Relation Schema and Instance', 'Keys', and 'Relational Query Languages'. At the bottom right is a navigation bar with icons for back, forward, search, and other presentation controls. The footer includes 'Database System Concepts - 8<sup>th</sup> Edition', '04.2', and '©Silberschatz, Korth and Sudarshan'.

So, in the last module we have talked about attributes relational schemas and instances in mathematical form and very importantly we have tried to introduce discuss about the concept of keys.

(Refer Slide Time: 00:53)



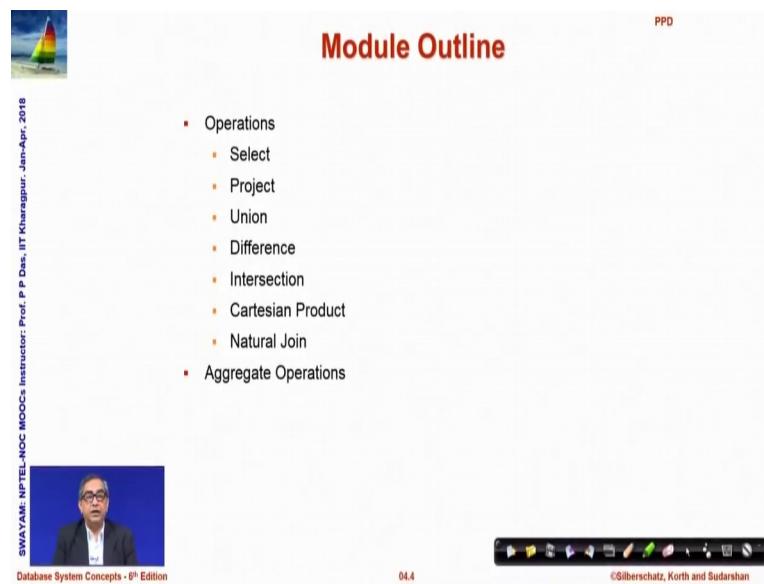
This slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. On the far right, there is some very small, illegible text. The main content is a bulleted list of objectives:

- To understand relational algebra
- To familiarize with the operators of relational algebra

The footer contains the text "SWAYAM: NIPTEL-NOC MOOCs Instructor: Prof. P Das, IIT Kharagpur, Jam-Apri. 2018", "Database System Concepts - 8<sup>th</sup> Edition", "04.3", and "©Silberschatz, Korth and Sudarshan". There is also a decorative toolbar at the bottom.

In this module we, will try to understand more on the relational algebra and familiarize with the operations of relational algebra.

(Refer Slide Time: 01:00)



This slide is titled "Module Outline" in red at the top right. It features a small sailboat icon in the top left corner. On the far right, there is some very small, illegible text. The main content is a bulleted list of operations:

- Operations
  - Select
  - Project
  - Union
  - Difference
  - Intersection
  - Cartesian Product
  - Natural Join
- Aggregate Operations

The footer contains the text "SWAYAM: NIPTEL-NOC MOOCs Instructor: Prof. P Das, IIT Kharagpur, Jam-Apri. 2018", "Database System Concepts - 8<sup>th</sup> Edition", "04.4", and "©Silberschatz, Korth and Sudarshan". There is also a decorative toolbar at the bottom.

So, these are the different operations that we will look at select project union and so on. Some of them are simple set theoretic operations some are newly defined operations and we look at the aggregators.

(Refer Slide Time: 01:18)

The slide features a logo of a sailboat on the left. The title 'Select Operation – selection of rows (tuples)' is at the top right. Below it, a note says 'Relation r' with a bullet point. A table 'r' is shown with columns A, B, C, D and rows for (α, α, 1, 7), (α, β, 5, 7), (β, β, 12, 3), and (β, β, 23, 10). To the right, a handwritten note shows the selection operation  $\sigma_{A=B \wedge D > 5}(r)$  with a checkmark. Below this, another table shows the result of the selection: rows (α, α, 1, 7) and (β, β, 23, 10). Handwritten notes next to the result table say 'σ: propositional condition'. The footer includes 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jam-Apr. 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '04.6', and '©Silberschatz, Korth and Sudarshan'.

So, relational operators, so what is a relation as we have seen already? A relation is nothing but a table. It has a set of columns and it has a set of rows or records that fill up data according to those columns. Select is an operation, which chooses a subset of rows from a relation based on a certain condition. So, it is written in terms of in relational algebra we write it with a notion of notation of  $\sigma$  and following a parenthesis we put the name of the relation.

So, we say we are selecting from the relation  $r$  and then we put a condition here, which is a propositional condition. So, if for this all rows of  $r$  will be checked if a row will satisfy this condition. Then it will be included in the result, if it does not satisfy the condition, then it will not be included in the result.

(Refer Slide Time: 02:53).

The slide illustrates the selection operation on a relation  $r$ . The relation  $r$  is defined as:

$$\begin{array}{|c|c|c|c|} \hline A & B & C & D \\ \hline \alpha & \alpha & 1 & 7 \\ \alpha & \beta & 5 & 7 \\ \beta & \beta & 12 & 3 \\ \beta & \beta & 23 & 10 \\ \hline \end{array}$$

A handwritten condition  $\Theta: A = B \wedge D > 5$  is shown next to the relation. The result of the selection query  $\sigma_{A=B \wedge D > 5}(r)$  is:

$$\begin{array}{|c|c|c|c|} \hline A & B & C & D \\ \hline \alpha & \alpha & 1 & 7 \\ \beta & \beta & 23 & 10 \\ \hline \end{array}$$

Annotations include a checkmark in the first row of the original relation and a checkmark in the second row of the result.

So, let us look at this example. So, our condition is sorry let this put this back. So, our condition  $\Theta$  is  $A=B \wedge D>5$ . So, you are saying that that any row to be selected, the value of it is  $A$  attribute should equal the value of it is  $B$  attribute and when that happens the value of it is  $D>5$ . So, you can easily if we look through this we can easily say by the first condition  $A=B$  you can say that this row does not satisfy this condition.

Because,  $A$  is  $\alpha$  and  $B$  is  $\beta$  whereas, these 3 rows satisfy because  $A=\beta$ . Then you again look at  $D$  we find that this  $D<5$ . We say this also does not satisfy, because it fails the second condition both have to hold. So, we finally, come to that this record and this record  $r$  the selected record in the result  $\alpha \alpha 1 7, \beta \beta 23 10$ . In both of these records  $A=B$  in both of these records  $D>5$ .

So, selection is a process which selects a subset of rows from a table, from a relation and creates a new relation. Based on a selection condition that must be true for all the rows for all the records that have been selected, but the set of columns do not change they remain the same.

(Refer Slide Time: 04:57)

The slide title is "Select Operation – selection of rows (tuples)". It shows a relation  $r$  with fields A, B, C, D and four tuples:  $\alpha\ \alpha\ 1\ 7$ ,  $\alpha\ \beta\ 5\ 7$ ,  $\beta\ \beta\ 12\ 3$ , and  $\beta\ \beta\ 23\ 10$ . A handwritten note "σ<sub>A=B</sub> (r)" is written next to the relation. Below it, a query  $\sigma_{A=B \wedge D > 5} (r)$  is shown with its result: a relation with the same structure and two tuples:  $\alpha\ \alpha\ 1\ 7$  and  $\beta\ \beta\ 23\ 10$ . A handwritten note "σ<sub>A=B</sub> (r)" is written next to the result relation.

Relation r

A	B	C	D
$\alpha$	$\alpha$	1	7
$\alpha$	$\beta$	5	7
$\beta$	$\beta$	12	3
$\beta$	$\beta$	23	10

$\sigma_{A=B \wedge D > 5} (r)$

A	B	C	D
$\alpha$	$\alpha$	1	7
$\beta$	$\beta$	23	10

σ<sub>A=B</sub> (r)

σ<sub>A=B</sub> (r)

So, for example, if I if in the contrary if we say, that  $\sigma_{A \neq B} (r)$  then naturally, I will have a relation with fields A B C D and that relation will be only this row. Where that is the only row where  $A \neq B$  we can.

(Refer Slide Time: 05:26)

The slide title is "Select Operation – selection of rows (tuples)". It shows a relation  $r$  with fields A, B, C, D and four tuples:  $\alpha\ \alpha\ 1\ 7$ ,  $\alpha\ \beta\ 5\ 7$ ,  $\beta\ \beta\ 12\ 3$ , and  $\beta\ \beta\ 23\ 10$ . A handwritten note "σ<sub>A=B</sub> (r)" is written next to the relation. Below it, a query  $\sigma_{A=B \wedge D > 5} (r)$  is shown with its result: a relation with the same structure and two tuples:  $\alpha\ \alpha\ 1\ 7$  and  $\beta\ \beta\ 23\ 10$ . A handwritten note "σ<sub>A=B</sub> (r)" is written next to the result relation.

Relation r

A	B	C	D
$\alpha$	$\alpha$	1	7
$\alpha$	$\beta$	5	7
$\beta$	$\beta$	12	3
$\beta$	$\beta$	23	10

$\sigma_{A=B \wedge D > 5} (r)$

A	B	C	D
$\alpha$	$\alpha$	1	7
$\beta$	$\beta$	23	10

σ<sub>A=B</sub> (r)

σ<sub>A=B</sub> (r)

I can have a selection saying  $r$  where  $C > 0$ . Naturally, this will satisfy, this will satisfy, this will satisfy, this will satisfy. So, this whole relation would be the result of the selection. So, it is possible that it is not necessary that some rows will have to get eliminated in the result.

(Refer Slide Time: 05:54)

**Select Operation – selection of rows (tuples)**

- Relation  $r$

	A	B	C	D
$\alpha$	$\alpha$	1	7	$\times$
$\alpha$	$\beta$	5	7	$\times$
$\beta$	$\beta$	12	3	$\times$
$\beta$	$\beta$	23	10	$\times$

- $\sigma_{A=B \wedge D > 5}(r)$

	A	B	C	D
$\alpha$	$\alpha$	1	7	
$\beta$	$\beta$	23	10	

I say that this is the selection is  $D < 1$  this will fail, this will fail, this will fail. So, all of them will fail so, it is possible that the result of an operation could be either the whole relation as we saw last time or a null relation which has where none of the record with feature because, none of the record satisfy the condition. So, that is a basic select operation.

(Refer Slide Time: 06:38)

**Project Operation – selection of columns (Attributes)**

- Relation  $r$ :

	A	B	C
$\alpha$	10	1	
$\alpha$	20	1	
$\beta$	30	1	
$\beta$	40	2	

- $\Pi_{A,C}(r)$

	A	C
$\alpha$	1	
$\alpha$	1	
$\beta$	1	
$\beta$	2	

$$\Pi_{A,C}(r) = \begin{array}{|c|c|} \hline A & C \\ \hline \alpha & 1 \\ \alpha & 1 \\ \beta & 1 \\ \beta & 2 \\ \hline \end{array}$$

$\text{TT}_{A,B,C}(r) = r'$

Let us move on look at the next one is called a projection operation. So, select chooses a subset of the rows. Projection necessarily chooses projects a set of columns from the

original relation. So, this is quite straightforward to see, it is written in terms of this notation  $\Pi$  and then you write the columns that you want in the result of projection. So, we say this is  $\Pi_{A,C}$ . So, which means basically the column, which is not selected in the projection, you can simply forget about that simply erase it. If you erase that you get this relation and once you get that, please recall that a relation is a set and in a set every element has to be distinct. So, after erasing B this first and the second row have become identical.

So, naturally both of them cannot be there, it will have to be made distinct by erasing any one of them. And hence, they become one row in the result; obviously, I can project on any of the individual single or all the fields also I can do a  $\Pi_{A,B,C}(r)$  of course, that means, in this case that will mean that it is a set which is equal to r anyway, but; obviously, I must have at least 1 column at least one attribute to project on. I cannot project on a null set of attributes because that does not give me a schema.

So, there will have to be some attribute one or more attribute on which I project. So, selection and projection, selection has given me the set of rows to retain and projection has given me what are the columns to retain in the result. And combining them I can do several different operations in a database table which can give me several interesting results. Before proceeding further,

Let us look into some of the typical other operations that relational algebra allows. The next one is  $U$  given two relations I can take a  $U$  this is nothing but a set theoretic union. The two relations  $r$  and  $s$  must have the same set of columns A and B because, if the columns are not same, then the  $U$  does not make sense. They because certainly if the columns are different attributes are different their types and type of data values would be different. So, they cannot be put to a same table so when two relations have the same set of attributes then their instances can be taken a union of.

(Refer Slide Time: 09:49)

The slide title is "Set difference of two relations". It shows two relations,  $r$  and  $s$ , represented as tables:

- Relations  $r, s$ :
  - $r$  (left):

A	B
$\alpha$	1
$\alpha$	2
$\beta$	1
  - $s$  (right):

A	B
$\alpha$	2
$\beta$	3

$r - s$ :

A	B
$\alpha$	1
$\beta$	1

Arrows indicate the records in  $r$  that are not present in  $s$  are selected for the result.

So, all records that exist in both these relations will be put together into a single table.

So, here  $\alpha 1$  is coming here  $\beta 1$  is coming here.

(Refer Slide Time: 10:09)

The slide title is "Union of two relations". It shows two relations,  $r$  and  $s$ , represented as tables:

- Relations  $r, s$ :
  - $r$  (left):

A	B
$\alpha$	1
$\alpha$	2
$\beta$	1
  - $s$  (right):

A	B
$\alpha$	2
$\beta$	3

$r \cup s$ :

A	B
$\alpha$	1
$\alpha$	2
$\beta$	1
$\beta$	3

Arrows indicate the records from both  $r$  and  $s$  are combined into a single table.

In terms of relation,  $\alpha 1$  is coming here,  $\alpha 2$  is coming here,  $\beta 1$  is coming here,  $\beta 3$  is coming here and  $\alpha 2$  is coming here. You can see that this record  $\alpha 2$  exists in both the relations. And by the set theoretic notion of uniqueness, in the  $\cup$  they have to be unified. So, one of them will be removed, it does not matter because they are identical

anyway. So, for relations having the same set of attributes we can simply make a  $\cup$  of all its records.

(Refer Slide Time: 10:57)

The slide is titled "Set difference of two relations". It shows two relations,  $r$  and  $s$ , represented as tables:

- Relations  $r, s$ :**

	A	B
$\alpha$	1	
$\alpha$	2	
$\beta$	1	

	A	B
$\alpha$	2	
$\beta$	3	
- $r - s$ :** A third table representing the result of the set difference operation, which contains tuples from  $r$  that are not in  $s$ .
 

	A	B
$\alpha$	1	
$\beta$	1	

On the right, there is a Venn diagram with two overlapping circles labeled  $r$  and  $s$ . The region of circle  $r$  that does not overlap with circle  $s$  is shaded, visually representing the set difference  $r - s$ .

Vertical text on the left edge of the slide reads: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr., 2018".

At the bottom, there is a small video thumbnail of a professor, the text "Database System Concepts - 8<sup>th</sup> Edition", the number "04.9", and the copyright notice "©Silberschatz, Korth and Sudarshan".

So, other the 3rd operation is the a 4th operation is doing a set difference it is works simply as set theoretic difference. Again, the two relations must have the same set of attributes and I can do a difference of  $r-s$  which mean that all tuples which exist in  $r$ , but do not exist in  $s$  will be included. So, this is included because, this is not here, but this is not included because it is in  $s$  this is included.

Because, this is this does not exist in the set  $s$  so it is the set, so you take the set  $r$  and then erase all the records like this which exist in  $s$  and you get  $r-s$ . So, if I look into just to recap if I look into the Venn diagram, then this is these are set  $r-s \in r$ , but  $\notin s$ . So, there is a 4th operation that one can do with the in relational algebra.

(Refer Slide Time: 12:28)

The slide is titled "Set intersection of two relations". It shows two relations,  $r$  and  $s$ , represented as tables:

- Relation  $r$ :

A	B
$\alpha$	1
$\alpha$	2
$\beta$	1
- Relation  $s$ :

A	B
$\alpha$	2
$\beta$	3

The intersection  $r \cap s$  is shown as:

A	B
$\alpha$	2

A hand-drawn Venn diagram illustrates the intersection as the overlap of two circles labeled  $r$  and  $s$ . A note at the bottom left says "Note:  $r \cap s = r - (r - s)$ ".

Other details on the slide include:

- SWAYAM, NIPEL-NOCs Instructor: Prof. P. Das, IIT Kharagpur - Jam-Apri. 2018
- Database System Concepts - 8<sup>th</sup> Edition
- 04.10
- ©Silberschatz, Korth and Sudarshan

Fifth is a set intersection( $\cap$ ) of 2 relations. So, again the two relations need to have the same set of attributes. You can take their intersection, which is the record which belongs to both. It is the record that belongs to both of them and as you are aware now, set intersection actually is not a new operation. It is not a fundamental operation because, if I have  $r$ , if I have  $s$ , then this is  $r - s$ .

Now, if I subtract this  $r - s$  which is this set from  $r$  which is this bigger set, then what will be remaining? This is what will be remaining? So, if I subtract  $r - s$  from  $r$  then what will remain? Is necessarily the intersection of this is  $r \cap s$ . So, set intersection is not a fundamental operation of relational algebra. But, can be used because it can be expressed in terms of set difference.

(Refer Slide Time: 14:03)

joining two relations -- Cartesian-product

- Relations  $r, s$ :

A	B
$\alpha$	1
$\beta$	2

 $r$ 

C	D	E
$\alpha$	10	a
$\beta$	10	a
$\beta$	20	b
$\gamma$	10	b

 $s$ 

- $r \times s$ :

A	B	C	D	E
$\alpha$	1	$\alpha$	10	a
$\alpha$	1	$\beta$	10	a
$\alpha$	1	$\beta$	20	b
$\alpha$	1	$\gamma$	10	b
$\beta$	2	$\alpha$	10	a
$\beta$	2	$\beta$	10	a
$\beta$	2	$\beta$	20	b
$\beta$	2	$\gamma$	10	b

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur. Jam-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
04.11 ©Silberschatz, Korth and Sudarshan

Next comes how can we join two different relations which have different set of attributes? So, relation  $r$  has A B and relation  $s$  has C D E. So, we can take a Cartesian product( $\times$ ). So, taking Cartesian product is making all possible combinations. So, necessarily if since this has two relations and this has 3 relations. So, this will have 1 2 3 4 5 6 7 8 this has 4 relations. So, these are 8, 8 total all possible pairing of relations of  $r$  or records of  $r$  and records of  $s$  are included. So, that is the Cartesian product all possible combinations. This is this is how we can join two relations, but certainly what is a important is something which we will discuss shortly.

(Refer Slide Time: 15:00)

Cartesian-product – naming issue

- Relations  $r, s$ :

A	B
$\alpha$	1
$\beta$	2

 $r$ 

C	D	E
$\alpha$	10	a
$\beta$	10	a
$\beta$	20	b
$\gamma$	10	b

 $s$ 

- $r \times s$ :

A	r.B	s.B	D	E
$\alpha$	1	$\alpha$	10	a
$\alpha$	1	$\beta$	10	a
$\alpha$	1	$\beta$	20	b
$\alpha$	1	$\gamma$	10	b
$\beta$	2	$\alpha$	10	a
$\beta$	2	$\beta$	10	a
$\beta$	2	$\beta$	20	b
$\beta$	2	$\gamma$	10	b

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur. Jam-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
04.12 ©Silberschatz, Korth and Sudarshan

Now, in the Cartesian product then issue may happen because there could be attributes which are common between two relations. So, if you if two attributes are common when you take Cartesian product how do you put their name? Because as with the example show here between r and s the attribute b is common so, how do you take care of that? So, when the, such common names happen then we actually change the name of the attribute with the name of the relation. So, b coming from r will be called **r.b** and b coming from s will be called **s.b**.

(Refer Slide Time: 15:46)

**Renaming a Table**

- Allows us to refer to a relation, (say E) by more than one name.  
 $\rho_x(E)$   
returns the expression  $E$  under the name  $X$
- Relations  $r$

	A	B
$\alpha$	1	
$\beta$	2	

$r$

$r \times \rho_s(r)$

	$r.A$	$r.B$	$s.A$	$s.B$
$\alpha$	1	a	1	
$\alpha$	1	$\beta$	2	
$\beta$	2	a	1	
$\beta$	2	$\beta$	2	

$r \times r$

$r.A$   $r.B$   $s.A$   $s.B$

©Silberschatz, Korth and Sudarshan

And accordingly, the relational algebra, gives you a way to rename a table and put it is name differently. So, this is given a this is given by this relation is given by this symbol  $\rho$ . So, you can using a relation  $r$  you can actually give it a different name  $s$  and do that in terms of. So, with that you can actually because otherwise you cannot compute  $r \times r$ . Because, if you try to do compute  $r \times r$  then you will have  $r.a r.b$  and you will again have  $r.a r.b$ . So, you are using this to rename  $r$  to  $s$  and then compute this. So, renaming a table is another feature which is provided of course, it is not a fundamental operation of the algebra, but this is what makes the any kind of Cartesian product possible.

(Refer Slide Time: 16:55)

The slide title is 'Composition of Operations'. It features two tables illustrating relational algebra operations:

- A table for the Cartesian product  $r \times s$ , showing the result of combining tuples from relations  $r$  and  $s$ .

	A	B	C	D	E
$\alpha$	1	$\alpha$	10	a	
$\alpha$	1	$\beta$	10	a	
$\alpha$	1	$\beta$	20	b	
$\alpha$	1	$\gamma$	10	b	
$\beta$	2	$\alpha$	10	a	
$\beta$	2	$\beta$	10	a	
$\beta$	2	$\beta$	20	b	
$\beta$	2	$\gamma$	10	b	
- A table for the selection  $\sigma_{A=C}(r \times s)$ , showing the result of applying a condition to the Cartesian product.

	A	B	C	D	E
$\alpha$	1	$\alpha$	10	a	
$\beta$	2	$\beta$	10	a	
$\beta$	2	$\beta$	20	b	

Other slide details include:

- SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur. Jam-Apr. 2018
- Database System Concepts - 8<sup>th</sup> Edition
- 04.14
- ©Silberschatz, Korth and Sudarshan

Finally, we can make composition of operations that if for example, what we show here is we have two relations  $r$  and  $s$  you have taken a Cartesian product and then we have taken a selection. So, taken a Cartesian product of  $r$  and  $s$  to produce the table as you can see the process( $r \times s$ ) and then  $\sigma_{A=C}(r \times s)$  based on that condition. So, all these operations can be combined in multiple different ways to give you really complex relational algebra operations.

(Refer Slide Time: 17:33)

The slide title is 'Joining two relations – Natural Join'. It describes the natural join operation:

- Let  $r$  and  $s$  be relations on schemas  $R$  and  $S$  respectively. Then, the "natural join" of relations  $R$  and  $S$  is a relation on schema  $R \cup S$  obtained as follows:
  - Consider each pair of tuples  $t_r$  from  $r$  and  $t_s$  from  $s$ .
  - If  $t_r$  and  $t_s$  have the same value on each of the attributes in  $R \cap S$ , add a tuple  $t$  to the result, where
    - $t$  has the same value as  $t_r$  on  $r$
    - $t$  has the same value as  $t_s$  on  $s$

Other slide details include:

- SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur. Jam-Apr. 2018
- Database System Concepts - 8<sup>th</sup> Edition
- 04.15
- ©Silberschatz, Korth and Sudarshan

There is a nice operation which is a derived one which can be written in terms of other operations which is called a natural join( $\bowtie$ ) which we will use very heavily.

(Refer Slide Time: 17:48)

**Natural Join Example**

- Relations r, s:

	A	B	C	D
r	$\alpha$	1	$\alpha$	$\alpha$
	$\beta$	2	$\gamma$	a
	$\gamma$	4	$\beta$	b
	$\alpha$	1	$\gamma$	a
	$\delta$	2	$\beta$	b

	B	D	E
s	1	a	$\alpha$
	3	a	$\beta$
	1	a	$\gamma$
	2	b	$\delta$
	3	b	$\epsilon$

- Natural Join
- $r \bowtie s$

	A	B	C	D	E
	$\alpha$	1	$\alpha$	$\alpha$	$\alpha$
	$\alpha$	1	$\alpha$	$\alpha$	$\gamma$
	$\alpha$	1	$\gamma$	a	$\alpha$
	$\alpha$	1	$\gamma$	a	$\gamma$
	$\delta$	2	$\beta$	b	$\delta$

$\Pi_{A, r.B, C, r.D, E} (\sigma_{r.B = s.B \wedge r.D = s.D} (r \times s))$

SHIVAM: NTEL NOC MOOC Instructor: Prof. P. P. Doshi, IIT Kanpur, Jan-Apr. 2014  
Database System Concepts - 6<sup>th</sup> Edition

04.16 ©Silberschatz, Korth and Sudarshan

Let me first show you an example of that, suppose I have two relations r and s and what is important is there are some attributes which are common between them. Now, we saw earlier that in Cartesian product, in terms of common attributes. We basically the attributes got renamed in terms of the table name, but this is not what we are looking at in relation natural joint.

What we want to say is, if an attribute is common between two tables then, while you join them, the records from two fields can be joined if their value on that common attribute is same. So, it is what it tries to do is it tries to make a Cartesian product of this two tables ( $r \times s$ ) first take all possible combinations, but then you select only those rows where the values are identical between columns having the same name. So, for example, if you if you look into this row and this row. So, what will happen in the Cartesian product I will have this  **$\alpha 1 \alpha a 1 a \alpha$** .

Let me write it in smaller. So, I am doing a Cartesian product I am looking at this row I am looking at this row.

(Refer Slide Time: 19:39)

Natural Join Example

- Relations  $r, s$ :

	A	B	C	D
$\alpha$	1	$\alpha$	a	
$\beta$	2	$\gamma$	a	
$\gamma$	4	$\beta$	b	
$\alpha$	1	$\gamma$	a	
$\delta$	2	$\beta$	b	

	B	D	E
1	a	$\alpha$	
3	a	$\beta$	
1	a	$\gamma$	
2	b	$\delta$	
3	b	$\epsilon$	

- Natural Join
- $r \bowtie s$

$$\Pi_{A, r.B, C, r.D, E} (\sigma_{r.B = s.B \wedge r.D = s.D} (r \times s))$$

So, I have A B C D I have B D E and  $\alpha \alpha a 1 a \alpha$ . Now, they match on B they match on D. So, I will say this is this will get retained, but in the Cartesian product I will also have the first row of r going with the second row of s  $\alpha 1 \alpha a 3 a \beta$ . Here, the B does not match D does not does match, but the B does not match. So, this particular entry will not go in the final result. To take the Cartesian product and you only retain those rows where the values match for the identically named attribute. That is why, so you take the Cartesian product now look at the expression the attributes common attributes are B and C, B and D.

So, the B attribute is  $r.B$  and  $s.B$ . So, we say that in the Cartesian product  $r.B = s.B$ . The name is common; the value will have to be same similarly D is a common attribute. So,  $r.D$  value in the  $r.D$  and the value in the  $s.D$  has to be same. So, based on the Cartesian product you do a selection for equality of values on attributes which are identical between the two column between the two relations between the two tables.

This is a final selection, as you do that you get a table where there are two Bs  $r.B$   $s.B$  there are two Ds  $r.D$   $s.D$ , but according to this selection for all at all records for all rows the value on  $r.B$  and value on  $s.B$  are same value on  $r.D$  and value on  $r.s.B$  are same. Because that is how we have done the selection. So, there is no reason to keep two B columns or two D columns.

(Refer Slide Time: 22:33)

**Natural Join Example**

- Relations  $r, s$ :

	A	B	C	D
$\alpha$	1	$\alpha$	a	
$\beta$	2	$\gamma$	a	
$\gamma$	4	$\beta$	b	
$\alpha$	1	$\gamma$	a	
$\delta$	2	$\beta$	b	

$r$

	B	D	E
1	a	$\alpha$	
3	a	$\beta$	
1	a	$\gamma$	
2	b	$\delta$	
3	b	$\epsilon$	

$s$

- Natural Join
- $r \bowtie s$

	A	B	C	D	E
$\alpha$	1	$\alpha$	a	$\alpha$	
$\alpha$	1	$\alpha$	a	$\gamma$	
$\alpha$	1	$\gamma$	a	$\alpha$	
$\alpha$	1	$\gamma$	a	$\gamma$	
$\delta$	2	$\beta$	b	$\delta$	

$\Pi_{A, r.B, C, r.D, E}(\sigma_{r.B = s.B \wedge r.D = s.D}(r \times s))$

SWAYAM, NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jam-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
©Silberschatz, Korth and Sudarshan

So now, you project based on **A r .B C r.D and E** which means that  $s.B$  and  $s.D$  are left out you do not project them. So, after you have done this projection, you get the final result of the natural join, which has a union of all the attributes that the two relations at A B C D and B D E union is A B C D E and you have all those records whose values matched on the common attributes between relation r and relation D. So, you can say that if I now do a selection, if I now do a projection on A B C or rather A B C D.

I will get a subset of r if I do a projection on B D E I will get a subset of s. So, this is the natural join operation in relational algebra as you can see this is a derived operation because we could use the Cartesian product selection and projection to get this, but as I tell you we will see more of this when we look at the look at all these different aj query coding, but natural join is one of the most widely used most fundamental relational algebra operation that you will often need beyond selection and projection. So, these were the 6 operations and the important derived operations of relational algebra. Besides that relational algebra has some aggregation operators.

(Refer Slide Time: 24:13)

The slide is titled "Aggregate Operators" in red at the top center. In the top left corner, there is a small image of a sailboat on water. The top right corner has the letters "PPD". On the left side, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jam-Apri- 2018". At the bottom left, it says "Database System Concepts - 8<sup>th</sup> Edition". At the bottom right, it says "04.18 ©Silberschatz, Korth and Sudarshan". The footer also contains several small icons for navigating through the presentation.

- Can we compute:
  - SUM
  - AVG
  - MAX
  - MIN

For example, given a table we could compute the sum of values on a column we could compute average of values, max of values, mean of values. So, these somehow aggregate values of multiple rows on a particular column. And therefore, these are called aggregate operators we will see when we talk about SQL. We will see how this really can be coded in SQL and used, but these are this become very convenient to use because often we will need to know.

If this is I mean these are the instructors. And so, let us see which instructor has a maximum load of courses? How many based on hours or which instructor has? What is the average load on the different instructors and so on? So, in every possible context different aggregate operators are frequently required and they are also available as operators in most of the pure as well as commercial query languages.

(Refer Slide Time: 25:18)

The slide has a header 'Notes about Relational Languages' with a sailboat icon. It lists four points: 'Each Query input is a table (or set of tables)', 'Each query output is a table', 'All data in the output table appears in one of the input tables', and 'Relational Algebra is not Turing complete'. The footer includes 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur', 'Database System Concepts - 8<sup>th</sup> Edition', '04.19', and '©Silberschatz, Korth and Sudarshan'.

- Each Query input is a table (or set of tables)
- Each query output is a table
- All data in the output table appears in one of the input tables
- Relational Algebra is not Turing complete

Finally, to note that relational algebra in relational average every query input is a table. And the output is also a table, it is always manipulating one or more tables into a single table that is what we are, I mean in very simple terms that is the way you can look at it. And all data in the output table appears in one of the input tables that is no new data gets generated. It is basically taking, selecting, combining data from different input tables. It does not generate a new data that is that has to be that is if I see that a, we an attribute for a particular row has a value 15.

Then there must be some input table where there is a record where in that field as an attribute value 50. Otherwise, this cannot happen again relational algebra is not turing complete in the sense that, there are algorithms which cannot be coded in relational algebra. We mentioned this earlier too and that is the reason that is a foundational reason of why the SQL language commercial SQL language is based on relational algebra is not turing complete there.

So, we might need to use other programming languages along with the relational algebra coding for solving some of the application problems.

(Refer Slide Time: 26:50)



### Summary of Relational Algebra Operators

Symbol (Name)	Example of Use
$\sigma$ (Selection)	$\sigma \text{ salary} \geq 85000 \text{ (instructor)}$ Return rows of the input relation that satisfy the predicate.
$\Pi$ (Projection)	$\Pi ID, \text{salary} \text{ (instructor)}$ Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
$\times$ (Cartesian Product)	$\text{instructor} \times \text{department}$ Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
$\cup$ (Union)	$\Pi name \text{ (instructor)} \cup \Pi name \text{ (student)}$ Output the union of tuples from the two input relations.
$-$ (Set Difference)	$\Pi name \text{ (instructor)} - \Pi name \text{ (student)}$ Output the set difference of tuples from the two input relations.
$\bowtie$ (Natural Join)	$\text{instructor} \bowtie \text{department}$ Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.


  
 Database System Concepts - 8<sup>th</sup> Edition      04.20      ©Silberschatz, Korth and Sudarshan

To summarize this is the, so this table is what you not only should remember, but you should become a expert of in terms of the operators of relational algebra which we will start using very heavily as we start doing the query coding and processing. So, we talked about selection, which takes rows selectively we talked about projection which takes out certain columns of a table. We talked about Cartesian product of two relations which make all possible combined relations.

We talked about union of records from two tables having identical set of attributes. We talked about set difference which again is the difference of records of one relation from another, given that they have identical set of attributes. We have shown that set difference can be used to also compute set intersection, it is not in the fundamental operation, but is the derived 1 and we have shown a very interesting operation based on Cartesian product selection and projection called natural join where two tables can be joined based on 1 or more common attributes they have.

Now, I am sure you have already noted that if I am doing a natural join between two tables. Which do not have any common attribute then the result is merely the Cartesian product because the selection around the Cartesian product has no condition to select any of the you know any of the fields any of the rows separately. So, it merely turns out to be a Cartesian product.

(Refer Slide Time: 28:38)

Module Summary

- Introduced relational algebra
- Familiarized with the operators of relational algebra

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur. Jam-Apri- 2018

Database System Concepts - 8<sup>th</sup> Edition

04.21

©Silberschatz, Korth and Sudarshan

So, in this module, we have introduced the relational algebra and we have familiarized ourselves with the fundamental and derived operators of relational algebra. Going forward in the next module, will take a deeper look into the relational model and start progressing towards the query design and database design.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture-06**  
**Introduction to SQL/1**

Welcome to module 6 of database management systems, this is the starting of week 2. In week 1 we have done 5 modules, after the overview of the course, we have primarily introduced the basic notions of DBMS and we have discussed about the relational module, the fundamentals of it.

(Refer Slide Time: 00:28)

**Week 01 Recap**

- **Module 01: Course Overview**
  - Why Databases?
  - KYC: Know Your Course
- **Module 02: Introduction to DBMS/1**
  - Levels of Abstraction
  - Schema & Instance
  - Data Models
  - DDL & DML
  - SQL
  - Database Design
- **Module 03: Introduction to DBMS/2**
  - Database Design
  - Database Engine, Users, Architecture
  - History of DBMS
- **Module 04: Introduction to Relational Model/1**
  - Attribute Types
  - Relation Schema and Instance
  - Keys
  - Relational Query Languages
- **Module 05: Introduction to Relational Model/2**
  - Relational Operations
  - Aggregate Operations

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jun-Aug' 2018

Database System Concepts - 6<sup>th</sup> Edition

06.2

©Silberschatz, Korth and Sudarshan

(Refer Slide Time: 00:54)

**Module Objectives**

- To understand relational query language
- To understand data definition and basic query structure

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: June-August, 2018

Database System Concepts - 6<sup>th</sup> Edition

06.3

©Silberschatz, Korth and Sudarshan

In this background, in this week we are primarily focusing on the query language the structured query language SQL. So, all the 5 modules will relate to discussions on query language and this module 6 7 and 8, the 3 modules will introduce SQL at a first level and the last 2 modules 9 and 10, I am sorry 9 and 10 will discuss about intermediate, that is somewhat advanced level of features in the SQL.

The objective of the current module is to, understand the relational query language and particularly the data definition and the basic query structure, that will hold for all SQL queries, particularly this the modules this week would be important for writing any kind of database applications, squaring the database to find information from the existing data and to manipulate it. So, please put a lot of focus in the whole material of this week and practice them well, to understand the basic issues of database systems, in depth in a well oriented manner. In this module we will first talk about the history and then we will see how to define data and start manipulating them.

(Refer Slide Time: 02:25)

Module Outline

- History of SQL
- Data Definition Language (DDL)
- Basic Query Structure (DML)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur.. Date-Apr-2018  
Database System Concepts - 6<sup>th</sup> Edition 06.4 ©Giberschatz, Korth and Sudarshan

(Refer Slide Time: 02:37)

## History of Query Language

- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
  - SQL-86
  - SQL-89
  - SQL-92
  - SQL:1999 (language name became Y2K compliant!)
  - SQL:2003
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
  - Not all examples here may work on your particular system

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur.. Date-Apr-2018  
Database System Concepts - 6<sup>th</sup> Edition 06.6 ©Giberschatz, Korth and Sudarshan

SQL was originally called IBM sequel language and was a part of system R, it was subsequently renamed as structured query language and like any other good programming language that we have, SQL also gets standardized by ANSI and ISO and there have been several standards of SQL, that has come up with SQL 92 being the most popular one, and the commercial system, most of them try to provide support for SQL 92 features, but they do vary between themselves. So, it is possible that the examples that we show here, may or may not all of them execute in the system that you are using. So, you will have to look at what standard your system is actually following.

So, the first what we will talk about is a DDL data definition language, as we had discussed earlier this is the features to create that schema, the tables in a database management system.

(Refer Slide Time: 03:59)

The slide has a decorative header featuring a sailboat icon. The title 'Data Definition Language' is centered in red. Below the title is a text block: 'The SQL data-definition language (DDL) allows the specification of information about relations, including:' followed by a bulleted list. At the bottom left is a video player showing a professor speaking, and at the bottom right are navigation icons.

The SQL data-definition language (DDL) allows the specification of information about relations, including:

- The schema for each relation
- The domain of values associated with each attribute
- Integrity constraints
- And as we will see later, also other information such as
  - The set of indices to be maintained for each relations
  - Security and authorization information for each relation
  - The physical storage structure of each relation on disk

So, it allows for the creation or definition of the schema, for each relation that we have in the database it specifies the domain of values associated with each attribute of the schema and it also defines a variety of integrity constraints, later in the course we will see that, it also has to specify other related information like indexing, security authorization, physical storage and so on.

(Refer Slide Time: 04:30)

The slide has a header 'Domain Types in SQL' with a sailboat icon. The content lists various SQL domain types with descriptions:

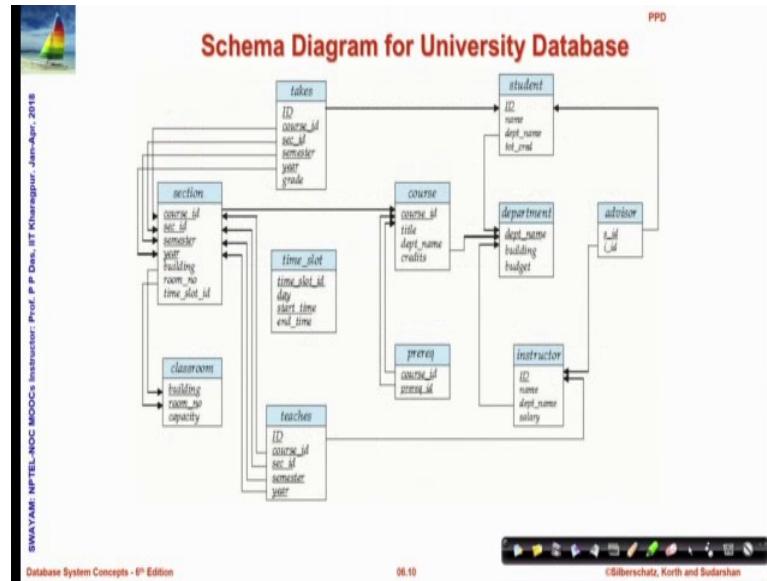
- **char(n).** Fixed length character string, with user-specified length  $n$
- **varchar(n).** Variable length character strings, with user-specified maximum length  $n$
- **int.** Integer (a finite subset of the integers that is machine-dependent)
- **smallint.** Small integer (a machine-dependent subset of the integer domain type)
- **numeric(p,d).** Fixed point number, with user-specified precision of  $p$  digits, with  $d$  digits to the right of decimal point. (ex., **numeric(3,1)**, allows 44.5 to be stored exactly, but not 444.5 or 0.32)
- **real, double precision.** Floating point and double-precision floating point numbers, with machine-dependent precision
- **float(n).** Floating point number, with user-specified precision of at least  $n$  digits
- More are covered in Chapter 4

Navigation icons and slide information are visible at the bottom.

So, first the domain of possible values. So, we have already specified that, every domain in SQL is more of an atomic nature. So, they are more like the primitive or built in data types of languages like c, c++, java. So, the common domain types are character which are basically strings of character, having a certain length then you can have variable character string which means that the length here specifies that, the maximum length that the string can take, but a string could be shorter than that integer; obviously, then small integer, which in a system may give a smaller range of integer values.

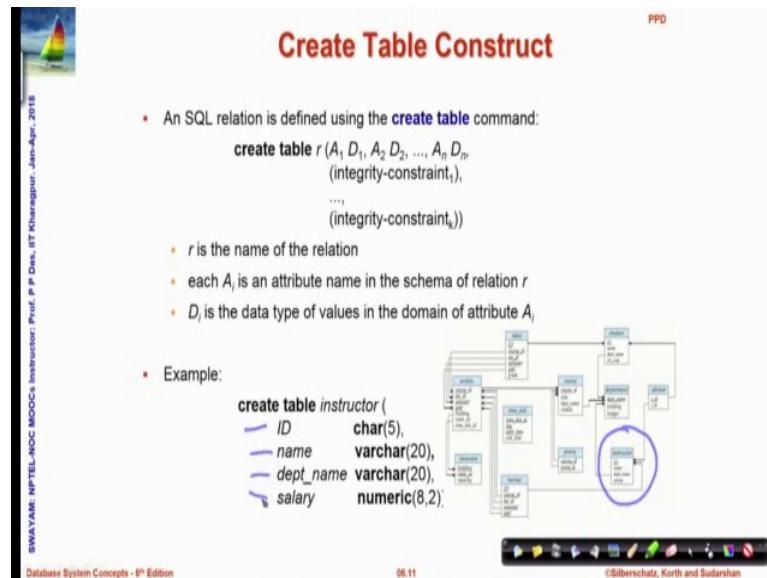
Then there is a numeric type which is often very important, which says what is the precision of the numbers that are to be written in this format stored in this format. So, d basically gives that precision value and p gives the size, then you can have real and double precision numbers you can have floating point numbers and so on, and there are some more data types, which we will discuss later in the course.

(Refer Slide Time: 05:52)



Given all these domain types; so, what we will try to do is, here is a schema for the university database, which has multiple different relations designed in that showing the attributes and marking out, what are the keys? And what are the foreign keys? So, we would take examples of some of these and try to code them in the SQL.

(Refer Slide Time: 06:19)

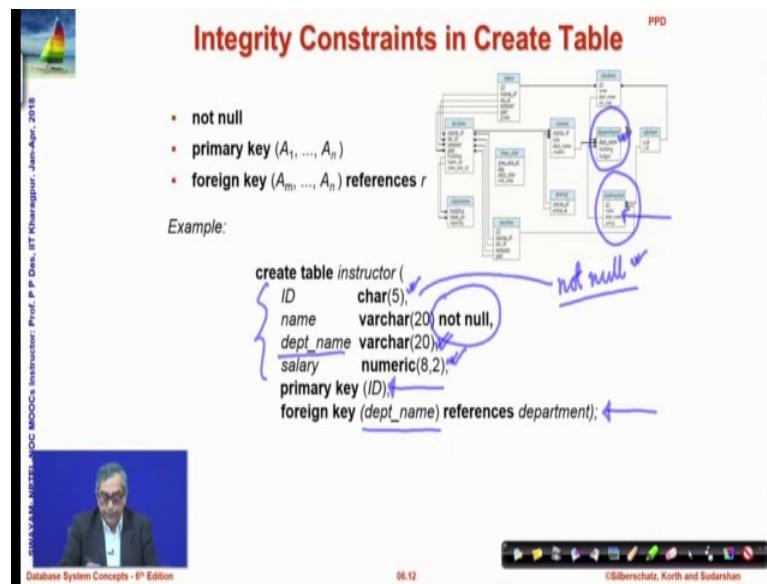


Now, to create a table this is how you go about, the SQL keywords CREATE TABLE is the basic command. So, with the CREATE TABLE you have to specify a name, here the name that is given is in terms of this name `r`, which is the name of the relation and then

you provide a series of attributes, separating by comma  $A_i$  are different attributes and for every attribute, there is a corresponding type domain type specified. So, it says that  $A_1$  is of domain type  $D_1$ ,  $A_2$  is of domain type  $D_2$  and so on. And all of these attribute descriptions, are then followed by a series of integrity constraints. It is possible that a CREATE TABLE may not provide any constraint, but often you will have a number of constraints to work with. So, here is one example.

So, in this we are trying to code the creation of this instructor table, as you can see it has 4 different fields ID, name, department name and salary and for each one we have specified the domain type. So, ID is char 5, this means that the identity of this table instructor will be strings of length 5 whereas, the name or the department name are strings, but they have a maximum length 20, but they could have be of variable length where a salary is of numeric type having specification (8, 2). So, it can have 2 decimal places and be of size 8 maximum. So, this is the basic form of definition that we have for creating a table, defining a table or defining a schema.

(Refer Slide Time: 08:28)



Now, we can add a number of integrity constraints to the CREATE TABLE, 3 integrity constraints we will discuss here, one is not null, one is primary key and other third is foreign key. So, not null we specify whether a field can be null or not, primary key as we have seen will specify the attributes which form the primary key, and the foreign key will specify the attributes which reference some other table and our key in that table. So,

here is an example, in the instructor relation here, we had seen this part, the attribute what we have added here is, this not null. So, we say the name is not null which means that, in the instructor table it is not possible to insert a record, where the name of the instructor is null that is unknown, but it is possible, if the same thing is not said about department name same thing is not said about salary. So, it is possible that these could be null.

Now, we additionally say that primary key is ID. So, this field ID is a primary key and it is a property of SQL CREATE TABLE command that, if an attribute is referred as a primary key, then it cannot be not null. So, you do not need to specify that is here you do not need to write not null, because it is a primary key it will be known to be not null, because certainly we have discussed that key is the distinguishing attribute in a database table. So, it cannot be null.

So, it will not be able to distinguish similarly, we have finally we have the third integrity constant which is foreign key which says that, it is referencing this table department and the foreign key of this is here, the dep\_name this particular field is a foreign key, which is a key of the department table. And so, we will be able to refer this, from this table as a foreign key and we know that it will be key in the department table. So, these are the ways to specify the integrity constraint.

(Refer Slide Time: 11:24)

**And a Few More Relation Definitions**

```

CREATE TABLE student (
    ID          VARCHAR(5),
    name        VARCHAR(20) NOT NULL,
    dept_name   VARCHAR(20),
    tot_cred    NUMERIC(3,0),
    PRIMARY KEY (ID),
    FOREIGN KEY (dept_name)
        REFERENCES department;
)

CREATE TABLE takes (
    ID          VARCHAR(5),
    course_id  VARCHAR(8),
    sec_id     VARCHAR(8),
    semester   VARCHAR(6),
    year       NUMERIC(4,0),
    grade      VARCHAR(2),
    PRIMARY KEY (ID, course_id, sec_id, semester, year),
    FOREIGN KEY (ID) REFERENCES student,
    FOREIGN KEY (course_id, sec_id, semester, year) REFERENCES section;
)

Note: sec_id can be dropped from primary key above, to ensure a student cannot be registered for two sections of the same course in the same semester

```

The diagram illustrates the following relationships:

- student** table has a primary key **ID**.
- student** table has a foreign key **dept\_name** referencing **department** table.
- takes** table has a primary key consisting of **ID**, **course\_id**, **sec\_id**, **semester**, and **year**.
- takes** table has a foreign key **ID** referencing **student** table.
- takes** table has a foreign key **course\_id**, **sec\_id**, **semester**, and **year** referencing **section** table.
- section** table has a primary key **course\_id**.
- section** table has a foreign key **professor\_id** referencing **professor** table.
- professor** table has a primary key **professor\_id**.
- professor** table has a foreign key **dept\_name** referencing **department** table.
- department** table has a primary key **dept\_name**.

So, here are couple of more examples. So, I will not go through them in detail, I will request you to take time and carefully understand them, again these are about different relations that exist here, about the student and about the courses that the students take, and in every case we have specified the set of fields, that you have in the table in the design of the schema are listed, in the CREATE TABLE the ID information about the primary key is provided and also the information about the foreign key.

Here department name is the foreign key which is mapping to this point, similar things can be observed about the takes relationship which show how students are actually taking courses. So, it relates different it has a set of fields but it has 2 kinds of foreign keys, one that relate to the student through the ID and this combination of attributes which refer to the section.

So, this is how different tables can be created using the data definition language, here is a note that you should observe that if you consider this section ID, the section ID is a part of the primary key which means that 2 records cannot be same, if they are different in the section ID, then such records are allowed. So which means that it is possible that, a student can attend or take a course in the same semester, in the same year with 2 different section IDs because they are primary keys.

So, they can be different. So, if we drop this from the primary key then we will enforce the condition that, no student will be able to take a course, in 2 sections in the same semester and the same year. So, these are the different design choices that we have and we will move on, here is one more example trying to show you the CREATE TABLE command for the course relation, that we have in the university database.

(Refer Slide Time: 13:59)

And more still

```
create table course (
    course_id      varchar(8),
    title          varchar(50),
    dept_name      varchar(20),
    credits         numeric(2,0),
    primary key (course_id),
    foreign key (dept_name) references department);
```

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur.. Date: Apr-2018

Database System Concepts - 6<sup>th</sup> Edition 06.14 ©Silberschatz, Korth and Sudarshan

(Refer Slide Time: 14:11)

Updates to tables

- **Insert**
  - insert into instructor values ('10211', 'Smith', 'Biology', 66000);
- **Delete**
  - Remove all tuples from the student relation
    - ↳ delete from student
- **Drop Table**
  - drop table r
- **Alter**
  - alter table r add A D
    - ↳ where A is the name of the attribute to be added to relation r and D is the domain of A
    - ↳ All existing tuples in the relation are assigned null as the value for the new attribute
  - alter table r drop A
    - ↳ where A is the name of an attribute of relation r
    - ↳ Dropping of attributes not supported by many databases

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur.. Date: Apr-2018

Database System Concepts - 6<sup>th</sup> Edition 06.15 ©Silberschatz, Korth and Sudarshan

Moving on, let us look at how to update or actually put in different records, in a table which has already been created, the basic command is insert, and the keywords for that is INSERT INTO and values, in between you write the name of the relation, where the record will have to be inserted and then the values, will have to be put as a tuple in the same order in which you have defined the attributes of that relation, and certainly each of the values like this is ID value, next is the name value, the department, the salary, each one of them should be from the same domain type, as has been specified during the CREATE TABLE come on.

So, these things will have to remember. And so, every record will get inserted through one insert command, similarly a deletion can be done by DELETE FROM students, if you do DELETE FROM students without specifying which record you want to delete, basically all records will get deleted. We will see how selective deletion will happen, that will come on later. DROP TABLE is a command to remove a table, a table that has been created can be removed from the database altogether, by doing DROP TABLE and the relation name you can also change the schema of a table by using ALTER TABLE.

So, the form is alter table is a key words, you can add a new attribute to relation are by writing the name of the attribute and the domain of the attribute one after the other. Similarly, it is possible also to drop an attribute that already exists and the syntax for that will be, ALTER TABLE the relation name drop is a keyword and the name of the attribute, mind you all database systems may not allow you to drop an attribute to ALTER TABLE to remove attributes. And so, it works in some and it does not work in the rest.

(Refer Slide Time: 16:54)

**Basic Query Structure**

- A typical SQL query has the form:

```
select A1, A2, ..., An
from R1, R2, ..., Rm
where P
```

- A<sub>i</sub> represents an attribute
- R<sub>j</sub> represents a relation
- P is a predicate

A <sub>1</sub>	A <sub>2</sub>	A <sub>n</sub>

UNIVARIATE METRIC-MOOC Instructor: Prof. P. P. Datta, IIT Kharagpur - Jain-Agarwal, 2015  
Database System Concepts - 6<sup>th</sup> Edition  
06.17  
©Übersetzung, Korith und Sudarshan

Now, that was about the definition of the table and the basic definition of the data. So, now we will get into the basics query structure which is with tables with existing data, how do I query and find out different information.

So, the structure of an SQL query and this you should observe very carefully is normally said to be, SELECT FROM WHERE colloquially will often say, let us have a SELECT

FROM WHERE. So, it has 3 keywords select which is followed by a set of; this is a set of attributes. So, this specifies that, when a select query runs it will finally give us a new relation and in that relation, the attributes that will be available are the attributes that feature in the select list. The next clause or the next keyword in this is from, which specifies a set of existing relations.

So,  $r_1, r_2, r_m$  represent different relations and these are the relations, which will be used to actually find the information extract the information. Finally, the where clause has a predicate as a condition, which specify that what condition has to be satisfied. So, that certain peoples from the relations  $r_1$  to  $r_m$ , will be chosen and put in this new selected result, table in terms of the attributes  $A_1$  to  $A_n$ .

(Refer Slide Time: 18:37)

The select Clause

- The **select** clause lists the attributes desired in the result of a query
  - corresponds to the projection operation of the relational algebra
- Example: find the names of all instructors:

```
select name  
      from instructor
```
- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
  - E.g., *Name*  $\equiv$  *NAME*  $\equiv$  *name*
  - Some people use upper case wherever we use bold font

SWAYAMPRABHA MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr-2018

Database System Concepts - 8<sup>th</sup> Edition

06.18

Gilberschatz, Korth and Sudarshan

So, this is the basic understanding of the structure of the SQL query and naturally as I mentioned that will result in a relation. Now, we will go over each and every clause carefully, the select clause as I said will list all the attributes. So, it is like a projection (**Π**) in terms of the relational algebra that we have done. So, if we write `select name from instructor` then this will result in finding the names of all instructors from the instructor table, because this is, this you know is a relation, because it is happening it is a featuring in the from clause and in select, we are saying that the attribute that we want to select is the attribute name.

So, it will the instructor table has 4 attributes ID, name, dept. name and salary from that it will simply take the name of the instructor and list that in the output table. So, the basic form of selection that happens and this point you may also note, that in SQL everything is case insensitive, it does not matter whether you write in upper case or lower case. So, you can choose the style that you prefer to use.

(Refer Slide Time: 19:51)

The slide is titled "The select Clause (Cont.)" in red. It contains a bulleted list of points about SQL duplicates:

- SQL allows duplicates in relations as well as in query results
- To force the elimination of duplicates, insert the keyword **distinct** after select
- Find the department names of all instructors, and remove duplicates

Below the list are two code snippets:

```
select distinct dept_name  
from instructor
```

and

```
select all dept_name  
from instructor
```

On the left side of the slide, there is a vertical sidebar with a sailboat icon at the top and a video thumbnail of a man speaking below it. The sidebar also contains some text and a logo at the bottom.

This is a very important factor, that you should keep in mind that we said, while introducing relational algebra, that in the relational algebra every relation is a set and which means that according to set theory. We cannot have 2 tuples in the same relation, which are identical in all its values, because set theory does not allow that, but please keep in mind that SQL actually allows duplicates in relations. So, it is possible that in the same relation in the same table, I may have more than one record which are identical in all the fields, in all the attributes of that table and this will have lot of consequences and we will see how often, this property will have to be used.

So, if you want a typical set theoretic kind of output, that is if you want the result to be distinct, all records to be distinct, then you have to explicitly say that you want distinct values to be selected. So, all that you are doing is, you are after select and before the attribute name, you introduce another keyword distinct.

So, select distinct dept name from instructor will actually, select the departments of all instructors and quite why if it just selects department name of all instructor, then it is

quite possible that the same department name will appear number of times, because every department has multiple instructors. But when we use distinct, then every name will feature only once in that selection, then you can also specify another keyword all, which ensures that the duplicates are not removed. So, if you do select all depth name, then all the names will feature with duplicates, if some department had 3 instructors the name of that department will feature thrice.

(Refer Slide Time: 22:05)

The slide title is "The select Clause (Cont.)". It features a small sailboat icon in the top left corner and a video camera icon in the bottom right corner. The main content is a bulleted list of points about the SELECT clause:

- An asterisk in the select clause denotes "all attributes"  
`select *  
from instructor`
- An attribute can be a literal with no `from` clause  
`select '437'`
  - Results is a table with one column and a single row with value "437"
  - Can give the column a name using:  
`select '437' as FOO`
- An attribute can be a literal with `from` clause  
`select 'A'  
from instructor`
  - Result is a table with one column and  $N$  rows (number of tuples in the `instructor` table), each row with value "A"

On the left side of the slide, there is a vertical sidebar with the text "SWAYAM-NIDHI-NOOC-MOOCs Instructor: Prof. P. Datta, IIT Kharagpur - Jan-Apr-2018". At the bottom left, there is a video thumbnail showing a man with glasses and a blue background. At the bottom right, it says "06.20" and "©Giberschutz, Korth and Sudarshan".

You can use an asterisk after select, to specify that you are interested in all the attributes that the relation or the collection of relations in the from clause has. You can also specify a select with a literal and without a from clause, if you do that then it will simply return you a table with a single row having that literal value and you can also rename that table, using what is known as clause as a command. So, this will give you a table Foo, where there is only one row and that row has an entry 437. You can use that, for other purposes also you can do a select of a literal, from a table with using a from clause where in, you will get a single column table, where as many is as there are records in the instructor will be produced.

(Refer Slide Time: 23:16)

The slide title is "The select Clause (Cont.)". The content discusses the use of arithmetic expressions in the select clause, mentioning division by 12 and renaming the result. A query example is shown:

```
select ID, name, salary/12
from instructor
```

The slide also includes a small video player showing a speaker and navigation icons.

Select clause can also use basic arithmetic operations for example, here we are showing a select where the third attribute as you can see, the third attribute is salary by 12, assuming that the instructor table has a salary number which is annual salary by 12 naturally you give you the monthly salary. So, those such arithmetic choices can also be made. you can also rename that field that particular salary by 12 field by a new name, as I said as can be used to rename. So, if we if you use that then, when you get the output you will get the column names are ID, name and monthly salary and in monthly salary you will actually have a computation, which is salary by 12 and in the same way, you can use multiple different kinds of arithmetic operators.

(Refer Slide Time: 24:16)

The slide has a header 'The where Clause' with a sailboat icon. The content includes a bulleted list, code snippets, and a footer with navigation icons.

- The **where** clause specifies conditions that the result must satisfy
  - Corresponds to the selection predicate of the relational algebra
- To find all instructors in Comp. Sci. dept

```
select name  
from instructor  
where dept_name = 'Comp. Sci.'
```
- Comparison results can be combined using the logical connectives **and**, **or**, and **not**
  - To find all instructors in Comp. Sci. dept with salary > 80000

```
select name  
from instructor  
where dept_name = 'Comp. Sci.' and salary > 80000
```
- Comparisons can be applied to results of arithmetic expressions

Navigation icons at the bottom include arrows, a magnifying glass, and other symbols.

Now, we come to the where clause, where clause specifies the condition is a predicate which corresponds to the selection predicate of relational algebra. So, it will specify some condition, here is an example if we want to find all instructors from the instructor table, who are associated with computer science department, then you can say select name from instructor and to specify that they are from the computer science department, you will specify department name is equal to computer science.

So, this will ensure that you select the records only when this condition is satisfied. So, all records for which department name is different from computer science will not be included here. You can also write predicates using the different logical connectives and or not and so on. So, here is an example, where you are finding all instructors in computer science with salary greater than 80000. So, here we have used and clause. So, only records where the department name is computer science and salary is greater than 8000 will be chosen in the result. So, and then the projection (**Π**) will be done on the name of those instructors, you can apply comparisons of arithmetic expression. So, where clause can really write different kind of things.

(Refer Slide Time: 26:04)

The slide has a header 'The from Clause' with a sailboat icon. It contains a bulleted list of points about the 'from' clause, followed by a SQL query example, and ends with a footer.

- The **from** clause lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra
- Find the Cartesian product *instructor* *X* *teaches*

```
select *  
from instructor, teaches
```

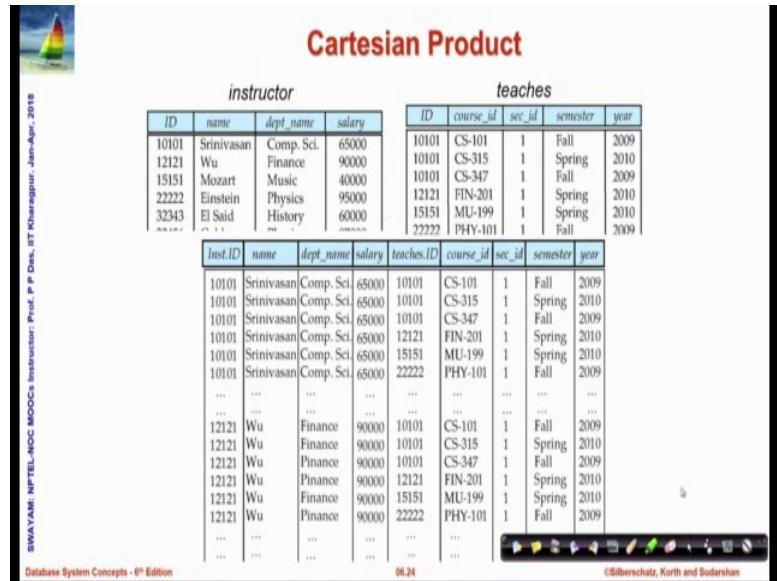
  - generates every possible *instructor* – *teaches* pair, with all attributes from both relations
  - For common attributes (e.g., *ID*), the attributes in the resulting table are renamed using the relation name (e.g., *instructor.ID*)
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra)

Database System Concepts - 8<sup>th</sup> Edition      06.23      ©Giberschatz, Korth and Sudarshan

Finally, the from clause is sets all the different relations from where you are actually looking for the records. So, it kind of corresponds to the Cartesian product of the relational algebra. So, if we want to say compute instructor Cartesian product teachers, then you can say SELECT \* instructor one table comma teachers. So, this will choose records from instructed relation, as well as from teacher's relation and in all possible combined way, it will put them in the output we have used a star.

So, all fields of in instructor and all fields of teaches will be there in the output, and since some fields may have identical name like ID in instructor and there is ID in teachers, they will be qualified by the name of the relation, from can have 1 relation, 2 relation any number of relations as you require. So, this will cause the Cartesian product to be computed which may not be very useful as an independent feature, but we will see in the next module how it can give very important computations, in terms of computing joints and so on.

(Refer Slide Time: 27:37)



The slide illustrates the Cartesian product of two relations: *instructor* and *teaches*. The *instructor* relation has attributes *ID*, *name*, *dept\_name*, and *salary*. The *teaches* relation has attributes *ID*, *course\_id*, *sec\_id*, *semester*, and *year*. The resulting Cartesian product relation has columns *inst.ID*, *name*, *dept\_name*, *salary*, *teaches.ID*, *course\_id*, *sec\_id*, *semester*, and *year*.

instructor				teaches				
<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>	<i>ID</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2009
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2010
32343	El Said	History	60000	15151	MU-199	1	Spring	2010
...	...	...	...	22222	PHY-101	1	Fall	2009

Database System Concepts - 6<sup>th</sup> Edition      06.24      ©Silberschatz, Korth and Sudarshan

So, here is an example of the Cartesian product that we talked of. So, here is a instructor relation, the teacher's relation and as you can see when we have done this cartesian product, that is `SELECT * from instructor comma teachers`, then all fields this is an ID of the instructor, there is an ID in teachers.

So, that is also specified here qualified by the name of the relation whereas, name comes in directly because there is no attribute called name in teachers, the department name comes in directly, salary comes in, directly course ID comes in, section ID comes in, semester comes in, year comes in and so on. And the combination of all tuples in their instructor relation, against all tuples of the teacher's relation all possible combinations have come in this result, which eventually is a cartesian product of the relational algebra.

(Refer Slide Time: 28:50)

**Module Summary**

- Introduced relational query language
- Familiarized with data definition and basic query structure

SWAYAM - NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: June-August, 2018

Database System Concepts - 6<sup>th</sup> Edition

06.25

©Silberschatz, Korth and Sudarshan

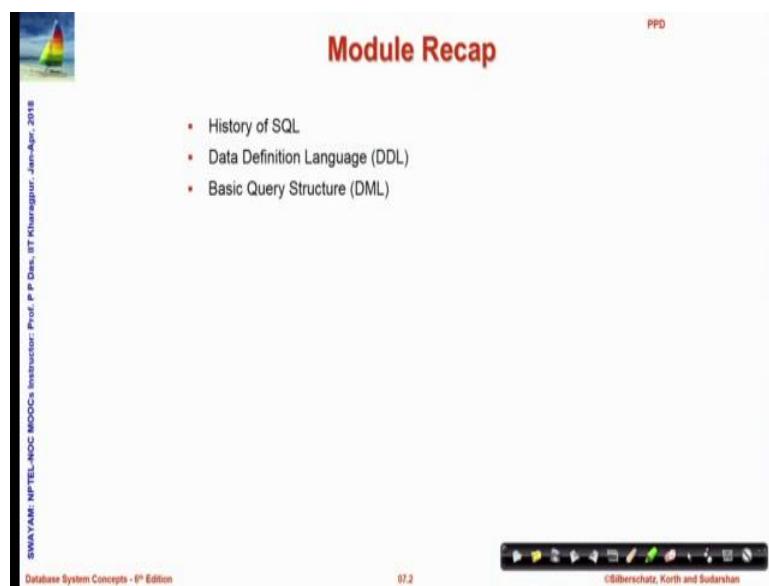
So, this is what we have to summarize, we have introduced the relational query language and particularly familiarize ourselves with the data definition that is creation of the table creation, of the schema with the attribute names, domain types and constraints. And the updates to the table in terms of insertion and deletion of values or addition or deletion of attributes or removing a table altogether and then, we have given the basic structure of the SELECT FROM WHERE query of SQL, which will be the key language feature of a query language, that we will continue to discuss all through this course.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 07**  
**Introduction to SQL/2**

Welcome to module 7 of database management systems, this is a second part out of total 3 of introduction to SQL.

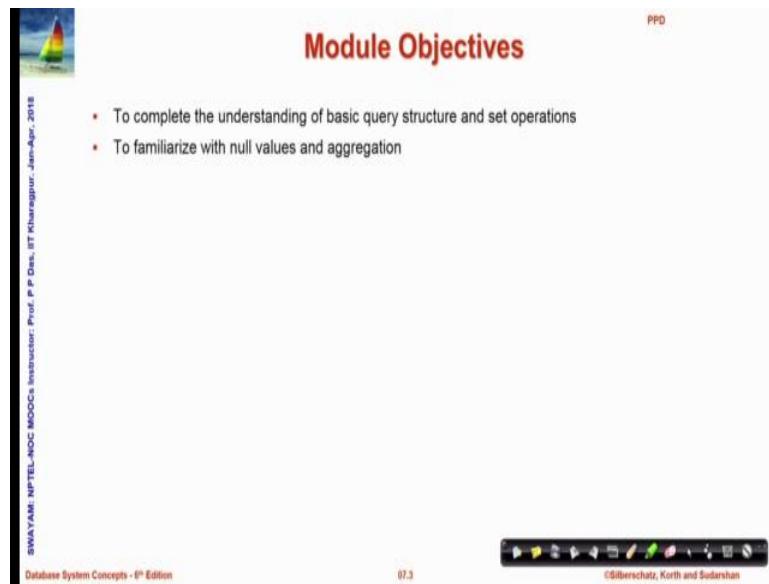
(Refer Slide Time: 00:31)



The slide is titled "Module Recap" in red at the top right. It features a small sailboat icon in the top left corner. A vertical sidebar on the left contains the text "SWAYAM: NPTEL-AHOC MOOC Instructor: Prof. P P Das, IIT Kharagpur, Jan-Apr., 2018". The main content area lists three bullet points: "History of SQL", "Data Definition Language (DDL)", and "Basic Query Structure (DML)". At the bottom, there is footer text: "Database System Concepts - 6<sup>th</sup> Edition", "07.2", and "©Silberschatz, Korth and Sudarshan". A navigation bar with various icons is at the very bottom.

In the last module, we have discussed about the evolution of SQL the data definition language part of it and the basic structure of queries.

(Refer Slide Time: 00:42)



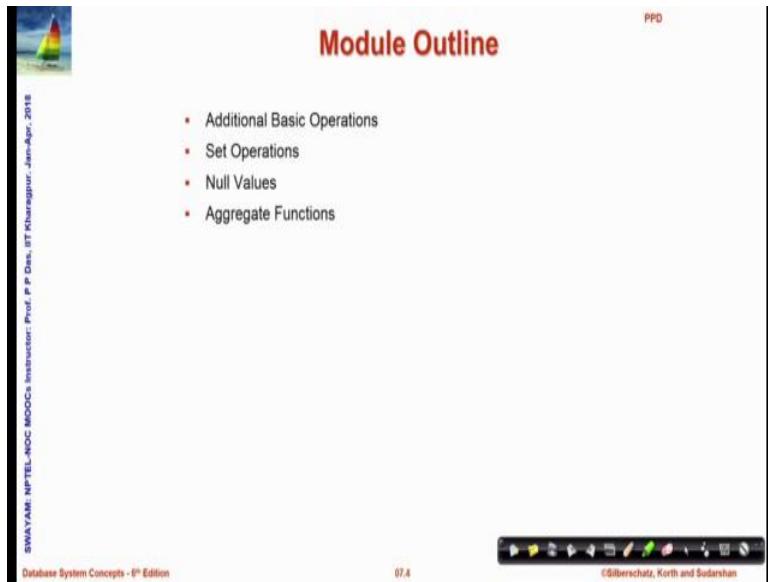
The slide has a header 'Module Objectives' in red. Below it is a bulleted list of objectives:

- To complete the understanding of basic query structure and set operations
- To familiarize with null values and aggregation

At the bottom left, it says 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Date, IIT Kharagpur, Jan-Apr., 2018'. At the bottom center, it says 'Database System Concepts - 6<sup>th</sup> Edition' and '07.3'. At the bottom right, it says '©Silberschatz, Korth and Sudarshan'.

In the current module we will complete the understanding of the basics queries structure and will see how, common set theoretic operations can be performed in terms of queries. We will familiarize ourselves with the handling of null values and aggregation operation that will be frequently required for forming queries.

(Refer Slide Time: 01:05)



The slide has a header 'Module Outline' in red. Below it is a bulleted list of topics:

- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions

At the bottom left, it says 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Date, IIT Kharagpur, Jan-Apr., 2018'. At the bottom center, it says 'Database System Concepts - 6<sup>th</sup> Edition' and '07.4'. At the bottom right, it says '©Silberschatz, Korth and Sudarshan'.

So, this is a module outline, these are the topics, that we will discuss. So, we started the discussion of some more basic operations in the query.

(Refer Slide Time: 01:15)

The slide has a header 'Cartesian Product' with a sailboat icon. The content includes a bulleted list and some SQL code:

- Find the Cartesian product *instructor* X *teaches*
- ```
select *  
from instructor, teaches
```
- generates every possible *instructor* – *teaches* pair, with all attributes from both relations
- For common attributes (e.g., *ID*), the attributes in the resulting table are renamed using the relation name (e.g., *instructor.ID*)
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra)

At the bottom, there is a video player showing a person speaking, the text 'Database System Concepts - 6th Edition', the number '07.6', and the copyright '©Übersetzung, Korth und Sudarshan'.

We have already discussed this that, if we do SELECT \* FROM 2 tables then it results in a Cartesian product we have seen this result earlier..

(Refer Slide Time: 01:26)

The slide has a header 'Cartesian Product' with a sailboat icon. It shows two tables: 'instructor' and 'teaches'.

| ID    | name       | dept_name  | salary |
|-------|------------|------------|--------|
| 10101 | Srinivasan | Comp. Sci. | 65000  |
| 12121 | Wu         | Finance    | 90000  |
| 15151 | Mozart     | Music      | 40000  |
| 22222 | Einstein   | Physics    | 95000  |
| 32343 | El Said    | History    | 60000  |

| ID    | course_id | sec_id | semester | year |
|-------|-----------|--------|----------|------|
| 10101 | CS-101    | 1      | Fall     | 2009 |
| 10101 | CS-315    | 1      | Spring   | 2010 |
| 10101 | CS-347    | 1      | Fall     | 2009 |
| 12121 | FIN-201   | 1      | Spring   | 2010 |
| 15151 | MU-199    | 1      | Spring   | 2010 |
| 22222 | PHY-101   | 1      | Fall     | 2009 |

The resulting Cartesian product table has 18 rows, combining each row from the first table with each row from the second table.

At the bottom, there is a video player showing a person speaking, the text 'Database System Concepts - 6th Edition', the number '07.7', and the copyright '©Übersetzung, Korth und Sudarshan'.

(Refer Slide Time: 01:32)

The slide shows a SQL query being executed:

```

    * Find the names of all instructors who have taught some course and the course_id
      select name, course_id
      from instructor, teaches
      where instructor.ID = teaches.ID
  
```

Annotations highlight the 'instructor' and 'teaches' tables in the query, and the 'ID' column in both tables.

Below the query, the results are shown as two tables:

| ID    | name       | dept_name  | salary |
|-------|------------|------------|--------|
| 10101 | Srinivasan | Comp. Sci. | 65000  |
| 12121 | Wu         | Finance    | 90000  |
| 15151 | Mozart     | Music      | 40000  |
| 22222 | Einstein   | Physics    | 95000  |
| 32343 | El Said    | History    | 60000  |

| ID    | course_id | sec_id | semester | year |
|-------|-----------|--------|----------|------|
| 10101 | CS-101    | 1      | Fall     | 2009 |
| 10101 | CS-315    | 1      | Spring   | 2010 |
| 10101 | CS-347    | 1      | Fall     | 2009 |
| 12121 | FIN-201   | 1      | Spring   | 2010 |
| 15151 | MU-199    | 1      | Spring   | 2010 |
| 22222 | PHY-101   | 1      | Fall     | 2009 |

Annotations show arrows pointing from the first table to the second, indicating an 'Equi-Join, Natural Join'.

At the bottom, there is a watermark: 'BINAYAK NPTEL-NOC MOOCs Instructor: Prof. P. Dini, IIT Kharagpur - Jan-2017 - 2018'.

Know by itself as we had said that, by itself the Cartesian product may not be very useful, but suppose we want to answer this kind of a query, that find all names of all instructors who have taught some course and for those you also write the course\_id. So, what we are interested in is, the name of the instructor and course\_id.

So, we put that on the select course these are the 2 tables that are required because, the name of the instructor is there in the instructor table relation and then the relationship between which instructors teach which course is in the teachers table. So, in the instructor table we have the name in the teachers table, we have the course\_id and also this relationship as to which course is taught by which teacher and here, we have the relationship between which id of the instructor has which name. So, what we do is, when we do this Cartesian product will get something like this, as we have already seen, but we want to qualify it by with a predicate which say that, we will out of all these combinations will choose only those where, the instructor.ID equals the teaches.ID. So, if we look at say in the first row the instructor.ID is this, teachers.ID is this, which is id of the teachers table they are same. So, which says the this instructors Srinivasan actually teaches the course CS-101 whereas, if you look into this row, it says that instructor.ID is this and teaches.ID is this and we are really not interested in this combination, because this combination does not convey anything meaningful. So, by the use of this where clause, we will try to choose only those records where, these 2 ids are same which will tell us that, this particular instructor actually teaches that particular course.

So, if we do that, then you will find that majority of the records that, actually came about in the Cartesian product are eliminated from this result. So, I have struck them out. So, you can currently see in this part you can see only 4 records the 3 courses taught by Srinivasan. So, and the one course taught by you. So, in the output you will have this name, this name part and this course\_id part because, you are projecting on these 2. So, this will be the output table which will get generated and just to remind you, this is the notion of natural join that, we had discussed in relational algebra in this case we will actually call it equi join because, we are using a equality condition after the Cartesian product to join these 2 relationship. So, this is a very critical operation in many of cases in our database query system.

(Refer Slide Time: 05:23)

The screenshot shows a presentation slide with the following details:

- Title:** Examples
- Content:**
  - A bullet point: "Find the names of all instructors in the Art department who have taught some course and the course\_id"
  - Below the bullet point is a SQL query:

```
select name, course_id
from instructor, teaches
where instructor.ID = teaches.ID and instructor.dept_name = 'Art'
```
- Navigation:** A toolbar at the bottom includes icons for back, forward, search, and other presentation controls.
- Page Number:** 07.8
- Credit:** ©Silberschatz, Korth and Sudarshan
- Source:** SWAYAM-NPTEL-NOC's INSTRUCTOR: Prof. P. R. Dand, IIT Kharagpur. -Jan-Apr-2018

This is another extension of this similar example. So, here we have added another predicate in the where clause, specifying that instructed or department name is Art.

So, which means this will now, give the names of all instructors in the art department only who have taught some course and specify their course\_id. So, in different such ways, you can manipulate and create queries.

(Refer Slide Time: 05:57)

The slide has a header 'The Rename Operation' with a sailboat icon. It contains the following text and code:

- \* The SQL allows renaming relations and attributes using the **as** clause:  
 $old-name \text{ as } new-name$
- \* Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.  

```
select distinct T.name  
from instructor as T, instructor as S  
where T.salary > S.salary and S.dept_name = 'Comp. Sci.'
```
- \* Keyword **as** is optional and may be omitted  
 $instructor \text{ as } T \equiv \text{instructor } T$

At the bottom left is a video thumbnail of a speaker, and at the bottom right are navigation icons.

It is possible to rename, we have already seen examples you can rename a relation you can rename an attribute and the style is to use AS. So, here you can see that, in the SELECT query we have said from instructor as T. So, the name of this relation can be treated as T and we again see that as instructor as S. So, actually what we are doing, we are doing a join between the same relation; instructor and instructor.

And we are trying to find out all instructor who have higher salary than some instructor in computer science. So, the some instructor in computer science is specified by this condition, because department has to be computer science and the fact that salary is higher. So, as if you treat that, though it is actually a join between a product between instructor and instructor the same relation, but by renaming you treat them as if they are 2 different tables having name T and S, and then it becomes easier to write this kind of query. So, otherwise it is quite difficult to write this query to find out, because you need to actually create a product of one relation with itself.

(Refer Slide Time: 07:30)

The slide features a small sailboat icon in the top left corner. The title 'Cartesian Product Example\*' is centered at the top in red. Below the title is a table with two columns: 'person' and 'supervisor'. The data is as follows:

| person | supervisor |
|--------|------------|
| Bob    | Alice      |
| Mary   | Susan      |
| Alice  | David      |
| David  | Mary       |

Below the table is a bulleted list of three items:

- Find the supervisor of "Bob"
- Find the supervisor of the supervisor of "Bob"
- Find ALL the supervisors (direct and indirect) of "Bob"

At the bottom left is a video thumbnail of a man speaking. The bottom right shows a navigation bar with icons and the text 'Database System Concepts - 6<sup>th</sup> Edition'.

Keyword AS is optional you can just write instructor, and then the name the new name that you want to give and that itself will work here. Is another Cartesian product example, here given a relation which lists a person and his or her supervisor, we want to find out all supervisors direct or indirect of that person.

So, I leave this as an exercise to you, to think over as to how we can actually compute this query?

(Refer Slide Time: 07:56)

The slide features a small sailboat icon in the top left corner. The title 'String Operations' is centered at the top in red. Below the title is a bulleted list of four items:

- SQL includes a string-matching operator for comparisons on character strings. The operator `like` uses patterns that are described using two special characters:
  - percent ( % ). The % character matches any substring
  - underscore ( \_ ). The \_ character matches any character
- Find the names of all instructors whose name includes the substring 'dar'

Below the list is a handwritten note in blue ink:

select name  
from instructor  
where name like "%dar%"

At the bottom left is a video thumbnail of a man speaking. The bottom right shows a navigation bar with icons and the text 'Database System Concepts - 6<sup>th</sup> Edition'.

SQL supports several string operations and of particular interest at 2 specific symbols, characters which allow us doing certain match, percentage is used to match any substring and underscore is used to match any particular character and we use a keyword LIKE, to find out different string patterns that can be matched. So, here we want to find the names of all instructors, whose name includes the substring “dar” and by writing this so saying the predicate is formed is named like this.

So, what is there is a percentage before there is a percentage after. So, anywhere “dar” will feature in the name, this predicate will turn out to be true otherwise if there is no “dar” in the name the predicate will turn out to be false and that particular record will not get selected. So, in this way we can using LIKE, we can actually do different kinds string operations as conditions in the weight loss.

(Refer Slide Time: 09:07)

The slide has a title 'String Operations' in red at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list of SQL string operations and a sample query. At the bottom, there is footer information and a navigation bar.

**String Operations**

- SQL includes a string-matching operator for comparisons on character strings. The operator `like` uses patterns that are described using two special characters:
  - percent ( % ). The % character matches any substring
  - underscore ( \_ ). The \_ character matches any character
- Find the names of all instructors whose name includes the substring "dar"

```
select name  
from instructor  
where name like '%dar%'
```
- Match the string "100%"

```
like '100 \%' escape '\'
```
- in that above we use backslash (\) as the escape character

BRUNNEN-NPTEL-NOOC MOOCs Instructor: Prof. P. P. Dini, IIT Kharagpur - 2018

Database System Concepts - 6<sup>th</sup> Edition 07.12 ©Güterschütz, Korth and Sudarshan

So now, naturally this brings in an issue of for what if my string itself has a percentage or an underscore character. So, the rule followed is you will need to escape that, with the escape character that you define. This is a style which you have seen in C programming as well.

(Refer Slide Time: 09:29)

The slide features a title 'String Operations (Cont.)' at the top right. On the left, there is a small video window showing a man speaking. The main content area contains a bulleted list of string matching patterns:

- Patterns are case sensitive
- Pattern matching examples:
  - 'Intro%' matches any string beginning with "Intro"
  - '%Comp%' matches any string containing "Comp" as a substring
  - '\_\_\_\_\_' matches any string of exactly three characters
  - '\_\_\_%' matches any string of at least three characters

At the bottom left, it says 'Database System Concepts - 8<sup>th</sup> Edition'. At the bottom center, it shows '07.13'. At the bottom right, it says '©Übersetzung, Korth und Sudarshan'.

Patterns are certainly case sensitive. So, it depends it will distinguish between upper case as well as lower case, and these are different examples of string matching that you can do where, you can match at this beginning of a string, end of a string, anywhere in the string, specific number of characters in a string and so on. SQL supports concatenation, conversion of lower to upper case and vice versa and different other common string operations, those are available as functions in SQL and can be used for convenience.

(Refer Slide Time: 10:09)

The slide features a title 'Ordering the Display of Tuples' at the top right. On the left, there is a small video window showing a man speaking. The main content area contains a bulleted list of ordering rules:

- List in alphabetic order the names of all instructors

```
select distinct name  
from instructor  
order by name
```
- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default.
  - Example: **order by name desc**
- Can sort on multiple attributes
  - Example: **order by dept\_name, name**

At the bottom left, it says 'Database System Concepts - 8<sup>th</sup> Edition'. At the bottom center, it shows '07.14'. At the bottom right, it says '©Übersetzung, Korth und Sudarshan'.

Now, let us address a different question. Let us say, we have computed a query and then often we would want that the result be ordered in according to certain order particularly the value of certain field if we want the result to be ordered, then SQL allows you to do that by another clause that you add to the query, which is called order by. So, what this will do, we have already seen this query this will find out the names of all the instructors and the names will occur in a distinct manner because, distinct is specified, but then the output will be in terms ordered by the name.

And the ordering can be descending or ascending, you can control that, by specifying whether you want descending or ascending by default the ordering is ascending. So, that makes the presentation of the result of and very easy and you can certainly sort on multiple fields as well, so it can be ordered based on combination of fields.

(Refer Slide Time: 11:15)

## Where Clause Predicates

- SQL includes a **between** comparison operator
- Example: Find the names of all instructors with salary between \$90,000 and \$100,000 (that is,  $\geq \$90,000$  and  $\leq \$100,000$ )
 

```
select name
from instructor
where salary between 90000 and 100000
```
- Tuple comparison
 

```
select name, course_id
from instructor, teaches
where (instructor.ID, dept_name) = (teaches.ID, 'Biology');
```

SQL where clause also allows between as a comparison parameter. So, between can specify 2 values. So that, whenever the field value will be between these 2 given values the condition will be predicated will be taken to be true otherwise is taken to be false. You can compare based on people as well. So, in this case you could have written, you could have checked for equality of instructor.ID with teachers.ID and department name with the literal biology, but you can compact it by writing a tuple notation as is shown here. So, these are common convenient ways of writing different where clause.

(Refer Slide Time: 12:02)

**Duplicates\***

- In relations with duplicates, SQL can define how many copies of tuples appear in the result!
- Multiset versions of some of the relational algebra operators – given multiset relations  $r_1$  and  $r_2$ :
  - $\sigma_\theta(r_1)$ : If there are  $c_1$  copies of tuple  $t_1$  in  $r_1$ , and  $t_1$  satisfies selections  $\sigma_\theta$ , then there are  $c_1$  copies of  $t_1$  in  $\sigma_\theta(r_1)$
  - $\Pi_A(r)$ : For each copy of tuple  $t_1$  in  $r_1$ , there is a copy of tuple  $\Pi_A(t_1)$  in  $\Pi_A(r_1)$  where  $\Pi_A(t_1)$  denotes the projection of the single tuple  $t_1$
  - $r_1 \times r_2$ : If there are  $c_1$  copies of tuple  $t_1$  in  $r_1$  and  $c_2$  copies of tuple  $t_2$  in  $r_2$ , there are  $c_1 \times c_2$  copies of the tuple  $t_1, t_2$  in  $r_1 \times r_2$

SWAMY: NPTEL-NOC MOOC Instructor Prof. P P Das, IIT Kharagpur Date: Jan-August, 2018  
Database System Concepts - 6<sup>th</sup> Edition  
07.16 ©Übersetzung, Korth und Sudarshan

Now, we have specified that SQL does carry duplicates. So, unlike relational algebra which set theoretically specify that, their duplicates should not be there if in SQL there could be duplicate entries in the same relation. So, there is a; this is called when duplicates are allowed in set theory, then such sets where duplicates are allowed and known as multi sets. So, there are multi set versions of the SQL queries or so to say the relational algebra operations.

So, you have a selection, which can be multi set selection, which means that, if there are certain  $c_1$  number of copies of a tuple in the relation, which satisfy the condition theta then all of them will feature in the result. And all those copies can be seen simultaneously, because it is a multi-set condition, similar definitions are hold for projection, as well as for Cartesian product. So, I will leave it to you to go through the details and convince yourself that, these multi set relations really extend the traditional single set distinct definition of the relational algebra.

(Refer Slide Time: 13:27)

The slide has a decorative background of a sailboat on water. The title 'Duplicates (Cont.)\*' is at the top. A list of bullet points discusses multiset relations  $r_1$  and  $r_2$ , their Cartesian product, and SQL semantics. Handwritten annotations show sets  $r_1 = \{(1, a), (2, a)\}$  and  $r_2 = \{(2, (3), (3))\}$ , and a set of tuples  $\{(a, 2), (a, 2), (a, 3), (a, 3), (a, 3), (a, 3)\}$  with a circled '2' over it. Below is an SQL query and its multiset equivalent.

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Datta, IIT Kharagpur - Jan-Aug - 2018

Duplicates (Cont.)\*

- Example: Suppose multiset relations  $r_1(A, B)$  and  $r_2(C)$  are as follows:  
 $r_1 = \{(1, a), (2, a)\}$        $r_2 = \{(2, (3), (3))\}$
- Then  $\Pi_B(r_1)$  would be  $\{(a), (a)\}$ , while  $\Pi_B(r_1) \times r_2$  would be  
 $\{(a, 2), (a, 2), (a, 3), (a, 3), (a, 3), (a, 3)\}$
- SQL duplicate semantics:  

```
select A1, A2, ..., An
  from r1, r2, ...
 where P
```

is equivalent to the multiset version of the expression:

$$\prod_{A_1, A_2, \dots, A_n} (\sigma_P (r_1 \times r_2 \times \dots \times r_m))$$

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Datta, IIT Kharagpur - Jan-Aug - 2018

Database System Concepts - 6<sup>th</sup> Edition

07.17

©Überschütz, Korth and Sudarshan

So, here is an example where, there are 2 multi set relations as you can see particularly this one, which has identical duplicate entries. So, using that you can define a projection of  $r_1$  on  $B$ , which will certainly give you it is you are doing projection on  $B$ . So, it will give you  $A$  only. So, you will have this result itself will be a multi set because you will get 2 as. So, this result will be like  $a, a$ , and then you have  $r_2$  with which you are doing the Cartesian product. So, you will have all possible combinations all these 6 are the result in the SQL whereas, in set theory typically the result should have been only these 2 tuples.

(Refer Slide Time: 14:38)

The slide has a decorative background of a sailboat on water. The title 'SET OPERATIONS' is at the top. To the right is a list of topics: Additional Basic Operations, Set Operations, Null Values, and Aggregate Functions. Below is a video player showing a professor speaking.

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Datta, IIT Kharagpur - Jan-Aug - 2018

SET OPERATIONS

- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Datta, IIT Kharagpur - Jan-Aug - 2018

Database System Concepts - 6<sup>th</sup> Edition

07.18

©Überschütz, Korth and Sudarshan

(Refer Slide Time: 14:43)

The slide has a header 'Set Operations' in red. On the left, there is a small sailboat icon. The main content consists of three bullet points with corresponding SQL queries:

- Find courses that ran in Fall 2009 or in Spring 2010  
(`select course_id from section where sem = 'Fall' and year = 2009`)  
`union`  
(`select course_id from section where sem = 'Spring' and year = 2010`)
- Find courses that ran in Fall 2009 and in Spring 2010  
(`select course_id from section where sem = 'Fall' and year = 2009`)  
`intersect`  
(`select course_id from section where sem = 'Spring' and year = 2010`)
- Find courses that ran in Fall 2009 but not in Spring 2010  
(`select course_id from section where sem = 'Fall' and year = 2009`)  
`except`  
(`select course_id from section where sem = 'Spring' and year = 2010`)

At the bottom left, it says 'SWAYAM: NPTEL-NOC MOOCs Initiator: Prof. P. P. Desai, IIT Kanpur Date: June-August, 2018'. At the bottom center, it says 'Database System Concepts - 6<sup>th</sup> Edition' and '07.19'. At the bottom right, there is a navigation bar with icons.

Now, we take a quick look into the common set operations. So, it is possible to do union, intersection, difference kind of operations very easily with SQL queries. So, suppose we want to find all courses, that ran in fall 2009 or in spring 2010. So, certainly the first part of the query is simple. This will give you all courses that run. So, you are taking out the course\_id from section is where, the course running information is provided and you part putting 2 conditions, which say that they actually this courses ran in fall 2009.

So, this is the first query the second query says the courses, that run in spring 2010 and you are you have an or condition in the statement of what you are looking for. So, you do a union, union is another keyword. So, this will simply give you a relation of the course\_id attribute as the only attribute, which has records from the first as well as the second query. Similarly, you can find out the courses that run, both in fall 2009 and spring 2010 by using intersect, which basically give you the intersection of the result of the first and the second query.

You could also do difference set difference by doing find courses, that ran in fall 2009, but not in 2010. So, what will that mean, that will mean that the result of the first query from the results of the second query be subtracted be done a difference from. So, those that, had run in the fall 2009 and then was again run in spring 2010 will get removed. So, that is done through the accept keyword.

So, in this way you can very easily do set operations, whenever that is easy to conceive; obviously, you can write these queries in several other different forms, but this is just to show you how set theoretic operations can be easily written.

(Refer Slide Time: 17:00)

The slide has a title 'Set Operations (Cont.)' at the top right. On the left, there is a small image of a sailboat on water. Below the title, there are three bullet points with corresponding SQL code:

- Find the salaries of all instructors that are less than the largest salary  

```
select distinct T.salary
from instructor as T, instructor as S
where T.salary < S.salary
```
- Find all the salaries of all instructors  

```
select distinct salary
from instructor
```
- Find the largest salary of all instructors  

```
(select "second query")
except
(select "first query")
```

At the bottom left, there is a video thumbnail of a man speaking, with the text 'SHAYAM: NPTEL-NOC MOOCs Instructor - Prof. P. P. Dayal, IIT Madras'. At the bottom center, it says '07.20'. At the bottom right, there is a navigation bar with icons for back, forward, search, and other presentation controls, and the text '©Übersetzung, Korth und Sudarshan'.

You can do set operations like this. in terms of fine salaries of all instructions that are less than a largest salary. So, again we are using renaming to think of the same relation as 2, and then as if from the relation T you are trying to look at relation S and finding out what are the salaries, which are smaller than that and certainly whatever comes out is the one which is not the largest because, certainly the largest will not satisfy this particular condition, because it will get compared with itself. You can find salaries of all instructors, and then you can find the largest salary.

So, this is all salaries which are less than largest, this is all salaries including the largest. So, what happens, if you subtract that is from this if you subtract this from all salaries, if you remove the salaries, that are not largest naturally what you get is a largest salary. So, this is a interesting way to find the largest salary we will see later on that there could be several other ways particularly the use of aggregate function, which make these computations easier to perform, but these are the typical ways to use set theoretic operations.

(Refer Slide Time: 18:29)

The slide is titled "Set Operations (Cont.)" in red. It features a small sailboat icon in the top left corner. In the bottom left, there is a video feed of a man speaking. The bottom right contains a navigation bar with icons for back, forward, search, and other presentation controls. The main content area lists points about set operations:

- Set operations **union**, **intersect**, and **except**
  - Each of the above operations automatically eliminates duplicates
- To retain all duplicates use the corresponding multiset versions **union all**, **intersect all** and **except all**.
- Suppose a tuple occurs  $m$  times in  $r$  and  $n$  times in  $s$ , then, it occurs:
  - $m + n$  times in  $r \text{ union all } s$
  - $\min(m,n)$  times in  $r \text{ intersect all } s$
  - $\max(0, m - n)$  times in  $r \text{ except all } s$

The set operations, so we have seen 3 of them union, intersect and except they automatically these operations are set theoretic. So, each of them automatically, eliminate the duplicate, unlike what SQL by default, SQL by default does what, allows duplicates, but set operations will eliminate duplicates, because they are set operations. So, if you want the SQL type of behaviour, if you want the duplicates to be preserved, then you can have a multi set version of these set operations, which are known as union all, intersect all, except all like that.

And naturally if you do these operations then, here is the simple formula of the number of tuples, that will get computed in different cases. you can study and convince yourself that these are the correct numbers. Let us, go to the treatment of we talked about null values, that we said that it is possible that certain records in a relation may have one or more attributes where, the value is not known and to represent, that the value is not known we are putting a placeholder called null.

(Refer Slide Time: 19:52)

The slide has a header 'Null Values' in red. On the left, there is a small video window showing a man speaking. The main content area contains a bulleted list of points about null values, followed by a SQL query. A blue arrow points from the word 'null' in the list to the word 'null' in the SQL code. The SQL code is:

```
select name  
from instructor  
where salary is null
```

At the bottom left, it says 'Database System Concepts - 8<sup>th</sup> Edition'. At the bottom right, it shows '07.23' and '©Silberschatz, Korth and Sudarshan'.

So, let us see what is the consequence of that null value in terms of doing these query operations. So, the null signifies an unknown value. So, if I do 5 plus null then naturally the result is null. So, what you are saying that I am adding an unknown quantity to 5. So, then what would you say is the result is unknown. So, that is the basic semantics of adding null to a number. So, it is possible to check, if particularly a field is a null for a record, and that is done by a predicate “is null”.

So, in this particular query we are trying to find all instructors, whose salary is null that is not known. So, this is a predicate. So, for a particular record for which salaries null, this will become true and that will get included in the result, but for all records for which, there is some value for the salary. So, salary is known it is not null those will not get included in the result.

(Refer Slide Time: 20:58)

## Null Values and Three Valued Logic

- Three values – *true, false, unknown*
- Any comparison with *null* returns *unknown*
  - Example:  $5 < \text{null}$  or  $\text{null} > \text{null}$  or  $\text{null} = \text{null}$
- Three-valued logic using the value *unknown*:
  - OR:  $(\text{unknown or true}) = \text{true}$ ,  
 $(\text{unknown or false}) = \text{unknown}$   
 $(\text{unknown or unknown}) = \text{unknown}$
  - AND:  $(\text{true and unknown}) = \text{unknown}$ ,  
 $(\text{false and unknown}) = \text{false}$ ,  
 $(\text{unknown and unknown}) = \text{unknown}$
  - NOT:  $(\text{not unknown}) = \text{unknown}$
  - "P is unknown" evaluates to true if predicate P evaluates to unknown

So, the basic semantics of null is then, combined with the truth values because, we know our basic predicate logic is 2 values true and false. But now, you have a third value unknown that is, you may not know the value of a predicate. So, how does it play around with the true and false values, you can reason through that quite easily if you are comparing with a null in whatever way, then naturally the result is unknown. So, it returns a null, if you are doing any connectives for example, if you are doing or of null or true, then the result should be true because, in or we say that if any of the components is true then the result is true.

So, here you do not need to know what is that unknown well you can say it is true, but if you do or with false or of unknown with false the second row or of unknown with false if you do this, then naturally this is unknown because, since this is false the result would be true only if unknown value is true and the result would be false if the unknown value is false, you do not know what that unknown value is. So, you have to say that you have result is unknown. So, using that same logic you could see verify, I would ask you to verify offline at home you please verify, that all these combinations of true false with unknown are valid. So, if P is unknown as a predicate will evaluate to true if P is not known..

(Refer Slide Time: 23:05)



## Aggregate Functions

- These functions operate on the multiset of values of a column of a relation, and return a value
  - avg: average value
  - min: minimum value
  - max: maximum value
  - sum: sum of values
  - count: number of values

Database System Concepts - 6<sup>th</sup> Edition      07.26      ©Büterschutz, Korth and Sudarshan

Now, we come to the aggregate functions, there are several aggregate functions they can be used for convenience and these are the common months, that operate on the multi set values naturally aggregate functions operate on a particular column they try to aggregate in a particular column and return a single value for example, average would be meaning, that you are trying to find average of the values of a particular column.

(Refer Slide Time: 23:28)



## Aggregate Functions (Cont.)

- Find the average salary of instructors in the Computer Science department

```
select avg (salary)  
from instructor  
where dept_name= 'Comp. Sci.';
```
- Find the total number of instructors who teach a course in the Spring 2010 semester

```
select count (distinct ID)  
from teaches  
where semester = 'Spring' and year = 2010;
```
- Find the number of tuples in the course relation

```
select count (*)  
from course;
```



Database System Concepts - 6<sup>th</sup> Edition      07.27      ©Büterschutz, Korth and Sudarshan

So, here is an example. So, we are trying to find the average salary of instructors in computer science department. So, naturally what you output is average salary. So, mind you this will output relation will have one attribute, which is average salary and since, average salary is a single quantity it will only have one, and here I have made use of this

aggregate function average. So, it says you do average on the attribute salary and where do you get that attribute, from you get it from the table instructor and then we are saying that we are not interested to find average of salary of all instructors. We are interested to find the average salary of those instructors who work for computer science. So, you put this back clause. So, this will ensure, that you find the average salary of instructors in computer science department.

So, in similar way you can use other aggregate functions LIKE, if you want to know the total number of instructors who teach a course in this semester. So, you first put the where clause, naturally you where will you find this information you will find this information in teachers, teachers is the relation which tells you which instructor is teaching what course. So, that comes in from, then you have to specify that teaching a course in spring 2010 semester. So, the where clause specifies, that the semester is spring and the year is 2010. So, this will give you all records, which show that some instructor is teaching the course in spring 2010 semester.

Now, naturally there could be multiple the same instructor could happen multiple times, because an instructor may be teaching more than one course. So, you make the instructor.ID, instructor.ID that you have here, you make that distinct. So, that you get only those instructors, every instructor who is teaching one course or more than one course will feature only once in this total list, and then you simply count it. use aggregate function count on that.

So, that will tell you how many instructors are teaching some course in spring 2010 mind you, this is critical to use this key word distinct, because unless you use that, then all that you will eventually find out is not the number of instructors who are teaching the course you will find out the number of courses, that are being offered in spring 2010 because, there could be the same instructor teaching more than one course. If you just want to count the number of tuples you can do count on star. because, what is star is all the attributes. So, from if we want to find out the number of course, you suggest to count \* on course.

(Refer Slide Time: 27:03)

**Aggregate Functions – Group By**

- Find the average salary of instructors in each department

```
select dept_name, avg(salary) as avg_salary
from instructor
group by dept_name;
```

| ID    | name       | dept_name  | salary |
|-------|------------|------------|--------|
| 76766 | Crick      | Biology    | 72000  |
| 45565 | Katz       | Comp. Sci. | 75000  |
| 10101 | Srinivasan | Comp. Sci. | 65000  |
| 83821 | Brandt     | Comp. Sci. | 92000  |
| 98345 | Kim        | Elec. Eng. | 80000  |
| 12121 | Wu         | Finance    | 90000  |
| 76543 | Singh      | Finance    | 80000  |
| 32343 | El Said    | History    | 60000  |
| 58583 | Califieri  | History    | 62000  |
| 15151 | Mozart     | Music      | 40000  |
| 33456 | Gold       | Physics    | 87000  |
| 22222 | Einstein   | Physics    | 95000  |

| dept_name  | avg_salary |
|------------|------------|
| Biology    | 72000      |
| Comp. Sci. | 77333      |
| Elec. Eng. | 80000      |
| Finance    | 85000      |
| History    | 61000      |
| Music      | 40000      |
| Physics    | 91000      |

So, this is showing you the computation of average salary of instructors in each department.

So now, what do you want to do is earlier you try to find out the average salary in one department. Now, you want that for all the departments for each department I want. So, my result now, is not a single row, it is not a single value it is a pair where, I show the department and the average salary in that department. So, this is what I want visible and this is what I have.

So, naturally the information comes from instructor, that is from what I want is a department name and the average salary and I want to give it a nice name avg\_salary. So, I have done a rename. So, I get avg\_salary here, but then what I want is I do not want an average d1 over this whole set of fields. I want separate average to be done here, to be done here, to be done on this to be on this. So, these are these are basically groupings by the department as you can see, that this particular relation has been sorted according to the department name.

So, when I want to do apply an aggregate function on certain subgroups of records, I use this particular clause GROUP BY, and use a name of a field. So, what it does is if the values in the GROUP BY field in this case department name are identical those records are put together and over those records and average is computed. So, the average, that is computed over these records are put in here, average that is computed in terms of these

records are put in here, there is only one record, so average that is computed in terms of that is put in here. So, GROUP BY is a very useful feature along with the aggregation functions and it allows you to club information based on certain attribute and then compute the aggregation on some other field.

(Refer Slide Time: 29:27)

The slide has a header 'Aggregation (Cont.)' with a small sailboat icon. It contains a bullet point and a code snippet:

- Attributes in **select** clause outside of aggregate functions must appear in **group by** list

```
/* erroneous query */  
select dept_name, ID, avg (salary)  
from instructor  
group by dept_name;
```

At the bottom, there is footer text: 'SWAYAM: NPTEL-NOC MOOC Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr-2018', 'Database System Concepts - 6<sup>th</sup> Edition', '07.29', and '©Übersetzung, Korth und Sudarshan'.

Mind you, you will have to do GROUP BY and create the result id result table you will have to make sure that, all your result table attribute are used in the GROUP BY, which is not an aggregate function. So, here ID is not used. So, this is not a query, that SQL would support.

(Refer Slide Time: 29:53)

The slide has a header 'Aggregate Functions – Having Clause' with a sailboat icon. It contains a bullet point: 'Find the names and average salaries of all departments whose average salary is greater than 42000'. Below it is a SQL query:

```
select dept_name, avg (salary)
from instructor
group by dept_name
having avg (salary) > 42000;
```

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

SWAYAM: NPTEL-NOOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur.. Date: Aug-18  
Database System Concepts - 6<sup>th</sup> Edition  
07:30  
©Übersetzung, Korth und Sudarshan

You can further refine your result by saying that, find names and average salary of all departments; this much you have already done.

Now, you are qualifying that, whose average salary is greater than 42000. So, of all that we have for example, if we look here for example, in this music department average salary is less than 42000. So, you do not want that in the result you want only those where, the average salary is greater than 42000 and the way to do that is to add another clause called HAVING they say that, the average salary is greater than 42000. So, you are adding another predicate for actually qualifying the aggregated value. Now, the HAVING clause actually applies after along with the GROUP BY because, naturally the HAVING relates to the grouping.

So, once the grouping has happened groups have been formed then the HAVING clause will be evaluated on that, in contrast where clause also has a predicate, but the where clause is applied before forming the groups. So, this point this note has to be understood carefully because, if you have a where clause to choose the records it will first apply, then out of those records chosen the grouping will happen and once the grouping has happened, then the aggregate function will evaluate and the HAVING clause will get evaluated, the predicate of HAVING clause will get evaluated.

(Refer Slide Time: 31:45)

The slide has a header 'Null Values and Aggregates' in red. On the left, there is a small sailboat icon and a vertical watermark that reads 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur... Jan-Apr., 2018'. The main content includes a bulleted list of points and a corresponding SQL query:

- Total all salaries

```
select sum (salary )  
from instructor
```

- Above statement ignores null amounts
- Result is *null* if there is no non-null amount

- All aggregate operations except **count(\*)** ignore tuples with null values on the aggregated attributes
- What if collection has only null values?
  - count returns 0
  - all other aggregates return *null*

At the bottom left, there is a video thumbnail showing a man in a suit. The bottom right corner contains a navigation bar with icons and the text '©Übersetzung, Korth und Sudarshan'.

Certainly, if there are null values in terms of aggregates, then there is a question of what will happen. So, the general strategy is that, whenever you perform aggregation then the null values are all ignored. So, if on that field there is no value which is not null, that is if all values are null then the result is null otherwise the result is computed by ignoring the null values.

So, these are what do you have of course, if you count then, if the collection has only null values the count will return you 0, but all other aggregates will return you simply null.

(Refer Slide Time: 32:29)

**Module Summary**

- Completed the understanding of basic query structure and set operations
- Familiarized with null values and aggregation

SWAYAM - NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur, Date: June-August, 2018  
Database System Concepts - 6<sup>th</sup> Edition

07:32

©Silberschatz, Korth and Sudarshan

So, to summarize we have started the basic understanding of the basics query structure in the last module. Now, we have completed that with some more additional operations, we have understood the set theoretic operations and very importantly we have familiarized with the treatment of null values and aggregation functions particularly the GROUP BY and HAVING clauses and how do null values and aggregation interact in terms of an SQL query.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 08**  
**Introduction to SQL/3**

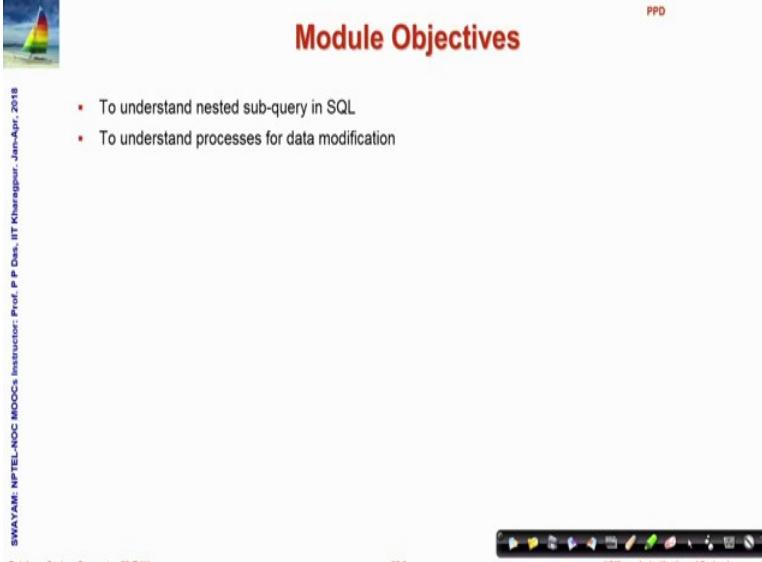
Welcome to module 8 of Database Management Systems. We have been discussing basic SQL queries and this is the third and closing part of that introductory discussion that we started in the 6th module.

(Refer Slide Time: 00:34)

The slide is titled "Module Recap" in red at the top right. It features a small sailboat icon in the top left corner. On the far left, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018". At the bottom left, it says "Database System Concepts - 8<sup>th</sup> Edition". In the center, there is a bulleted list of topics: "Additional Basic Operations", "Set Operations", "Null Values", and "Aggregate Functions". The bottom right contains a navigation bar with icons for back, forward, search, and other presentation controls, along with the text "©Silberschatz, Korth and Sudarshan". The slide is numbered "08.2" at the bottom center.

So, just to quickly recap, these are the things that we did in the last module completing the understanding of basic operations the null values and aggregate functions.

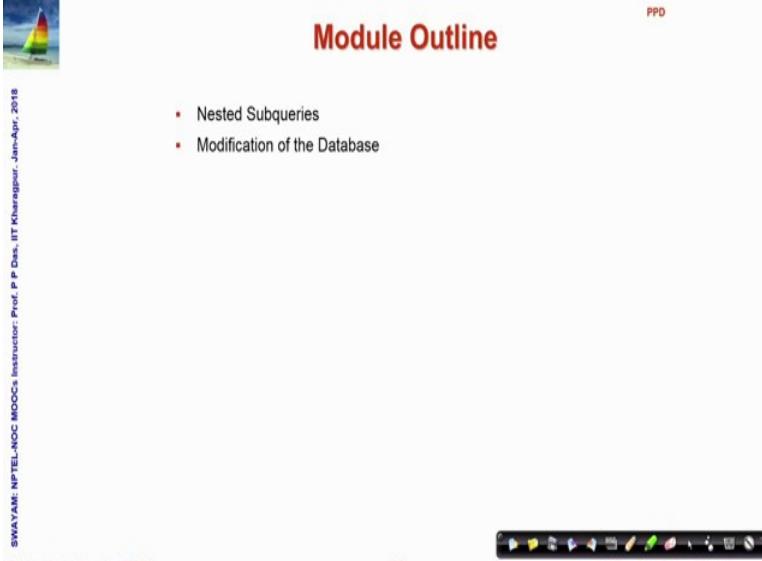
(Refer Slide Time: 00:44)



The slide is titled "Module Objectives" in red. It features a small sailboat icon in the top left corner. On the right side, there is a vertical footer bar with the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jam-Agr., 2018" and "PPD" at the top. The main content area contains two bullet points: "To understand nested sub-query in SQL" and "To understand processes for data modification". At the bottom, it says "Database System Concepts - 8<sup>th</sup> Edition" and "08.3". A navigation bar with various icons is at the very bottom.

In the current module, we want to understand a feature which is very very important in SQL query forming it is called the nested query or more formally nested sub query. In SQL and we would like to understand the process of data modification.

(Refer Slide Time: 01:05)



The slide is titled "Module Outline" in red. It features a small sailboat icon in the top left corner. On the right side, there is a vertical footer bar with the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jam-Agr., 2018" and "PPD" at the top. The main content area contains two bullet points: "Nested Subqueries" and "Modification of the Database". At the bottom, it says "Database System Concepts - 8<sup>th</sup> Edition" and "08.4". A navigation bar with various icons is at the very bottom.

And those are the two things that are outlined here.

So, let us start with nested sub queries.

(Refer Slide Time: 01:12)

The slide has a header 'Nested Subqueries' with a sailboat icon. On the left, there's a vertical sidebar with text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018'. The main content area contains a bulleted list, a SQL query example, and a detailed explanation of subquery components.

- SQL provides a mechanism for the nesting of subqueries
- A **subquery** is a **select-from-where** expression that is nested within another query
- The nesting can be done in the following SQL query

```
select A1, A2, ..., An
  from r1, r2, ..., rm
  where P
```

as follows:

- $A_i$  can be replaced by a subquery that generates a single value
- $r_i$  can be replaced by any valid subquery
- $P$  can be replaced with an expression of the form:  
 $B <\text{operation}> (\text{subquery})$   
where  $B$  is an attribute and  $<\text{operation}>$  to be defined later

At the bottom, there's a video player showing a man speaking, the text 'Database System Concepts - 8th Edition', the number '08.6', and the copyright '©Silberschatz, Korth and Sudarshan'.

A sub query is necessarily a SELECT FROM WHERE expression that is nested within another query, this is. So, these are the key things where expression. So, it is nothing new over; what we have already learned? But it is a part of another query it is nested within another query and that is the reason it is called a sub query. So, it by itself is not the result. this will be used, this is a SELECT FROM WHERE expression that will be used in a nested form in another some other queries to actually generate the result.

So, that is a nested sub query. So, the nesting can be done in one or more of the three clauses that a SELECT FROM WHERE SQL query has. An attribute can be replaced a relation can be any valid sub query or a sub query could form the part of a predicate in the where clause all of these are possible. So, we will discuss them one by one. So, first we will start by discussing how sub queries work in the where clause.

(Refer Slide Time: 02:44)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018

## Subqueries in the Where Clause

- A common use of subqueries is to perform tests:
  - For set membership
  - For set comparisons
  - For set cardinality

Database System Concepts - 8<sup>th</sup> Edition 08.8 ©Silberschatz, Korth and Sudarshan

The common use for you having sub queries in the where clause is to perform different kind of tests for membership comparison cardinality and so, on.

(Refer Slide Time: 02:58)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018

## Set Membership

- Find courses offered in Fall 2009 and in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
course_id in (select course_id
from section
where semester = 'Spring' and year= 2010);
```

Set Member ship

Database System Concepts - 8<sup>th</sup> Edition 08.8 ©Silberschatz, Korth and Sudarshan

So, let us look at this; you have already seen this query before find courses offered in fall 2009, and in fall in spring 2010. Earlier we have shown different ways of coding this, now we are showing yet another way to be able to code this in SQL.

So, first from the beginning certainly we need the courses. So, the SELECT clause is trivial it has to be the distinct course id is certainly the information will come from

section which has information about offering of courses as we have also seen already. So, those two are no brainer. Now let us see what how do I find courses that are offered in fall 2009 and in spring 2010. So, again the first part, the courses offered in fall 2009 is coded in this part; in part of that where clause predicate when he says semester has to be fall and here is 2009. So, this part is also done.

Now, we need to ensure that whatever I mean, if I assume that after this this part were not there, then this will only give me courses which are offered in fall 2009, but we want the courses that are in fall 2009 and in spring 2010. So, we do something interesting what I do is, we write a separate query here which is courses that are offered in spring 2010. SELECT course\_Id just in the same way, SELECT course\_Id, section, semester year. So, this particular query will give me the courses offered in spring 2009. So, what do we have in one part? I have so, if you look at this part this courses that happened in fall 2009; if we look at this part courses that happened in spring 2010 good.

Now, what I want, I need that the; it I am interested in the courses that happened in both. So, for a course that exists here I want to specify that that course Id must be present here. So, what I am checking for? I am checking for a set membership; this is a set right. So, I am trying to check, whether that course Id which is being selected in the first part exists in, in is another keyword. In this particular, this particular relation that is specified by the second part of this query which is courses offered in spring 2010.

If it is, if the course Id is present, then that course must have been offered in both the semesters if it is not present, then it is offered only in fall 2009, and not in spring 2010 and certainly the courses that were not offered in fall 2009, and only offered in spring 2010 will feature here, but they do not exist here. So, they will never come up for tests. So, as a result what I get finally, is the effect of computing courses that are offered in fall 2009 and in spring 2010. this part of the query which I used as a part of the where clause is my nested sub query.

And in this case as we have seen it is used for set membership and this is a basic idea of using nested sub queries; that is a nested sub query will always give you a relation. So, you try to put that relation in the right context of the where clause from clause or SELECT clause, and then make use of it in building up your logical.

(Refer Slide Time: 07:14)

The slide features a small sailboat icon in the top left corner. The title 'Set Membership' is centered at the top in a red font. On the left side, there is vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. The main content consists of two bullet points with corresponding SQL queries:

- Find courses offered in Fall 2009 and in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
course_id in (select course_id
from section
where semester = 'Spring' and year= 2010);
```

- Find courses offered in Fall 2009 but not in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
course_id not in (select course_id
from section
where semester = 'Spring' and year= 2010);
```

At the bottom, there is footer text: 'Database System Concepts - 8<sup>th</sup> Edition', '08.9', and '©Silberschatz, Korth and Sudarshan'.

So, let us run through some examples this is, what you are saying is, earlier one was the courses offered in both here we are trying to do kind of the difference saying the courses offered in fall 2009, but not in spring 2010 certainly we easily get that by changing the membership to negation of the membership earlier it was in now you do not in you will simply get that it is up to you to take some examples and convince yourself that this kind of a nesting will work.

(Refer Slide Time: 07:43)

The slide features a small sailboat icon in the top left corner. The title 'Set Membership (Cont.)' is centered at the top in a red font. On the left side, there is vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. The main content consists of two bullet points with corresponding SQL queries:

- Find the total number of (distinct) students who have taken course sections taught by the instructor with ID 10101

```
select count (distinct ID)
from takes
where (course_id, sec_id, semester, year) in
(select course_id, sec_id, semester, year
from teaches
where teaches.ID= 10101);
```

- Note: Above query can be written in a much simpler manner.  
The formulation above is simply to illustrate SQL features.

At the bottom, there is footer text: 'Database System Concepts - 8<sup>th</sup> Edition', '08.10', and '©Silberschatz, Korth and Sudarshan'.

We find the set of the find the total number of distinct students, we have taken the course section taught by the instructor Id, some Id is given. So, again we form a nested query here is a nested query, which tells me the courses taught by this particular teacher 10101, and then we check set membership in terms of this course Id, section Id that is fields of the takes relation to see that, whether that particular tuple can exist in the course offered by the specific teacher; if it does then take out that I d which is which will turn out to be the student Id. In this case because that is the text relation has the student I d take out the student I d and count it as distinct. So, this can simply give you the answer to this squared; obviously, we agree that this can be written in a simpler form also, but we are including it here just for the sake of illustrating the feature.

(Refer Slide Time: 09:03)

## Set Comparison – “some” Clause

- Find names of instructors with salary greater than that of some (at least one) instructor in the Biology department
 

```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept name = 'Biology';
```
- Same query using > **some** clause
 

```
select name
from instructor
where salary > some (select salary
from instructor
where dept name = 'Biology');
```

**SWAYAM: NPTEL-NOC MOOCs Instruction:** Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018  
**Database System Concepts - 5<sup>th</sup> Edition**

There is another clause called the some clause look at this fine names of instructor; salary is greater than that of some which means at least one instructor in biology department and we have already seen this coding before.

Now, we can do this in terms of the nested query by using, again this is certainly the salary of instructors in biology department and we are doing greater than sum; that means, that the salary here being checked must find at least one record here. So that it is greater than that salary value. So, it is greater than some is a nice way to find existential records using the nested sub query.

(Refer Slide Time: 10:02)



## Definition of “some” Clause\*

▪  $F <\text{comp}> \text{some } r \Leftrightarrow \exists t \in r \text{ such that } (F <\text{comp}> t)$   
Where  $<\text{comp}>$  can be:  $<$ ,  $\leq$ ,  $>$ ,  $=$ ,  $\neq$

|                                                                                                                        |                                        |
|------------------------------------------------------------------------------------------------------------------------|----------------------------------------|
| $(5 < \text{some} \begin{array}{ c } \hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{true}$            | (read: 5 < some tuple in the relation) |
| $(5 < \text{some} \begin{array}{ c } \hline 0 \\ \hline 5 \\ \hline 0 \\ \hline \end{array}) = \text{false}$           |                                        |
| $(5 = \text{some} \begin{array}{ c } \hline 0 \\ \hline 5 \\ \hline \end{array}) = \text{true}$                        |                                        |
| $(5 \neq \text{some} \begin{array}{ c } \hline 0 \\ \hline 5 \\ \hline \end{array}) = \text{true}$ (since $0 \neq 5$ ) |                                        |

$(= \text{some}) = \text{in}$   
However,  $(\neq \text{some}) \neq \text{not in}$

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition 08.12 ©Silberschatz, Korth and Sudarshan

The logic of some clause, I have detailed out here. So, we will not go through each one of them in this discussion.

(Refer Slide Time: 10:09)



## Set Comparison – “all” Clause

▪ Find the names of all instructors whose salary is greater than the salary of all instructors in the Biology department

```
select name
from instructor
where salary > all (select salary
                     from instructor
                     where dept name = 'Biology');
```

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition 08.13 ©Silberschatz, Korth and Sudarshan

I leave it unto you to study and convince yourself, that you understand the semantics of some. So, similarly we have an all clause which say that if we want to say, they find the names of all instructors; whose salary is greater than the salary of all instructors in the biology department in case of sum we can write or we will write all and it will check every salary will check with the whole set of salaries in this sub query. And only if that is

greater than that particular record, that particular name will be included in the result. Otherwise it will be excluded from the result.

(Refer Slide Time: 10:52)



## Definition of "all" Clause\*

- $F <\text{comp}> \text{all } r \Leftrightarrow \forall t \in r (F <\text{comp}> t)$

|   |
|---|
| 0 |
| 5 |
| 6 |

$(5 <\text{all } \boxed{5} ) = \text{false}$

|    |
|----|
| 6  |
| 10 |

$(5 <\text{all } \boxed{10} ) = \text{true}$

|   |
|---|
| 4 |
| 5 |

$(5 = \text{all } \boxed{5} ) = \text{false}$

|   |
|---|
| 4 |
| 6 |

$(5 \neq \text{all } \boxed{6} ) = \text{true (since } 5 \neq 4 \text{ and } 5 \neq 6\text{)}$

$(\neq \text{all}) = \text{not in}$   
However,  $(= \text{all}) \neq \text{in}$

↳

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition      08.14      ©Silberschatz, Korth and Sudarshan

Similar to some there is a basic semantics of all which is also worked out here and I leave that to your study at home.

(Refer Slide Time: 11:03)



## Use of "exists" Clause

- Yet another way of specifying the query "Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester"

```

select course_id
from section as S
where semester = 'Fall' and year = 2009 and
exists (select *
from section as T
where semester = 'Spring' and year= 2010
and S.course_id = T.course_id);

```

- Correlation name – variable S in the outer query
- Correlated subquery – the inner query



↳

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition      08.16      ©Silberschatz, Korth and Sudarshan

You can test for empty relations by using the exists construct. So, if you say exists r, then that is a predicate which mean that r is not empty; if r is empty then exist is false and not exist is the negation of exist. So, it can be used to specify the query like find all courses

taught both in fall 2009 and spring 2010. So, all that you have to do earlier you did it by set membership.

So, now you are trying to do this by this exists. So, you are saying that this is again the same query which gives you the courses that are in spring 2010 and also in this fall 2009 and you check whether this relation, whether this particular nested query is an empty one or not. if it is an empty one, then exist will fill and the whole where clause will fail, if it was not an empty one then you have found such an entry it was offered and therefore, it will get included.

So, these are just different ways of expressing similar things, but what you should note is the nested sub query is a very convenient way to frame up the logic in multiple different ways that you would like to do. So, these are the different names given the correlation name and the correlated sub query. incidentally; the nested query is often referred to as the inner query and the query in which the nesting has happened, is known as the outer query.

(Refer Slide Time: 13:04)

The slide features a title 'Use of “not exists” Clause' in red font at the top right. To the left of the title is a small icon of a sailboat on water. Below the title, there is a list of bullet points and a SQL query. At the bottom of the slide, there is footer information and a navigation bar.

**Use of “not exists” Clause**

▪ Find all students who have taken all courses offered in the Biology department.

```
select distinct S.ID, S.name
from student as S
where not exists ( (select course_id
                   from course
                   where dept_name = 'Biology')
                  except
                  (select T.course_id
                   from takes as T
                   where S.ID = T.ID));
```

▪ First nested query lists all courses offered in Biology  
▪ Second nested query lists all courses a particular student took

▪ Note:  $X - Y = \emptyset \Leftrightarrow X \subseteq Y$

▪ Note: Cannot write this query using = all and its variants

SWAYAM: NPTEL-NOCOCS Instructor: Prof. P. P. Desai, IIT Kharagpur - Jam-Astr. 2018

Database System Concepts - 8<sup>th</sup> Edition 08.17 ©Silberschatz, Korth and Sudarshan

Here is another example which illustrate the use of not exist; so which I leave it for your own study.

(Refer Slide Time: 13:15)

The slide has a header 'Test for Absence of Duplicate Tuples' with a sailboat icon. It contains a bulleted list of three items, followed by a SQL query with a nested subquery highlighted in a blue box.

- The **unique** construct tests whether a subquery has any duplicate tuples in its result
- The **unique** construct evaluates to "true" if a given subquery contains no duplicates
- Find all courses that were offered at most once in 2009

```
select T.course_id
from course as T
where unique
  select R.course_id
  from section as R
  where T.course_id= R.course_id
    and R.year = 2009);
```

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition 08.18 ©Silberschatz, Korth and Sudarshan

We can check for uniqueness that is; test for absence of duplicate tuples by using the unique keyword. So, we can see here that here is a nested query and using unique to find out all courses that were offered at most once in 2009. So, if a course was offered more than once, then naturally multiple records will feature and the result the unique will fail. unique will be true only if; there is only one entry which shows that it is offered at most once in that semester.

Now, I move on. So, we have been discussing about sub queries in the where clause; now we move on to sub queries in the from clause.

(Refer Slide Time: 14:09)

The slide has a header 'Subqueries in the From Clause'. It contains a bulleted list:

- SQL allows a subquery expression to be used in the **from** clause
- Find the average instructors' salaries of those departments where the average salary is greater than \$42,000

A SQL query is shown:

```
select dept_name, avg_salary  
from (select dept_name, avg(salary) as avg_salary  
      from instructor  
     group by dept_name)  
   where avg_salary > 42000;
```

Handwritten annotations include:

- A blue box encloses the subquery part of the code.
- A blue bracket on the right side groups the entire subquery with the outer select statement.
- Handwritten text '(dept\_name, avg\_salary)' is written next to the bracketed area.

The footer of the slide includes:

SWAYAM: NIFTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr., 2018  
Database System Concepts - 8<sup>th</sup> Edition  
08.20  
©Silberschatz, Korth and Sudarshan

So, as we have already seen a nested sub query is a relation. So, it can naturally be used in the place of any relation that we have in the from clause. So, we are trying to find out average instructor salaries of those departments where the average salary is greater than 42,000. So, look at. this is a nested sub query. So, where what is been found here this will compute the average salary department wise average salary which we have already seen group by department name and then you do average on the salary and you give it that field a new name.

So which means that; this is equivalent to having a relation which has two attributes depth name and avg\_salary. So, from that you are now trying to do the selection and what is the condition that the average salary has to be written. So, you already have that as a part of the field the average salary. So, you just need to put that in the where clause and you have only those coming out of this particular relation where average salary is greater than 42,000 to be selected in this SELECT query. So, these will feature in the output of this selection.

(Refer Slide Time: 15:56)

The slide features a sailboat icon in the top left corner. The title 'Subqueries in the From Clause' is centered at the top in red. Below the title is a bulleted list of points:

- SQL allows a subquery expression to be used in the **from** clause
- Find the average instructors' salaries of those departments where the average salary is greater than \$42,000
  - `select dept_name, avg_salary  
from (select dept_name, avg(salary) as avg_salary  
 from instructor  
 group by dept_name)  
 where avg_salary > 42000;`
- Note that we do not need to use the **having** clause
- Another way to write above query
  - `select dept_name, avg_salary  
from (select dept_name, avg(salary)  
 from instructor  
 group by dept_name) as dept_avg(dept_name, avg_salary)  
 where avg_salary > 42000;`

At the bottom left, there is a small video thumbnail of a man speaking. The footer contains the text 'Database System Concepts - 8<sup>th</sup> Edition', '08.20', and '©Silberschatz, Korth and Sudarshan'.

So, that is how you can very easily use a nested sub query in the; from clause and for this we did earlier. We solve this problem using the having clause, but they here we will not need we did not need the having clause to do this there is this is another way. So, here what we have done is the same; if we if we look into the nested sub query. This is actually the same. all that we have done we have given it a new name by the renaming feature, and then this as if becomes a relation and on that the computation is done rest of it is similar.

(Refer Slide Time: 16:50)

The slide features a sailboat icon in the top left corner. The title 'With Clause' is centered at the top in red. Below the title is a bulleted list of points:

- The **with** clause provides a way of defining a temporary relation whose definition is available only to the query in which the **with** clause occurs
- Find all departments with the maximum budget

Below the list is a block of SQL code:

```
with max_budget(value) as  
      (select max(budget)  
       from department)  
select department.name  
      from department, max_budget  
     where department.budget = max_budget.value;
```

At the bottom left, there is a small video thumbnail of a man speaking. The footer contains the text 'Database System Concepts - 8<sup>th</sup> Edition', '08.21', and '©Silberschatz, Korth and Sudarshan'.

There is a with clause that provides a way of computing a temporary relation and that can be subsequently used.

So, let us look at example. we are trying to find all departments with maximum budget. So, this is my basic query, we want to find department name, department dot name, that is; a departments name from the department table and the budget must be same as maximum budget. So, for that I need to know the maximum budget that exists across the department. So, look at what has been done, here we have a nested query which aggregates the maximum budget from the departments.

So, this gives you the value of the maximum budget. We make that into a temporary table max budget with an attribute value. So, this is renaming. So, you cannot see the renaming is being used in very interesting ways. So, this is my nested query that gives me a relation and this is my definition of the relation. So, max budget now is a temporary relation a relation that I used subsequently in my from clause and with allows me to do that this relation will not be available.

Otherwise after this query this relation will not exist this is just a temporary one computed for this query. So, this gives me the budget value, this gives me the department and department specific budget, and this condition tells me that I can choose all the departments which has the maximum budget very nice way of using this.

(Refer Slide Time: 19:06)

The slide title is "Complex Queries using With Clause\*" in red. On the left, there is a small image of a sailboat on water. Below the title, there is a bulleted list:

- Find all departments where the total salary is greater than the average of the total salary at all departments

Below the list is a block of SQL code:

```
with dept_total (dept_name, value) as
  (select dept_name, sum(salary)
   from instructor
   group by dept_name),
dept_total_avg(value) as
  (select avg(value)
   from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value > dept_total_avg.value;
```

At the bottom of the slide, there is a video player showing a person speaking. The video player includes controls for volume, brightness, and other media functions. The video is titled "Database System Concepts - 8<sup>th</sup> Edition".

So, with clause can be used in even more involved way again this is an example which is more complex use and I leave it to you to practice study and understand. we move on to sub queries in the SELECT clause finally.

(Refer Slide Time: 19:26)

The slide has a decorative header with a sailboat icon and the title 'Scalar Subquery' in red. The content includes a list of bullet points and a SQL query. The footer contains a photo of the speaker, the course name 'Database System Concepts - 8<sup>th</sup> Edition', the date 'Sat-Apr-2018', and the professor's name 'Prof. P. P. Dass, IIT Kharagpur'. There is also a copyright notice '©Silberschatz, Korth and Sudarshan'.

- Scalar subquery is one which is used where a single value is expected
- List all departments along with the number of instructors in each department

```
select dept_name,  
(select count(*)  
from instructor  
where department.dept_name = instructor.dept_name)  
as num_instructors  
from department;
```

- Runtime error if subquery returns more than one result tuple

A scalar sub query is one; where there is a single value is expected. So, we can very easily use that in the SELECT. So, what if you look at this part which is the sub query? So, you are saying list all departments along with number of instructors each department has. So, this condition tells that, the from the instructor; we are taking out those that department name where the instructor works to count them and then you form that as a new attribute mind you in while we were using this in the from clause.

We were treating that as a relation because nested query will give a relation, but here in the SELECT clause the entities are attributes. So, this as is renaming of attribute which means, but this is a relation that is why this notion of scalar sub query is required, that is though this is a relation what does the relation compute it computes a single value. So, that value we are putting as an attribute named num instructors.

So, we have department name and the number of instructor, then for each and every department; that we actually have from the department list. So, it is a very interesting way of using this nested sub query in terms of the SELECT clause. naturally; since in the SELECT clause, I cannot have, I mean every entry in the SELECT clause has to be an attribute pure relations are not possible.

So, if the sub query returns more than one table which cannot be conceived as one or more attributes, then it will be runtime error that will not be allowed; because we do not know how to handle multiple tuples in terms of a select clause. ok. Next we move on to discussing the modifications to the database, how do we modify the database?

(Refer Slide Time: 22:12)

The slide features a small sailboat icon in the top left corner. The title 'Modification of the Database' is centered at the top in red. On the left edge, there is vertical text: 'SWAYAM/NFTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. In the center, there is a video frame showing a man with glasses and a dark suit. Below the video frame, the text 'Database System Concepts - 8th Edition' is visible. To the right of the video frame, there is a progress bar showing '08.26' and a copyright notice '©Silberschatz, Korth and Sudarshan'. A decorative footer bar with various icons is at the bottom.

- Deletion of tuples from a given relation
- Insertion of new tuples into a given relation
- Updating of values in some tuples in a given relation

So, we will look into some of the ways of changing the records or removing records from that earlier. We saw a delete of record which removed all records from a relation, but now we will see selective deletion insertion and update of values.

(Refer Slide Time: 22:33)

The slide features a small sailboat icon in the top left corner. The title 'Deletion' is centered at the top in red. On the left edge, there is vertical text: 'SWAYAM/NFTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. In the center, there is a video frame showing a man with glasses and a dark suit. Below the video frame, the text 'Database System Concepts - 8th Edition' is visible. To the right of the video frame, there is a progress bar showing '08.27' and a copyright notice '©Silberschatz, Korth and Sudarshan'. A decorative footer bar with various icons is at the bottom.

- Delete all instructors  
`delete from instructor`
- Delete all instructors from the Finance department  
`delete from instructor  
where dept_name= 'Finance';`
- Delete all tuples in the *instructor* relation for those instructors associated with a department located in the Watson building  
`delete from instructor  
where dept_name in (select dept_name  
from department  
where building = 'Watson')`

Now, deleting all instructors are easy DELETE FROM instructor all records sorry this and this becomes an empty table, but suppose we want to delete all instructors from the finance department, then like we do in the SELECT FROM WHERE we again use the where clause as a predicate and say that DELETE FROM instructor, but you do the deletion provided this condition is satisfied that is; department name is same as finance. So, it is very similar to the SELECT FROM WHERE, but the effect is unlike SELECT FROM WHERE, where no tables change in the database here.

The table is actually changing; because these instructor records are deleted whose department name was finance. The third example shows delete all tuples in the instructor relation for those instructors, associated with the department located in the Watson building. So, Watson building may have multiple departments. So, all instructors who worked on those departments which are located in the Watson building that will go. So, you do, this is again you are using nested query.

Now, you know how to use a nested query. So, you use nested query which will give you the names of departments, which gives you a relation with a single attribute with names of departments housed in the Watson building, then you use the set membership to check whether a particular department belongs to that set, if it does then it is in Watson building; otherwise, it is not in Watson building if it does belong to the Watson building then this where clause becomes true and the corresponding instructor record is deleted and that is of this whole different kinds of selective deletion can happen.

(Refer Slide Time: 24:34)

The slide has a header 'Deletion (Cont.)'. On the left, there is a small video window showing a professor. The main content area contains the following text and code:

- Delete all instructors whose salary is less than the average salary of instructors

```
delete from instructor  
where salary < (select avg (salary)  
from instructor);
```

- **Problem:** as we delete tuples from deposit, the average salary changes
- Solution used in SQL:
  1. First, compute `avg (salary)` and find all tuples to delete
  2. Next, delete all tuples found above (without recomputing `avg` or retesting the tuples)

At the bottom, it says 'Database System Concepts - 8<sup>th</sup> Edition' and '08.28'.

Delete all instructors whose salary is less than the average salary of instructor. Again this is; so, you compute the selection sub query which computes the average salary and check if the salary is less than the average salary and delete that. sounds straightforward, but just wait, just wait, I mean did we do it do a right thing an average salary is computed by taking the sum of all salaries in the relation. And then dividing it by the number of relations this has to be the average salary certainly if I remove a record then the average itself will change.

So, if I write the query in this manner then what I am saying on the face of it looks correct, but then actually can it be correct because the moment a condition is satisfied and that record is deleted this average value itself has changed. So, that is not. So, that will depend then the result will depend on the order in which the deletion is happening, but that is not what was meant what was meant is take all the records for the present find out the average find out all records which have a salary less than that average and remove them.

So, this initially you know easy trivial looking solution is not actually correct. So, you will have to do the solution in two stages: first compute the average salary, find all tuples to delete? Next delete all tuples found above without re-computing. The average in the present solution the average is recomputed, which is the wrong thing.

(Refer Slide Time: 26:32)

The slide has a header 'Insertion' in red. On the left, there is a small sailboat icon and some vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018'. The main content includes a bulleted list and two SQL code snippets:

- Add a new tuple to *course*  

```
insert into course  
values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```
- or equivalently  

```
insert into course (course_id, title, dept_name, credits)  
values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```
- Add a new tuple to *student* with *tot\_creds* set to null  

```
insert into student  
values ('3003', 'Green', 'Finance', null);
```

At the bottom, there is footer text: 'Database System Concepts - 8<sup>th</sup> Edition', '08.29', and '©Silberschatz, Korth and Sudarshan'.

So, again I will leave that for you to solve. we move on to looking at modifications in terms of insertion. So, we had seen this earlier we can add a new tuple by inserting to then the relation names, and then you save values and the tuple of values. We can specify the; if we if we do not remember the order of attributes in the relation then we can also specify the order in which we are spaced actually giving the information.

So, you are saying INSERT INTO course and what we have done here is we have actually specified the order in which that tributes occurred and that order and the order of values must be the same. In the first case, this order of values is decided by the order of the attributes that exist in terms of the create table. Add a new tuple to student with total credit set to null that is I do not know; if we were adding a student initially does not have a credit right. The credit is a nullable field the credit will be earned after the student has gone through the courses and all that. So, if I do not know the value of a field then I can write null which is a special value designating unknown for at the time of insertion.

(Refer Slide Time: 28:00)



## Insertion (Cont.)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jam-Apr- 2018

- Add all instructors to the *student* relation with tot\_creds set to 0

```
insert into student
select ID, name, dept_name, 0
from instructor
```
- The **select from where** statement is evaluated fully before any of its results are inserted into the relation
- Otherwise queries like

```
insert into table1 select * from table1
```

would cause problem



Database System Concepts - 8<sup>th</sup> Edition 08.30 ©Silberschatz, Korth and Sudarshan

And all instructors to the student relation with total credit set to 0. So, I can also combine insert with select. So, we are taking the first part this part is selection which generates a whole lot of records having ID, name, department name and the total credit set to 0. From the instructor and insert them into the students. So, these will get inserted into the students SELECT FROM WHERE statement is evaluated fully. So, this first select from will be done before any of its results are inserted in the relation. So, that is the basic condition that SQL guarantees; because, if that were not the case then such situations will become circular and will cause problem.

(Refer Slide Time: 29:03)



## Updates

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jam-Apr- 2018

- Increase salaries of instructors whose salary is over \$100,000 by 3%, and all others by a 5%
  - Write two **update** statements:

```
update instructor
set salary = salary * 1.03
where salary > 100000;
update instructor
set salary = salary * 1.05
where salary <= 100000;
```
  - The order is important
  - Can be done better using the **case** statement (next slide)



Database System Concepts - 8<sup>th</sup> Edition 08.31 ©Silberschatz, Korth and Sudarshan

Updates can be done based on particular values. So, you can update a table and what it means that? It you could update the values of specific fields.

So, here in the first case; we are giving trying to give a 3 percent salary raise for salaries which are more than 100,000 and some 5 percent raise for salaries which are less than equal to 100,000 and mind you this order in which you do the update is important, because if you do the later update first then someone who was what qualified in the later part was less than 100,000 with the increase will become more than 100,000 and will also qualify for the second one. So, that will become wrong. So, update often is dependent on the order.

(Refer Slide Time: 29:58)

The slide has a title 'Case Statement for Conditional Updates' with a sailboat icon. On the left, there's a vertical watermark-like text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018'. The main content includes a bullet point: 'Same query as before but with case statement' followed by an SQL update statement:

```
update instructor  
set salary = case  
    when salary <= 100000 then salary * 1.05  
    else salary * 1.03  
end
```

At the bottom, there's a video player showing a person speaking, the text 'Database System Concepts - 8<sup>th</sup> Edition', the time '08:32', and the copyright notice '©Siberschatz, Korth and Sudarshan'.

And therefore, you have yet another ah feature to take care of this when you have a specific order to do things it is called the case. So, you say when salary case is a new keyword, when is a key word when salary is less than equal to 100,000, then this is how you hike otherwise this is how you hike. So

(Refer Slide Time: 30:28)

The slide has a title 'Updates with Scalar Subqueries' at the top right. On the left, there is a small logo of a sailboat on water. The main content area contains a bulleted list of points and some SQL code. The footer includes the text 'SWAYAM: NPTEL-NOOC MOOCs Instructor: Prof. P. Des., IIT Kharagpur - Jam-Agr.', 'Database System Concepts - 8<sup>th</sup> Edition', '08.33', and '©Silberschatz, Korth and Sudarshan' along with a set of navigation icons.

- Recompute and update tot\_creds value for all students

```
update student S
set tot_cred = (select sum(credits)
    from takes, course
    where takes.course_id = course.course_id and
        S.ID= takes.ID and
        takes.grade <> 'F' and
        takes.grade is not null);

```

- Sets tot\_creds to null for students who have not taken any course
- Instead of `sum(credits)`, use:  

```
case
when sum(credits) is not null then sum(credits)
else 0
end
```

It can it looks more like the; if statement of C, C++ you can do updates with scalar sub queries. We have seen scalar sub queries already. So, you can use a scalar sub query, again I would not go through that details please study and you will be able to understand.

(Refer Slide Time: 30:50)

The slide has a title 'Module Summary' at the top right. On the left, there is a small logo of a sailboat on water. The main content area contains a bulleted list of points. The footer includes the text 'SWAYAM: NPTEL-NOOC MOOCs Instructor: Prof. P. Des., IIT Kharagpur - Jam-Agr.', 'Database System Concepts - 8<sup>th</sup> Edition', '08.34', and '©Silberschatz, Korth and Sudarshan' along with a set of navigation icons.

- Introduced nested sub-query in SQL
- Introduced data modification

So, these are different examples. So to summarize, we have introduced a very powerful feature in SQL query known as nested sub query, where we can write a SELECT FROM WHERE expression as a part of the where predicate or as a relation in the from clause or as one or more collection of attributes in the select clause and it can be used in several

other places also. We have seen the ways to perform data modification in terms of deleting inserting and updating records. And we have also seen how nested sub query often may be very useful not only in terms of performing a query, but also in terms of performing certain data modifications.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 09**  
**Intermediate SQL/1**

Welcome to module 9 of database management systems. We have discussed about the introductory level of SQL the structured query language. in this module and the next we will take up some more intermediate level features of SQL. So, these modules are called intermediate SQL.

(Refer Slide Time: 00:42)

**Module Recap**

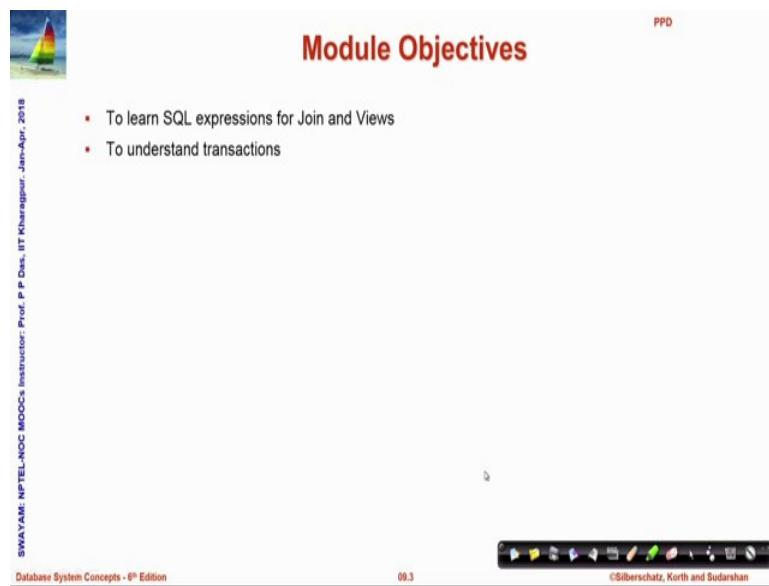
- Nested Subqueries
- Modification of the Database

PPD

Jun-Apr. 2018  
Prof. P. P. Das, IIT Kharagpur - Jun-Apr. 2018  
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jun-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
00.2  
©Silberschatz, Korth and Sudarshan

So, in the last module this is what we have done which was part of the closing part of the introductory SQL. The nested sub queries and modifications to the database.

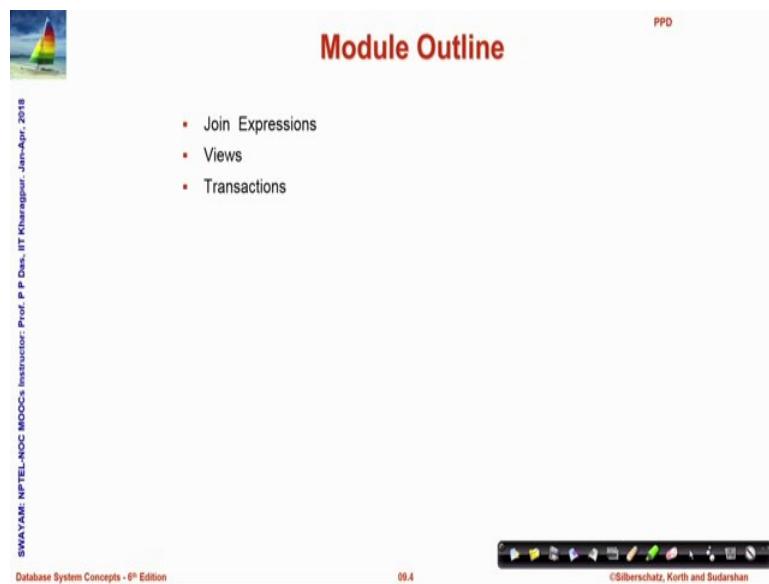
(Refer Slide Time: 00:55)



This slide is titled "Module Objectives" in red text at the top center. In the top right corner, there is a small red "PPD" label. On the left side, there is a vertical watermark-like text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018". At the bottom left, it says "Database System Concepts - 8<sup>th</sup> Edition". The bottom right features a copyright notice: "©Silberschatz, Korth and Sudarshan". The main content consists of two bullet points: "To learn SQL expressions for Join and Views" and "To understand transactions".

Today, we will, in this module learn about SQL expressions for join and views and we will take a quick look into understanding the transaction.

(Refer Slide Time: 01:11)



This slide is titled "Module Outline" in red text at the top center. In the top right corner, there is a small red "PPD" label. On the left side, there is a vertical watermark-like text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018". At the bottom left, it says "Database System Concepts - 8<sup>th</sup> Edition". The bottom right features a copyright notice: "©Silberschatz, Korth and Sudarshan". The main content consists of three bullet points: "Join Expressions", "Views", and "Transactions".

This is the module outline as it will span.

(Refer Slide Time: 01:15)

The slide features a small sailboat icon in the top left corner. In the top right, there is a red 'PPD' logo. Below it, a bulleted list includes 'Join Expressions', 'Views', and 'Transactions'. The main title 'JOIN EXPRESSIONS' is centered in large red capital letters. At the bottom, there is a navigation bar with icons for back, forward, search, and other presentation controls.

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

JOIN EXPRESSIONS

PPD

- Join Expressions
- Views
- Transactions

Database System Concepts - 8<sup>th</sup> Edition

09.5

©Silberschatz, Korth and Sudarshan

So, we start with the join expressions in SQL join as we have already introduced.

(Refer Slide Time: 01:21)

The slide has a small sailboat icon in the top left. The title 'Joined Relations' is centered in red capital letters. Below the title is a bulleted list of four points about join operations. At the bottom, there is a video frame showing a man speaking, a navigation bar, and course information.

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Joined Relations

- **Join operations** take two relations and return as a result another relation
- A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition).
- It also specifies the attributes that are present in the result of the join
- The join operations are typically used as subquery expressions in the **from** clause

Database System Concepts - 8<sup>th</sup> Edition

09.6

©Silberschatz, Korth and Sudarshan

Takes two relations and returns a result as another relation. So, it is two different instances of two schemas and we try to connect them according to certain properties.

So, a join operation, is primarily a Cartesian product, which relates tuples in two relations under certain conditions of match. It also specifies that after the joining has been done, what are the tuples, which will be present in the output join. So, join

operation typically uses sub query, is used in a sub query, in the FROM clause we will see those usages later.

(Refer Slide Time: 02:19)

The slide has a header 'Types of Join between Relations' with a small sailboat icon. On the right, it says 'PPD'. The left margin contains vertical text: 'SWAYAM: NPTEL-NOOC MOOCs', 'Instructor: Prof. P. P. Das, IIT Kharagpur', and 'Date: Jan-Apr., 2018'. The main content is a bulleted list of join types:

- Cross join
- Inner join
  - Equi-join
    - Natural join
- Outer join
  - Left outer join
  - Right outer join
  - Full outer join
- Self-join

At the bottom, there's a video player showing a man speaking, the text 'Database System Concepts - 8<sup>th</sup> Edition', the number '09.7', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, if we look into the different types of join that SQL supports, these are the different classifications. So, we have CROSS JOIN, we have INNER JOIN, which specifically could be EQUI JOIN and even more specifically NATURAL JOIN and we will see there are variety of OUTER JOIN, that are possible. There could be a SELF JOIN also, where one relation is joined with itself.

(Refer Slide Time: 02:51)

The slide has a header 'Cross Join' with a small sailboat icon. On the right, it says 'PPD'. The left margin contains vertical text: 'SWAYAM: NPTEL-NOOC MOOCs', 'Instructor: Prof. P. P. Das, IIT Kharagpur', and 'Date: Jan-Apr., 2018'. The main content is a bulleted list explaining CROSS JOIN:

- CROSS JOIN returns the Cartesian product of rows from tables in the join
  - Explicit

```
select *  
from employee cross join department;
```
  - Implicit

```
select *  
from employee, department;
```

At the bottom, there's a video player showing a man speaking, the text 'Database System Concepts - 8<sup>th</sup> Edition', the number '09.8', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

CROSS JOIN is just a formal join based name for Cartesian product of two rows. So, you could explicitly do a CROSS JOIN, which you can see here or you can implicitly also do a CROSS JOIN, but by specifying two or more relations in from clause, and taking all the attributes from there, we have seen these kind of Cartesian products earlier. So, CROSS JOIN here is more as placeholder. in the context of the join semantics, the pure Cartesian product is a CROSS JOIN, but what would be more interesting is, when we take different kinds of inner and OUTER JOINs.

(Refer Slide Time: 03:30)

**Join operations – Example**

- Relation *course*

| <i>course_id</i> | <i>title</i> | <i>dept_name</i> | <i>credits</i> |
|------------------|--------------|------------------|----------------|
| BIO-301          | Genetics     | Biology          | 4              |
| CS-190           | Game Design  | Comp. Sci.       | 4              |
| CS-315           | Robotics     | Comp. Sci.       | 3              |

- Relation *prereq*

| <i>course_id</i> | <i>prereq_id</i> |
|------------------|------------------|
| BIO-301          | BIO-101          |
| CS-190           | CS-101           |
| CS-347           | CS-101           |

- Observe that
  - prereq information is missing for CS-315 and
  - course information is missing for CS-437

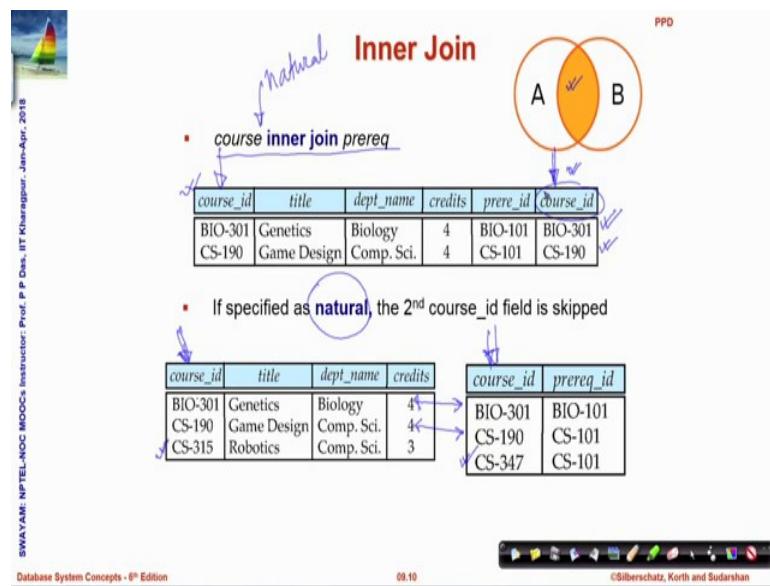
Navigation icons: back, forward, search, etc.

SWAYAM: NPTEL-NOCO MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur  
 Database System Concepts - 8<sup>th</sup> Edition      09.9      ©Silberschatz, Korth and Sudarshan

So, let us start with a simple example to understand the issues. So, there is a relation *course*, which has four attributes and in this particular instance, it has three tuples; three rows and there is another relation *prereq*, which specifies the prerequisite for every course.

So, it has two attributes; the *course\_id* and the corresponding *prereq\_id*. It also has three rows; three tuples here, and if you look at the instances carefully, you will find that the three courses that are specified in the *course* relation all are not specified in the *prereq* relation. Bio-301 and CS-190 is present in *prereq*, but CS-315 is not present in at the same time, the *prereq* has one particular tuple, specifying the prerequisite of CS-347, which in turn is not present in the *course* relation. So, with this observation, let us start trying to see what different joint mean.

(Refer Slide Time: 04:54)



So, in a join is computed, then in terms of the two relations that we have, there is an attribute `course_id`, which is common. So, once we have taken the cross product, we will from the cross product, only retain those rows, where the `course_id` in relation `course` and the `course_id` in relation `prereq` are same. So, when we do this particular record, when it gets mapped with this corresponding record, it will generate the corresponding output record. Similarly, CS 190, when it is mapped to the CS 190, in the `prereq`, it will generate the second record, we have already understood this, the third record in the courses CS 315 has no match here in `prereq`.

So, that will not feature in the output. Similarly, in the `prereqs` C S 347 that exist has no match in `courses`. So, that also will not appear in the output and also in the output you find that the `course_id` has actually featured twice. This is the first column `course_id` comes from `course`. So, it should more formally be called **course.course\_id** whereas, the second one comes from `prereq`. So, it should that should formally be called **prereq.course\_id**.

Now, if in addition to saying that, this is an INNER JOIN, if I also specify the word natural, I can say natural here, if say natural, then this second duplicate attribute `course_id` will be dropped from the output that becomes a NATURAL JOIN, INNER JOIN as the name suggests, finds out the inner part of the two relations.

So, if we look at the two relations as A and B only those records, which are both have instance in A as well as B, in terms of equality of this course\_id attribute will come in the output. So, this is the basic type of join, INNER JOIN which is most commonly used.

(Refer Slide Time: 07:34)

The slide has a title 'Outer Join' in red font. To the left of the title is a small image of a sailboat on water. On the right side of the slide is a list of bullet points:

- An extension of the join operation that avoids loss of information
- Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join
- Uses *null* values

At the bottom of the slide, there is footer text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Date: Apr. 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '09.11', and '©Silberschatz, Korth and Sudarshan'.

Now, we can extend this into a different kind of join, known as OUTER JOIN in the INNER JOIN. As you have seen that courses that exist in the course relation, but are not there in the prereq or the ones that exist in the prereq and is not there in the course are not featuring in the final INNER JOIN output. So, there is some loss of information in terms of this. So, why we are doing this we can compute and add tuples from one relation that may not match with any tuple in the other relation and if we want to do that then naturally for the other attributes of that tuple in the target relation we will not know the values. So, we will use null values this is the basic idea of OUTER JOIN.

(Refer Slide Time: 08:34)

| course_id | title       | dept_name  | credits | prereq_id |
|-----------|-------------|------------|---------|-----------|
| BIO-301   | Genetics    | Biology    | 4       | BIO-101   |
| CS-190    | Game Design | Comp. Sci. | 4       | CS-101    |
| CS-315    | Robotics    | Comp. Sci. | 3       | null      |

| course_id | prereq_id |
|-----------|-----------|
| BIO-301   | BIO-101   |
| CS-190    | CS-101    |
| CS-347    | CS-101    |

So, let us see what it specifically means? We first talked about left OUTER JOIN, left in the sense that we have, this is how it is written. Left OUTER JOIN is a sequence of commands that you give, you are also saying it is natural, which means that the common attribute will not feature twice in the output and this is the left relation and this is the right relation. So, left OUTER JOIN specifies that in the output all records of the left relation, in this case the course relation must feature.

So, naturally when we do the join, we will get these two records as we have got in terms of the INNER JOIN, in terms of course 315 the CS 315, the third course there is no instance in the prereq, we will still have that in the output, but since the prereq value for that, the prerequisite value is not known, the prerequisite id will be set to null here. So, left OUTER JOIN ensure that, all relations of the left relation, all tuples of the left relation will necessarily feature in the output and that is a reason. If you see in the Venn diagram, the whole of this set, A is shown whereas, this part certainly will not feature.

(Refer Slide Time: 10:05)

The slide features a sailboat icon in the top left corner and a Venn diagram in the top right showing two overlapping circles labeled A and B, with the text 'PPD' above it. The title 'Right Outer Join' is centered in red. Below the title is a bulleted list: 'course natural right outer join prereq'. To the right of the list is a table showing the result of a right outer join between 'course' and 'prereq' relations.

| course_id | title       | dept_name  | credits | prereq_id |
|-----------|-------------|------------|---------|-----------|
| BIO-301   | Genetics    | Biology    | 4       | BIO-101   |
| CS-190    | Game Design | Comp. Sci. | 4       | CS-101    |
| CS-347    | null        | null       | null    | CS-101    |

| course_id | title       | dept_name  | credits |
|-----------|-------------|------------|---------|
| BIO-301   | Genetics    | Biology    | 4       |
| CS-190    | Game Design | Comp. Sci. | 4       |
| CS-315    | Robotics    | Comp. Sci. | 3       |

| course_id | prereq_id |
|-----------|-----------|
| BIO-301   | BIO-101   |
| CS-190    | CS-101    |
| CS-347    | CS-101    |

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jam-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition 09.13 ©Silberschatz, Korth and Sudarshan

Now, similarly we can have a right OUTER JOIN, where the concept is the same except that. Now, we ensure that all records of the right relation, in this case the prereq relation will feature and therefore, C S 2347 for which there is no entry in the course relation will also come as a record and since we do not know the title department name and credits for these fields, we will put them as null and this again is a natural one. So, course\_id is featuring only once. So, you will understand that since, we have a left version and we have a right version.

(Refer Slide Time: 10:47)

The slide features a sailboat icon in the top left corner and a toolbar at the bottom. The title 'Joined Relations' is centered in red. Below the title is a bulleted list of four points about join operations. To the right of the list are two tables: 'Join types' and 'Join Conditions'.

- Join operations take two relations and return as a result another relation
- These additional operations are typically used as subquery expressions in the **from** clause
- Join condition** – defines which tuples in the two relations match, and what attributes are present in the result of the join
- Join type** – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated

| Join types       |
|------------------|
| inner join       |
| left outer join  |
| right outer join |
| full outer join  |

| Join Conditions                  |
|----------------------------------|
| natural                          |
| on <predicate>                   |
| using ( $A_1, A_2, \dots, A_n$ ) |

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jam-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition 09.14 ©Silberschatz, Korth and Sudarshan

We can actually have a full version as well. So, if we look into the join relations in general, it takes two relations and returns a result and those additional operations are used in the sub query in from and there is a set of join conditions. So, these are the join conditions that we are specifying, whether it is natural and we will soon see that we can actually not depend only on the attributes that are common, we can actually specify that which attributes should be used in computing the joint. So, those are on condition and the using clause, we will just illustrate them soon and finally, there are four types of join that can happen, that is the INNER JOIN. We have seen the left OUTER JOIN, right OUTER JOIN and we will soon see the full OUTER JOIN.

(Refer Slide Time: 11:52)

**Full Outer Join**

PPD

- course natural full outer join prereq

| course_id | title       | dept_name  | credits | prereq_id |
|-----------|-------------|------------|---------|-----------|
| BIO-301   | Genetics    | Biology    | 4       | BIO-101   |
| CS-190    | Game Design | Comp. Sci. | 4       | CS-101    |
| CS-315    | Robotics    | Comp. Sci. | 3       | null      |
| CS-347    | null        | null       | null    | CS-101    |

| course_id | title       | dept_name  | credits |
|-----------|-------------|------------|---------|
| BIO-301   | Genetics    | Biology    | 4       |
| CS-190    | Game Design | Comp. Sci. | 4       |
| CS-315    | Robotics    | Comp. Sci. | 3       |

| course_id | prereq_id |
|-----------|-----------|
| BIO-301   | BIO-101   |
| CS-190    | CS-101    |
| CS-347    | CS-101    |

SWAYAM NPTEL-MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur. Jam-Apr. 2015

Database System Concepts - 8<sup>th</sup> Edition 09.15 ©Silberschatz, Korth and Sudarshan

So, full OUTER JOIN as you must have guessed will ensure that you get certainly the tuples from the INNER JOIN, which is here, you will get the tuple from the left OUTER JOIN that is here, that is a tuple which exists in course and there is no corresponding matching tuple in the prereq and you will also get the tuple from the right OUTER JOIN, that is for tuple, which exists in the prereq relation, but there is no corresponding tuple in the course relation and corresponding missing values are all set to null. So, these three kinds of OUTER JOIN are possible.

(Refer Slide Time: 12:38)

The slide has a header 'Joined Relations – Examples' with a sailboat icon. On the left, vertical text reads 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018'. At the bottom, it says 'Database System Concepts - 8<sup>th</sup> Edition' and '09.16 ©Silberschatz, Korth and Sudarshan'.

**course inner join prereq on**  
`course.course_id = prereq.course_id`

| course_id | title       | dept_name  | credits | prere_id | course_id |
|-----------|-------------|------------|---------|----------|-----------|
| BIO-301   | Genetics    | Biology    | 4       | BIO-101  | BIO-301   |
| CS-190    | Game Design | Comp. Sci. | 4       | CS-101   | CS-190    |

**▪ What is the difference between the above (equi\_join), and a natural join?**

**▪ course left outer join prereq on**  
`course.course_id = prereq.course_id`

| course_id | title       | dept_name  | credits | prere_id | course_id |
|-----------|-------------|------------|---------|----------|-----------|
| BIO-301   | Genetics    | Biology    | 4       | BIO-101  | BIO-301   |
| CS-190    | Game Design | Comp. Sci. | 4       | CS-101   | CS-190    |
| CS-315    | Robotics    | Comp. Sci. | 3       | null     | null      |

So, you can also specify join by saying that explicitly saying what attribute we want to join on and if you specify that, then you are saying it is a course INNER JOIN prereq. This part was same then you are putting an on clause, saying in the on clause, you will have to provide a predicate that is, which field should equate or match with what field. So, you are saying `course.course_id = prereq.course_id`.

So, this result incidentally happens to be same as just doing the INNER JOIN, but we are illustrating that, on clause can explicitly use. For example, between the two relations, we have more than one common attribute, but we may want to actually do the INNER JOIN based on only one of them or equality on two of them and so on..

So, this kind of a join, where INNER JOIN, where you set two fields to be equal or two or more fields to be equal is also known as EQUI JOIN and since we have not specified natural, you can again observe, then the `course_id` attribute has occurred twice. If it was said natural then the second `course_id` attribute would not have come in the result, this is a showing. The left OUTER JOIN in terms of on clause and we have seen similar results. And now, this can be seen in terms of the on clause as well, and you can see in that second `course_id` field. This entry is null, because actually you do not have that in the prerequisite set and; obviously, this set will be null, this field will be null..

(Refer Slide Time: 14:35)

The slide title is 'Joined Relations – Examples'. It features two tables illustrating joins:

- course natural right outer join prereq**:  
A table showing courses and their prerequisites. Courses BIO-301 and CS-190 have prerequisites, while CS-347 does not.

| course_id | title       | dept_name  | credits | prere_id |
|-----------|-------------|------------|---------|----------|
| BIO-301   | Genetics    | Biology    | 4       | BIO-101  |
| CS-190    | Game Design | Comp. Sci. | 4       | CS-101   |
| CS-347    | null        | null       | null    | CS-101   |
- course full outer join prereq using (course\_id)**:  
A table showing courses and their prerequisites, using course\_id for the join. Courses BIO-301, CS-190, CS-315, and CS-347 are listed with their respective prerequisites.

| course_id | title       | dept_name  | credits | prere_id |
|-----------|-------------|------------|---------|----------|
| BIO-301   | Genetics    | Biology    | 4       | BIO-101  |
| CS-190    | Game Design | Comp. Sci. | 4       | CS-101   |
| CS-315    | Robotics    | Comp. Sci. | 3       | null     |
| CS-347    | null        | null       | null    | CS-101   |

Other slide details: SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Des. IIT Kharagpur - Jam-Astr. 2018. Database System Concepts - 8<sup>th</sup> Edition. 09.17. ©Silberschatz, Korth and Sudarshan.

So, this is another example, showing you the natural right OUTER JOIN, this is you, showing you full OUTER JOIN and we are showing the use of the using clause. We can say using and put a set of attributes and the meaning is the join will be performed, based on those attributes. So, here in this case again the join will be based on course\_id.

(Refer Slide Time: 15:01)

The slide title is 'VIEWS'. It lists three types of views:

- Join Expressions
- Views
- Transactions

Other slide details: SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Des. IIT Kharagpur - Jam-Astr. 2018. Database System Concepts - 8<sup>th</sup> Edition. 09.18. ©Silberschatz, Korth and Sudarshan.

So, that was about different kinds of join that we can do, which we going forward. We will see that form, a very critical has a very critical place, in terms of query formulation. Now, we take you to a different concept known as views now.

(Refer Slide Time: 15:20)

**Views**

- In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database.)
- Consider a person who needs to know an instructors name and department, but not the salary. This person should see a relation described, in SQL, by

```
select ID, name, dept_name  
from instructor
```

- A **view** provides a mechanism to hide certain data from the view of certain users

relation that is not of the conceptual model but is made available to a user as a "virtual relation" is called a **view**

SWAYAM: NPTEL-NOOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

09.19

©Silberschatz, Korth and Sudarshan

We have seen that. So, far we have been computing certain query results, based on one or more relations one or more instances. Now, in some cases, we may want the results to be restrictive in terms of based on the user or based on the context, in which the result should be used. So, we may not want all fields of a result to be visible to all the users or to the application. So, we may not expose the whole logical model and in those cases, we introduce a view. So, here we are showing one, where, from the instructor relation, we are only picking up three fields and we are not picking up the salary field.

Now, you would think that, well this is what we can do in terms of the normal query and certainly then, what is the point of using this? Now, what we can do is, we can create this not adjust as a query, but as a view one, once we create this as a view, it actually this query expression is treated as what is known as a view expression and every time you want to use that view. The actual tuples in that view are computed, but this is not actually a relation that exists in the database. So, it is a kind of, can be thought of as a kind of virtual relation, which exists, which can be seen only when you use that.

(Refer Slide Time: 17:22)

The slide has a header 'View Definition' in red. On the left, there is a small sailboat icon and a portrait of a man. The main content is a bulleted list about views:

- A view is defined using the `create view` statement which has the form
  - `create view v as < query expression >`
  - where `<query expression>` is any legal SQL expression
  - The view name is represented by `v`
  - Once a view is defined, the view name can be used to refer to the virtual relation that the view generates
  - View definition is not the same as creating a new relation by evaluating the query expression
    - Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view

So, there is a subtle but very strong difference between, actually computing a result, through a select query and defining a view, based on a select query and then making use of the view, as if it were actually a relation that existed. So, to do this, this is how we go about. It is the send text, is very similar to the create table. So, you do a create view give a name and then you specify as is the connective and specify the query expression, which is an SQL query, which will let you compute the view, every time you actually need it..

So, this is a view name, once a view is defined, the view name can be used as a virtual relation, it can be used exactly as we use any of the really existing relation, the conceptual relations that we have created, through create table. So, it is the difference, this is what needs to be understood very well, the view definition is not the same as creating a new relation, once you create the new relation, the time you have created it, you get the result and that result is explicitly available as a set of tuples as a table rather a view is a definition, which you stored in the database as an expression. So, every time you make use of that view, at that time the set of tuples are computed. It is not existing in the database as stored like the real relations and based on that computation, all the rest of the query will actually be executed.

(Refer Slide Time: 18:58)

The slide has a header 'Example Views' with a sailboat icon. On the left, there's vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018'. The main content lists three examples:

- A view of instructors without their salary:

```
create view faculty as
select ID, name, dept_name
from instructor
```
- Find all instructors in the Biology department:

```
select name
from faculty
where dept_name = 'Biology'
```
- Create a view of department salary totals:

```
create view departments_total_salary(dept_name, total_salary) as
select dept_name, sum(salary)
from instructor
group by dept_name;
```

Handwritten annotations include blue circles around 'dept\_name' in the first two examples and 'dept\_name' and 'sum(salary)' in the third example. There are also blue arrows pointing from the circled text to the corresponding parts in the third example's SQL code.

At the bottom, it says 'Database System Concepts - 8<sup>th</sup> Edition' and '09.21 ©Silberschatz, Korth and Sudarshan'.

So, let us take a quick look, this is a create view, we have created the view of a, of view called faculty, from instructor relation. Instructor relation is the real one, I existing one and faculty is a view expression being created and in that, what we have done? Simply, we have taken a done, a projection, we have lived left out the salary field. Now, we can make use of that view, you can see that we are doing from faculty.

So, this actually is a view, but this behaves as if this is varying relation. So, from faculty we are trying to find out the name of all those faculties; who belong to the biology department. So, what will happen, when they want to execute this query? This will refer to this view. So, to execute this query, it will have to first execute this query, get the temporary virtual instance of the virtual relation, created and based on that, this query will be computed and the results will be given accordingly.

So, that is the basic purpose of the view that the whole thing, the whole view expression remains as an abstraction in the database and computed whenever it is used. So, this is showing you another view, which shows certain computed information. For example, we are creating a view for departmental total salary, which will show as two fields department name and total salary, which has been created by aggregation.

So, anytime we make use of this view in a from clause, we will get, we will feel as if such a relation really exists where the department name and the total salary of the instructors in that department are stored, but it really does not exist. It is computed

whenever it is needed, whenever it is used, you can actually use views to create other views.

(Refer Slide Time: 20:52)

The slide has a title 'Views Defined Using Other Views' at the top right. On the left, there is a small image of a sailboat on water. A vertical column of text on the left edge reads: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018'. The main content area contains two bullet points with SQL code:

- `create view physics_fall_2009 as  
select course.course_id, sec_id, building, room_number  
from course, section  
where course.course_id = section.course_id  
and course.dept_name = 'Physics'  
and section.semester = 'Fall'  
and section.year = '2009';`
- `create view physics_fall_2009_watson as  
select course_id, room_number  
from physics_fall_2009  
where building='Watson';`

At the bottom left, there is a video thumbnail showing a man speaking. At the bottom right, there is a navigation bar with icons for back, forward, search, and other presentation controls. The footer of the slide says 'Database System Concepts - 6<sup>th</sup> Edition' and '09.22'.

For example, this is one view which is the view of `physics_fall_2009`, which are all courses that are offered in physics, from the physics department in the semester fall of year 2009 and using that we can create another view. See here again, we are in the, from clause we are using this view. So, creating this using this view, we are creating yet another view, which shows the courses that ran in the Watson building. So, views can be used as I have already said as any other.

(Refer Slide Time: 21:35)

The slide has a header 'View Expansion' with a sailboat icon. On the left, there's a vertical footer bar with text: 'SWAYAM: NPTEL-NOOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018'. The main content area contains a bulleted list and a code snippet. The list includes: 'Expand use of a view in a query/another view'. The code snippet is:

```
create view physics_fall_2009_watson as
(select course_id, room_number
from (select course.course_id, building, room_number
      from course, section
     where course.course_id = section.course_id
       and course.dept_name = 'Physics'
       and section.semester = 'Fall'
       and section.year = '2009')
   where building= 'Watson';
```

A large curly brace is placed over the inner select statement. At the bottom, there's a video frame of a professor, a progress bar showing '09.23', and a footer '©Silberschatz, Korth and Sudarshan'.

Actual relation, but they do not really exist. So, if you expand out, if you just put the physics fall 2009 expression, within the view definition of physics for 2009 Watson, this is your earlier view relation. So, this is known as view expansion. So, this is actually the query that, you are executing and that has a lot of value.

(Refer Slide Time: 22:04)

The slide has a header 'Views Defined Using Other Views' with a sailboat icon. On the left, there's a vertical footer bar with text: 'SWAYAM: NPTEL-NOOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018'. The main content area contains a bulleted list:

- One view may be used in the expression defining another view
- A view relation  $v_1$  is said to *depend directly* on a view relation  $v_2$  if  $v_2$  is used in the expression defining  $v_1$
- A view relation  $v_1$  is said to *depend on* view relation  $v_2$  if either  $v_1$  depends directly to  $v_2$  or there is a path of dependencies from  $v_1$  to  $v_2$
- A view relation  $v$  is said to be *recursive* if it depends on itself

At the bottom, there's a video frame of a professor, a progress bar showing '09.24', and a footer '©Silberschatz, Korth and Sudarshan'.

So, as we have said views can be defined indirectly from one relation. So, these are called direct dependence or they could be defined in terms of a chain of relations  $v_1$  in

terms of v2, v2 in terms of v3 and so on. And a view relation can be recursive also, that a view could be in terms of itself.

(Refer Slide Time: 22:29)

The slide features a small sailboat icon in the top left corner. The title 'View Expansion\*' is centered in red. On the left, vertical text reads 'SWAYAM: NPTEL-NOOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. The main content is a bulleted list:

- A way to define the meaning of views defined in terms of other views
- Let view  $v_i$  be defined by an expression  $e_1$  that may itself contain uses of view relations
- View expansion of an expression repeats the following replacement step:

```
repeat
    Find any view relation  $v_i$  in  $e_1$ 
    Replace the view relation  $v_i$  by the expression defining  $v_i$ 
    until no more view relations are present in  $e_1$ 
```
- As long as the view definitions are not recursive, this loop will terminate

At the bottom, there's a video player showing a man speaking, with controls for volume, brightness, and navigation. The text 'Database System Concepts - 8<sup>th</sup> Edition' and '09.25' are at the bottom left, and '©Silberschatz, Korth and Sudarshan' is at the bottom right.

A lot of power view expansion is the process that SQL uses to evaluate a view. So, I would request you to study this and understand that this process works. This is pretty much like pseudo code C program.

(Refer Slide Time: 22:47)

The slide features a small sailboat icon in the top left corner. The title 'Recursive View' is centered in red. On the left, vertical text reads 'SWAYAM: NPTEL-NOOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. The main content is a bulleted list:

- In SQL, recursive queries are typically built using these components:
  - A non-recursive seed statement
  - A recursive statement
  - A connection operator
    - The only valid set connection operator in a recursive view definition is UNION ALL
  - A terminal condition to prevent infinite recursion

At the bottom, there's a video player showing a man speaking, with controls for volume, brightness, and navigation. The text 'Database System Concepts - 8<sup>th</sup> Edition' and '09.26' are at the bottom left, and '©Silberschatz, Korth and Sudarshan' is at the bottom right.

Now, moving to recursive views, the views where the same relation can be used in the view to define another view, we need like every other recursive structure. We need first a

non-recursive statement, which is called the seed statement. we need a recursive statement, which can recur, we need a connection operator, which can connect the non-recursive and the recursive results together put them together, the only connective that is valid is union, all that is multi set union and we also need some kind of a terminal condition to guarantee that the recursion really a terminus, it does not go on forever.

(Refer Slide Time: 23:36)

**Recursive View – Example**

- In the context of a relation *flights*:

```
create table flights (
    source varchar(40),
    destination varchar(40),
    carrier varchar(40),
    cost decimal(5,0));
```

| source   | destination | carrier           | cost |
|----------|-------------|-------------------|------|
| Paris    | Detroit     | KLM               | 7    |
| Paris    | New York    | KLM               | 6    |
| Paris    | Boston      | American Airlines | 8    |
| New York | Chicago     | American Airlines | 2    |
| Boston   | Chicago     | American Airlines | 6    |
| Detroit  | San Jose    | American Airlines | 4    |
| Chicago  | San Jose    | American Airlines | 2    |

- Find all the destinations that can be reached from 'Paris'

SWAYAM: NPTEL-NOCs Instructor: Prof. P. P. Desai, IIT Kharagpur Date: 2018

Database System Concepts - 8<sup>th</sup> Edition

09.27

©Silberschatz, Korth and Sudarshan

So, let us take an example. So, this is in context of a relation *flights*, where the four fields are as specified and there is an instance shown, which show that different source destination of different carriers, carrying people from one source to the other destination and what do you want to find is all destinations that can be reached from Paris. Now, you can see that from Paris, if I can reach Detroit and from Detroit I can reach San Jose, then I can actually reach San Jose from Paris. So, that is the basic reachability. So, that will necessarily, if I want to compute that, then I will be able to compute this very easily by doing NATURAL JOIN of flights with flights provided, I take say source.

(Refer Slide Time: 24:38)

**Recursive View – Example**

PPD

SWAYAM: NPTEL-NOOC MOOCs Instructor: Prof. P. Deshpande, IIT-Kharagpur - Jan-Apr., 2018

- In the context of a relation *flights*:

```
create table flights (
    source varchar(40),
    destination varchar(40),
    carrier varchar(40),
    cost decimal(5,0));
```

| source   | destination | carrier           | cost |
|----------|-------------|-------------------|------|
| Paris    | Detroit     | KLM               | 7    |
| Paris    | New York    | KLM               | 6    |
| Paris    | Boston      | American Airlines | 8    |
| New York | Chicago     | American Airlines | 2    |
| Boston   | Chicago     | American Airlines | 6    |
| Detroit  | San Jose    | American Airlines | 4    |
| Chicago  | San Jose    | American Airlines | 2    |

- Find all the destinations that can be reached from 'Paris'

flights f1  $\bowtie$  flights f2  
 $f_1.destination = f_2.source$

Database System Concepts - 8<sup>th</sup> Edition 09.27 ©Silberschatz, Korth and Sudarshan

Let us compute it like this, flights f1 join, flights f2 and I will have **f1.destination = f2.sources**. So, the idea is, if something goes from Paris to Detroit that is in f1 and if some flight goes from Detroit to San Jose that is in f2, then the destination in f1 and the source in f2 have to be equated. So, if we do this kind of a self EQUI JOIN, then we will be able to find out all flights that go from Paris to San Jose or all places that you can reach from Paris in one hop.

Naturally, once you reach, once you do that then you may be able to go to another destination in two hops and once you do that then, you may be able to reach another, yet another destination in three hops and so on. So, we do not really know how many hops, maximum would be required to compute this reachability information. So, that is the reason we need to make use of the recursion and so, this is how we express it.

(Refer Slide Time: 25:54)

**Recursive View – Example**

```

create recursive view reachable_from (source,destination,depth) as (
  select root.source, root.destination, 0 as depth
  from flights as root
  where root.source = 'Paris'
union all
  select in1.source, out1.destination, in1.depth + 1
  from reachable_from as in1, flights as out1
  where in1.destination = out1.source and
        in1.depth <= 100);
  
```

- A non-recursive seed statement
- A recursive statement
- A connection operator
- A terminal condition to prevent infinite recursion

Get the result by simple selection on the view:

```

select distinct source, destination
from reachable_from;
  
```

| source   | destination | carrier           | cost |
|----------|-------------|-------------------|------|
| Paris    | Detroit     | KLM               | 7    |
| Paris    | New York    | KLM               | 6    |
| Paris    | Boston      | American Airlines | 8    |
| New York | Chicago     | American Airlines | 2    |
| Boston   | Chicago     | American Airlines | 6    |
| Detroit  | San Jose    | American Airlines | 4    |
| Chicago  | San Jose    | American Airlines | 2    |
| Paris    | Detroit     |                   |      |
| Paris    | New York    |                   |      |
| Paris    | Boston      |                   |      |
| Paris    | Chicago     |                   |      |
| Paris    | San Jose    |                   |      |

This example view, `reachable_from`, is called the *transitive closure* of the `flights` relation

Source: [https://info.teradata.com/HTMLPubs/DB\\_TTU\\_16\\_00/index.html#page/SQL\\_Reference%2FB035-1184-160K%2Fname1472241335807.html%23wwID0EJ23T](https://info.teradata.com/HTMLPubs/DB_TTU_16_00/index.html#page/SQL_Reference%2FB035-1184-160K%2Fname1472241335807.html%23wwID0EJ23T)

So, if we look into this, we are specifying that is a recursive view. It will happen with itself, this is the name and this is what we want to compute source destination and we take another dummy attribute kind of which specify the depth of recursion. So, the present instance of the relation is at depth 0.

So, which defines your non recursive seat part. so, it says select. So, you have renamed it as flights as route, you are specified that it has to start from Paris and you can find out the source destination pair at depth 0, then you specify the recursive part that is the second hop has to be defined. So, we are saying that, if you had the reachability then call, let us call it in one. This reachability maybe in one hop that is a depth 0 maybe in 2 hops that is a depth 1 maybe at 3 hops; that is a depth 2 and you take another instance of flight as out 1 and what you need is the destination in the first in one has to be same as the source in the other so that they get connected and then you output the source from the first one destination from the second one and naturally the depth has got incremented, because you have done added one more hop and so, this is the and you add another condition saying that **in1.depth** should be  $\leq$  to 100. This is as I mentioned is a terminal condition which makes sure that, you do not get into infinite recursion. So, this view, recursive view cannot be used to compute any reachability, which has more than 101 hops.

So, that is to be noted and finally, we need to connect these two results, which is the initial start seed and the recursive one. So, this is the connection operator.

So, this is basically the idea of the recursive view, those of you who are more familiar with discrete structure, would have known among relations in some more depth, you would know that, we can define a transitive closure of a binary relation. So, this recursive view is necessarily computing the transitive closure from the flights relation. So, this is the instance of the flights and on the final computation, this is what you get. This gives you all the destinations that can eventually be reached from the source Paris.

(Refer Slide Time: 28:41)

The slide has a title 'The Power of Recursion' in red at the top right. To its left is a small icon of a sailboat on water. On the far left edge, there is vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. At the bottom left is the text 'Database System Concepts - 8<sup>th</sup> Edition'. In the bottom right corner, there is a navigation bar with icons for back, forward, search, and other presentation controls. The main content area contains the following bullet points:

- Recursive views make it possible to write queries, such as transitive closure queries, that cannot be written without recursion or iteration
  - Intuition: Without recursion, a non-recursive non-iterative program can perform only a fixed number of joins of *flights* with itself
    - This can give only a fixed number of levels of reachable destinations
    - Given a fixed non-recursive query, we can construct a database with a greater number of levels of reachable destinations on which the query will not work

So, the recursive is very powerful in the sense that without recursion a non-recursive version can only find flights up to a certain number of hops and whatever ma query you write it is always possible to write out a database instance which will have more hops and your query will necessarily fail.

(Refer Slide Time: 29:05)

The slide title is "The Power of Recursion". The text on the slide includes:

- Computing transitive closure using iteration, adding successive tuples to `reachable_from`
  - The next slide shows a `flights` relation
  - Each step of the iterative process constructs an extended version of `reachable_from` from its recursive definition
  - The final result is called the *fixed point* of the recursive view definition.
- Recursive views are required to be **monotonic**. That is, if we add tuples to `flights` the view `reachable_from` contains all of the tuples it contained before, plus possibly more

A hand-drawn diagram is present on the slide, showing a ladder leaning against a wall, with the word "union all" written above it.

At the bottom of the slide, there is a footer with the text "Database System Concepts - 8<sup>th</sup> Edition", "09.30", and "©Silberschatz, Korth and Sudarshan".

So, we make use of the recursion here to make sure that you can actually extend this to whatever depth you want and to compute this we keep on computing till no changes are possible and in that sense these recursive views are said to be monotonic in that every time you compute your result necessarily becomes larger and that is the reason you for the purpose of being monotonic you are actually making use of the union all. So, that makes it all inclusive

(Refer Slide Time: 29:49)

The slide title is "Example of Fixed-Point Computation". It features a table of flight data and a table showing the progression of tuples in closure through three iterations.

**Flight Data Table:**

| source   | destination | carrier           | cost |
|----------|-------------|-------------------|------|
| Paris    | Detroit     | KLM               | 7    |
| Paris    | New York    | KLM               | 6    |
| Paris    | Boston      | American Airlines | 8    |
| New York | Chicago     | American Airlines | 2    |
| Boston   | Chicago     | American Airlines | 6    |
| Detroit  | San Jose    | American Airlines | 4    |
| Chicago  | San Jose    | American Airlines | 2    |

**Iterations Table:**

| Iteration # | Tuples in Closure                            |
|-------------|----------------------------------------------|
| 0           | Detroit, New York, Boston                    |
| 1           | Detroit, New York, Boston, San Jose, Chicago |
| 2           | Detroit, New York, Boston, San Jose, Chicago |

At the bottom of the slide, there is a footer with the text "Database System Concepts - 8<sup>th</sup> Edition", "09.31", and "©Silberschatz, Korth and Sudarshan".

So, now, if we go and this is the instance and this you can. Here, I have shown that how the iteration actually happens in the iteration 0, in the flights itself, you had three destinations, then you add two more in iteration 1, in iteration 2, you do not add anything else. So, your result henceforth will not change. So, you have reached a fixed point and computations, are over.

(Refer Slide Time: 30:10)

The slide has a header 'Update of a View' with a sailboat icon. The main content is a bulleted list:

- Add a new tuple to *faculty* view which we defined earlier

SQL code:

```
insert into faculty values ('30765', 'Green', 'Music');
```

This insertion must be represented by the insertion of the tuple

```
('30765', 'Green', 'Music', null)
```

into the *instructor* relation

At the bottom left is a video frame showing a man speaking, with text 'SWAYAM: NPTEL-NOCs Instructor: Prof. P. P. Deshpande'. At the bottom right is a navigation bar with icons and the text 'Database System Concepts - 8<sup>th</sup> Edition' and '09.32'.

You can also update a view you can insert a record directly into a view, but since view only is partial information on the relation, when you INSERT INTO a view since, view is virtual. There will have to be an insertion, in the real relation and in the real relation you may not know certain fields. So, if you are doing this insertion into faculty, which is a view of instructor, then the salary field is not known. So, in the actual instructor a null will have to get inserted in the salary field. So, the salary field needs to be null able kind of field so updates on views have certain restrictions.

(Refer Slide Time: 30:47)

**Some Updates cannot be Translated Uniquely**

- ```
create view instructor_info as
    select ID, name, building
    from instructor, department
    where instructor.dept_name= department.dept_name;
```
- ```
insert into instructor_info values ('69987', 'White', 'Taylor');
```

  - which department, if multiple departments in Taylor?
  - what if no department is in Taylor?
- Most SQL implementations allow updates only on simple views
  - The **from** clause has only one database relation
  - The **select** clause contains only attribute names of the relation, and does not have any expressions, aggregates, or **distinct** specification
  - Any attribute not listed in the **select** clause can be set to null
  - The query does not have a **group by** or **having** clause

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jam-Aut., 2018  
Database System Concepts - 8<sup>th</sup> Edition  
09.33  
©Silberschatz, Korth and Sudarshan

So, there are some more instances that I have given, which you can study and try to understand that what are the difficulties of updating on the view. So, it can be done, but it has to be done in a restrictive sense. So, these are the different conditions that has to happen for views to be updated.

(Refer Slide Time: 31:08)

**And Some Not at All**

```
create view history_instructors as
select *
from instructor
where dept_name= 'History';
```

- What happens if we insert ('25566', 'Brown', 'Biology', 100000) into *history\_instructors*?

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jam-Aut., 2018  
Database System Concepts - 8<sup>th</sup> Edition  
09.34  
©Silberschatz, Korth and Sudarshan

(Refer Slide Time: 31:11)



## Materialized Views

- **Materializing a view:** create a physical table containing all the tuples in the result of the query defining the view
- If relations used in the query are updated, the materialized view result becomes out of date
  - Need to **Maintain** the view, by updating the view whenever the underlying relations are updated

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr - 2018


Database System Concepts - 8<sup>th</sup> Edition
09.35
©Silberschatz, Korth and Sudarshan

So, please go through these slides to understand what are there in terms of the views? Finally, view is a virtual relation, but it can be materialized also, that is materializing is basically computing a physical relation at the instance of the view, but naturally if you materialized then there is a certain point of time, where you have materialized where you have made it into a physical relation and hence, if your original source data in the view changes in future the materialized view also need to be updated otherwise, your data will get bad.

(Refer Slide Time: 31:43)



PPD

- Join Expressions
- Views
- Transactions

## TRANSACTIONS


Database System Concepts - 8<sup>th</sup> Edition
09.36
©Silberschatz, Korth and Sudarshan

(Refer Slide Time: 31:49)

**Transactions**

- Unit of work
- Atomic transaction
  - either fully executed or rolled back as if it never occurred
- Isolation from concurrent transactions
- Transactions begin implicitly
  - Ended by **commit work** or **rollback work**
- But default on most databases: each SQL statement commits automatically
  - Can turn off auto commit for a session (e.g. using API)
  - In SQL:1999, can use: **begin atomic .... end**
  - Not supported on most databases

SWAYAM: NPTEL-NOOC Instructor: Prof. P. Desai, IIT Kharagpur - Jan-Apr - 2015

Database System Concepts - 8<sup>th</sup> Edition 09.37 ©Silberschatz, Korth and Sudarshan

Finally, in this module, we mentioned that, there is something called transactions, which we will take up at a later stage, in much depth. This is just to get you familiar with atom a transaction is a unit of work, which is usually atomic, which is either fully executed or if it fails, it will be rolled back as if it never occurred and this is required for isolation in concurrent transactions. So, we will talk about this lot more when we take up concurrency and related issues.

So, come transactions implicitly begin and they end by either committing the work that they have successfully finished or rolling back that, this cannot be done. So, there are some features in the SQL for doing transactions and, but usually you can transactions, commit by default and the only it is exceptions, when the rollback is happening and we will see more of that later.

(Refer Slide Time: 32:49)

Module Summary

- Learned SQL expressions for Join and Views
- Introduced transactions

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

09.38

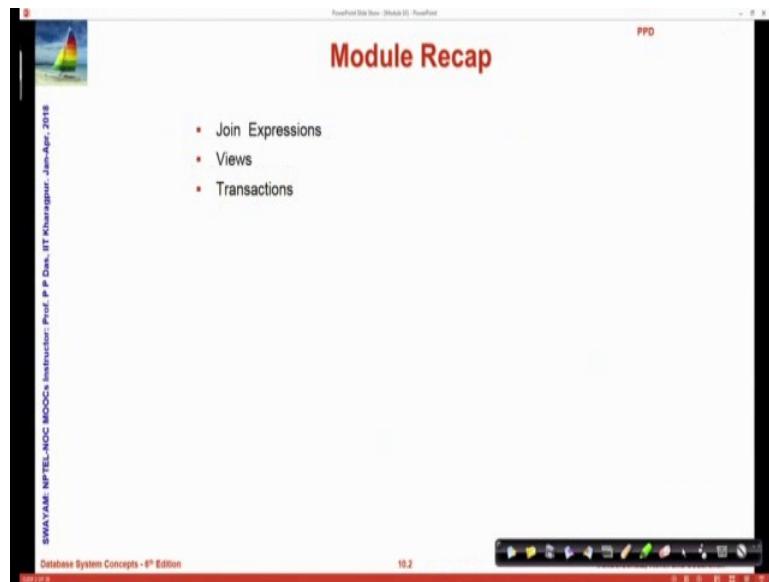
©Silberschatz, Korth and Sudarshan

So, to summarize in this module, we have learnt about two important SQL features in terms of join and views and we just introduced the basic notion of committing transactions.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 10**  
**Intermediate SQL/2**

(Refer Slide Time: 07:37)



Module Recap

- Join Expressions
- Views
- Transactions

SMAV YAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - June-April - 2018  
Database System Concepts - 6<sup>th</sup> Edition

Welcome to module ten of database management systems. We have been discussing about intermediate level features in SQL; and this is a second and closing module on that. We have in the last module; talked about join expressions, views and transaction in a bit.

(Refer Slide Time: 07:46)

The slide is titled "Module Objectives" in a large, bold, red font at the top center. Below the title is a bulleted list of three items, also in red font. The footer of the slide includes a red bar with the text "Database System Concepts - 6<sup>th</sup> Edition" and "103". On the left side of the slide, there is a vertical sidebar with some text and a logo.

- To learn SQL expressions for integrity constraints
- To understand more data types in SQL
- To understand authorization in SQL

In this module, we will try to learn SQL expressions that are responsible for maintaining in the integrity of the database. We have talked about integrity a little. We will now see how explicitly integrity can be checked and how different kinds of integrity can be ensured through SQL. We will also talk about more data types.

We have seen the basic primitive data types, and we had promised that we will talk more about the data types including user defined data types here. And finally, we will talk about a very important aspect of authorisation as to who can do what in a database in through SQL.

(Refer Slide Time: 08:27)

The screenshot shows a Microsoft PowerPoint slide titled "Module Outline". The slide content is as follows:

- Integrity Constraints
- SQL Data Types and Schemas
- Authorization

At the bottom left of the slide, there is vertical text: "SWAYAM: NPTEL-NOC: MOOCs Instructor: Prof. P. P. Chakrabarti - IIT Kharagpur - Jan-Apr - 2016". At the bottom center, it says "Database System Concepts - 6<sup>th</sup> Edition". The bottom right corner shows the slide number "10.4". The top right corner has the word "PPD". The top bar of the slide indicates "PowerPoint Slide Show - Module 10 - PowerPoint".

So, this is a module out line.

(Refer Slide Time: 08:30)

The screenshot shows a Microsoft PowerPoint slide with the title "INTEGRITY CONSTRAINTS" in large red capital letters. Below the title, the same list of topics from the previous slide is displayed:

- Integrity Constraints
- SQL Data Types and Schemas
- Authorization

At the bottom left of the slide, there is vertical text: "SWAYAM: NPTEL-NOC: MOOCs Instructor: Prof. P. P. Chakrabarti - IIT Kharagpur - Jan-Apr - 2016". At the bottom center, it says "Database System Concepts - 6<sup>th</sup> Edition". The bottom right corner shows the slide number "10.5". The top right corner has the word "PPD". The top bar of the slide indicates "PowerPoint Slide Show - Module 10 - PowerPoint".

We start with integrity constraints.

(Refer Slide Time: 08:31)

The slide is titled "Integrity Constraints" in red. It contains a bulleted list of integrity constraints:

- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency
  - A checking account must have a balance greater than \$10,000.00
  - A salary of a bank employee must be at least \$4.00 an hour
  - A customer must have a (non-null) phone number

On the left margin, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Ch. BT Kumar aggrawal - Jatin Agarwal". At the bottom, it says "Database System Concepts - 8<sup>th</sup> Edition" and "10.6".

Integrity constraints guard against accidental damage to the database that is there are certain real world facts that must be ensured in the database all the time. For example, in bank accounts, we have a minimum balance that need to be maintained, for a particular customer we might want that the customer's phone number must be present. We may have certain age bar in terms of entering into certain memberships or certain employment and so on. All these kinds of real world constraints need to be represented and maintained in the database and that is the purpose of the integrity constraint.

(Refer Slide Time: 09:16)

The slide is titled "Integrity Constraints on a Single Relation" in red. It contains a bulleted list of constraints:

- not null
- primary key
- unique
- check (P), where P is a predicate

On the left margin, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Ch. BT Kumar aggrawal - Jatin Agarwal". At the bottom, it says "Database System Concepts - 8<sup>th</sup> Edition" and "10.7".

And we will first look at the issues of integrity constraint for a single relation, and we have seen the use of NOT NULL and PRIMARY KEY. We will also talk about what is UNIQUE and we will see how actually general constraints can be checked in terms of a CHECK clause with a predicate P.

(Refer Slide Time: 09:43)

The slide has a title 'Not Null and Unique Constraints' in red. It contains two main sections:

- not null**
  - Declare *name* and *budget* to be **not null**
  - name* *varchar(20)* **not null**
  - budget* *numeric(12,2)* **not null**
- unique ( $A_1, A_2, \dots, A_m$ )**
  - The unique specification states that the attributes  $A_1, A_2, \dots, A_m$  form a candidate key
  - Candidate keys are permitted to be null (in contrast to primary keys).

On the right side of the slide, there is a hand-drawn diagram of a table with rows labeled  $t_1$  and  $t_2$ . The columns are labeled  $A_1, A_2, \dots, A_m$ . The first column  $A_1$  has a checkmark in the  $t_1$  row and a question mark in the  $t_2$  row. The second column  $A_2$  has a question mark in the  $t_1$  row and a checkmark in the  $t_2$  row. This illustrates that both rows must differ in at least one attribute value for them to be unique.

The bottom left corner shows a video player window with a man speaking, and the bottom right corner shows the slide number 10.8.

So, NOT NULL. we had seen this before we can while creating the database table. Create table we can specify a field to be NOT NULL and then in that case null values will not be allowed in those fields. So, they will must be some value given. You can say one or more attributes to be UNIQUE. If you specify them to be UNIQUE; that means, that in any instance of the table in future that cannot be two tuples which match in all of those attribute.

So, if you say  $A_1$  to  $A_m$  are UNIQUE; that means, that if we have two different tuples in the table anytime in future. Two different rows in the table  $t_1$  and  $t_2$  then across  $A_1, A_2, A_m$  they must differ in at least one attribute value. So, uniqueness is a basic requirement for being a CANDIDATE KEY, but they are, but still permitted to be null which in contrast to what is the true for PRIMARY KEY already you know that PRIMARY KEY values cannot be null, but uniqueness allows a null value, but they have to be different.

(Refer Slide Time: 10:58)

The slide is titled "The check clause". It contains the following text and code:

- check (P)  
where P is a predicate

Example: ensure that semester is one of fall, winter, spring or summer:

```
create table section (
    course_id varchar (8),
    sec_id varchar (8),
    semester varchar (6),
    year numeric (4,0),
    building varchar (15),
    room_number varchar (7),
    time_slot_id varchar (4),
    primary key (course_id, sec_id, semester, year),
    check (semester in ('Fall', 'Winter', 'Spring', 'Summer'))
);
```

A blue arrow points from the word "P" in the first bullet point to the "check" keyword in the SQL code.

The CHECK clause is where you say CHECK and then you put a predicate. So, the idea is like this, suppose I know that I have specified an attribute called semester. It is a varchar 6 which means that it can have a string maximum of length 6, but naturally I can write anything there I can write morning in that field. I can write welcome in that field and so on, but those are not valid names of semester. So, I want that in my design semester must have any of these values only.

So, we say semester in. So, I have listed the values that are allowed in is a set membership. So, it says the semester in so which means the value of the semester is be one of this fall. And this whole thing now becomes the predicate P on which I give a check which means that, whenever I am creating once, you have created the table. when I want to insert or update the values in the table, the records in the table; the value of semester has to be always within this. Otherwise, the CHECK integrity constraint will fail, and the update or insert will not happen and an exception will be raised. This is the basic idea of CHECK constraints.

(Refer Slide Time: 12:25)

The slide is titled "Referential Integrity". It includes the following text and a diagram:

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation
- Example: If "Biology" is a department name appearing in one of the tuples in the *instructor* relation, then there exists a tuple in the *department* relation for "Biology"

Hand-drawn diagram illustrating referential integrity:

- A diagram showing two tables: "Instructor" and "Department".
- The "Instructor" table has columns labeled "InstID" and "DeptName".
- The "Department" table has columns labeled "DeptID" and "DeptName".
- A blue arrow points from the "DeptName" column in the "Instructor" table to the "DeptName" column in the "Department" table.
- A handwritten note "InstID" is written above the first column of the "Instructor" table.
- A handwritten note "DeptName" is written above the second column of both the "Instructor" and "Department" tables.
- A handwritten note "DeptID" is written above the first column of the "Department" table.

Now, let us move on to more involved integrity CHECK which goes beyond one table. So, let us suppose that we are talking about the instructor table we have the instructor table. An instructor table has a department name. Similarly, we have a department table which naturally has a department name. Now, we know that this is the key in the department table and therefore, here it is the foreign key.

Now, while we are inserting records in the instructor table how do we guarantee that the record that we insert has a corresponding entry in the foreign key table that is difference table. So, it is when we are inserting and in a faculty in the saying that the faculty belongs to biology department there needs to be a biology entry in the department table as well. So, this is known as a referential integrity that is once you refer from one table to the other that reference must also be a valid one; otherwise, all your computations will go wrong.

(Refer Slide Time: 13:48)

The slide has a title 'Referential Integrity' in red. Below it is a bulleted list:

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation
  - Example: If "Biology" is a department name appearing in one of the tuples in the *instructor* relation, then there exists a tuple in the *department* relation for "Biology"
- Let A be a set of attributes. Let R and S be two relations that contain attributes A and where A is the primary key of S. A is said to be a **foreign key** of R if for any values of A appearing in R these values also appear in S

Navigation icons and a progress bar are visible at the bottom.

So, this is a for saying it, formally there are two relations and one relation as a PRIMARY KEY which is used in the other relation as a foreign key then there is a referential integrity that needs to be maintained.

(Refer Slide Time: 14:02)

The slide has a title 'Cascading Actions in Referential Integrity'. Below it is a bulleted list of SQL code snippets:

- ```
* create table course (
    course_id char(5) primary key,
    title      varchar(20),
    dept_name varchar(20) references department
)
```
- ```
* create table course (
    ...
    dept_name varchar(20),
    foreign key (dept_name) references department
        on delete cascade
        on update cascade,
    ...
)
```
- alternative actions to cascade: **no action, set null, set default**

Navigation icons and a progress bar are visible at the bottom.

So, here we are just showing the effect of that. So, we have created a table this is the first one is what you have seen earlier creating the table course and that table course needs the name of the department. So, we are specifying that it references the department table. Now, if it references the department table, it must ensure the referential integrity. So, this

just says that this refers to the department table, but I can be more specific to say what will happen if the integrity gets violated. For example, I have created this and the course table has an entry, which has a department name say biology. Naturally, biology department should have the entry in the department table with this department name biology for this to be valid.

Now, say for some reason the biology department is abolished and that particular record from the department table is removed, naturally, the course which is referring to biology in terms of its department name that particular record will become invalid. So, we can say that on delete, what you should be doing, one most common action that we specify in referential integrity is cascade that if the referred entity is deleted then the referring entity should also be deleted. So, if you delete the biology entry from the department table, then all courses which have biology through references to department as their field value should also get deleted.

Similar thing can be there on update also. For example, biology department say tomorrow changes the name to bioscience. Now, if I have a referential integrity put on the course table as on update cascade, then as I change the bioscience the name to bioscience. All records in the table course, which had the department name as biology will necessarily get updated. So, this is the way to maintain referential integrity. Cascading is one of the most common way to handle this, but there could be other ways to take action also that could be no action that you say ok, I do not care. Let that happen in that case, because of the violation that could be some exceptions thrown or even say that if this happens then I will set that field to null or I will set that field to some default value and so on.

(Refer Slide Time: 17:01)

The slide is titled "Integrity Constraint Violation During Transactions". It contains the following SQL code:

```
create table person (
ID char(10),
name char(40),
mother char(10),
father char(10),
primary key ID,
foreign key father references person,
foreign key mother references person)
```

Below the code is a bulleted list:

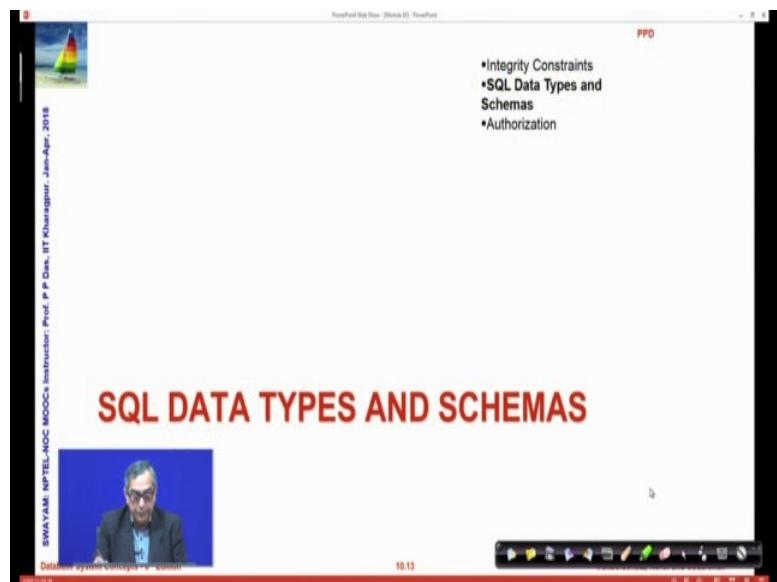
- How to insert a tuple without causing constraint violation ?
  - insert father and mother of a person before inserting person
  - OR, set father and mother to null initially, update after inserting all persons (not possible if father and mother attributes declared to be **not null**)
  - OR defer constraint checking (will discuss later)

At the bottom left, it says "DRAFT: NPTEL-NOCO-MOCOCs Instructional IP-Draft, IIT-Kharagpur - Jain-Agarwal" and "Database System Concepts - 6th Edition". At the bottom right, it shows "10.12" and some navigation icons.

So, this is how the referential integrity has to be handled. There could be integrity violation during transactions also. This is an example of a self-referential table which of persons which where every person's entry needs the name of the mother and the father which are also entries in this table. So, necessarily if you are entering a person record you need this feels to be populated and that can be populated only if those records already exists.

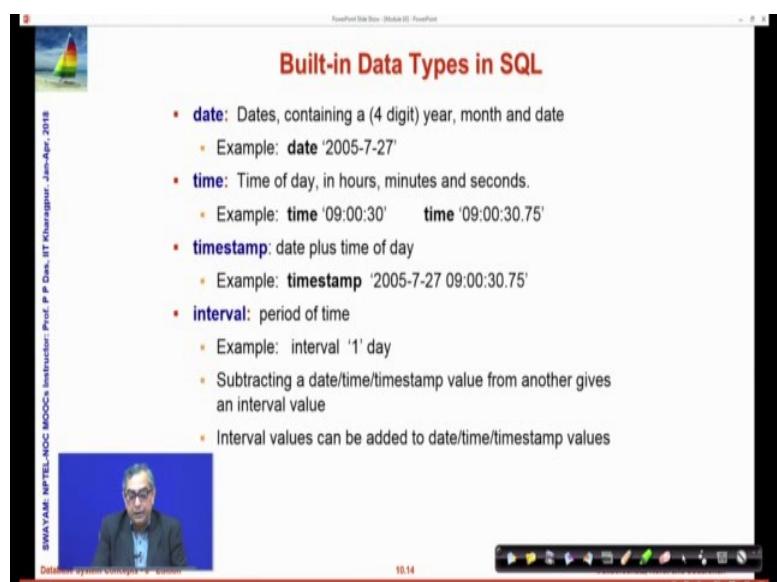
So, there is some order in which you have to enter the records or you have to set them as null and then update them in future and or some ways to say that well do not check this integrity now. We will talk about this integrity at a later point of time. So, these are the issues that necessarily you will have to be addressed.

(Refer Slide Time: 17:54)



Let us move on and look at the SQL data types and schemas.

(Refer Slide Time: 18:02)



So, in addition to the data types like char, varchar, int and all that you have an explicit date data type which gives you a year, month, date kind of format with a four-digit year. because date is very frequently required, you have a time type to give you hour, minute, second time format. You have a timestamp, which is date and time together and you have what is known as interval where you can do a date or time difference between two different dates, two different time, two different time stamps and so on. So, these are the

common added built in types which makes it very easy to handle the temporal aspects in SQL queries.

(Refer Slide Time: 18:54)

Index Creation

```
create table student  
(ID varchar(5),  
name varchar(20) not null,  
dept_name varchar(20),  
tot_cred numeric(3,0) default 0,  
primary key (ID))
```

- create index studentID\_index on student(ID)
- Indices are data structures used to speed up access to records with specified values for index attributes

```
select *  
from student  
where ID = '12345'
```

- can be executed by using the index to find the required record, without looking at all records of student
- More on indices in Chapter 11

Database System Concepts - 6th Edition

In addition, the next that you can do is you can create an index. So, let us look at this. So, this create table definition you understand well by now. You can do this, I can say create index and give a name for the index and specify which field on which the index should be created. So, here we are saying that the index should happen here. This is the name of the relation; this is the name of the attribute name of the field. Now, this does not change any data neither does it change any schema, but it creates certain additional structure so that it becomes easier to search this particular table using IDs.

So, if I have a query like this that I am trying to find out all information about a particular student then as we have said that by default the different entries the rows of a relation are unordered. So, the only way to find out this particular row and in fact, whether it actually exist would be to go over all the relations one by one. But if we index it, then it creates some kind of an efficient data structure through which it can be searched out very efficiently very easily. with a later module we will talk about indexing. But just to give you the idea that this is similar to finding out a value in an unordered array if you are thinking of C.

In contrast, we all know that this can be done, but takes a whole lot of time it takes order in time. But I could keep those numbers in terms of some binary search tree, balanced

binary search tree like red black tree or two three four tree kind of, where the search can be conducted in a login time. Or I could keep it in terms of some efficient hashing mechanism where the search could happen in terms of an ordered one time also. So, indexing has a lot of importance and we will talk about that more, but this is how you create index in SQL.

(Refer Slide Time: 21:18)

User-Defined Types

- **create type** construct in SQL creates user-defined type

```
create type Dollars as numeric (12,2) final
```

- **create table department**  
(dept\_name varchar (20),  
building varchar (15),  
budget Dollars);

You can have user defined types, you can say create type and use some specific. you know sub types of a type as and give it a name. So, you send numeric 12, 2. So, which is the 12 digit number with two decimal places of precision. you can call it dollar and then use that as a type name. So, type name doing this helps in to visit make sure that wherever you actually have to conceptually refer to dollars you are talking about dollars it is easier to understand and you are making sure that everywhere the same numeric precision is used.

(Refer Slide Time: 22:02)

The screenshot shows a PowerPoint slide with the title 'Domains'. The slide contains the following text:

- **create domain** construct in SQL-92 creates user-defined domain types

```
create domain person_name char(20) not null
```

- Types and domains are similar
- Domains can have constraints, such as **not null**, specified on them

```
create domain degree_level varchar(10)
constraint degree_level_test
check (value in ('Bachelors', 'Masters', 'Doctorate'));
```

A video player window is visible at the bottom left, showing a man speaking. The video player interface includes a progress bar and a timestamp of 10.17.

You can also actually go further and create domains which is very similar to create type. But domains are more powerful in the sense that, in a domain, you can also add constraints like NOT NULL and you say that this is person name, say that this once you have set that this person name is 20 characters long and it cannot be null then you do not specifically have to every time.

You define a field based on this domain type you do not have to specifically say that it is NOT NULL. You could also create a specific constraint in terms of the CHECK clause and make it easier. So, now if you say degree level you do not have to put CHECK clause explicitly in the SQL query, because it is already specified in the created domain.

(Refer Slide Time: 22:52)

The slide is titled "Large-Object Types" in red. It contains a bulleted list:

- Large objects (photos, videos, CAD files, etc.) are stored as a *large object*.
  - **blob**: binary large object -- object is a large collection of uninterpreted binary data (whose interpretation is left to an application outside of the database system)
  - **clob**: character large object -- object is a large collection of character data
  - When a query returns a large object, a pointer is returned rather than the large object itself

Navigation icons at the bottom include back, forward, and search symbols.

SQL supports certain large-objects which are either called blobs if they are binary, or called clob if they are character objects. The only the major difference in terms of the large object types are they are not stored as a part of the table. They stored elsewhere and you actually maintain a kind of a reference a pointer to that large object. So, this is very useful in terms of handling photos, videos and no big binary files, character files also.

(Refer Slide Time: 23:21)

The slide has a red header "PPD". Below it, there is a bulleted list:

- Integrity Constraints
- SQL Data Types and Schemas
- Authorization

The main title "AUTHORIZATION" is displayed in large red capital letters. Navigation icons at the bottom include back, forward, and search symbols.

Let us move to authorization.

(Refer Slide Time: 23:29)

The screenshot shows a PowerPoint slide with the title 'Authorization' in red at the top center. The slide content is organized into two main bullet points:

- Forms of authorization on parts of the database:
  - **Read** - allows reading, but not modification of data
  - **Insert** - allows insertion of new data, but not modification of existing data
  - **Update** - allows modification, but not deletion of data
  - **Delete** - allows deletion of data
- Forms of authorization to modify the database schema
  - **Index** - allows creation and deletion of indices
  - **Resources** - allows creation of new relations
  - **Alteration** - allows addition or deletion of attributes in a relation
  - **Drop** - allows deletion of relations

Handwritten notes are present on the right side of the slide:

- A bracket groups the first four items under 'Read', 'Insert', 'Update', and 'Delete' with the handwritten note 'DBA Prof.'.
- A bracket groups the last three items under 'Index', 'Resources', 'Alteration', and 'Drop' with the handwritten note 'Analysts'.

Other slide details include:

- Left margin: 'SHIVAYAM: NPTEL MOOC Instructor: Prof. P. Das, IIT Kharagpur - Jatin-Das'
- Bottom left: 'Database System Concepts - 4. Control'
- Bottom center: '10.20'
- Bottom right: Standard Mac OS X window controls

Next, authorization is the process by which you restrict different users to be able to do different kind of operations. You recall in the early modules on database overview. We mentioned that there could be several types of users for a database. There could be absolutely application users who may necessarily do not feature as a part of the database development. But there could be application developers expectedly most of you would become application developers or there could be intermediate higher level of analyst who design databases, design constraints, decide on indices and so on and that could be database administrator.

And also in terms of different application programs and programmers there is a need to separate out who can access which part of the database. For example, if you look at a add a banking system, then while I am, by net banking application is accessing different information about my account, one part I need to ensure that I can only access my account.

And also what the database system needs to ensure is that a net banking application should in no way be able to access the information about the specific employees, because, in the same database information about the bank employees will also be there. It should not be possible for possible to access information about different physical information about the branches as to where, how many square feet of area that branch has and so on and so forth.

So, we need to put variety of restrictions and as we will see that authorisation or this process of restricting or allowing different access and different authority to operate is decided based on two different factors. One is what you want to do, and two is who wants to do that. So, what and who, so we identify different operations or different operations on certain tables or operations on certain attributes as what needs to be done. And on the other side, we will identify who in terms of specific individual user IDs or groups of user IDs or roles that exist.

So, here we will just try to show you how we can do that in SQL. So, the first part of the authorisation is being able to do different things with the database that means, the instances of the database. So, there are authorisations to read insert, update and delete. So, read is where you can access the data, but you cannot modify; insert is when you can add new data, but you do not insert rights authorisation, you cannot update an existing data, you can only insert data. You can have update rights, where we can change make modifications, but you may not be, you are not allowed to delete data, and you can that would be delete right where it allows you to delete data and my new this authorisations or not these are all independent authorisation.

So, you may have I mean certain authorisations may need certain other authorisations to be present. For example, if you are updating naturally evenly to read, but it is these are all independent authorisations, and you may have one or more of them to be able to do the appropriate actions. Similar set of another set of authorisations will exist, if you want to for those want to modify the database schema, naturally, this is primarily for the applications and application programmers, and this primarily would be for the analysts that you can index the different table you can do.

You can have authorisation for resources which mean you can create new relations, create new schemas, you can alter schemas, you can drop schemas and so on. So, these are the different kinds of authorisation that are possible.

(Refer Slide Time: 28:01)

The screenshot shows a PowerPoint slide with the title 'Authorization Specification in SQL' in red at the top. Below the title is a bulleted list of points about the GRANT statement:

- The **grant** statement is used to confer authorization
  - grant <privilege list>
  - on <relation name or view name> to <user list>
- <user list> is:
  - a user-id
  - **public**, which allows all valid users the privilege granted
  - A role (more on this later)
- Granting a privilege on a view does not imply granting any privileges on the underlying relations
- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator)

At the bottom left of the slide, there is a small video thumbnail showing a man speaking, with the text 'Database System - Lecture 8' and 'Dr. S. Ravi'. The slide has a footer with the text 'SWANAMI\_NPTEL\_ANOCA\_Instructional\_Protocol\_PPT\_2015' and '10.21'.

So, let us see how it works the authorisation is specified in terms of a statement called grant. So, you grant an authorisation to a privilege list and on certain relation to a group of users. So, grant what kind of authorisation you are granting that is the privilege less list on what relation on view you are granting that is on condition and to whom are you granting those. So, user list could be a specific user ID or you could say public which in this case everybody will have that or this could be a role which will see, what a role is.

Granting a privilege on a view does not imply granting any privileges on the underline relation, please mind this one, because, you have seen that a view can be formed from multiple different relations. So, if somebody has been granted a particular privilege say read privilege on a view, then it does not mean that the corresponding underline relationship, you been granted a read privilege on faculty relation that we faculty view that we did that does not mean that the user will automatically get a read privilege on the underline in structure relation. So, that has to be kept in mind.

The grantor of the privilege must already hold the privilege that is you cannot grant. Naturally, grant will be done also by somebody in some of the users you may be, so that user who is granting must also have the privilege, same privilege on the specific item. So, you cannot grant privilege on something, some relation or view on which you yourself do not have that or it has to be the database administrative who naturally has privilege for everything.

(Refer Slide Time: 30:13)

The screenshot shows a PowerPoint slide with the title 'Privileges in SQL' in red at the top center. Below the title is a bulleted list of SQL privileges:

- **select**: allows read access to relation, or the ability to query using the view
  - Example: grant users  $U_1$ ,  $U_2$ , and  $U_3$  **select** authorization on the *instructor* relation:  
`grant select on instructor to  $U_1$ ,  $U_2$ ,  $U_3$`
- **insert**: the ability to insert tuples
- **update**: the ability to update using the SQL update statement
- **delete**: the ability to delete tuples.
- **all privileges**: used as a short form for all the allowable privileges

At the bottom left of the slide, there is a small video player showing a man speaking, with the text 'SWANAMI\_NPTEL\_ANOCC\_MOCOCs\_Instructor: Prof. P. Das, IIT Kharagpur' and 'Database system (University of Calicut)'. The video player has a timestamp of '10.22'.

The privileges on SQL there is a SELECT privilege, which is so you know this is the privilege list. SELECT privilege which basically means read access. So, it is saying SELECT privileges is read access on instructor and these are the different users. So, this is how typically, you can have insert to ability to insert tuples, the update privilege this should be easy to understand now delete privilege.

So, only thing is read is a called SELECT here, so these are the different privileges that you have in a SQL. And you have one all in compassing privilege which is called all privileges so as a short form of allowing all these allowable privileges.

(Refer Slide Time: 31:00)

The screenshot shows a PowerPoint slide with the title 'Revoking Authorization in SQL' in red at the top center. Below the title is a bulleted list of points about the REVOKE statement. The slide has a footer with the text 'Database System Concepts - 6th Edition' and '10.23'. On the left side, there is vertical text that reads 'DATA MAINTAINABILITY: NPTEL-NOCOCS Instructional IP: Dr. ET Khurana - Jain-Agarwal - 2015'. The slide is part of a presentation titled 'Module 10 - PowerPoint'.

- The **revoke** statement is used to revoke authorization  
**revoke <privilege list>**  
**on <relation name or view name> from <user list>**
- Example:  
**revoke select on branch from U<sub>1</sub>, U<sub>2</sub>, U<sub>3</sub>**
- <privilege-list> may be all to revoke all privileges the revoker may hold
- If <revoker-list> includes **public**, all users lose the privilege except those granted it explicitly
- If the same privilege was granted twice to the same user by different grantors, the user may retain the privilege after the revocation
- All privileges that depend on the privilege being revoked are also revoked

Certainly, if you can grant an authorization, then needs to be a reverse process that is if we want to withdraw authorization of certain privileges on certain items for certain users, so that is known as a REVOKE statement. So, we can revoke it the structure looks exactly similar to the grant. So, you REVOKE a privilege list, SELECT, insert this kind of on certain relation and view from a set of users. So, you can say that REVOKE SELECT ON branch from this. So, once this is done, then U<sub>1</sub>, U<sub>2</sub> and U<sub>3</sub> will not be able to read the branch relation or the branch view. The privilege list may be all to REVOKE all privileges. So, instead of revoking SELECT, insert separately, you can just say all and revoke all of that.

The list of revoking can include public which means all users lose that privilege and or though those were granted. If the same privileges granted twice, now it is possible that a user gets privilege granted to him or her by two different granting authorities, then if ones is one of them is revoked the other will still continue to remain; So, every privilege that is granted needs to be explicitly revoked that is a basic meaning.

So, all privileges that depend on the privilege being revoked are also revoked. So, some privilege which is dependent on some other privilege, if you REVOKE the update privilege then this SELECT privilege will remain. But, if you REVOKE the SELECT privilege then if you also had the update privilege that will certainly get revoked, because if you cannot read then naturally you cannot change.

(Refer Slide Time: 33:01)

The slide is titled "Roles". It contains the following bulleted list:

- `create role instructor;`  
    `grant instructor to Amit;`
- Privileges can be granted to roles:  
    `grant select on takes to instructor;`
- Roles can be granted to users, as well as to other roles  
    `create role teaching_assistant`  
    `grant teaching_assistant to instructor;`  
        • Instructor inherits all privileges of *teaching\_assistant*
- Chain of roles  
    `create role dean;`  
    `grant instructor to dean;`  
    `grant dean to Satoshi;`

The SQL also allows you to create certain roles. Roles are kind of like virtual use that so, we all say that we all play certain role. So, I have an entity as an individual say I may be user called PPD, but I have a role as an instructor, I have a role as say the head of the department, I have a role as a chairman of committee and so on. So, often times it becomes easier to grant privileges to different roles.

You do not really care immediately about who that individual could be who that particular user could be who has that privilege, whoever plays that role gets that privilege, whoever becomes the director of IIT Kharagpur has the privilege to appoint faculty members it is of that kind. It does not specifically. So, role is of that kind of a concept. So, you CREATE ROLE we are saying that role is the role instructor is created.

And then you are saying that you grant instructor to Amit which says that Amit now plays the role of instructor. So, any privilege that the instructor role has Amit will enjoy that. So, let us see more of this the privileges can be grant to roles. So, earlier we said it could be public, it could be users, but now you are saying that it could be two roles. So, here this role was creates and the privilege is being granted to that. And since Amit plays that role it will mean that Amit with this GRANT SELECT ON takes to instructor, Amit will actually get a privilege of SELECT on takes relation that is the kind of derived structure that roles give you roles can be granted to users as well as to other roles.

So, roles are becoming like virtual users. So, you can CREATE ROLE teaching assistant and grant teaching assistant to instructor, which means that if you do that you are granting this. So, it means that any privilege the teaching assistant will have, the instructor will get those privileges, because, you have made in instructor to also play the role teaching assistant mind you instructor itself is virtual entity. So, if Amit is an instructor by this, then Amit plays this role and this role plays teaching assistant role and this teaching assistant role has certain privileges, so naturally through this chain process Amit will get those privileges. So, in an instructor inherits all privileges of teaching assistant.

So, this is what exactly, what I was talking of you can have a chain of roles create a role dean new one, you have created, then grant instructor to dean grant dean to Satoshi. So, which means; that once you grant dean to instructor; so anybody who plays the dean's role will get all privileges of instructor. Here you are saying that Satoshi is going to play the dean role. So, the Satoshi in terms of chaining gets all the privileges that instructor has.

(Refer Slide Time: 36:41)

**Authorization on Views**

```

create view geo_instructor as
(select *
from instructor
where dept_name = 'Geology');
grant select on geo_instructor to geo_staff

```

- Suppose that a geo\_staff member issues

```

select *
from geo_instructor;

```

- What if
  - geo\_staff does not have permissions on instructor?
  - creator of view did not have some permissions on instructor?

So, once this has been done, then you can have authorization on views as well. So, you have created a view here. So, this is the view created the geo instructor the Geology instructor. And on that view particularly you have given the privilege to the geo staff. So, a geo staff member would be able to access this view. And if this query is fired by a geo

staff member, which I am assuming is a role then this view will get executed and the results of all instructors in the department geology will be update. But, what if the geo staff does not have permission on instructor, does not matter that is the beauty of the whole thing. The geo staff may not have permission to do SELECT on instructor, but the geo staff has permission to SELECT on geo instructor. So, the geo staff will be able to execute this view, but the geo staff will not be able to do a SELECT on from the instructor database instructor table.

(Refer Slide Time: 38:16)

The slide is titled "Other Authorization Features\*". It lists the following points:

- references privilege to create foreign key
  - grant reference (*dept\_name*) on *department* to Mariano;
  - why is this required?
- transfer of privileges
  - grant select on *department* to Amit with grant option;
  - revoke select on *department* from Amit, Satoshi cascade;
  - revoke select on *department* from Amit, Satoshi restrict;

There are several other authorization features, the references a privilege to create foreign key. So, we talked about basic read, write, data manipulation privileges, but there could be other privileges like whether you can create a foreign key, whether you can transfer of privilege. So, whether you can give one privilege to another, so whether you can cascade, whether you can restrict and so on.

So, transfer of privileges is also privilege. I am naturally will have to think of this is an authorization. So, actually what we can authorize is also a privilege that needs to be authorized. So, these are the derived privileges that I have.

(Refer Slide Time: 39:00)

The screenshot shows a Microsoft PowerPoint slide titled "Module Summary". The slide content includes a bulleted list of three items:

- Learnt SQL expressions for integrity constraints
- Familiarized with more data types in SQL
- Discussed authorization in SQL

At the bottom left of the slide, there is a vertical footer text: "DATA MAINTAINABILITY: INTEGRITY CONCEPTS", "Database System Concepts - 6<sup>th</sup> Edition", and "Prof. Dr. P. Gajjar, IIT Kharagpur". At the bottom right, there is a slide number "10.27". The top of the slide has a navigation bar with icons for back, forward, and search.

So, in summary, we have learnt about SQL expressions to deal with integrity constraints. We are familiarized with more data types particularly user defined types and domains creation of index and we have discussed about authorization in SQL. Each one of them particularly authorization has lot more details, but at this intermediate level we just wanted to get a basic idea about authorization to be able to deal with that.

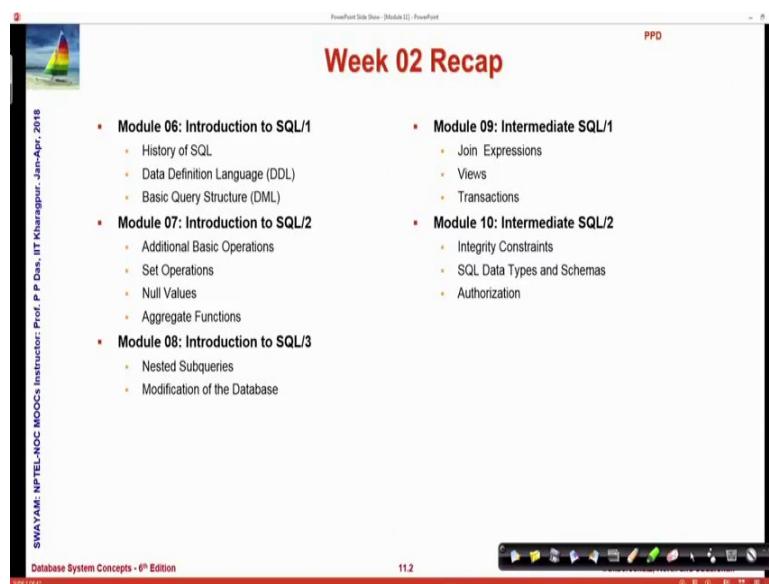
So, with this we close our discussion on the intermediate level SQL features. In the next module that we start next week, we will talk about some of the advanced SQL features.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 11**  
**Advanced SQL**

Welcome to module eleven of database management system. This will be on advanced SQL.

(Refer Slide Time: 00:31)



The slide is titled "Week 02 Recap" in red at the top right. It features a small sailboat icon in the top left corner. The content is organized into two columns of bullet points:

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                    |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>▪ <b>Module 06: Introduction to SQL/1</b><ul style="list-style-type: none"><li>▫ History of SQL</li><li>▫ Data Definition Language (DDL)</li><li>▫ Basic Query Structure (DML)</li></ul></li><li>▪ <b>Module 07: Introduction to SQL/2</b><ul style="list-style-type: none"><li>▫ Additional Basic Operations</li><li>▫ Set Operations</li><li>▫ Null Values</li><li>▫ Aggregate Functions</li></ul></li><li>▪ <b>Module 08: Introduction to SQL/3</b><ul style="list-style-type: none"><li>▫ Nested Subqueries</li><li>▫ Modification of the Database</li></ul></li></ul> | <ul style="list-style-type: none"><li>▪ <b>Module 09: Intermediate SQL/1</b><ul style="list-style-type: none"><li>▫ Join Expressions</li><li>▫ Views</li><li>▫ Transactions</li></ul></li><li>▪ <b>Module 10: Intermediate SQL/2</b><ul style="list-style-type: none"><li>▫ Integrity Constraints</li><li>▫ SQL Data Types and Schemas</li><li>▫ Authorization</li></ul></li></ul> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

At the bottom left, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur, Jan-Apr., 2018". At the bottom center, it says "Database System Concepts - 6<sup>th</sup> Edition". The bottom right corner shows slide number "11.2".

Before we start let me quickly recap what we did last week in the five modules. Last week we totally spent on discussing first the introductory features of SQL how to create data and how to write basic queries, and we studied about all different kinds of SQL operations at theoretic operation, handling of null values aggregation, nested queries and so on. And then we did an intermediate level of SQL query formation in terms of joint expression, views and integrities different kinds of SQL data types and importantly authorization.

(Refer Slide Time: 01:24)

The slide is titled "Module Outline" in red. It contains a bulleted list of topics:

- Accessing SQL From a Programming Language
- Functions and Procedural Constructs
- Triggers

At the bottom left, it says "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur, Jan-Apr., 2018". At the bottom center, it says "Database System Concepts - 6<sup>th</sup> Edition". At the bottom right, it shows slide number 11.4.

In the context, in this context, we now take up some more of the SQL features which are somewhat advanced. And we will try to understand how SQL can be used from a programming language, and familiarize with functions and procedures in SQL. This will violate some of the basic premises that we started with in saying that SQL is a declarative language only because as you understand functions procedure as procedural language features we will see how to handle those, and we will take a look into another important feature of triggers.

(Refer Slide Time: 02:07)

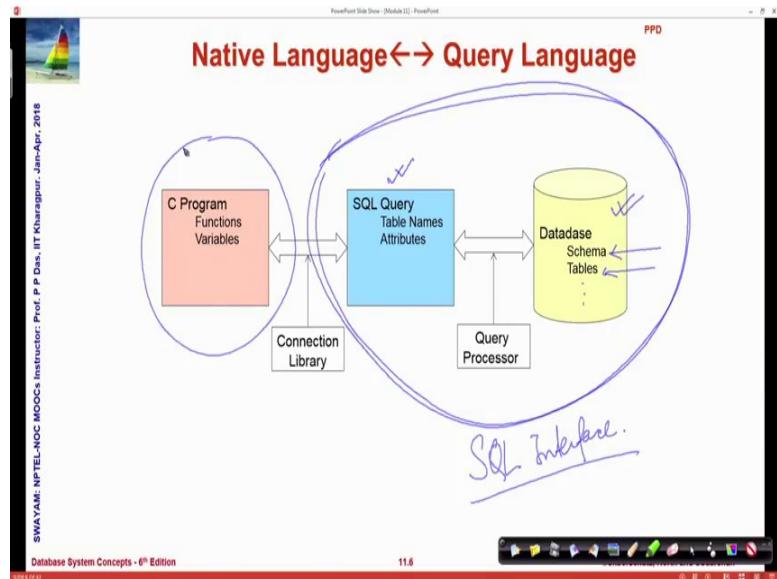
The slide has a large red title "ACCESSING SQL FROM A PROGRAMMING LANGUAGE" at the top. Below it is a bulleted list of topics:

- Accessing SQL From a Programming Language
- Functions and Procedural Constructs
- Triggers

At the bottom left, it says "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur, Jan-Apr., 2018". At the bottom center, it says "Database System Concepts - 6<sup>th</sup> Edition". At the bottom right, it shows slide number 11.5.

First with accessing SQL so, this is the module outline accessing SQL from a programming language.

(Refer Slide Time: 02:11)



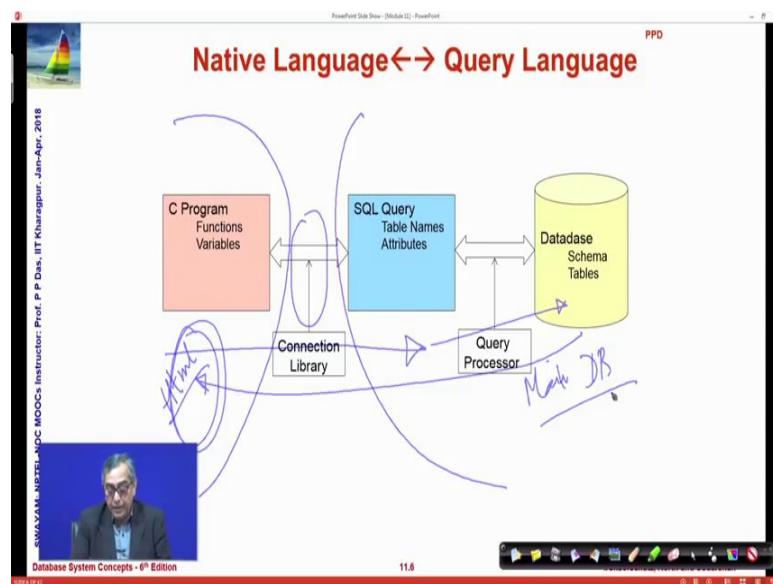
So, this is just kind of an abstract view that you can think of that what we have been doing so far naturally there is a database which has primarily two things schema and tables. Of course, there are many other things there are index, there is authorization that triggers and all that, but there is a database which stores everything.

And so far we have been dealing with only this part of the information that I can write certain SQL query to define table, using table names, attributes and certain logic and that goes through certain query processor works on the database to get me either create the desired effect. Either that is creating table instances or creating index or extracting certain relation values, defining some views and so on. So, all these have to happen through an SQL interface. So, this has to happen through an SQL interface.

So, whatever system we are using whether you are using MySQL, Postgres, Oracle, Sybase SQL server whatever you using that has some interface through which these SQL queries can be executed, so that is kind of a standalone complete system. But, in general, we will have lot of more requirements in terms of the application. For example, the application might require some graphics, SQL does not have support for graphics; application might require certain numerical algorithms to be executed might require some geometric computations to be done poly intersection of polygons to be computed

and so on and so forth. It might require some network programming and so on. So, there is a need to do all these and certainly these are best done in terms of certain native language that could be C++, java, C#, visual basic, python any of the native languages which has support for a variety of different tasks.

(Refer Slide Time: 04:20)



So, what is critical is can we make a bridge between these two that is can we take the advantages of the SQL domain and the advantages of the normal imperative procedural programming domain together. That is I can do some graphic representation and then certainly go to the database extract some information and then present it in the graphics and vice versa. For example, whenever we access say gmail, we get to see them in terms of an html presentation which is some kind of a graphics rendering, but the all the mail entry certainly come from some database.

So, somewhere there is a connection by which when I say get my inbox mails and somewhere the information goes over from this to the SQL query up to the database and the result is brought back to me. I see that in the html page here, but here there must be certain mail database, where all these information exists. So, what I am trying to come at is it is critical that the application programs or you know high level programming normal programming languages native languages should be able to interface with SQL. And what we discuss here is two different mechanisms for this interfacing. The first one is using a connection library and this is what I will specifically show you.

(Refer Slide Time: 05:50)

Accessing SQL From a Programming Language

- API (application-program interface) for a program to interact with a database server
- Application makes calls to
  - Connect with the database server
  - Send SQL commands to the database server
  - Fetch tuples of result one-by-one into program variables
- Various tools:
  - JDBC (Java Database Connectivity) works with Java
  - ODBC (Open Database Connectivity) works with C, C++, C#, Visual Basic, and Python
    - Other API's such as ADO.NET sit on top of ODBC
  - Embedded SQL

Database System Concepts - 6<sup>th</sup> Edition  
CAYAM-NPTEL-ODC MOOCs Instructor: Prof. P. Das, IIT Kharagpur, Jan-Apr, 2018

11.7

So, for if you are using a connection library then you have a set of APIs application programming interfaces these are basically functions in that library. So, that with that application can connect to the database because certainly the database is somewhere else is a different server. And send an SQL command to the database server so that you can say that this is what I want. And then a result is computed the result of that computation the table the can be brought back those tuples can be brought back to the application program. Mind you when we are here when you are sending the information in terms of the database server, we are talking SQL command, we are talking about attributes tables of the SQL space.

Whereas, when I want it in the program I want it in terms of program variables. So, there has to be certain correspondence made between them. There are a variety of tools available which allow you to do this JDBC is common very commonly known which is specific for java. We have an open database connectivity APIs which has different versions for different languages these are the common languages that it is used with. And as we will see later on that there is another mechanism for doing the same thing called the embedded SQL, we will come to that later.

(Refer Slide Time: 07:15)

The screenshot shows a PowerPoint slide with the title 'JDBC' in red at the top center. Below the title is a bulleted list of points about JDBC:

- JDBC is a Java API for communicating with database systems supporting SQL
- JDBC supports a variety of features for querying and updating data, and for retrieving query results.
- JDBC also supports metadata retrieval, such as querying about relations present in the database and the names and types of relation attributes
- Model for communicating with the database:
  - Open a connection
  - Create a "statement" object
  - Execute queries using the Statement object to send queries and fetch results
  - Exception mechanism to handle errors

At the bottom left of the slide, there is a small video window showing a man speaking. The video window has a blue background and the text 'Database System Concepts - 6<sup>th</sup> Edition' at the bottom. The overall interface is a standard Windows desktop with a taskbar at the bottom.

So, JDBC is a java API, I will not go into details here if you know java you should be able to quickly look up. So, it is a it communicates with the database by opening a connection creating what java calls a statement object and executes the query using the statement objects and that is used to send the query as well as to get back the result. So, java is object oriented as you know so statement object is used as a as an encapsulation which travels between the java program and the SQL query processor. And since the query gets back the result since the query gets back there is that kind of there is exception mechanism to handle errors which is common for java.

(Refer Slide Time: 08:00)

The screenshot shows a PowerPoint slide with the title 'ODBC' in red at the top center. Below the title is a bulleted list of points about ODBC:

- Open DataBase Connectivity (ODBC) standard
  - standard for application program to communicate with a database server
  - application program interface (API) to
    - open a connection with a database,
    - send queries and updates,
    - get back results
- Applications such as GUI, spreadsheets, etc. can use ODBC

At the bottom left of the slide, there is a small video window showing a man speaking. The video window has a blue background and the text 'Database System Concepts - 6<sup>th</sup> Edition' at the bottom. The overall interface is a standard Windows desktop with a taskbar at the bottom.

What you see in contrast does a similar thing, but since it is to cater for different languages it has got a softer model it has got less powerful model. It is a standard application program to communicate with database server. So, again it has to open a connection to the database, send queries and updates and get back results. So, these are these are the three basic things, these three other three basic things that certainly needs to be done if I want to easily work across the application programming domain and the database programming domain. So, applications such as GUI, spreadsheet, etcetera can use ODBC.

(Refer Slide Time: 08:39)

**ODBC – Python Example**

- The code uses a data source named "SQLS" from the odbc.ini file to connect and issue a query.
- It creates a table, inserts data using literal and parameterized statements and fetches the data

```

import pyodbc

conn = pyodbc.connect('DSN=SQLS;UID=test01;PWD=test01')
cursor=conn.cursor()
cursor.execute("create table rvtest (col1 int, col2 float,
col3 varchar(10))")
cursor.execute("insert into rvtest values(1, 10.0,
'ABC')")
cursor.execute("select * from rvtest")

while True:
    row=cursor.fetchone()
    if not row:
        break
    print(row)

cursor.execute("delete from rvtest")
cursor.execute("insert into rvtest values (?, ?, ?)", 2,
20.0, 'XYZ')
cursor.execute("select * from rvtest")

while True:
    row=cursor.fetchone()
    if not row:
        break
    print(row)

```

Source: <https://dzone.com/articles/tutorial-connecting-to-odbc-data-sources-with-python>

So, yeah I am just quickly showing you an example we will talk about application programming mode in a later module. So, this is a python example. So, python for this the ODBC for python is known as pyodbc library. So, you need to import that. And then using that you can connect. So, if you have to connect you have to say which database who is the user, what is the password, because authentication needs to happen. So, SQLS is a database here and with this user with this password is connecting you that is successful you get a conn object and on the conn object you have something what is known as a cursor. Cursor is nothing but if you think about it is kind of a pointer. So, it can be used to point to either a row or a whole query or a table.

So, you get back a cursor object. So, then this is if you look into this part, this part is nothing but a pure SQL query. So, you take that as a string and pass it on to the execute

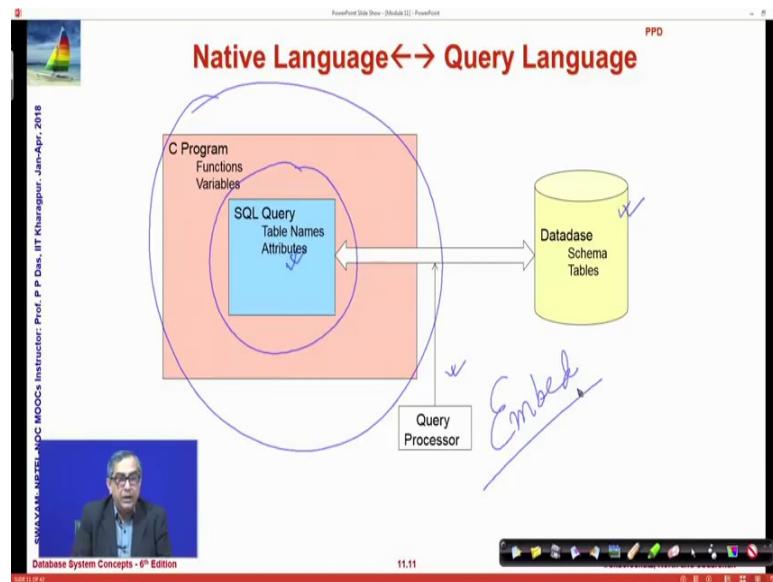
method of cursor. So, what it does is, so this is this has done the first task which is connecting to the database. Now, you are basically putting the query to the database in terms of doing saying that this execute. Similarly, so if that will get executed, so the table is created naturally there is no result to get if you have created a table. Now, you do an insert. So, again this particular record, you can see that within this double quote, this whole insert syntax you take that as a string and just give it to execute as a parameter.

In the third, that you do is do a select. So, you have done this we have inserted a record, created a table inserted a record in that and then now you are doing a select. So, certainly we expect one record to come back. So, these are the SQL executions and then your get back result. So, cursor has another method which is known as fetch one. So, what it will do that if your result table has multiple entries, then the cursor will if will be will start with the first row and fetch one will bring the whole of this first row as a row vector.

So, it will bring in as all the components as one as a python string. And then you check whether it is empty or not, if it is empty then the I mean there is nothing to bring back, so you are done. So, you break otherwise you simply print the row, so you are printing there. And then you go back again this is while true. So, what happens is the cursor will advance to the next row and get you the next row. Again you do the same thing go back it will come back to the same, but once you get an empty result, you know that there is nothing more to proceed and you break.

So, this is what is illustrated then there are some more this particular record is deleted, then again another record is inserted. And another select is done. So, this is how a native program here in this case python can interact with the database and do any of the SQL tasks that we were doing earlier with SQL interface, now we can be done from the python, so that is a basic ODBC mechanism. I particularly chose python to just give you a different flavour, you can we will have assignments on doing it for C and using jdbc on java as well.

(Refer Slide Time: 12:19)



Now, I am come I will come back to the same interaction issue, but this time you can see that I am using a different diagram. The database is the same; this part is the same; the query processing is same, but instead of having a connection library, now I have put the SQL query itself as a part of the native program, the C program. So, this is what is called embedding.

So, you say I embed I put the SQL as a part of C, but naturally SQL is not C. So, we need to put certain additional syntax in C, so that I can directly write SQL as a part of the C program. I am talking about C, again just as an example if this is true for other several other languages which can be used as used for embedding SQL within them.

(Refer Slide Time: 13:09)

The slide is titled "Embedded SQL". It contains the following text and bullet points:

EXEC SQL <embedded SQL statement>;

Note: this varies by language:

- In some languages, like COBOL, the semicolon is replaced with END-EXEC
- In Java embedding uses # SQL{ .... };

Below the slide, a video player window shows a man speaking, identified as Prof. P. Das, BT Kharagpur, Jan-Apr., 2018. The video player has a progress bar at 11.12 and a red bar at the bottom.

So, this is called the embedded SQL, it works for C, C++ , java fortran etcetera. And the language native language in which your embedding is called the is known as a host language. And basic form of these languages allow that the are come from the System R. So, what is important is this particular statement EXEC-SQL. This EXEC-SQL written inside the body of a C program will tell the C compiler that this part is not C; this part is actually embedded SQL.

And it will be treated differently; it will be compiled by the SQL compiler within the C compiler, so that is the basic structure, so EXEC-SQL. And then you put the pure SQL statement the embedded SQL statement. So, let us go forward and see some of this, there are different syntax for different native language embedding.

(Refer Slide Time: 14:11)

The slide is titled "Embedded SQL (Cont.)". It contains the following text:

- Before executing any SQL statements, the program must first connect to the database. This is done using:  
EXEC-SQL connect to server user user-name using password;  
Here, *server* identifies the server to which a connection is to be established
- Variables of the host language can be used within embedded SQL statements. They are preceded by a colon (:) to distinguish from SQL variables (e.g., :credit\_amount )
- Variables used as above must be declared within DECLARE section, as illustrated below. The syntax for declaring the variables, however, follows the usual host language syntax

EXEC-SQL BEGIN DECLARE SECTION  
int credit-amount ;  
EXEC-SQL END DECLARE SECTION;

Below the slide, there is a video player window showing a man speaking. A red box highlights this video player area.

So, to be able to connect you say EXEC-SQL and connect to server username using password. So, we saw similar things in terms of ODBC based connection these three information needs to be specified which database server, who is the user, what is the password. So, here you say that in this form. And this particular statement you can write as a part of the C program. Now, naturally the question here is in the in the earlier case when we are using the connection library, your results came back through the cursor which you then could deal because the cursors are necessarily objects in your native language. So, in python the cursor was a particular object in the pyodbc library.

But now you have embedded the SQL in terms of your c program. So, naturally the results or whatever you are doing in C which needs to have a communication with the SQL statement need to be differentiated from the SQL names themselves. So, any native language variable that needs to be treated in SQL will be marked with a colon preceding it. So, credit amount of this a colon credit amount, so it becomes a SQL variable provided credit amount itself is a C variable, so that is the basic you know connection mechanics. There are far more details in that, but I am just giving you the glimpse.

Now, any region where you write the SQL, you can write it as exact SQL begin declare section, and END declare section this is where you specify what are the different what are the different C variables C declarations that need to be used for this SQL definition. I will just show you an example soon so that.

(Refer Slide Time: 16:15)

The screenshot shows a PowerPoint slide titled "Embedded SQL (Cont.)". The slide content includes:

- To write an embedded SQL query, we use the `declare c cursor for <SQL query>` statement. The variable `c` is used to identify the query
- Example:
  - From within a host language, find the ID and name of students who have completed more than the number of credits stored in variable `credit_amount` in the host language
  - Specify the query in SQL as follows:

Below the text, there is a block of SQL code:

```
EXEC SQL
declare c cursor for
select ID, name
from student
where tot_cred > :credit_amount
END_EXEC
```

Annotations include a blue curly brace under "declare c cursor for", a blue arrow pointing from "credit\_amount" in the SQL code to the variable "credit\_amount" in the explanatory text, and a blue circle highlighting "credit\_amount" in the explanatory text.

Similar to the ODBC style, you have to declare a cursor. So, this is to write the embedded SQL, you use declared C cursor for such and such SQL queries, so then that C, variable C will become your handle in the C language to be able to answer handle the query results. So, here is an example. So, you can see that credit amount is a variable in the host language.

So, you are using it in SQL with colon credit amount which says that it is this host language variable. So, that you can set a particular credit amount and go with that. And this is the query, and you have set a cursor on that which you can make use of in the exact SQL.

(Refer Slide Time: 17:14)

The slide is titled "Embedded SQL (Cont.)". It contains the following text:

- Example:
  - From within a host language, find the ID and name of students who have completed more than the number of credits stored in variable `credit_amount` in the host language
- Specify the query in SQL as follows:

```
EXEC SQL
declare c cursor for
select ID, name
from student
where tot_cred > :credit_amount
END_EXEC
```
- The variable `c` (used in the cursor declaration) is used to identify the query

A video player window is visible on the right side of the slide, showing a man speaking. The video player has a progress bar at 11.15 and a red status bar at the bottom.

So, let us this is the example continued.

(Refer Slide Time: 17:18)

The slide is titled "Embedded SQL (Cont.)". It contains the following text:

- The `open` statement for our example is as follows:

```
EXEC SQL open c;
```

This statement causes the database system to execute the query and to save the results within a temporary relation. The query uses the value of the host-language variable `credit-amount` at the time the `open` statement is executed.
- The `fetch` statement causes the values of one tuple in the query result to be placed on host language variables.

```
EXEC SQL fetch c into :si, :sn END_EXEC
```

Repeated calls to `fetch` get successive tuples in the query result

A video player window is visible on the right side of the slide, showing a man speaking. The video player has a progress bar at 11.16 and a red status bar at the bottom.

Let us look at other features. You can once you have set a query you can actually execute that by the `open` statement. So, you say `EXEC SQL open c`, the cursor. So, that will execute the query that you have associated with that cursor. And then once that has been done, then you can fetch the results into that cursor one by one; one tuple at a time.

(Refer Slide Time: 17:47)

The slide is titled "Embedded SQL (Cont.)". It contains the following bullet points:

- A variable called SQLSTATE in the SQL communication area (SQLCA) gets set to '02000' to indicate no more data is available
- The close statement causes the database system to delete the temporary relation that holds the result of the query.

```
EXEC SQL close c;
```

Note: above details vary with language. For example, the Java embedding defines Java iterators to step through result tuples.

Speaker photo: A man with glasses and a suit, speaking into a microphone. The video player shows "Database System Concepts - 6th Edition" and "11.17".

Once you are done with all that then you simply close.

(Refer Slide Time: 17:52)

The slide is titled "Embedded SQL – C Example". It contains the following bullet point:

- The program prompts the user for an order number, retrieves the customer number, salesperson, and status of the order, and displays the retrieved information on the screen

```
int main() {  
    /* Execute the SQL query */  
    EXEC SQL INCLUDE SQLCA;  
    EXEC SQL BEGIN DECLARE SECTION;  
    int orderId; /* Employee ID (from user) */  
    int custId; /* Retrieved customer ID */  
    char salesPerson[10]; /* Retrieved salesperson name */  
    char status[5]; /* Retrieved order status */  
    EXEC SQL END DECLARE SECTION;  
  
    /* Set up error processing */  
    EXEC SQL WHENEVER SQLERROR GOTO query_error;  
    EXEC SQL WHENEVER NOT FOUND GOTO bad_number;  
  
    /* Prompt the user for order number */  
    printf ("Enter order number: ");  
    scanf ("%d", &orderId);  
  
    /* Execute the SQL query */  
    EXEC SQL SELECT CustID, SalesPerson, Status  
        FROM Orders  
        WHERE OrderID = :orderId  
        INTO :custId, :salesPerson, :status;  
  
    /* Display the results */  
    printf ("Customer number: %d\n", custId);  
    printf ("Salesperson: %s\n", salesPerson);  
    printf ("Status: %s\n", status);  
    exit();  
  
    /* Error handling */  
    query_error:  
    printf ("SQL error: %d\n", sqlca->sqlcode);  
    exit();  
  
    bad_number:  
    printf ("Invalid order number.\n");  
    exit();  
}
```

Source: <https://docs.microsoft.com/en-us/sql/odbc/reference/embedded-sql>

So, let us look at a example. This is a program which will prompt the user for an order number, and retrieves the customer number I mean given an order number it will retrieve the customer number, salesperson, status of the order and it will display that as the retrieved information on the screen. So, here is a C program. It starts on here with the main. So, you can see that this says that EXEC-SQL INCLUDE SQLCA, SQLCA is the communication area. So, there is a exchange going on between the c program

and SQL program. So, the area that is used by both for this transaction is known as SQLCA.

Then you have the declare section. So, you are saying these are SQL exec declaration. So, these are, but within that what you have are pure C declaration, but all the declarations of C that are put within this can be used in SQL query with a colon at the beginning. So, that the value can be exchanged to the SQLCA then in the next you are specifying what will happen if you have an error.

So, you say SQL look at this EXEC-SQL whenever SQL error go to query error. So, it will go to this level. If it is not found that is no result is there it will go to this. So, you by making sure that if there is some error that happens in the SQL part, what will happen in your C program. And then subsequently you have simple C program which reads the order number, and after having read that you are doing this.

So, what does it do, EXEC-SQL is to say that this is embedded; this is the select query starts, the fields, the relation, the condition. And then there are three attributes. So, you are setting an association with three variables in the C program. So, if I want to the result of this select will be a table of three attributes three columns, so we are giving a name to each one of these three attributes in terms of our C program.

So, there is a cust id. So, I say the cust id attribute in SQL is colon cust id here which is basically cust id variable in the C program. Let me clean up again. So, this is cust id; this is in SQL; this is my C program, and this is my C program variable in SQL which corresponds to this its similarly order based. Similarly, this is in SQL attribute this is a array of character here. And this is a correspondence; this is a correspondence.

So, now, you can easily see that once this has been done I will be able to get all these values here, normally the program would be longer the you will have to iterate over the cursor as you did in case in the ODBC case. But in this case since we are using one order number we know that there will be only one record. So, we have not shown the iteration on the cursor.

So, once you have got this you are using those values to print out the result. And in case this query has got into some problem because of SQL, then it will automatically jump to SQL query this particular level. If it has got a bad number which means that no

such record was found, it was a null table then it will immediately take you to this. So, this is how the embedded SQL worked.

So, there are these are two different styles the ODBC style and the embedding style are two different styles. If you have if you depending on the preference you use that earlier days people used to use more of embedding, I believe that now the preference is more for the ODBC kind of connection oriented system ODBC JDBC kind of because they are certainly more programmer friendly this.

(Refer Slide Time: 22:05)

The slide has a title 'Updates Through Embedded SQL' in red. Below the title is a bulleted list:

- Embedded SQL expressions for database modification (**update**, **insert**, and **delete**)
- Can update tuples fetched by cursor by declaring that the cursor is for update

Below the list is a section titled 'EXEC SQL' containing the following code:

```
declare c cursor for
select *
from instructor
where dept_name = 'Music'
for update
```

After the code, another bullet point states:

- We then iterate through the tuples by performing **fetch** operations on the cursor (as illustrated earlier), and after fetching each tuple we execute the following code:

```
update instructor
set salary = salary + 1000
where current of c
```

At the bottom left, it says 'Database System Concepts - 6<sup>th</sup> Edition'. At the bottom right, it shows '11.19' and a navigation bar.

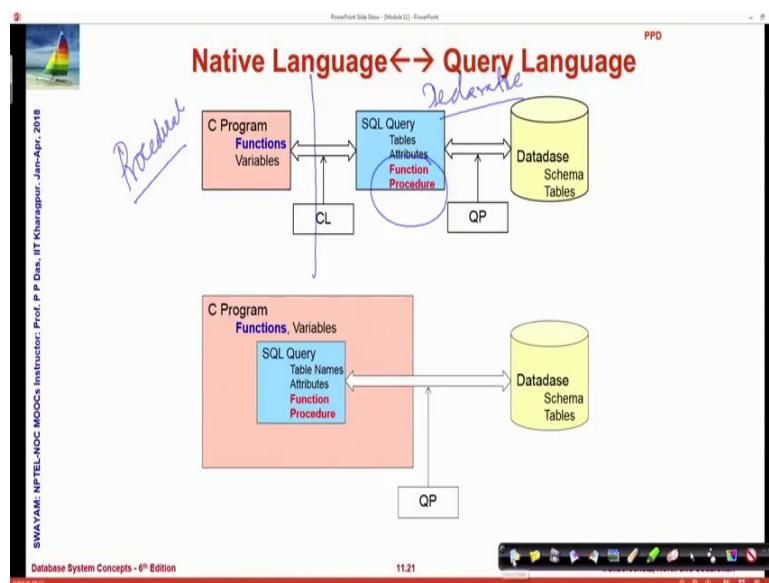
You can do updates through embedded SQL as well. So, I will not go through the details you can just go through this and understand that.

(Refer Slide Time: 22:15)



Let me move onto the next advanced part of SQL which is function and procedural construct.

(Refer Slide Time: 22:27)



Now mind you again this is these are the two models that we have I have already shown you. The two models in which the application program the native language program and the query language can interact the ODBC mechanism and the Embedded mechanism. But what we are now empowering is so far our basic premise was that this site is a procedural I discussed this at the very beginning. And this side is declarative. So, in SQL

you do not say that how you find out a result, you say what you want as a result, these are more like predicates. And in C program, you cannot specify what you want as a result you would rather say that do step one, step two, step three and you will get this result.

So, C program is all full of functions again I am talking about C as a placeholder it is true for most of the programming languages we use otherwise. So, there are procedural languages. So, procedures in C are functions; whereas, in SQL you had select from where kind of conditional clause.

Now, what we are saying that SQL also in later version have allowed certain functions and procedures, which can be part of SQL. It also has allowed certain imperative constructs like case like loop like while, repeat those kind of to make certain kind of procedural programming easier in SQL. And naturally these again can be used in conjunction with the connection oriented applications with the native or embedded oriented mechanisms. So, we will just take a quick look into some of these function and procedural features.

(Refer Slide Time: 24:15)

The screenshot shows a Microsoft PowerPoint slide titled "Functions and Procedures" in red font. The slide content includes a bulleted list of points about SQL:1999 support for functions and procedures, mentioning table-valued functions and imperative constructs like loops and assignments. The slide is part of a presentation titled "Database System Concepts - 6th Edition". On the left side of the slide, there is a vertical sidebar with the text "SWAYAM AND IIT-MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018". At the bottom of the slide, there is a small video player showing a person speaking, with the text "Database System Concepts - 6th Edition" below it. The overall interface is that of a Windows operating system, with a taskbar at the bottom showing various application icons.

So, this started coming in from SQL 1999. And you can have functions and procedure written in SQL itself and function and procedures written external language or the host language also. So, both of them are available. What is interesting is some database systems support functions which are table valued. Functions, we always know functions

written objects only, but in SQL you can have functions which have table valued which written new functions. So, and certain imperative constructs have come in.

(Refer Slide Time: 24:56)

SQL Functions

- Define a function that, given the name of a department, returns the count of the number of instructors in that department.

```
create function dept_count (dept_name varchar(20))  
returns integer  
begin  
declare d_count integer;  
select count(*) into d_count  
from instructor  
where instructor.dept_name = dept_name  
return d_count;  
end
```
- The function `dept_count` can be used to find the department names and budget of all departments with more than 12 instructors.

```
select dept_name, budget  
from department  
where dept_count(dept_name) > 12
```

SWAYAM MOOCs Instructor: Prof. P. Das, BT Kharagpur, Jain-Apr-2018

Database System Concepts - 6<sup>th</sup> Edition

11.23

So, there are several databases have their proprietary constructs and all that also. So, at this level of the course since we are not focusing particularly on any specific database systems, we will not talk about those. We can do enough in terms of the standard SQL itself. So, this is how you define a function which looks very similar to the SQL definition, create function, give it a name certainly here are the parameters. And here is a return type. So, if you are familiar with C, C++, Java you I mean it is just that the syntax is different, but the elements are same.

There is a begin end in the scope. And this is the pure SQL that you are writing here. So, this is a function which is not a function in C, this is a function in SQL. So, this you can write this function in SQL. And once you have written that function then you can actually use that. So, if you look at the function is department name, then separately I am writing a query, I am using this department name as function. So, whenever I whenever this query will get executed, this function will be called, and the corresponding values will get returned. So, this is the basic mechanism ok.

(Refer Slide Time: 26:16)

The slide is titled "SQL functions (Cont.)" in red. It contains a bulleted list of details about SQL functions:

- Compound statement: **begin ... end**
  - May contain multiple SQL statements between **begin** and **end**.
- **returns** – indicates the variable-type that is returned (e.g., integer)
- **return** – specifies the values that are to be returned as result of invoking the function
- SQL function are in fact parameterized views that generalize the regular notion of views by allowing parameters

A video player interface is visible at the bottom, showing a thumbnail of a person speaking, the title "Database System Concepts - 6th Edition", the time "11:24", and other control buttons.

So, so there are SQL functions have several details which you can go through it has returned naturally.

(Refer Slide Time: 26:23)

The slide is titled "Table Functions" in red. It contains a bulleted list of details about table functions:

- SQL:2003 added functions that return a relation as a result
- Example: Return all instructors in a given department

```
create function instructor_of(dept_name char(20))  
    returns table  
        ID varchar(5),  
        name varchar(20),  
        dept_name varchar(20),  
        salary numeric(8,2)  
    return table  
        (select ID, name, dept_name, salary  
         from instructor  
         where instructor.dept_name = instructor_of.dept_name)
```

▪ Usage

```
select *  
from table (instructor_of('Music'))
```

A video player interface is visible at the bottom, showing a thumbnail of a person speaking, the title "Database System Concepts - 6th Edition", the time "11:25", and other control buttons.

And you can have table functions where it can return table. So, the syntax is given again its clear to see what will happen if you return a table which is computed from the select from where query that you have within the function.

(Refer Slide Time: 26:37)

The slide is titled "SQL Procedures". It contains a bulleted list and some code snippets.

- The `dept_count` function could instead be written as procedure:

```
create procedure dept_count_proc (
    in dept_name varchar(20),
    out d_count integer)
begin
    select count(*) into d_count
    from instructor
    where instructor.dept_name = dept_count_proc.dept_name
end
```

- Procedures can be invoked either from an SQL procedure or from embedded SQL, using the `call` statement.

```
declare d_count integer;
call dept_count_proc('Physics', d_count);
```

- Procedures and functions can be invoked also from dynamic SQL
- SQL:1999 allows more than one function/procedure of the same name (called name **overloading**), as long as the number of arguments differ, or at least the types of the arguments differ

Database System Concepts - 6<sup>th</sup> Edition 11.26

I can have SQL procedures which just performs some action, but does not have a return value so to say. And you can use them they can declare and call those procedures explicitly. Procedures can be functions and procedures can be overloaded as well if you are on SQL 99.

(Refer Slide Time: 27:08)

The slide is titled "Language Constructs for Procedures & Functions". It contains a bulleted list and some code snippets.

- SQL supports constructs that gives it almost all the power of a general-purpose programming language.
  - Warning: most database systems implement their own variant of the standard syntax below.
- Compound statement: `begin ... end`,
  - May contain multiple SQL statements between `begin` and `end`.
  - Local variables can be declared within a compound statements
- While** and **repeat** statements:

```
while boolean expression do
    sequence of statements;
end while

repeat
    sequence of statements;
until boolean expression
end repeat
```

Database System Concepts - 6<sup>th</sup> Edition 11.27

Now, there are several language constructs as I mentioned SQL does allow while repeat. So, I am just covering them in terms of completeness it is not that these are frequently used features or I recommend that you do lot of them right here. If you need

to do procedural thing its always better to do them outside, but in some cases it may be easier to code a query if you can write a while, repeat kind of loop.

(Refer Slide Time: 27:37)

The screenshot shows a PowerPoint slide with the title 'Language Constructs (Cont.)' in red. On the left, there is a vertical sidebar with text: 'SWAYAM-NPTEL-NC MOOCs Instructor: Prof. P P Das, BT Kharagpur - Jain-Apr-2015'. The main content area contains a bulleted list under the heading 'For loop':

- For loop
  - Permits iteration over all results of a query
- Example: Find the budget of all departments

```
declare n integer default 0;
for r as
    select budget from department
do
    set n = n + r.budget
end for
```

Below the text is a video player window showing a man speaking. The video player has a blue background and the text 'Database System Concepts - 6<sup>th</sup> Edition' at the bottom. The bottom right corner of the slide shows the number '11.28'.

You can write a for loop which iterates over the records of a table which is certainly very convenient. So, if you want to do something over the records of a table compute something that the for loop will become easier.

(Refer Slide Time: 27:50)

The screenshot shows a PowerPoint slide with the title 'Language Constructs (Cont.)' in red. On the left, there is a vertical sidebar with text: 'SWAYAM-NPTEL-NC MOOCs Instructor: Prof. P P Das, BT Kharagpur - Jain-Apr-2015'. The main content area contains a bulleted list under the heading 'Conditional statements (if-then-else)':

- Conditional statements (if-then-else)  
SQL:1999 also supports a case statement similar to C case statement
- Example procedure: registers student after ensuring classroom capacity is not exceeded
  - Returns 0 on success and -1 if capacity is exceeded
  - See book (page 177) for details
- Signaling of exception conditions, and declaring handlers for exceptions

```
declare out_of_classroom_seats condition
declare exit handler for out_of_classroom_seats
begin
...
.. signal out_of_classroom_seats
end
```

  - The handler here is exit -- causes enclosing begin..end to be exited
  - Other actions possible on exception

Below the text is a video player window showing a man speaking. The video player has a blue background and the text 'Database System Concepts - 6<sup>th</sup> Edition' at the bottom. The bottom right corner of the slide shows the number '11.29'.

You have a conditional statement case actually we have seen the case already. So, which is very easy in terms of coding many of the features so, here are some of them then you

have exceptions. So, I am not going through these, these in depth, but this is just to making you aware that while SQL continues to be predominantly a declarative language, it does have quite a bit of procedural support which in an appropriate time can be used if required. So, you can look up the manual for that. And there are a whole lot of things you can do with the external language routines.

(Refer Slide Time: 28:30)

The screenshot shows a PowerPoint slide with the title 'External Language Routines\*' in red at the top. Below the title, there is a bulleted list of two items:

- SQL:1999 permits the use of functions and procedures written in other languages such as C or C++
- Declaring external language procedures and functions

Following the list, there are two code snippets in black font:

```
create procedure dept_count_proc(in dept_name varchar(20),
                                 out count integer)
language C
external name '/usr/avi/bin/dept_count_proc'

create function dept_count(dept_name varchar(20))
returns integer
language C
external name '/usr/avi/bin/dept_count'
```

At the bottom left, there is a vertical watermark-like text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur. Jain-Apr., 2018'. At the bottom right, there is a small toolbar and the text 'Database System Concepts - 6<sup>th</sup> Edition' and '11.30'.

That is can write a function in C and actually call it from SQL, this is just doing the other way round. Earlier in terms of embedding or in terms of ODBC, a C function was executing a query. Now, you are doing the reverse you are saying that I can write a SQL query which uses a C function that already exist. So, there are external ways of binding, I will not go through these slides in depth, because again the you will have to come a certain way before you can actually start using such features.

But get to know that there could be as from SQL you can use any external language library; and if some library is good for certain computation which is needed for your query which is a non database kind of computation then you can make use of this external language routines.

(Refer Slide Time: 29:23)

The slide is titled "External Language Routines (Contd.)\*" in red. It contains a bulleted list of points and a block of SQL code. The SQL code defines a procedure and a function that call external C programs.

- SQL:1999 allows the definition of procedures in an imperative programming language, (Java, C#, C or C++) which can be invoked from SQL queries.
- Functions defined in this fashion can be more efficient than functions defined in SQL, and computations that cannot be carried out in SQL can be executed by these functions.
- Declaring external language procedures and functions

```
create procedure dept_count_proc(in dept_name varchar(20),
                                 out count integer)
language C
external name '/usr/avi/bin/dept_count_proc'

create function dept_count(dept_name varchar(20))
returns integer
language C
external name '/usr/avi/bin/dept_count'
```

Database System Concepts - 6<sup>th</sup> Edition  
11.31

So, I have put together all the basic features that external language routines will need.

(Refer Slide Time: 29:30)

The slide is titled "External Language Routines (Cont.)\*" in red. It contains a bulleted list of points under two main categories: Benefits and Drawbacks.

- Benefits of external language functions/procedures:
  - more efficient for many operations, and more expressive power
- Drawbacks
  - Code to implement function may need to be loaded into database system and executed in the database system's address space.
    - risk of accidental corruption of database structures
    - security risk, allowing users access to unauthorized data
  - There are alternatives, which give good security at the cost of potentially worse performance
  - Direct execution in the database system's space is used when efficiency is more important than security

Database System Concepts - 6<sup>th</sup> Edition  
11.32

But again I would tell you that there are benefits, but there are lot of drawbacks in using them. And I would not recommend that you frequently use these features. You should primarily restrict to SQL programming, and then anything that you need to do in the native, you should go to choose your language and do that.

(Refer Slide Time: 29:48)

The slide is titled "Security with External Language Routines\*" in red. It contains a bulleted list of points:

- To deal with security problems, we can do one of the following:
  - Use **sandbox** techniques
    - That is, use a safe language like Java, which cannot be used to access/damage other parts of the database code.
  - Run external language functions/procedures in a separate process, with no access to the database process' memory.
    - Parameters and results communicated via inter-process communication
- Both have performance overheads
- Many database systems support both above approaches as well as direct executing in database system address space.

There are security issues also that you will need to understand here.

(Refer Slide Time: 29:58)

The slide has a title "TRIGGERS" in large red capital letters. To the right of the title, there is a bulleted list of features:

- Accessing SQL From a Programming Language
- Functions and Procedural Constructs
- Triggers

Finally, before we close I will just mention another feature called triggers, triggers are very important.

(Refer Slide Time: 30:04)

The screenshot shows a PowerPoint slide titled "Triggers" in red. The slide content includes a bulleted list about triggers and a small video player window showing a person speaking.

▪ A **trigger** is a statement that is executed automatically by the system as a side effect of a modification to the database

▪ To design a trigger mechanism, we must:

- Specify the conditions under which the trigger is to be executed.
- Specify the actions to be taken when the trigger executes.

▪ Triggers introduced to SQL standard in SQL:1999, but supported even earlier using non-standard syntax by most databases.

- Syntax illustrated here may not work exactly on your database system; check the system manuals

Database System Concepts - 6<sup>th</sup> Edition

11:35

A trigger is a statement that is executed automatically when something happens in the database. So, you want that well things are happening. And well I want to know if a particular value has exceeded a certain level or if something has become null or some violatory things are happening and so on. So, how do you know that because a database is being accessed by hundreds and thousands of people and with hundreds of tables and millions of records.

So, triggers are a mechanism by which you can set that under this condition, I want a trigger, I want something specifically to happen. So, again they were introduced in 99, but earlier also triggers were there, but in 99 standard they became formal earlier they were somewhat you know differently structured. So, you might find that the system that you are using for practice the trigger in that may have a different format and semantics than the what we are discussing here.

(Refer Slide Time: 31:12)

The slide is titled "Triggering Events and Actions in SQL". It lists several points about triggers:

- Triggering event can be **insert**, **delete** or **update**
- Triggers on update can be restricted to specific attributes
  - For example, **after update of takes on grade**
- Values of attributes before and after an update can be referenced
  - referencing old row as** : for deletes and updates
  - referencing new row as** : for inserts and updates
- Triggers can be activated before an event, which can serve as extra constraints. For example, convert blank grades to null.

Handwritten notes on the slide:

```
create trigger setnull_trigger before update of takes
referencing new row as nrow
for each row
when (nrow.grade = '')
begin atomic
    set nrow.grade = null;
end;
```

So, the most common triggering events are insert, delete, update. So, if something some update is happening, so I can say that after this update, I want such and such things to happen. Or I can during the update I can one that well I am doing an update. So, there is an old value which is typically referred to as old row, the row that is getting updated. And there is a new row the new set of values that are getting created. So, I might want between the old row and the new row that certain things happen.

So, I can say that well just look into this. So, again the syntax is all similar. Create trigger, trigger there is a name of the trigger and this is the condition. Before update of takes, takes is a relation **referencing new row as nrow**. So, this is the new value that we set. Now, what are you saying, saying that for each row what you do when n grid is blank n row dot grade is blank that is if this is if you are updating and you have got a got you are going to update, a grade value which is blank then you simply set it to null.

So, it is possible that the grade that has come in and grades are characters and what has come in from the input and is going to get updated is a show a certain grade to be now because it may not have been decided. Now, you do not want those blank values to be present. You want because blank cannot be checked, we have we have checkers for null and so on.

So, you want to set that to null. So, trigger can make this thing happened because otherwise how will you know what value is actually getting changed. So, there could be several ways trigger can be used.

(Refer Slide Time: 33:23)

The screenshot shows a PowerPoint slide with the title "Trigger to Maintain credits\_earned value". Handwritten notes are overlaid on the slide, pointing to specific parts of the code. The notes include:

- create trigger credits\_earned after update of takes on (grade)
- referencing new row as nrow
- referencing old row as orow
- for each row
- when nrow.grade <> F and nrow.grade is not null
- and (orow.grade = F or orow.grade is null)
- begin atomic
- update student
- set tot\_cred= tot\_cred +
- (select credits
- from course
- where course.course\_id= nrow.course\_id)
- where student.id = nrow.id;
- end;

At the bottom left, there is a small video thumbnail of a professor speaking. The bottom right corner shows the Windows taskbar with icons for various applications like File Explorer, Word, and Excel.

For example, this is showing one that as you change the grades then certainly based on the grades credit earned value is computed. So, as a greatest change if it is now once grades have been entered say for 200 students. Now, after reviews grades for three of them are getting changed. So, how do you know that for those students, the change of the grade may impact the computation of the on credits.

So, you would like to update that on credits or it may or may not be required. So, the trigger will tell you that after update. So, whenever the update happens you take the old and the new value n row and o row, and then you are putting some conditions that if that new row is grade is f, and is not null; old row is grade or it is not null, then you try to do this.

So, if it was failure, and if it continues to be failure, new grade is not f; if it is f then you do not have to do anything; if it is null it do not have to do anything. But if it is if it was f or it was null, and now it has become a different grade then certainly the computation to update the credits earned is required. So, and the trigger gives you the right point when you can do this because otherwise you will not know in terms of millions of updates happening when this particular thing is going on.

(Refer Slide Time: 35:08)

The slide is titled "Statement Level Triggers" in red. It contains a bulleted list of points:

- Instead of executing a separate action for each affected row, a single action can be executed for all rows affected by a transaction
  - Use **for each statement** instead of **for each row**
  - Use **referencing old table** or **referencing new table** to refer to temporary tables (called **transition tables**) containing the affected rows
  - Can be more efficient when dealing with SQL statements that update a large number of rows

At the bottom left, there is a small video thumbnail showing a man speaking. The video player interface shows "Database System Concepts - 6<sup>th</sup> Edition" and a timestamp of "11:38".

Triggers can be on statements as well you can decide leave for your reading.

(Refer Slide Time: 35:12)

The slide is titled "When Not To Use Triggers" in red. It contains a bulleted list of points:

- Triggers were used earlier for tasks such as
  - Maintaining summary data (e.g., total salary of each department)
  - Replicating databases by recording changes to special relations (called **change** or **delta** relations) and having a separate process that applies the changes over to a replica
- There are better ways of doing these now:
  - Databases today provide built in materialized view facilities to maintain summary data
  - Databases provide built-in support for replication
- Encapsulation facilities can be used instead of triggers in many cases
  - Define methods to update fields
  - Carry out actions as part of the update methods instead of through a trigger

At the bottom left, there is a small video thumbnail showing a man speaking. The video player interface shows "Database System Concepts - 6<sup>th</sup> Edition" and a timestamp of "11:39".

But you have to be careful that triggers sounds very interesting and what happens particularly with the early stage of programming people get overboard with triggers and start using them severely, but triggers do have a lot of overhead. So, you should not many of the things that triggers can do, can be done through other means for example, by materialized, views and so on. So, as we go along we will mention that these are the problems that need to solve get solved by triggers; otherwise normally you should think

twice before you actually use a triggers. So, there could be different other ways of solving the same problem.

(Refer Slide Time: 35:55)

The screenshot shows a PowerPoint slide titled "When Not To Use Triggers (Cont.)". The slide contains a bulleted list of risks associated with triggers:

- Risk of unintended execution of triggers, for example, when
  - Loading data from a backup copy
  - Replicating updates at a remote site
  - Trigger execution can be disabled before such actions.
- Other risks with triggers:
  - Error leading to failure of critical transactions that set off the trigger
  - Cascading execution

Below the slide, there is a video player window showing a man speaking. The video player interface includes a play button, volume control, and a progress bar indicating the video is at 11:40. The video title is "Database System Concepts - 6th Edition". On the left side of the video player, there is vertical text: "SWAYAM-NPTEL-NC MOOCs Instructor: Prof. P P Das, BT Kharagpur, Jain-Apr-2018".

And because you have to keep in mind that triggers are expensive because once you have triggers and actually internally database for every update. If you have an update trigger on a field or a relation, then with every transaction with every change the database has to check if your trigger is true or not and so therefore, there is a cost to that. The other thing is there are triggers are for the live execution.

So, if you have offline for example, you are loading the data from a backup copy or you are replicating your database at a remote site and so on, then you have to put off the trigger; otherwise you know falsely the triggers will start happening and that may have a catastrophic effect that might trigger of different alarms and all that. So, you have to be careful with triggers in that manner so, cascading executions and all those.

(Refer Slide Time: 36:46)

The screenshot shows a PowerPoint slide titled "Module Summary". The slide contains a bulleted list of learning objectives:

- Introduced the use of SQL from a programming language
- Familiarized with functions and procedures in SQL
- Understood the triggers

Below the list is a video player window showing a man speaking. The video player interface includes a play button, volume control, and a progress bar indicating the video is at 11:41. The bottom of the slide has a red footer bar with the text "Database System Concepts - 6<sup>th</sup> Edition". On the left side of the slide, there is vertical text: "CIVASAM-NIPER-ANDC MOOCs Instructor: Prof. P. Das, IIT Kharagpur, Jan-Apr., 2018".

So, to summarize we have and this kind of closes our direct discussion on SQL. So, we have introduced the use of the very important aspect the use of SQL from a programming language, the interface between the native language and query language boundary. And that is something which will be extremely useful for application programming. And we are familiarized with you know the imperative extensions of SQL, the procedural extensions of SQL in terms of functions and procedures that you can directly write in SQL. And we have just introduced concept of triggers, so that you can sniff what is going on in your database.

So, there is the quite a few other features of SQL as well majority of them are advanced features dealing with olap and several others, which I chose to skip at this level of the course. So, this will close our discussion on SQL. And next we will move onto the design of the database looking into the algebra and the modelling.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 12**  
**Formal Relational Query Languages**

Welcome to module twelve of database management systems, in this module we will talk about the formal relational query languages. In the last couple of modules we have discussed about SQL at length introducing it dealing with the intermediate level of SQL features, and then exposing to some of the advanced features as well. The foundational mathematical model of SQL the query languages are to be discussed in this present module.

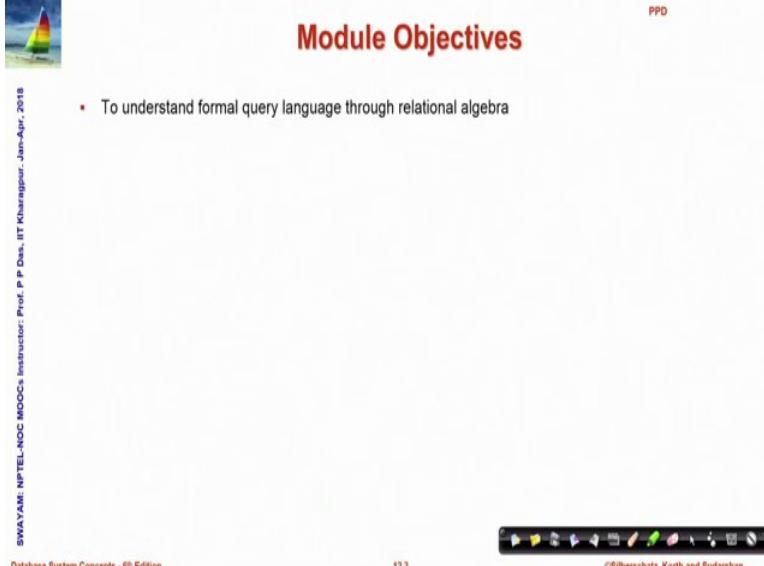
(Refer Slide Time: 01:04)

The slide is titled "Module Recap" in red text at the top right. It features a small sailboat icon on the left. A vertical watermark on the left side reads "SWAYAM: NPTEL-NOC INOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018". The main content is a bulleted list of three items: "Accessing SQL From a Programming Language", "Functions and Procedural Constructs", and "Triggers". At the bottom, there is footer text: "Database System Concepts - 8<sup>th</sup> Edition", "12.2", and "©Silberschatz, Korth and Sudarshan". There is also a decorative navigation bar at the bottom.

- Accessing SQL From a Programming Language
- Functions and Procedural Constructs
- Triggers

So, this is what we had done in the last module.

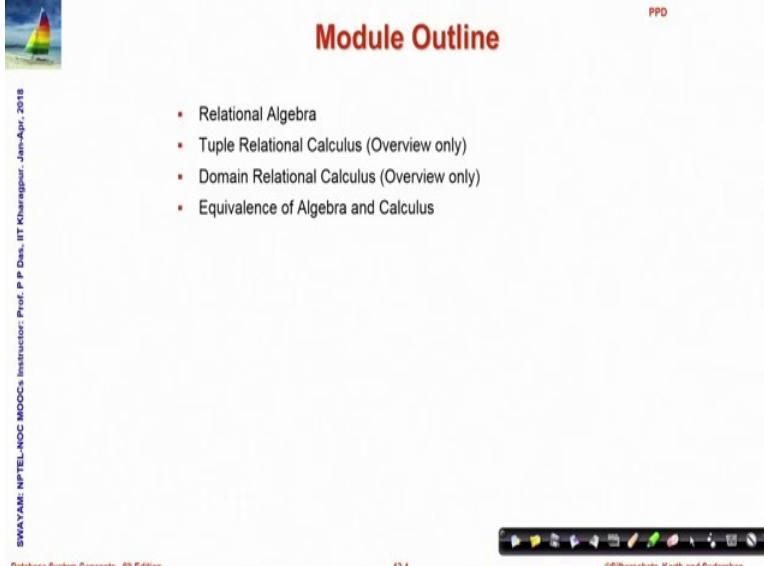
(Refer Slide Time: 01:10)



This slide is titled "Module Objectives" in red text at the top right. It features a small sailboat icon in the top left corner. On the far left, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs", "Instructor: Prof. P P Deshpande", and "Jan-Apr, 2018". In the top right corner, it says "PPD". A bulleted list under the title includes: "To understand formal query language through relational algebra". At the bottom, it shows "Database System Concepts - 8<sup>th</sup> Edition", "12.3", and "©Silberschatz, Korth and Sudarshan". A navigation bar with various icons is at the very bottom.

In the current 1 we will work to understand the formal query languages.

(Refer Slide Time: 01:19)



This slide is titled "Module Outline" in red text at the top right. It features a small sailboat icon in the top left corner. On the far left, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs", "Instructor: Prof. P P Deshpande", and "Jan-Apr, 2018". In the top right corner, it says "PPD". A bulleted list under the title includes: "Relational Algebra", "Tuple Relational Calculus (Overview only)", "Domain Relational Calculus (Overview only)", and "Equivalence of Algebra and Calculus". At the bottom, it shows "Database System Concepts - 8<sup>th</sup> Edition", "12.4", and "©Silberschatz, Korth and Sudarshan". A navigation bar with various icons is at the very bottom.

Primarily through relational algebra, and then we will also take a look into some of the calculus aspects tuple relational calculus, and domain relational calculus and we will show by example the equivalence between the algebra and the two calculus.

(Refer Slide Time: 01:40)

The slide has a header 'Formal Relational Query Language' with a sailboat icon. On the left, there is vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. The main content is a bulleted list:

- Relational Algebra
  - Procedural and Algebra based
- Tuple Relational Calculus
  - Non-Procedural and Predicate Calculus based
- Domain Relational Calculus
  - Non-Procedural and Predicate Calculus based

At the bottom, it says 'Database System Concepts - 8<sup>th</sup> Edition' and '12.5'. A video player interface shows a man speaking.

So, formal relational query languages are of 3 types 1 is known as relational algebra this is procedural in nature. So, we specify what operations need to be done to achieve the result and the whole formulation is based on set algebra. The second formal query language is tuple relational calculus which is non procedural and is based on predicate calculus. The third one the domain relational calculus is a minor variant of the tuple relational calculus is and is also non procedural and predicate calculus based.

(Refer Slide Time: 02:35)

The slide has a header 'RELATIONAL ALGEBRA' with a sailboat icon. On the left, there is vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. To the right, there is a list of topics:

- Relational Algebra
- Tuple Relational Calculus
- Domain Relational Calculus
- Equivalence of Algebra and Calculus

At the bottom, it says 'Database System Concepts - 8<sup>th</sup> Edition' and '12.6'. A video player interface shows a man speaking.

So, we start with the relational algebra in the relational algebra.

(Refer Slide Time: 02:40)



## Relational Algebra

PPD

- Created by Edgar F Codd at IBM in 1970
- Procedural language
- Six basic operators
  - select:  $\sigma$
  - project:  $\Pi$
  - union:  $\cup$
  - set difference:  $-$
  - Cartesian product:  $\times$
  - rename:  $\rho$
- The operators take one or two relations as inputs and produce a new relation as a result



SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018  
Database System Concepts - 8<sup>th</sup> Edition  
12.7 ©Silberschatz, Korth and Sudarshan

It was created by Edgar F Codd at IBM in 1970. So, you can see that it is quite an old formulation it is a procedural language it has six operators we have taken a quick view of these earlier in this module we will look at them at length. The select( $\sigma$ ) project( $\Pi$ ) union( $\cup$ ) set difference( $-$ ) Cartesian product( $\times$ ) and rename( $\rho$ ), we will also look at few derived operations like intersection and division which can be expressed in terms of these basic operators. And each one of these operators can take one or two relations as input and they produce one relation as a result.

(Refer Slide Time: 03:35)



## Select Operation

PPD

- Notation:  $\sigma_p(r)$
- $p$  is called the **selection predicate**
- Defined as:  
$$\sigma_p(r) = \{t \mid t \in r \text{ and } p(t)\}$$

where  $p$  is a formula in propositional calculus consisting of **terms** connected by :  $\wedge$  (and),  $\vee$  (or),  $\neg$  (not)

| A        | B        | C  | D  |
|----------|----------|----|----|
| $\alpha$ | $\alpha$ | 1  | 7  |
| $\alpha$ | $\beta$  | 5  | 7  |
| $\beta$  | $\beta$  | 12 | 3  |
| $\beta$  | $\beta$  | 23 | 10 |

Each **term** is one of:  
$$<\text{attribute}> \text{ op } <\text{attribute}> \text{ or } <\text{constant}>$$

where  $\text{op}$  is one of:  $=, \neq, >, \geq, <, \leq$

- Example of selection:  
$$\sigma_{\text{dept\_name}=\text{"Physics}}(\text{instructor})$$
  
$$\sigma_{A=B \wedge D > 5}(r)$$



SWAYAM: NPTEL: NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018  
Database System Concepts - 8<sup>th</sup> Edition  
12.8 ©Silberschatz, Korth and Sudarshan

So, we start with the select operation which you know has a notation  $\sigma_p$  where p is a predicate it is called the selection predicate and within parentheses we have a relation r (r) on which this predicate applies. So, it is defined as a set where you collect all the tuples all the rows designated by t and you specify that  $t \in r$ . So, it already exists in that relation and it satisfies the particular selection predicate.

So, any tuple that satisfies this predicate is included in the result any that does not satisfy is excluded from the result, p here particularly is a propositional calculus formula or expression. Where we have different terms that are connected by conjunction ( $\wedge$ ) or disjunction ( $\vee$ ) or negation ( $\neg$ ) or not, and each term by itself could be something like this it is an attribute operator and an attribute where operators are different comparisons operators one of the any six or a term could be an attribute operator a constant.

So, given that we can write any expression, which is a predicate and applying that we can select the tuples from the relation r which satisfy this predicate. So, here we show a simple example instructor is a relation, department\_name is an attribute within, course physics is a constant or literal. So, this selection show will select all the tuples where the attribute department name is equal to physics and all the others will be eliminated for reference I have also quoted here the example that we had shown at the time of introducing relational algebra.

So, you can see that here we have a more complex propositional term propositional formula where there are two terms, the  $a = b \wedge d > 5$ . So, in the selection result both of these conditions must be satisfied by all the tuples which feature here. So, this is the first operation that relational algebra has the select operation.

(Refer Slide Time: 06:40)

**Project Operation**

PPD

- Notation:  $\Pi_{A_1, A_2, \dots, A_k}(r)$   
where  $A_1, A_2$  are attribute names and  $r$  is a relation name
- The result is defined as the relation of  $k$  columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets
- Example: To eliminate the `dept_name` attribute of `instructor`

|          | A  | B | C |
|----------|----|---|---|
| $\alpha$ | 10 | 1 |   |
| $\alpha$ | 20 | 1 |   |
| $\beta$  | 30 | 1 |   |
| $\beta$  | 40 | 2 |   |

$$\Pi_{ID, name, salary}(instructor)$$

|          | A | C |
|----------|---|---|
| $\alpha$ | 1 |   |
| $\alpha$ | 1 |   |
| $\beta$  | 1 |   |
| $\beta$  | 2 |   |

$$=$$

|          | A | C |
|----------|---|---|
| $\alpha$ | 1 |   |
| $\beta$  | 1 |   |
| $\beta$  | 2 |   |

$$\Pi_{A,C}(r)$$

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - June-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
12.9  
©Silberschatz, Korth and Sudarshan

Next we move on to the second operation which is a project operation where a relation can be projected in terms of a number of attributes. So,  $\pi(\Pi)$  is the notation  $r$  again is the relation and the subscript at  $A_1, A_2, A_k$ , are key attribute names  $k$  has to be at least one and these attributes will be retained in the relation.  $\Pi_{A_1, A_2, A_k}$

So, the result is defined as the relation of  $k$  columns by erasing all the columns of  $r$  which are not listed amongst this  $A_1$  to  $A_k$ . Naturally, if you erase some columns it is possible that two rows that were distinct in those columns, but are identical in  $A_1$  to  $A_k$  feature in the relation since every relation is a set no distinct no two copies of the same people are allowed. So, the duplicate rows will be removed from the result remind you this is in contrast to what SQL does by default where duplicates or multi sets are allowed by default here we are talking about the formal relational algebra where it is purely set theoretic.

So, duplicate rows on projection will be removed from the result. So, we have an example from the instructor relation we had seen earlier we are projecting id name and salary( $\Pi_{id\_name, salary}$ ). So, we are removing the department name, which also exists in the same relation and as a reference you can see the example that we had seen earlier while introducing the relational algebra where projection is done from three columns A, B, C into two columns A and C and this results in at least results in two rows which are identical and therefore, in the final result one of those the duplicate one is removed.

(Refer Slide Time: 08:57)

**Union Operation**

PPD

|   |   |
|---|---|
| A | B |
| α | 1 |
| α | 2 |
| β | 1 |

|   |   |
|---|---|
| A | B |
| α | 2 |
| β | 3 |

r                            s

|   |   |
|---|---|
| A | B |
| α | 1 |
| α | 2 |
| β | 1 |
| β | 3 |

r ∪ s

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kanpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition      12.10      ©Silberschatz, Korth and Sudarshan

Moving on that the third operation is quite simple it is set theoretic union. So,  $r \cup s$  where  $r$  and  $s$  are two relations our set of peoples which either belong to  $r$  or belongs to  $s$ , or belongs to both, the condition is you can take union of both these relations have the same arity and the order of the attributes must satisfy that every corresponding attribute must have compatible domains. So, if we talk about the second column of  $r$ , and if we talk about the second column of  $s$  they must be of the same type and this must hold for all columns that the union forms that is all columns of  $r$  as well as  $s$  , otherwise this operation is not defined.

So, as an example we show that to find all courses taught in fall 2009 semester, this is a this is the query where we do a selection to find all tuples which are taught, which represent courses taught in fall 2009 semester, from the section relation we do a projection to get the ids of those courses only  $\Pi_{course\_id}(\sigma_{semester='Fall'} \wedge year=2009(section))$ . And the second row tells you the courses that are taught in the spring 2010 semester  $\Pi_{course\_id}(\sigma_{semester='Spring'} \wedge year=2010(section))$  , and we do a union to get courses that are taught either in fall 2009 semester, or in spring 2010 semester, or both. This is how the union is performed and this is the earlier example repeated here.

$$\Pi_{course\_id}(\sigma_{semester='Fall'} \wedge year=2009(section)) \cup \Pi_{course\_id}(\sigma_{semester='Spring'} \wedge year=2010(section))$$

(Refer Slide Time: 10:54)

**Set Difference Operation**

- Notation  $r - s$
- Defined as:  

$$r - s = \{t \mid t \in r \text{ and } t \notin s\}$$
- Set differences must be taken between **compatible** relations
  - $r$  and  $s$  must have the **same** arity
  - attribute domains of  $r$  and  $s$  must be compatible
- Example: to find all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

$$\Pi_{course\_id}(\sigma_{semester="Fall"} \wedge year=2009(section)) - \Pi_{course\_id}(\sigma_{semester="Spring"} \wedge year=2010(section))$$

|          |     |
|----------|-----|
| $A$      | $B$ |
| $\alpha$ | 1   |
| $\alpha$ | 2   |
| $\beta$  | 1   |

$r$

|          |     |
|----------|-----|
| $A$      | $B$ |
| $\alpha$ | 2   |
| $\beta$  | 3   |

$s$

|          |     |
|----------|-----|
| $A$      | $B$ |
| $\alpha$ | 1   |
| $\beta$  | 1   |

$r - s$

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kanpur Date: Jan-Apr., 2018  
Database System Concepts - 8<sup>th</sup> Edition      12.11      ©Silberschatz, Korth and Sudarshan

set difference is again just simple difference of sets  $r$  minus  $s$  where a tuple belongs to  $r$  and it does not belong to  $s$ . So, you remove all the tuples belonging to  $s$  that exist in  $r$  to get  $r - s$  again they must have the compatibility of having the same arity and attribute corresponding attribute domains must be compatible , this is an example to show to find all courses taught in fall 2009, but not in spring 2010.

$$\Pi_{course\_id}(\sigma_{semester="Fall"} \wedge year=2009(section)) - \Pi_{course\_id}(\sigma_{semester="Spring"} \wedge year=2010(section))$$

So, as opposed to union in the last slide you do a set difference to get this result. So, this is how you can use set theoretic operation to get different relational results this is also the result from the earlier example.

(Refer Slide Time: 11:49)

**Set-Intersection Operation**

- Notation:  $r \cap s$
- Defined as:  
 $r \cap s = \{t \mid t \in r \text{ and } t \in s\}$
- Assume:
  - $r, s$  have the same arity
  - attributes of  $r$  and  $s$  are compatible
- Note:  $r \cap s = r - (r - s)$

| A        | B |
|----------|---|
| $\alpha$ | 1 |
| $\alpha$ | 2 |
| $\beta$  | 1 |

$r$

| A        | B |
|----------|---|
| $\alpha$ | 2 |
| $\beta$  | 3 |

$s$

| A        | B |
|----------|---|
| $\alpha$ | 2 |

$r \cap s$

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018  
Database System Concepts - 8<sup>th</sup> Edition  
12.12  
©Silberschatz, Korth and Sudarshan

Set intersection can be supported is supported, but it is not a basic operation because as it is defined by all tuples which belong to both  $r$  and  $s$ . It can actually be computed by applying set difference twice  $r - (r - s)$  and certainly for set intersection also the same assumption about arity and compatibility of types hold and this is the earlier example.

(Refer Slide Time: 12:20)

**Cartesian-Product Operation**

- Notation  $r \times s$
- Defined as:  
 $r \times s = \{t q \mid t \in r \text{ and } q \in s\}$
- Assume that attributes of  $r(R)$  and  $s(S)$  are disjoint. (That is,  $R \cap S = \emptyset$ )
- If attributes of  $r(R)$  and  $s(S)$  are not disjoint, then renaming must be used

| A        | B |
|----------|---|
| $\alpha$ | 1 |
| $\beta$  | 2 |

$r$

| C        | D  | E |
|----------|----|---|
| $\alpha$ | 10 | a |
| $\beta$  | 10 | a |
| $\beta$  | 20 | b |
| $\gamma$ | 10 | b |

$s$

| A        | B | C        | D  | E |
|----------|---|----------|----|---|
| $\alpha$ | 1 | $\alpha$ | 10 | a |
| $\alpha$ | 1 | $\beta$  | 10 | a |
| $\alpha$ | 1 | $\beta$  | 20 | b |
| $\alpha$ | 1 | $\gamma$ | 10 | b |
| $\beta$  | 2 | $\alpha$ | 10 | a |
| $\beta$  | 2 | $\beta$  | 10 | a |
| $\beta$  | 2 | $\beta$  | 20 | b |
| $\beta$  | 2 | $\gamma$ | 10 | b |

$r \times s$

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018  
Database System Concepts - 8<sup>th</sup> Edition  
12.13  
©Silberschatz, Korth and Sudarshan

Next is Cartesian product, where we take two relations and for the Cartesian product we make we juxtapose one relation with the other. So,  $t$  is a tuple from  $r$  ( $t \in r$ ),  $q$  is a tuple from  $s$  ( $q \in s$ ) and we put them side by side to get a  $tq$  in the Cartesian

product  $r \times s$ , which basically means that you compute all possible combinations of pupils from  $r$  and of  $s$ . It is assumed that the attributes of  $r$  and  $s$  are disjoint that is a schema of  $r$  intersection schema of  $s$  is null.  $R \cap S = \emptyset$

If the attributes are not disjoint then we must use renaming which we will soon see, and here is an example that we had shown earlier of  $r$  and  $s$  computing  $r$  process, Cartesian product is a very useful operation particularly for computing join as we have seen in sql already.

(Refer Slide Time: 13:36)

The slide features a small sailboat icon in the top left corner. The title 'Rename Operation' is centered at the top in a red font. Below the title is a bulleted list of three points:

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.
- Example:

Below the list is the mathematical expression  $\rho_X(E)$  in black font. To its right, it says 'returns the expression  $E$  under the name  $X$ '. Further down, another expression  $\rho_{x(A_1, A_2, \dots, A_n)}(E)$  is shown, with the note 'returns the result of expression  $E$  under the name  $X$ , and with the attributes renamed to  $A_1, A_2, \dots, A_n$ '.

At the bottom of the slide, there is footer text: 'SWAYAM: NPTEL-NOC: Instructor: Prof. P. P. Deshpande, IIT Kanpur - Jan-Apr., 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '12.14', and '©Silberschatz, Korth and Sudarshan'.

Rename operation basically allows you to rename some expression attribute into another. So, the operator is  $P$  and you have an expression to which you give the name  $x$  and that is how the renaming of you can have multiple attributes of  $x$  as well.

(Refer Slide Time: 13:58)

PPD

**Division Operation**

- The division operation is applied to two relations
- $R(Z) \div S(X)$ , where  $X$  subset  $Z$ . Let  $Y = Z - X$  (and hence  $Z = X \cup Y$ ); that is, let  $Y$  be the set of attributes of  $R$  that are not attributes of  $S$

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
12.15  
©Silberschatz, Korth and Sudarshan

Division is another operator in relational algebra that can be applied between two relations, but it is a derived operation. So, it says that if I have, so let me just show you by, by a little bit of sketch that if I have two relations which has a set of attributes  $z$  and  $s$  which is a set of attribute  $x$ , such that actually the set  $z$  is a superset of  $x$ . So,  $z$  has more attribute the relation  $r$  has more attributes.

So, if you take the difference of attributes between  $z$  and  $x$  and call it  $y$ ,  $Y = Z - X$  then we are interested what happens on these remaining set of attributes  $y$ ..

(Refer Slide Time: 15:02)

PPD

**Division Operation**

- The division operation is applied to two relations
- $R(Z) \div S(X)$ , where  $X$  subset  $Z$ . Let  $Y = Z - X$  (and hence  $Z = X \cup Y$ ); that is, let  $Y$  be the set of attributes of  $R$  that are not attributes of  $S$

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
12.15  
©Silberschatz, Korth and Sudarshan

So, this is what we have this is my X set of attributes this is where X occurs this difference is Y this whole set is Z. Now in this what we want is we want to in the output we want a relation having only the y attribute such that for every tuple in that relation if I consider all the tuples of s then their cross product must be a part of r.

So, for every tuple here if there are say four tuples here, a tuple here must have all these four tuples along with it in the result. If it does not have any one or more of them then that tuple will not feature in the final result.

(Refer Slide Time: 15:58)

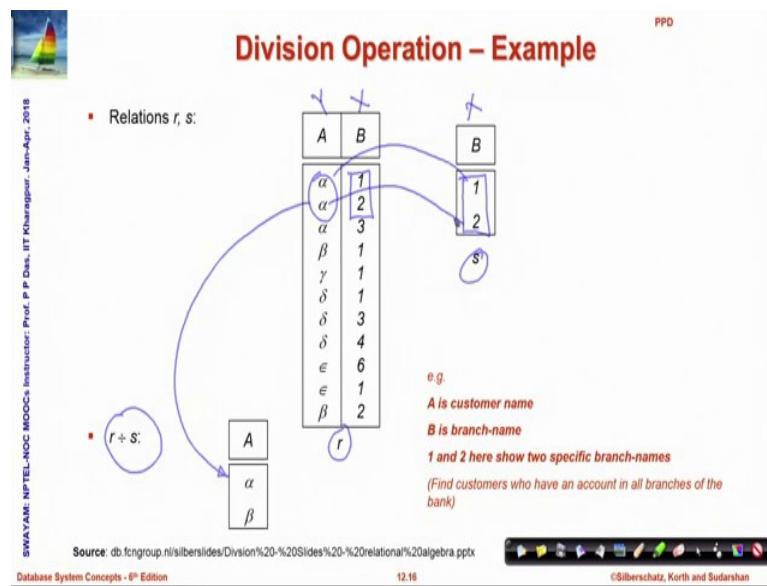
The slide has a header 'Division Operation' in red. On the left, there is a small logo of a sailboat on water. The right side contains a bulleted list of points about the division operation. At the bottom, there is footer information including the source 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018', the slide number '12.15', and copyright information '©Silberschatz, Korth and Sudarshan'.

- The division operation is applied to two relations
- $R(Z) \div S(X)$ , where  $X$  subset  $Z$ . Let  $Y = Z - X$  (and hence  $Z = X \cup Y$ ); that is, let  $Y$  be the set of attributes of  $R$  that are not attributes of  $S$
- The result of DIVISION is a relation  $T(Y)$  that includes a tuple  $t$  if tuples  $t_R$  appear in  $R$  with  $t_R[Y] = t$ , and with
  - $t_R[X] = t_s$  for every tuple  $t_s$  in  $S$ .
- For a tuple  $t$  to appear in the result  $T$  of the DIVISION, the values in  $t$  must appear in  $R$  in combination with every tuple in  $S$
- Division is a derived operation and can be expressed in terms of other operations

So, the result of a division is a relation  $T(Y)$  that include tuple  $t$  if tuples  $t_R$  that is the part of the tuple the tuple that appear in  $r$  and on that on the y part the difference part it matches. So, that you have that  $t_R[X] = t_s$  where  $t_s$  actually exist in  $s$ .

This must happen for all tuples in  $s$ , so division is a very good interesting operator which is often required for coding different queries. So, it is a derived operation.

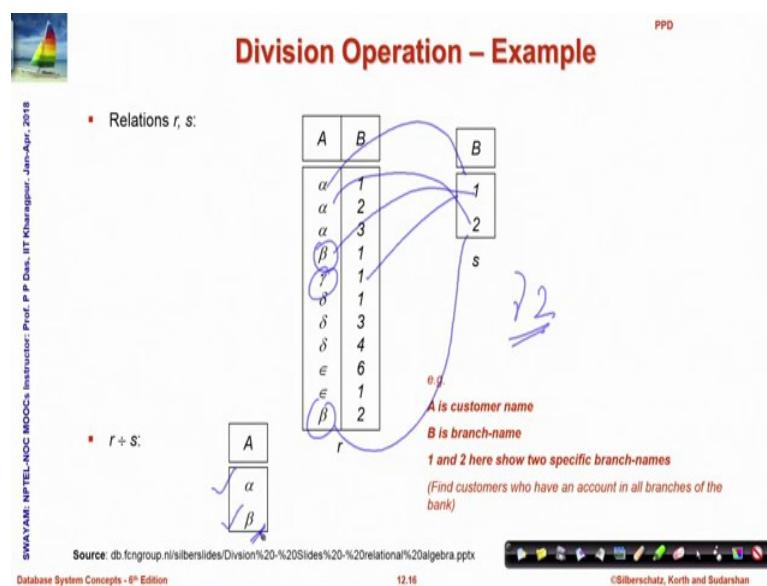
(Refer Slide Time: 16:44)



Let us take an example, let us say this is this is the relation  $r$  and this is a relation  $s$  and I am trying to compute  $r \div s$ .

So, what I want is over the attributes of this is therefore,  $X$  this is  $x$  this attribute  $y$  attribute set  $Y$ . So, all over  $X$  all the values that I have, I must have those values in the relation  $r$ , if I do then the attribute the particular tuple matching on the attribute  $Y$  goes onto the result. So,  $\alpha$  goes onto the result because you have both  $\alpha 1$  as well as  $\alpha 2$  in the set  $r$ , in the relation  $r$ .

(Refer Slide Time: 17:44)



$\beta_1$  is there and  $\beta_2$  is also there, so  $\beta$  also goes into the relation  $\alpha$  goes in because 1 is there 2 is also there, but  $\gamma$  will not go in because I have  $\gamma_1$ , but I do not have a tuple  $\gamma_2$  in  $r$  if I had  $\gamma_2$  in  $r$  that will go in the result.

(Refer Slide Time: 18:13)

**Division Operation – Example**

PPD

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018

- Relations  $r, s$ :

| A          | B |
|------------|---|
| $\alpha$   | 1 |
| $\alpha$   | 2 |
| $\alpha$   | 3 |
| $\beta$    | 1 |
| $\gamma$   | 1 |
| $\delta$   | 1 |
| $\delta$   | 3 |
| $\delta$   | 4 |
| $\epsilon$ | 6 |
| $\epsilon$ | 1 |
| $\beta$    | 2 |

- $r \div s$ :

| A        |
|----------|
| $\alpha$ |
| $\beta$  |

**s**

e.g.  
A is customer name  
B is branch-name  
1 and 2 here show two specific branch-names  
(Find customers who have an account in all branches of the bank)

Source: db.fcnetwork.ni/silberslides/Division%20-%20Slides%20-%20relational%20algebra.pptx

Database System Concepts - 8<sup>th</sup> Edition 12.16 ©Silberschatz, Korth and Sudarshan

So, if I can say it again the whole of the relation  $s$  must happen over the  $X$  attributes of  $r$ , consider these two together. If that happens then the attributes on  $y$  the tuples would be chosen and that is how we get the result having  $\alpha$  and  $\beta$ .

(Refer Slide Time: 16:47)

**Another Division Example**

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018

- Relations  $r, s$ :

| A        | B | C        | D | E |
|----------|---|----------|---|---|
| $\alpha$ | a | $\alpha$ | a | 1 |
| $\alpha$ | a | $\gamma$ | a | 1 |
| $\alpha$ | a | $\gamma$ | b | 1 |
| $\beta$  | a | $\gamma$ | a | 1 |
| $\beta$  | a | $\gamma$ | b | 3 |
| $\gamma$ | a | $\gamma$ | a | 1 |
| $\gamma$ | a | $\gamma$ | b | 1 |
| $\gamma$ | a | $\beta$  | b | 1 |

**r**

- $r \div s$ :

| A        | B | C        |
|----------|---|----------|
| $\alpha$ | a | $\gamma$ |
| $\gamma$ | a | $\gamma$ |

**s**

e.g.  
Students who have taken both "a" and "b" courses, with instructor "1"  
(Find students who have taken all courses given by instructor 1)

Source: db.fcnetwork.ni/silberslides/Division%20-%20Slides%20-%20relational%20algebra.pptx

Database System Concepts - 8<sup>th</sup> Edition 12.17 ©Silberschatz, Korth and Sudarshan

Let us look at one more example, so this is got r has five attributes this has X as two. So, this is this is two attributes X, these are three attributes Y and what I have to look for is those tuples in r where the values over Y would be same and I should be able to get the whole table of X over whole table of the relation S over the X attributes. So, if we look at here this is a 1, b 1, a 1, b 1, here these are identical.

So, this particular tuple will go into the result if I look in here this tuple will go into the result, but if I consider this tuple  $\beta$  a  $\gamma$  which has a 1 over d, but  $\beta$  a  $\gamma$  does not have b 1 it has b 3, so it will not go into the result. So, if you conceptually look at that is the reason this is called a division. So, you get this here you get this here. So, this is like the way we divide that this is the whole and wherever it goes in if the tuples that are identical on the wide set of attributes we will collect them into the final result.

(Refer Slide Time: 20:30)

**Another Division Example**

Relations  $r, s$ :

|          | A | B        | C        | D | E |
|----------|---|----------|----------|---|---|
| $\alpha$ | a | a        | $\alpha$ | a | 1 |
| $\alpha$ | a | a        | $\gamma$ | a | 1 |
| $\alpha$ | a | $\gamma$ | $\gamma$ | b | 1 |
| $\beta$  | a | $\gamma$ | a        | a | 1 |
| $\beta$  | a | $\gamma$ | $\gamma$ | b | 3 |
| $\gamma$ | a | $\gamma$ | $\gamma$ | a | 1 |
| $\gamma$ | a | $\gamma$ | $\gamma$ | b | 1 |
| $\gamma$ | a | $\beta$  | $\beta$  | b | 1 |

$r$

$r \div s$ :

|          | A | B        | C        |
|----------|---|----------|----------|
| $\alpha$ | a | a        | $\gamma$ |
| $\gamma$ | a | $\gamma$ | $\gamma$ |

**e.g.**  
Students who have taken both "a" and "b" courses, with instructor "1"  
(Find students who have taken all courses given by instructor 1)

Source: db.fnctgroup.nl/silberslides/Division%20-%20Slides%20-%20relational%20algebra.pptx  
Database System Concepts - 8<sup>th</sup> Edition  
12.17  
©Silberschatz, Korth and Sudarshan

So, this is the division operation which by which we can compute the students who have taken both a and b courses instructor 1 will be found out from this division operation.

(Refer Slide Time: 20:51)

The slide features a small sailboat icon in the top left corner. The title 'Formal Definition' is centered at the top in a red font. On the right side, there is a video frame showing a man speaking. The main content consists of two bullet points:

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation
- Let  $E_1$  and  $E_2$  be relational-algebra expressions; the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_p(E_1)$ ,  $P$  is a predicate on attributes in  $E_1$
  - $\Pi_S(E_1)$ ,  $S$  is a list consisting of some of the attributes in  $E_1$
  - $\rho_x(E_1)$ ,  $x$  is the new name for the result of  $E_1$

At the bottom left, it says 'Database System Concepts - 8<sup>th</sup> Edition'. In the bottom center, it shows '12.18'. At the bottom right, there is a 'CSlib' logo.

so formally speaking a basic expression in relational algebra consists either of a relation in the database, which is a instance or a constant relation which does not change which is given. And then we have six operations of union set difference cross product, selection projection, and renaming that can give us all sorts of different relational algebra formula and also the derived operations and whatever we have seen of sql can be expressed in terms of this relational algebra formula.

(Refer Slide Time: 21:30)

The slide features a small sailboat icon in the top left corner. The title 'TUPLE RELATIONAL CALCULUS' is centered at the top in a red font. On the right side, there is a video frame showing a man speaking. To the right of the video frame, there is a list of topics under the heading 'PPD':

- Relational Algebra
- Tuple Relational Calculus
- Domain Relational Calculus
- Equivalence of Algebra and Calculus

At the bottom left, it says 'Database System Concepts - 8<sup>th</sup> Edition'. In the bottom center, it shows '12.19'. At the bottom right, there is a 'CSlib' logo.

Now, relational algebra is not something totally unique the same thing can be done in terms of other formulations also.

(Refer Slide Time: 21:41)

The slide has a title 'Tuple Relational Calculus' in red at the top right. To the left of the title is a small image of a sailboat on water. On the far left edge of the slide, there is vertical text that reads 'SWAYAM: NPTEL-NOC MOOCs Initiator: Prof. P. P. Das, IIT Kanpur - Jan-Apr. 2018'. Below the title, there is a bulleted list of five points:

- A nonprocedural query language, where each query is of the form  $\{t \mid P(t)\}$
- It is the set of all tuples  $t$  such that predicate  $P$  is true for  $t$
- $t$  is a *tuple variable*,  $t[A]$  denotes the value of tuple  $t$  on attribute  $A$
- $t \in r$  denotes that tuple  $t$  is in relation  $r$
- $P$  is a *formula* similar to that of the predicate calculus

At the bottom of the slide, there is a video player interface showing a thumbnail of a person speaking, the text 'Database System Concepts - 8<sup>th</sup> Edition', the time '12:20', and the CSlib logo.

A second formulation which is also used is known as tuple relational calculus, which is non-procedural relational algebra was procedural because you are actually doing the explaining what the operations or you are detailing out what the operations are in tuple relational calculus you are specify what the condition is you are specifying what this condition is.

So, those tuples which satisfy this condition form the relation, so  $p$  is a predicate. So, whatever  $t$  satisfies the predicate are included and if  $a$  is an attribute then  $t[A]$  will denote the value of the tuple on attribute  $A$ ,  $A$  could be a single attribute it could be  $A$  set of attributes also and  $t \in r$ ,  $P$  is as I said it is a its a predicate calculus formula.

(Refer Slide Time: 22:43)

The slide features a sailboat icon in the top left corner. The title 'Predicate Calculus Formula' is centered at the top in red. Below the title is a numbered list of five items:

1. Set of attributes and constants
2. Set of comparison operators: (e.g.,  $<$ ,  $\leq$ ,  $=$ ,  $\neq$ ,  $>$ ,  $\geq$ )
3. Set of connectives: and ( $\wedge$ ), or ( $\vee$ ), not ( $\neg$ )
4. Implication ( $\Rightarrow$ ):  $x \Rightarrow y$ , if  $x$  is true, then  $y$  is true  
$$x \Rightarrow y \equiv \neg x \vee y$$
5. Set of quantifiers:
  - $\exists t \in r (Q(t))$  = "there exists" a tuple in  $t$  in relation  $r$  such that predicate  $Q(t)$  is true
  - $\forall t \in r (Q(t))$  =  $Q$  is true "for all" tuples  $t$  in relation  $r$

At the bottom left, vertical text reads: SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P.P. Deshpande, IIT Kharagpur. At the bottom center, it says Database System Concepts - 8<sup>th</sup> Edition and 12.21. On the right, there is a video player interface showing a video of a professor and the CSlib logo.

So, it could be a set of attributes or constants this I am just included for your help if in case you have become rusted with predicate calculus you can refer to a predicate calculus as a set of attributes and constant. It has set of comparison operators the six of them, there are a set of connectives these are all same as the propositional calculus there is  $\Rightarrow$  which says if  $x$  is true then  $y$  is true if  $x$  is false then the whole thing is true vacuously.

And what makes it primarily predicate calculus is a fact that it has existential quantifier, which says that the formula  $\exists t \in r Q(t)$  holds if I can find at least one tuple  $t$  which satisfies  $Q(t)$ .

Similarly, there is a universal quantifier where I will say that for all  $t \in r$ ,  $Q(t)$  is true if for all tuples of  $r$ ,  $t$  satisfies  $Q(t)$ . So, this in tuple relational calculus all conditions all predicates are formula of this kind and with that we can represent any.

(Refer Slide Time: 24:07)

The slide has a header 'Safety of Expressions' with a sailboat icon. The main content is a bulleted list:

- It is possible to write tuple calculus expressions that generate infinite relations
- For example,  $\{t \mid \neg t \in r\}$  results in an infinite relation if the domain of any attribute of relation  $r$  is infinite
- To guard against the problem, we restrict the set of allowable expressions to safe expressions
- An expression  $\{t \mid P(t)\}$  in the tuple relational calculus is *safe* if every component of  $t$  appears in one of the relations, tuples, or constants that appear in  $P$

NOTE: this is more than just a syntax condition

- E.g.  $\{t \mid t[A] = 5 \vee \text{true}\}$  is not safe --- it defines an infinite set with attribute values that do not appear in any relation or tuples or constants in  $P$

Navigation icons and footer text: SWAYAM-NPTEL-NOOC MOOCs Instructor: Prof. P.P. Desai, IIT Kanpur Date: June-Apr., 2018 Database System Concepts - 8<sup>th</sup> Edition 12.22 ©Silberschatz, Korth and Sudarshan

Relational set in full there is a word of caution because it is possible to write tuple relational calculus expression that can potentially generate infinite relations. Now, infinite relations are naturally not representable for example, if I write simply this that  $r$  is a relation and I write this predicate that  $\neg t \in r$ , which is basically complement set of  $r$  now a complement set of  $r$  potentially may be infinite if the domain is infinite..

So, such expressions tuple relational expressions are not acceptable as a part of the design. So, whenever we want to do this we would like to guard this by putting some additional condition and we have to make sure that any expression that we have in tuple relational calculus is a safe expression, in the sense that it does give me finite number of tuples in the relation.

(Refer Slide Time: 25:14)

The slide features a small sailboat icon at the top left. On the right side, there is a vertical list of topics under the heading 'PPD': Relational Algebra, Tuple Relational Calculus, Domain Relational Calculus, and Equivalence of Algebra and Calculus. The main title 'DOMAIN RELATIONAL CALCULUS' is centered in large red capital letters. Below the title, there is a navigation bar with icons for back, forward, search, and other presentation controls. At the bottom, it shows 'Database System Concepts - 8<sup>th</sup> Edition', the page number '12.23', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

A third formalism that exists that is used is known as domain relational calculus.

(Refer Slide Time: 25:25)

The slide has a sailboat icon at the top left. The main title 'Domain Relational Calculus' is centered in large red capital letters. Below the title, there is a bulleted list: 'A nonprocedural query language equivalent in power to the tuple relational calculus' and 'Each query is an expression of the form:'. To the right of the list is a mathematical expression:  $\{ \underline{x_1, x_2, \dots, x_n} | P(x_1, x_2, \dots, x_n) \}$ . Below the expression, there are two bullet points: 'x<sub>1</sub>, x<sub>2</sub>, ..., x<sub>n</sub> represent domain variables' and 'P represents a formula similar to that of the predicate calculus'. At the bottom right, there is a video frame showing a person speaking. The footer includes 'Database System Concepts - 8<sup>th</sup> Edition', the page number '12.24', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

Which is also non procedural and equivalent in power to tuple relational calculus again, which is very similar to tuple relational calculus the only difference being if you just recall tuple relational calculus. We are writing collection of tuples t such that P(t) that is the predicate P is satisfied by t here, instead of writing a tuple variable t we write expand it out in terms of all its components..

So, we write the values of the different components of  $t$  over different  $n$  attributes and write it as a  $n$  tuple and so here instead of having one variable  $t$  we have  $n$  variables  $\langle x_1, x_2, x_3, \dots, x_n \rangle$  and therefore, the predicate is formed of this  $n$  variables where  $x_1$  to  $x_n$  are represent the different domain values, domain variables and that leads to the reason for the name domain relational calculus.

(Refer Slide Time: 26:28)

The slide features a small sailboat icon in the top left corner. In the top right corner, the text 'PPD' is written above a bulleted list of topics:

- Relational Algebra
- Tuple Relational Calculus
- Domain Relational Calculus
- Equivalence of Algebra and Calculus

Below the title, there is a decorative footer bar with various icons. On the far left, vertical text reads 'SWAYAM NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018'. At the bottom left, it says 'Database System Concepts - 8<sup>th</sup> Edition'. In the center bottom, the page number '12.25' is shown. At the bottom right, it says '©Silberschatz, Korth and Sudarshan'.

## EQUIVALENCE OF ALGEBRA AND CALCULUS

Now, of the three formalisms that we have seen we will not go into direct mathematical proofs, but in the next couple of slides, I just show that they are equivalent in nature. What means that if I can write an expression in relational algebra then it is possible to write an equivalent expression in tuple relational calculus and in domain relational calculus and vice versa..

So, if I can write an expression in any one of these formalisms then there are equivalent expressions in the other two formalisms as well which is probably very easy to see between, tuple relational calculus and domain relational calculus because 1 is just representing the whole tuple as a single variable whereas, the other is representing it in terms of  $n$  domain variables.

So, their equivalence is pretty much very similar the fact that your predicate calculus formula has to change, but it is not, so obvious for the equivalence between relational algebra and the calculi.

(Refer Slide Time: 27:41)

The slide has a header 'Equivalence of RA, TRC and DRC' and a footer with a logo and text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018'.

**Select Operation:**

$R = (A, B)$

Relational Algebra:  $\sigma_{B=17}(r)$

Tuple Calculus:  $\{t \mid t \in r \wedge B = 17\}$

Domain Calculus:  $\{\langle a, b \rangle \mid \langle a, b \rangle \in r \wedge b = 17\}$

Source: [http://www.cs.sfu.ca/CourseCentral/354/louie/Equiv\\_Notes.pdf](http://www.cs.sfu.ca/CourseCentral/354/louie/Equiv_Notes.pdf)

Database System Concepts - 8<sup>th</sup> Edition

12.26

CSlib

A video player interface is visible on the right side of the slide.

So, we just show a few examples of the basic operations for example, say select operation. So, I am just not showing the proof, I am just giving you some example cases to show through a relation  $r$  has two attributes  $A, B$  this is what you wanted to write in relational algebra that you want to collect all tuples where  $b = 17$ .

Naturally in tuple relational calculus you can easily write the first condition is you are doing it on  $r$ . So,  $t$  must belong to  $r$  and your condition is  $B$  should be  $17$   $\{t \mid t \in r \wedge B = 17\}$ . So, this predicate will represent the same set or the same relation as in tuple calculus in domain calculus there are two components  $a$  and  $b$ . So, you have to say component taken together must belong to  $r$  and the component  $b$  must be equal to  $17$ .  $\{\langle a, b \rangle \mid \langle a, b \rangle \in r \wedge b = 17\}$

So, you can see that it is pretty straightforward to see the equivalence between a relational algebra expression involving select and the corresponding tuple calculus or domain calculus expressions this is through an example, but you can certainly easily generalize this as a proof.

(Refer Slide Time: 28:49)

The slide is titled "Equivalence of RA, TRC and DRC". It contains the following text and formulas:

*Project Operation*

$R = (A, B)$

Relational Algebra:  $\Pi_A(r)$

Annotation: A blue bracket underlines the entire formula  $\Pi_A(r)$ , and two blue arrows point down to the first letter 'r' and the symbol ' $\Pi_A$ '.

Tuple Calculus:  $\{t \mid \exists p \in r (t[A] = p[A])\}$

Annotation: A blue bracket underlines the entire formula, and two blue arrows point down to the first letter 't' and the symbol ' $\exists$ '.

Domain Calculus:  $\{<a> \mid \exists b (<a, b> \in r)\}$

Annotation: A blue bracket underlines the entire formula, and two blue arrows point down to the symbol ' $\exists$ ' and the letter 'b'.

Source: [http://www.cs.sfu.ca/CourseCentral/354/louie/Equiv\\_Notes.pdf](http://www.cs.sfu.ca/CourseCentral/354/louie/Equiv_Notes.pdf)

Database System Concepts - 8<sup>th</sup> Edition

12.27

cSlib

Similarly, for projection if we do a projection on a then all that we are trying to do is we are trying to create a new relation where only the a attribute exists. So, in the tuple t the a attribute exists and if I have projected and got this tuple t then in my original relation r there must be some tuple p such that on the attribute a they match they are same.

So, it is the same thing in relational algebra we said that keep a and erase everything else here we are saying that if we have been able to get a tuple t which has a value t[a] then there must be a tuple p in r, which has the same value over the same attribute. So, this condition is equivalent representative of the projection, and the same thing can be written in domain calculus you can go through it carefully and convince yourself.

(Refer Slide Time: 29:59)

The slide has a header 'Equivalence of RA, TRC and DRC'. It contains sections for 'Combining Operations' and 'Union'. The 'Combining Operations' section shows examples for Relational Algebra, Tuple Calculus, and Domain Calculus. The Relational Algebra example is  $\Pi_A(\sigma_{B=17}(r))$ . The Tuple Calculus example is  $\{t \mid \exists p \in r (t[A] = p[A] \wedge p[B] = 17)\}$ . The Domain Calculus example is  $\{\langle a \rangle \mid \exists b (\langle a, b \rangle \in r \wedge b = 17)\}$ . The 'Union' section is partially visible below. A video player interface at the bottom right shows a video of a professor speaking.

You can combine this as in the relational algebra as well as in tuple calculus. So, here you apply one relation one operation and then the other one selection then projection here you are combining this is part of projection this is also part of projection, but this condition has come from the selection and get a total predicate calculus predicate which will give you the tuple calculus expression for this combined expression of relational algebra , domain calculus will certainly happen in a similar manner.

(Refer Slide Time: 30:34)

The slide has a header 'Equivalence of RA, TRC and DRC'. It contains sections for 'Combining Operations' and 'Union'. The 'Union' section shows examples for Relational Algebra, Tuple Calculus, and Domain Calculus. The Relational Algebra example is  $r \cup s$ . The Tuple Calculus example is  $\{t \mid t \in r \vee t \in s\}$ . The Domain Calculus example is  $\{\langle a, b, c \rangle \mid \langle a, b, c \rangle \in r \vee \langle a, b, c \rangle \in s\}$ . A video player interface at the bottom right shows a video of a professor speaking.

Union certainly straightforward, so you can do it yourself.

(Refer Slide Time: 30:40)

The slide is titled "Equivalence of RA, TRC and DRC". It features a small sailboat icon in the top left corner and a "PPD" logo in the top right corner. The main content is organized into sections for "Set Difference", "Intersection", and "Union".

**Set Difference**

$R = (A, B, C) \quad S = (A, B, C)$

Relational Algebra:  $r - s$

Tuple Calculus:  $\{t \mid t \in r \wedge t \notin s\}$

Domain Calculus:  $\{\langle a, b, c \rangle \mid \langle a, b, c \rangle \in r \wedge \langle a, b, c \rangle \notin s\}$

Source: [http://www.cs.sfu.ca/CourseCentral/354/louie/Equiv\\_Note.pdf](http://www.cs.sfu.ca/CourseCentral/354/louie/Equiv_Note.pdf)

Database System Concepts - 8<sup>th</sup> Edition      12.30      CSlib

Set difference is again very straight forward because that is. In fact, in set theoretically whatever operations we have their relational algebraic definition itself is a tuple calculus formula you can expand them out and write in the domain calculus as well.

(Refer Slide Time: 30:57)

The slide is titled "Equivalence of RA, TRC and DRC". It features a small sailboat icon in the top left corner and a "PPD" logo in the top right corner. The main content is organized into sections for "Intersection", "Union", and "Difference".

**Intersection**

$R = (A, B, C) \quad S = (A, B, C)$

Relational Algebra:  $r \cap s$

Tuple Calculus:  $\{t \mid t \in r \wedge t \in s\}$

Domain Calculus:  $\{\langle a, b, c \rangle \mid \langle a, b, c \rangle \in r \wedge \langle a, b, c \rangle \in s\}$

Source: [http://www.cs.sfu.ca/CourseCentral/354/louie/Equiv\\_Note.pdf](http://www.cs.sfu.ca/CourseCentral/354/louie/Equiv_Note.pdf)

Database System Concepts - 8<sup>th</sup> Edition      12.31      CSlib

Intersection plays out in the same way tuples that belong to both  $r$  and  $s$ .

(Refer Slide Time: 31:04)

The slide has a header 'Equivalence of RA, TRC and DRC' with a small logo of a sailboat on the left and 'PPD' on the right. On the left margin, vertical text reads 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018'. Below the title, 'Cartesian/Cross Product' is defined with the formula  $R = (A, B) \quad S = (C, D)$ . Under 'Relational Algebra:', it shows  $r \times s$  above the expression  $\{t \mid \exists p \in r \exists q \in s (t[A] = p[A] \wedge t[B] = p[B] \wedge t[C] = q[C] \wedge t[D] = q[D])\}$ . Under 'Tuple Calculus:', it shows the same expression with the condition  $t[B] = p[B]$  underlined. Under 'Domain Calculus:', it shows the expression  $\{(a, b, c, d) \mid (a, b) \in r \wedge (c, d) \in s\}$ . At the bottom, it says 'Source: http://www.cs.sfu.ca/CourseCentral/354/louie/Equiv\_Notes.pdf' and '12.32'. A video player interface is at the bottom right.

Cartesian product is a little bit more involved because all that you are saying here is if I have a Cartesian product then if that product has a tuple  $t$ . Then there must be a tuple  $p$  in the relation  $r$  ( $p \in r$ ) the left relation there must be a tuple  $q$  in the in  $s$  ( $q \in s$ ) the right relation. So, that the final tuple  $t$  matches  $p$  on the  $a$  attributes the  $b$  attributes that is attributes of relation  $r$  and the components of  $t$  matches the tuple  $q$  in the attributes of  $s$ .

If all these conditions happen together then naturally this tuple  $t$  is a valid tuple for the Cartesian product. So, you could take examples and work this out and convince yourself that these are really equivalent.

(Refer Slide Time: 31:58)

PPD

## Equivalence of RA, TRC and DRC

### Natural Join

R = (A, B, C, D) S = (B, D, E)

Relational Algebra:  $r \bowtie s$

$$\Pi_{r.A,r.B,r.C,r.D,s.E}(\sigma_{r.B=s.B \wedge r.D=s.D}(r \times s))$$

Tuple Calculus:  $\{t \mid \exists p \in r \exists q \in s (t[A] = p[A] \wedge t[B] = p[B] \wedge t[C] = p[C] \wedge t[D] = p[D] \wedge t[E] = q[E] \wedge p[B] = q[B] \wedge p[D] = q[D])\}$

Domain Calculus:  $\{\langle a, b, c, d, e \rangle \mid \langle a, b, c, d \rangle \in r \wedge \langle b, d, e \rangle \in s\}$

Source: [http://www.cs.sfu.ca/CourseCentral/354/louie/Equiv\\_NaturalJoin.html](http://www.cs.sfu.ca/CourseCentral/354/louie/Equiv_NaturalJoin.html)

Database System Concepts - 8<sup>th</sup> Edition      12.33      ©Silberschatz, Korth and Sudarshan

We can define a natural join in a similar way, which I will leave it as an exercise for you to convince yourself that this relational algebra expression for natural join indeed has similar equivalents in tuple and domain calculi.

(Refer Slide Time: 32:17)

PPD

## Equivalence of RA, TRC and DRC

### Division

R = (A, B) S = (B)

Relational Algebra:  $r \div s$

Tuple Calculus:  $\{t \mid \exists p \in r \forall q \in s (p[B] = q[B] \Rightarrow t[A] = p[A])\}$

Domain Calculus:  $\{\langle a \rangle \mid \langle a \rangle \in r \wedge \forall \langle b \rangle (\langle b \rangle \in s \Rightarrow \langle a, b \rangle \in r)\}$

Source: [http://www.cs.sfu.ca/CourseCentral/354/louie/Equiv\\_Division.html](http://www.cs.sfu.ca/CourseCentral/354/louie/Equiv_Division.html)

Database System Concepts - 8<sup>th</sup> Edition      12.34      ©Silberschatz, Korth and Sudarshan

Division we just showed as a derived operation, we have not showed how in relational algebra you can write division using the other operations. I will leave that as an exercise as well, but here what I show is in tuple calculus how you can write division using the quantifiers.

Here you can see that here for the first time we do need to use the universal quantifier to make sure that while I divide that the whole of the table of s must be available against the part of the tuple part of the y attributes as we said that will be collected in the result.

(Refer Slide Time: 33:02)

Module Summary

- Discussed relational algebra with examples
- Introduced tuple relational and domain relational calculus
- Illustrated equivalence of algebra and calculus

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

12.35

CSlib

So, to summarize we have discussed primarily the relational algebra with some examples, we have introduced the tuple relational and domain relational calculus and through a set of examples. We have shown that we have illustrated that the algebra and the calculi are equivalent and I would request you to work out more examples to understand the equivalence, or if you are really enthused please try out proving their equivalence formally as well.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 13**  
**Entity-Relationship Model/1**

Welcome to module 13 of Database Management Systems. In this module and the next 2 we will discuss about Entity Relationship Model.

(Refer Slide Time: 00:34)

The slide is titled "Module Recap" in red at the top right. It features a small sailboat icon on the left. A vertical watermark on the left side reads "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur, Jan-Apr, 2018". At the bottom left is the course title "Database System Concepts". The bottom right contains copyright information "©Silberschatz, Korth and Sudarshan". The main content is a bulleted list of topics:

- Relational Algebra
- Tuple Relational Calculus (Overview only)
- Domain Relational Calculus (Overview only)
- Equivalence of Algebra and Calculus

At the very bottom center is the number "13.2".

So, far we have had a good look into the SQL language, the query language and it is formal basis in terms of relational algebra and calculi.

(Refer Slide Time: 00:44)

The slide is titled "Module Objectives" in red text at the top center. In the top right corner, there is a small logo with the letters "PPD". On the left side, there is a small image of a sailboat on water. The bottom left corner contains the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur, Jan-Apr. 2018" and "Database System Concepts". The bottom right corner shows a video player interface with a play button, volume controls, and a timestamp "13.3". A small video frame in the bottom right shows a man with glasses speaking.

- To understand the Design Process for Database Systems
- To study the E-R Model for real world representation

In this module we will try to understand the design process for database systems, because so far whatever we have done we have assumed that the schema is known to us, that some instance is given to us and then we have tried to extract different query information from the relation; but now we will look into how do we model the real world and actually get into the design process.

So, after an overview of the design process we would study entity relationship model, which is used to represent the real world whatever exists in the real world that will have to be represented for our use and final representation in terms of different relations.

(Refer Slide Time: 01:32)



## Module Outline

PPD

- Design Process
- E-R Model
  - Entity and Entity Set
  - Relationship
    - Cardinality
  - Attributes
  - Weak Entity Sets

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr. 2018

Database System Concepts

13.4



The design process at an abstract level.

(Refer Slide Time: 01:43)



## Design Phases

- The initial phase of database design is to characterize fully the data needs of the prospective database users
- Next, the designer chooses a data model and, by applying the concepts of the chosen data model, translates these requirements into a conceptual schema of the database
- A fully developed conceptual schema also indicates the functional requirements of the enterprise. In a "specification of functional requirements", users describe the kinds of operations (or transactions) that will be performed on the data

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr. 2018

Database System Concepts

13.6



The initial phase of database design certainly has to characterize what data is required to be maintained for an enterprise. So, whether I am doing, if I am doing an university database naturally we will need to identify that what are the data needs the students need to be described, the instructors need to be described, the courses sections time slots grades examinations etc; but if I am trying to deal with an world which is say railway

reservation, then I will need to deal with stations trains date (Refer Time: 02:26) the different classes of coach that the train has and so on.

So, the initial phase is to characterize the data requirement next the designer has to choose a data model because, unless we can we cannot deal with a natural language or English kind of description and work towards getting a particular schema.

So, we will need to use a data model and apply the concepts of the data model that we choose and translate the requirements into what is known as a conceptual schema of the database which is not a not a very concrete 1. But, a conceptual one this is what grossly what I want to do and a fully developed conceptual schema will indicate my functional requirements, in terms of what usually is called a specification of functional requirements system requirements. If it will specify what kind of users will be involved what kinds of operations transactions will be performed and so on.

(Refer Slide Time: 03:43)

The slide is titled "Design Phases (Cont.)" in red. On the left, there is a small image of a sailboat on water. The right side contains text and a list of bullet points. At the bottom right is a video player showing a person speaking.

The process of moving from an abstract data model to the implementation of the database proceeds in two final design phases.

- Logical Design – Deciding on the database schema.  
Database design requires that we find a "good" collection of relation schemas.
  - Business decision – What attributes should we record in the database?
  - Computer Science decision – What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
- Physical Design – Deciding on the physical layout of the database

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur, Jan-Apr. 2018  
Database System Concepts

Now, once we have that kind of a conceptual model that abstract that is a conceptual more abstract data model, we will go to the next phase of the design which is finding out what is the more concrete design through a process of logical design. In the process of logical design we will first decide on the database schema, we need to decide on what is a good schema.

So, there are principles to say that what is good and what is not so good, we need to make business decisions to find out which attributes we record in the database; we need to make computer science decision as to how the relational schemas will be interrelated between themselves, how the attributes will be distributed and at a last phase we need to also decide on the physical design which will tell us what is the physical layout of the data.

So, conceptual design refined into logical design finalized with physical design is our gross process of design.

(Refer Slide Time: 05:06)

The slide has a title 'Design Approaches' in red at the top right. On the left is a small image of a sailboat on water. The main content is a bulleted list under the heading 'Entity Relationship Model (covered in this chapter)'. The list includes:

- Entity Relationship Model (covered in this chapter)
  - Models an enterprise as a collection of *entities* and *relationships*
    - Entity: a "thing" or "object" in the enterprise that is distinguishable from other objects
      - Described by a set of *attributes*
    - Relationship: an association among several entities
    - Represented diagrammatically by an *entity-relationship diagram*:
  - Normalization Theory (Chapter 8)
    - Formalize what designs are bad, and test for them

At the bottom left is the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. At the bottom center is 'Database System Concepts' and '13.8'. At the bottom right is a video player showing a man speaking, with the text 'CS101' next to it.

Now, in this for the conceptual design we primarily follow a model called entity relationship model; that tries to identify the collection of entities and relationships. An entity is nothing, but it is an object is a thing that is distinguishable from other objects. So, if I say that student is an entity then the student is distinguishable from another entity course both of them are distinguishable from a third entity instructor and so on.

So, every entity for the purpose of distinction is described by a set of attributes or properties and these entities will have relations between them. For example, you can say that a course will be attended by students; students will be advised by instructors. So, this attended by advised by these are relationships or association between several entities and the model which represents initially diagrammatically and then in textual form this kind

of relationship is known as the entity relationship model or the entity relationship diagram.

We will then use it to get a relational set of relational schema which subsequently we normalize; the normalization is nothing but refinement of the design which improves a design to make it better in terms of correctness, in terms of ease of manipulation performance and so on. So, it basically removes bad designs from the database and converts them into good designs; we will talk about this normalization theory later in the course. Right now we are interested only in the entity relationship model, which will be used for conceptual design and then will give us the basis for the logical design in terms of the schemas.

So, let us take a deeper look into the entity relationship model and entity relationship model as I said is developed to facilitate the database design.

(Refer Slide Time: 07:45)

**ER model – Database Modeling**

The ER data model was developed to facilitate database design by allowing specification of an enterprise schema that represents the overall logical structure of a database

The ER model is very useful in mapping the meanings and interactions of real-world enterprises onto a conceptual schema. Because of this usefulness, many database-design tools draw on concepts from the ER model

The ER data model employs three basic concepts:

- entity sets
- relationship sets
- attributes

The ER model also has an associated diagrammatic representation, the ER diagram, which can express the overall logical structure of a database graphically

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Dass, IIT Kharagpur. Jan-Apr. 2018  
Database System Concepts 13.10 ©Sib

Get the overall logical structure it is useful in mapping the meaning and interactions of the real world in terms of certain diagrammatic schemas and it employs 3 basic concepts entities or entity sets we talked about entities, all entities that share the same set of properties like if student is an entity, then the collection of student is an entity set a instructor is an entity collection of instructors is an entity set.

So, all entities in an entity set will share the same set of attributes, will have relationship sets which define relationship between multiple entity sets and certainly in the process will make use of attributes. These are the 3 key components of an ER model it also has a ER diagram as we will show soon.

(Refer Slide Time: 08:51)

The slide features a small sailboat icon in the top-left corner. The title 'Entity Sets' is centered at the top in a red font. On the left edge, there is vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof P P Das, IIT Kharagpur, Jan-Apr. 2018'. The main content area contains a bulleted list of definitions and examples:

- An **entity** is an object that exists and is distinguishable from other objects.
  - Example: specific person, company, event, plant
- An **entity set** is a set of entities of the same type that share the same properties.
  - Example: set of all persons, companies, trees, holidays
- An entity is represented by a set of attributes; i.e., descriptive properties possessed by all members of an entity set.
  - Example:  
 $\text{instructor} = (\text{ID}, \text{name}, \text{street}, \text{city}, \text{salary})$   
 $\text{course} = (\text{course\_id}, \text{title}, \text{credits})$
- A subset of the attributes form a **primary key** of the entity set; i.e., uniquely identifying each member of the set.

At the bottom right, there is a video player interface showing a video of a man speaking, with controls for play, pause, and volume. The bottom of the slide has a navigation bar with icons for back, forward, and search, and the text 'Database System Concepts' and '13.11'.

So, as already defined entity is an object that exist and is distinguishable from other objects, entity set is a set of entities of the same type that share the same properties and an entities is represented by the set of attributes or properties that describe it.

So, when we say instructive for example, if we say here these are my attributes you have already learned this in terms of studying SQL. So, it has there is 5 attributes and these 5 attributes together or the values of these 5 attributes for a particular instructor defines my entity set instructor, collection of these attributes define my entity set courses. So, these are my different entity sets that exist that can be defined.

So, a subset of attributes in the entity set forms a key called the primary key, which can uniquely identify every entity in that entity set we have already been familiar with this concept of primary key the same concept continues.

(Refer Slide Time: 10:00)



## Entity Sets – *instructor* and *student*

| instructor_ID     | instructor_name | student-ID                                                                                                                                                                                                                                                                                                                                                                                                                   | student_name |
|-------------------|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|
| 76766             | Crick           | 98988                                                                                                                                                                                                                                                                                                                                                                                                                        | Tanaka       |
| 45565             | Katz            | 12345                                                                                                                                                                                                                                                                                                                                                                                                                        | Shankar      |
| 10101             | Srinivasan      | 00128                                                                                                                                                                                                                                                                                                                                                                                                                        | Zhang        |
| 98345             | Kim             | 76543                                                                                                                                                                                                                                                                                                                                                                                                                        | Brown        |
| 76543             | Singh           | 76653                                                                                                                                                                                                                                                                                                                                                                                                                        | Aoi          |
| 22222             | Einstein        | 23121                                                                                                                                                                                                                                                                                                                                                                                                                        | Chavez       |
| <i>instructor</i> |                 | <i>student</i>                                                                                                                                                                                                                                                                                                                                                                                                               |              |
|                   |                 |      |              |

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur, Jan-Apr, 2018



So, these are examples of entity sets instructor with 2 attributes and student with 2 attributes as well.

(Refer Slide Time: 10:14)

**Relationship Sets**

- A **relationship** is an association among several entities

Example:

|                 |                  |                   |
|-----------------|------------------|-------------------|
| 44553 (Peltier) | <i>advisor</i>   | 22222 (Einstein)  |
| student entity  | relationship set | instructor entity |

- A **relationship set** is a mathematical relation among  $n \geq 2$  entities, each taken from entity sets

$$\{(e_1, e_2, \dots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \dots, e_n \in E_n\}$$

where  $(e_1, e_2, \dots, e_n)$  is a relationship

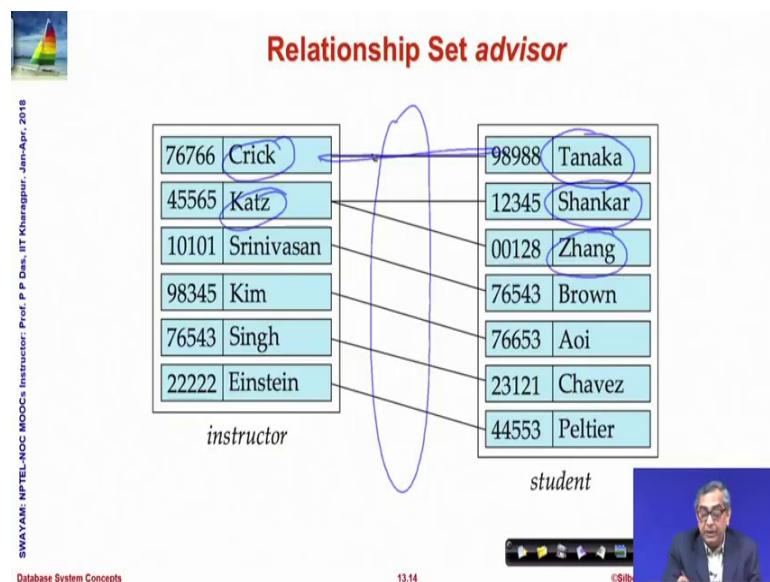
- Example:  
 $(44553, 22222) \in \text{advisor}$

A relationship is an association among two or more entities, so here we have an entity here shown as a student this is a student entity, identified by the student id which is a primary key in the student entity set. We have an instance of an instructor entity identified by the id of the instructor Einstein, which identifies any instructor uniquely and then adviser is a relationship set which relates these 2.

So, what I want to mean is if I say adviser relates 44553 to 2222, what I want to mean is peltier the student peltier is advised by the instructor Einstein. So, whenever we relate two or more entity sets like this we get relationships. So, a relationship is a mathematical relation Emma more than two or more entities, each taken from the entity set.

So, you can see that it can have components  $e_1 e_2 \dots e_n$ ,  $n$  entity sets and each entity  $e_i$  should belong to entity set capital  $E_1$ ,  $e_2$  should belong to entity set capital  $E_2$  and so on and is called a relationship we have already seen the advisor relationship as above.  
 $\{(e_1, e_2, e_3, e_4, \dots, e_n) | e_1 \in E_1, e_2 \in E_2, e_3 \in E_3, \dots\}$

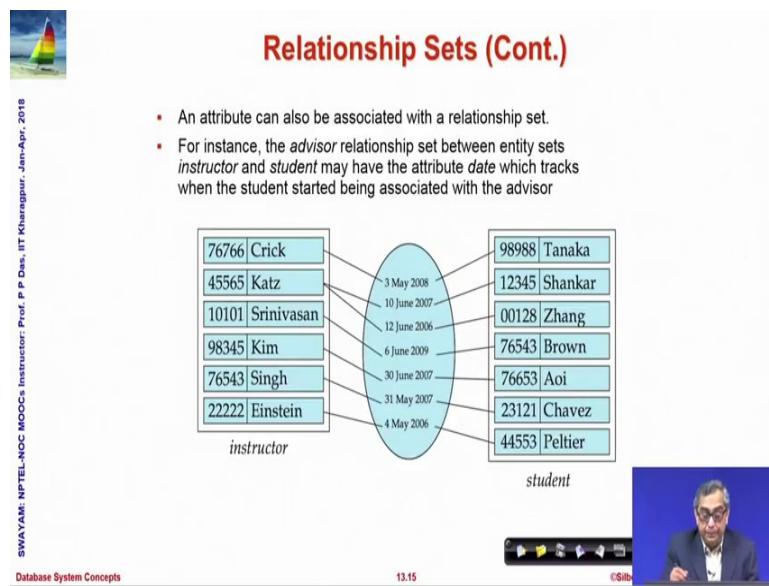
(Refer Slide Time: 11:58)



So, here what we show is a relationship advisor by these arrows these lines. So, what we are showing is this connection between these two, show that this student is advised by this instructor where as you can see. So, crick advises Tanaka where as Shankar and Zhang both are advised by Katz.

So, this group of associations between instructor and student is the gives me the relationship adviser as to who advises whom.

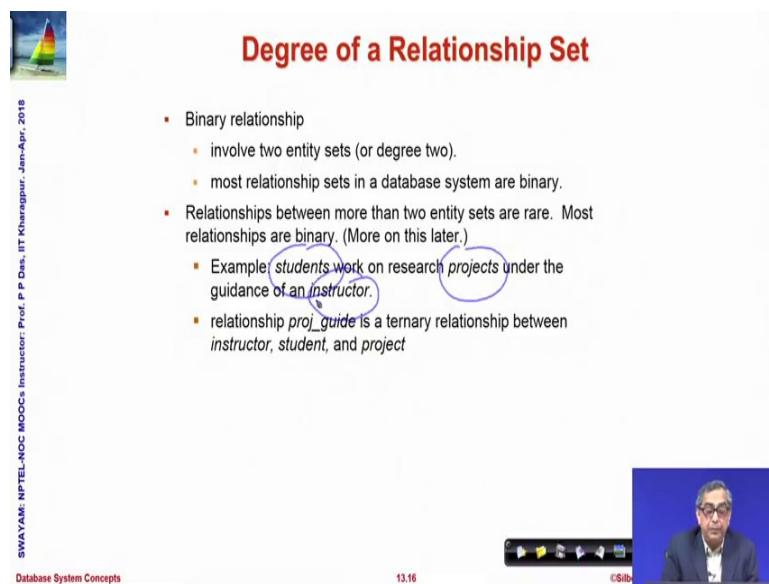
(Refer Slide Time: 12:39)



A relationship also like the entity sets the relationship also can have some additional attribute. For example, when I say that crick advises Tanaka I may associate an attribute date type attribute set third May 2018, to mean that when did this process of crick advising Tanaka started, we can it can be some other attribute also.

So, all that I am trying to highlight is attributes can be assigned to relationships as well.

(Refer Slide Time: 13:15)



Now, how will a relationship span out, we have said that a relationship must involve two entity sets. So, primarily relationships are binary it involves two and most relationships

in most databases are binary in nature, but it could be that there are we will see later that there are possibilities of having relationships which are more than binary ternary and higher.

So, here are example students works on research projects under the guidance of an instructor. So, here we have as you can see student's research projects and instructors, so there are 3 entity sets. So, if I want to maintain a relationship of say project guidance between them then that turns out to be a ternary relationship we will talk about this more later.

(Refer Slide Time: 14:16)

**Mapping Cardinality Constraints**

- Express the number of entities to which another entity can be associated via a relationship set.
- Most useful in describing binary relationship sets.

Diagram illustrating cardinality constraints:

Hand-drawn diagram showing two overlapping circles labeled  $E_1$  and  $E_2$ . Arrows point from various points in  $E_1$  to various points in  $E_2$ , representing associations between the two sets.

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT-Kharagpur, Jan-Apr. 2018  
Database System Concepts  
13:17 CS101

There are constraints in terms of the cardinality of the relationship, the cardinality basically talks of that when we have when I have a relation entity set  $E_1$  and identity set  $E_2$ .

So, there are different entities in them and I have different associations between them, then the question is how many of the entity of one entity set is related to how many of the entities of the other entity set and certain types of cardinality measures are very important.

(Refer Slide Time: 14:58)

The slide features a sailboat icon in the top left corner. The title 'Mapping Cardinality Constraints' is centered at the top in red. On the left, there is vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018'. Below the title is a bulleted list of four points:

- Express the number of entities to which another entity can be associated via a relationship set.
- Most useful in describing binary relationship sets.
- For a binary relationship set the mapping cardinality must be one of the following types:
  - One to one
  - One to many
  - Many to one
  - Many to many

At the bottom, there is a navigation bar with icons for back, forward, search, and other controls, along with the text 'Database System Concepts' and '13.17'.

To track and we say it is whether it is 1 to 1 to many many to one or many to many.

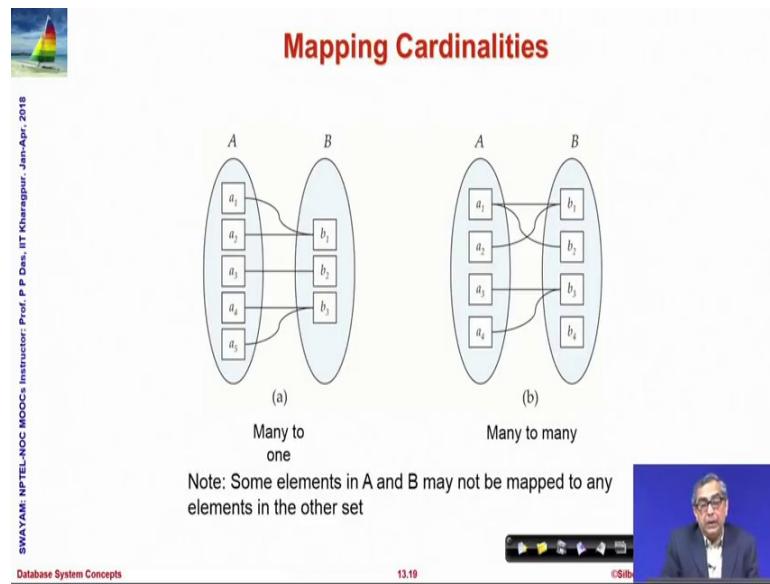
(Refer Slide Time: 15:08)

The slide features a sailboat icon in the top left corner. The title 'Mapping Cardinalities' is centered at the top in red. On the left, there is vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018'. Below the title are two diagrams labeled (a) and (b).  
Diagram (a) illustrates a 'One to one' relationship. It shows two ovals, A and B. Oval A contains four boxes labeled  $a_1, a_2, a_3, a_4$ . Oval B contains three boxes labeled  $b_1, b_2, b_3$ . Arrows connect  $a_1$  to  $b_1$ ,  $a_2$  to  $b_2$ ,  $a_3$  to  $b_3$ , and  $a_4$  to  $b_1$ .  
Diagram (b) illustrates a 'One to many' relationship. It shows two ovals, A and B. Oval A contains four boxes labeled  $a_1, a_2, a_3, a_4$ . Oval B contains five boxes labeled  $b_1, b_2, b_3, b_4, b_5$ . Arrows connect  $a_1$  to  $b_1$  and  $b_2$ ;  $a_2$  to  $b_3$  and  $b_4$ ;  $a_3$  to  $b_5$ ; and  $a_4$  to  $b_1$ .  
Below the diagrams, the text 'One to one' and 'One to many' is centered under their respective diagrams. At the bottom, there is a note: 'Note: Some elements in A and B may not be mapped to any elements in the other set'. There is also a navigation bar with icons for back, forward, search, and other controls, along with the text 'Database System Concepts' and '13.18'.

So, here the examples or the schematics, so in the first one in the diagram A you see that every entity from the entity set A relates to exactly one entity in the entity set B or you can say at most one entity in then they decide B similarly every entity in entity set B relates to exactly one entity in entity set A or at most 1 entity and entity set a if this holds then we say this relationship is one to one whereas, in diagram B you see that a 1 relates to b 1 as well as b 2 a 2 relates to b 3 as well as b4.

So, 1 entity in A relates to more than 1 entity may relate to more than 1 entity in B, but if you look from B side every entity in B is related to at most 1 entity in A then we say from A to B it is 1 to many.

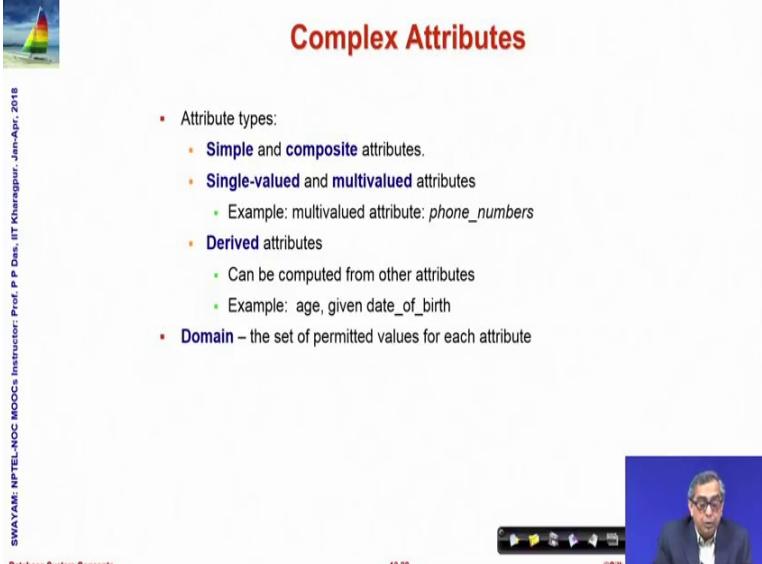
(Refer Slide Time: 16:04)



Now naturally since I can put the relations in any order as we have one to many if you look in the other direction it becomes many to one. So, many to one is from A to B many to one is where more than one entity in set A may relate to one entity inside B, but all entities in set B relates to at most one entity inside A and when there is no restriction at all that is any number of entities in set A may relate to any number of entities inside B and any number of entities in set B may relate to any number of entities in set A we say it is a many to many relation.

So, we have one too many one to one, we have one to many and many to one and we have many to many and it often helps in the design to be able to characterize which type of relationship we do have.

(Refer Slide Time: 16:51)



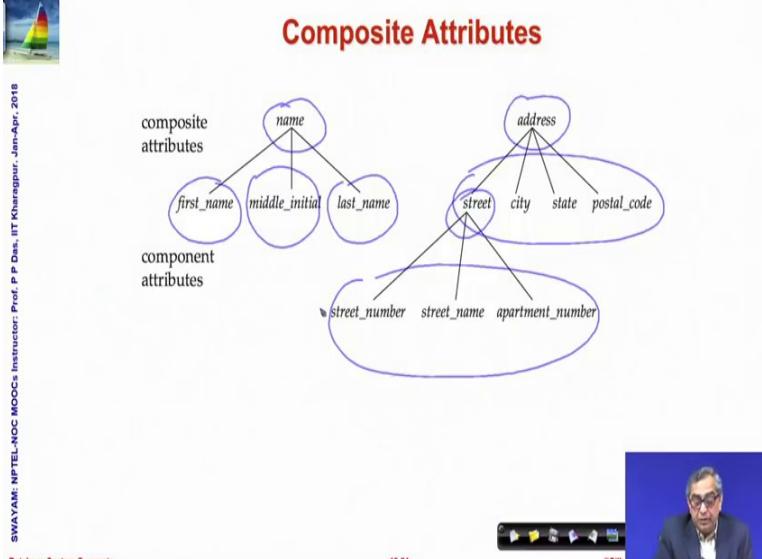
The slide is titled "Complex Attributes" in red. On the left, there is a vertical sidebar with the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018". At the bottom of the sidebar, it says "Database System Concepts". In the center, there is a list of attribute types:

- Attribute types:
  - Simple and composite attributes.
  - Single-valued and multivalued attributes
    - Example: multivalued attribute: `phone_numbers`
  - Derived attributes
    - Can be computed from other attributes
    - Example: age, given date\_of\_birth
  - Domain – the set of permitted values for each attribute

On the right side of the slide, there is a video player interface showing a video of a professor speaking. The video player has controls for play, pause, and volume. The time is indicated as 13.20. The video is from the channel "OSlib".

Coming to the attributes we can note that attributes are of different types, one is they could be simple or composite a simple attribute is just one single domain value like a salary number like an id like a name string and so on; whereas, a composite attribute may comprise of multiple parts.

(Refer Slide Time: 17:16)



The slide is titled "Composite Attributes" in red. On the left, there is a vertical sidebar with the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018". At the bottom of the sidebar, it says "Database System Concepts". In the center, there is a diagram illustrating composite attributes. It shows two main composite attributes: "name" and "address". The "name" attribute is composed of three component attributes: "first\_name", "middle\_initial", and "last\_name". The "address" attribute is composed of four component attributes: "street", "city", "state", and "postal\_code". The "street" attribute is further decomposed into "street\_number", "street\_name", and "apartment\_number". Arrows point from the composite attributes to their respective components, and from the components to their sub-components.

On the right side of the slide, there is a video player interface showing a video of a professor speaking. The video player has controls for play, pause, and volume. The time is indicated as 13.21. The video is from the channel "OSlib".

So consider this, this is a composite attribute, so name is an attribute if I think of then it has different parts, it is a first name middle name last name if I think of address it has. So, many different parts then street itself has so many different parts. So, whenever an

attribute is comprised some more of the components, when it is not a simple value then it is called a composite attribute we will see how to handle them; then some attributes may be single valued, for example a person has a has 1 name let us say, but has one address, but may have 2 or more phone numbers, the attributes which can take more than 1 value is known to be multi valued attribute.

So, we also need to specify whether certain specifying in the design whether certain attributes are single valued or multiple valued multi valued; of course, single value attributes are easy to deal with if it is multi valued we need to do some design changes. Certain attributes can be derived for example age, now I cannot keep the age of some a person in the database because, with every day the age changes. So, what we typically keep is a date of birth and the age is computed on the day when the particular query is made to find out what the age is so it is called a derived attribute and each 1 of them will have corresponding set of domains.

(Refer Slide Time: 18:53)

SWAYAM NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur. Date: Apr - 2018

**Redundant Attributes**

- Suppose we have entity sets:
  - *instructor*, with attributes: *ID, name, dept\_name, salary*
  - *department*, with attributes: *dept\_name, building, budget*

inst\_dept

Database System Concepts 13.22 ©Silb

Some attributes in the design may turn out to be redundant also consider this you have already seen this this is an instructor, which has a department name along with the different attributes and certainly I have a department table, so which department relation which gives the details of the department. Now since every instructor belongs to a department, so naturally we might want to have a relation *inst\_department* which could give the instructor and his or her department name. So, if we maintain that then this

becomes a redundant attribute this is not required, because if that information is already there in this relation.

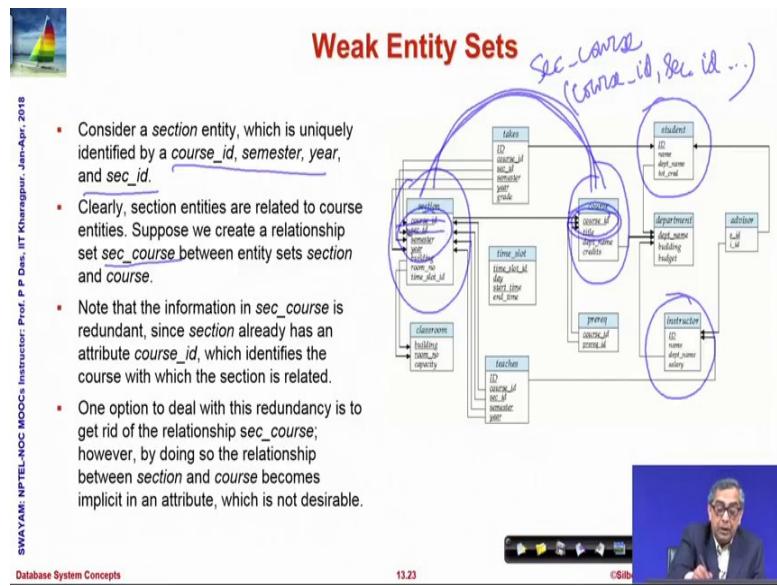
(Refer Slide Time: 19:49)

The slide has a decorative header featuring a sailboat on water. The title 'Redundant Attributes' is centered in red. On the left, vertical text reads: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P.P. Das, IIT Kharagpur - Jan-Apr. 2018'. At the bottom left is the text 'Database System Concepts'. In the bottom right corner is a video player showing a man speaking, with the time '13.22' and a 'CSlib' logo.

- Suppose we have entity sets:
  - *instructor*, with attributes: *ID, name, dept\_name, salary*
  - *department*, with attributes: *dept\_name, building, budget*
- We model the fact that each instructor has an associated department using a relationship set *inst\_dept*
- The attribute *dept\_name* appears in both entity sets. Since it is the primary key for the entity set *department*, it replicates information present in the relationship and is therefore redundant in the entity set *instructor* and needs to be removed
- BUT: when converting back to tables, in some cases the attribute gets reintroduced, as we will see later

So, in several cases there is a question of whether we maintain some information in terms of a relation or we can make that directly include that directly in the entity set and get rid of that relation. So, if I have the *inst\_dept* relation and then the attribute department name appears on both these sets, *inst\_dept* as well as on the *instructor* and there is duplication replication of the data which we want to avoid. But we will see the different cases when which style of design, whether we would be better to maintain the department name as a part of the *instructor* relation or it would be better not to have it there and have a separate relation which maps *instructor id* against the department name.

(Refer Slide Time: 20:48)



Finally comes a concept of weak entity sets you need to understand this a little bit, consider the university database example. So, we have courses we have students we have instructors and we have section, a section is if a course is large then it needs to be taught in multiple sections. So, for the same course at the same semester in the same year I may have different sections, in which the students are divided and naturally there could be multiple instructors each teaching 1 section of that course and students will be distributed on the sections not on the course.

Now, consider this section entity, if you look into this then this is how what we maintained we did a course id semester year and section id, but if you look into specifically and if you want to now know you know that there is a section and there is a course. So, you may want to relate these two section with the course and set up an entity between them. So, what will it relate it will relate the course id of the course with all of these, but the course id is already there as a part of the section right. So, you would say that well it is not required to have the course id, since it already has that and it it identifies it.

So, we can can we remove this course id from here, well if we remove the course id now we have a different problem. If you remove the course id then you have section id semester and year. But this does not uniquely represent the tuples of this relation

because, there could be 2 section as in the same semester in the same year for 2 different courses how do you distinguish them.

So, you get into a situation where the course the section gets identified uniquely provided, either you know the relationship between the section and the course in terms of the sec\_course relationship or you include the primary key of course, into the relation section which we did in the design and this is not a coincidence this is something which happens regularly and is the characteristics that specify the existence of weak entity sets.

(Refer Slide Time: 24:11)

**Weak Entity Sets (Cont.)**

- An alternative way to deal with this redundancy is to not store the attribute `course_id` in the `section` entity and to only store the remaining attributes `section_id`, `year`, and `semester`. However, the entity set `section` then does not have enough attributes to identify a particular `section` entity uniquely; although each `section` entity is distinct, sections for different courses may share the same `section_id`, `year`, and `semester`.
- To deal with this problem, we treat the relationship `sec_course` as a special relationship that provides extra information, in this case, the `course_id` required to identify `section` entities uniquely.
- The notion of **weak entity set** formalizes the above intuition. A weak entity set is one whose existence is dependent on another entity, called its **identifying entity**; instead of associating a primary key with a weak entity, we use the identifying entity, along with extra attributes called **discriminator** to uniquely identify a weak entity. An entity set that is not a weak entity set is termed a **strong entity set**.

So, the weak entity set is one whose existence depends on another entity set. So, if I just say section having section id year and semester then it is not uniquely specified, until I have a relation section course which relates the section to the particular course id. When such relationships are used to identify entities of a particular entity set, then the unique side the course side which is unique is known as the identifying entity and the other attributes in this case section id year semester are known as the discriminators.

So, we have a relationship between a weak entity set which is section, we have a strong entity set which is course why is it strong because course is identified by course id itself. Section is not unless you have a set course kind of relationship set between the course and the section which specifies that well, for this course this is the section this is the year this is the semester. So, this is the identifying entity through which the entities of this set

gets specified and whenever that situation happens then we say that we have a weak entity set.

(Refer Slide Time: 26:21)

The slide has a title 'Weak Entity Sets (Cont.)' in red. To the left is a small image of a sailboat on water. On the right is a video player showing a man in a suit. The video player has a progress bar at 13.25 and some control icons. The background of the slide is white.

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts

Weak Entity Sets (Cont.)

- Every weak entity must be associated with an identifying entity; that is, the weak entity set is said to be **existence dependent** on the identifying entity set. The identifying entity set is said to **own** the weak entity set that it identifies. The relationship associating the weak entity set with the identifying entity set is called the **identifying relationship**.
- Note that the relational schema we eventually create from the entity set *section* does have the attribute *course\_id*, for reasons that will become clear later, even though we have dropped the attribute *course\_id* from the entity set *section*.

So, weak entity sets naturally cannot happen by themselves their existence dependent on identifying entity set and the identifying entity set owns the weak entity set. So, the courses in that way own the section and the identifying relationship between them is necessary to uniquely identify every entity of this weak entity set or section in our case.

So, this notion is very important for the design as we will see that the relational schema that we eventually created, in this case from the entity set section we did include course id as a part of the primary key not using the sec course kind of relationship and we will show how this design style for dealing with entity weak entity sets influences the different database designs. So, weak entity sets are critical notions that you need to be aware of need to be confident of.

(Refer Slide Time: 27:34)

- Introduced the Design Process for Database Systems
- Elucidated the E-R Model for real world representation with entities, entity sets, relationships, etc.

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts

13.26

©Silberschatz, Korth and Sudarshan

So, in summary we have introduced the design process for database systems I will just quickly recap, the first stage is identifying the data items which is leading to the conceptual design, which will primarily do in terms of the entity relationship model identifying the entities the entity sets, the attributes that define the entity set, describe the entity set, the subset of attributes forming primary key that uniquely specifies every entity set, every entity in the entity set and the relationships typically binary may be non binary also, relationships that hold between the different entity sets.

So, this is the conceptual design that will lead to more detail logical design of how the relationship should be organized, what is the cardinality of that, what kind of attributes do I have, whether it is simple whether it is composite whether certain attributes are derived or not. So, all those are different aspects will have to be detailed out and we need to identify what are the weak entity sets and what are the strong entity sets, what are the identifying entities and with that we could complete the logical design and then we will need to make it in terms of it express it in terms of a relational schema.

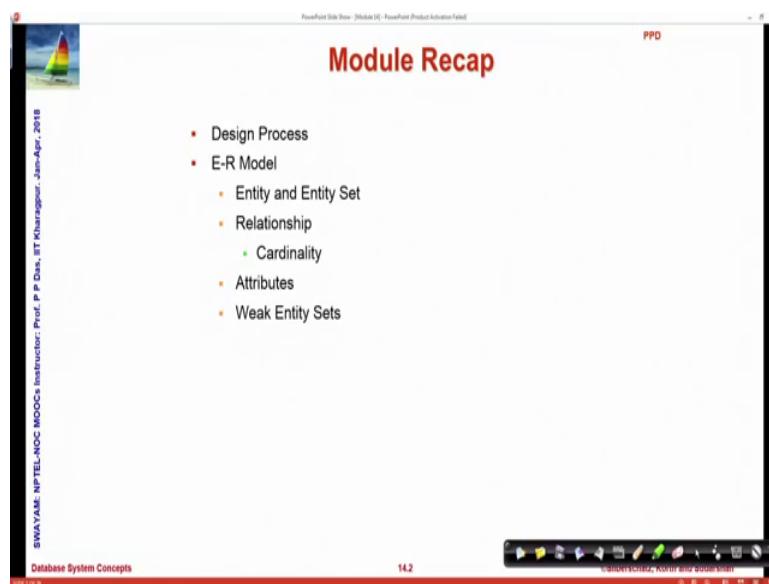
So, in this module we have just taken a look in the first part the entity relationship model and the very basic of how the conceptual design. We will go forward, in the model we have seen all the different primitives required to represent the reality represent what holds in the real world.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 14**  
**Entity-Relationship Model/2**

Welcome to module 14 of database management systems. In the last module we will started our discussions on the entity relationship model.

(Refer Slide Time: 00:29)



The screenshot shows a PowerPoint slide with the title "Module Recap" in red. The slide content is a bulleted list of topics:

- Design Process
- E-R Model
  - Entity and Entity Set
  - Relationship
    - Cardinality
  - Attributes
  - Weak Entity Sets

At the bottom left, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018". At the bottom right, it says "Database System Concepts" and "14.2". The top right corner has a small "PPD" label.

We will continue that in this module as well, and actually conclude it in the next module. So, these are the items that we had discussed in the last module.

(Refer Slide Time: 00:39)

The screenshot shows a PowerPoint slide with the title 'Module Objectives' in red at the top center. Below the title is a bulleted list of objectives:

- To illustrate E-R Diagram notation for E-R Models
- To explore translation of E-R Models to Relational Schemas

On the left side of the slide, there is vertical text: 'SWAYAM-NPTEL-MOOCs Instructor: Prof. P. P. Das, ST Kharegpur - Jan-Apr- 2018' and 'Database System Concepts'. At the bottom right, there is a video player showing a man speaking, with the number '143' next to it.

In the present one, we will first illustrate the entity relationship diagram notation; that graphical notation for E-R model, how nicely this can be shown in terms of certain diagrams. And then we will explore how E-R models can be translated to relational schemas, which is a basic step of the logical design.

(Refer Slide Time: 01:11)

The screenshot shows a PowerPoint slide with the title 'Module Outline' in red at the top center. Below the title is a bulleted list of topics:

- E-R Diagram
- E-R Model to Relational Schema

On the left side of the slide, there is vertical text: 'SWAYAM-NPTEL-MOOCs Instructor: Prof. P. P. Das, ST Kharegpur - Jan-Apr- 2018' and 'Database System Concepts'. At the bottom right, there is a video player showing a man speaking, with the number '144' next to it.

So, these are the topics. So, we start with the E-R diagram.

(Refer Slide Time: 01:16)

**Entity Sets**

- Entities can be represented graphically as follows:
  - Rectangles represent entity sets.
  - Attributes listed inside entity rectangle
  - Underline indicates primary key attributes

|                   |                 |
|-------------------|-----------------|
| <i>instructor</i> | <i>student</i>  |
| <u>ID</u>         | <u>ID</u>       |
| <i>name</i>       | <i>name</i>     |
| <i>salary</i>     | <i>tot_cred</i> |

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, ST Khartoum - Jan-Apr., 2018  
Database System Concepts

Naturally the first thing to represent in an E-R model is the entity set. Every entity set is represented by a rectangle. On the top, we write the name of that entity set so you can see examples here the instructor. And student are the two entity sets and below that we write the names of the attributes that are involved. And we underline the attribute or attributes that form the primary key of that entity set.

(Refer Slide Time: 01:49)

**Relationship Sets**

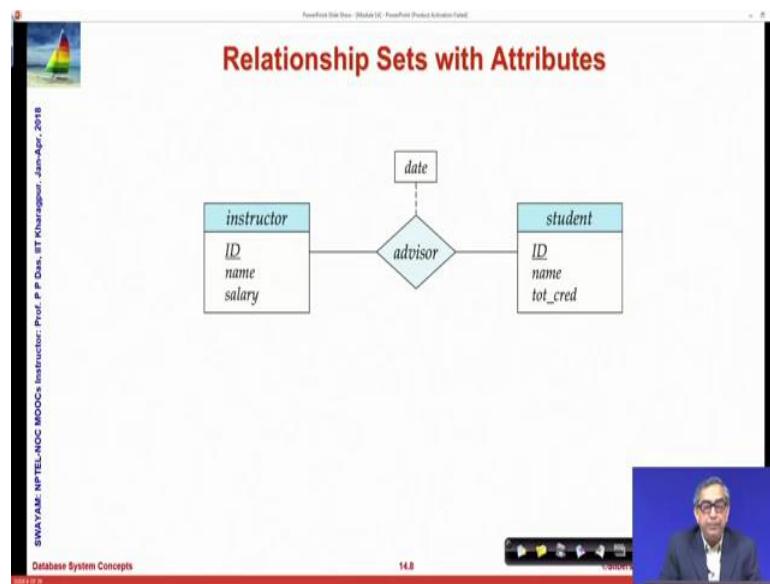
- Diamonds represent relationship sets.

|                   |                 |
|-------------------|-----------------|
| <i>instructor</i> | <i>student</i>  |
| <u>ID</u>         | <u>ID</u>       |
| <i>name</i>       | <i>name</i>     |
| <i>salary</i>     | <i>tot_cred</i> |

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, ST Khartoum - Jan-Apr., 2018  
Database System Concepts

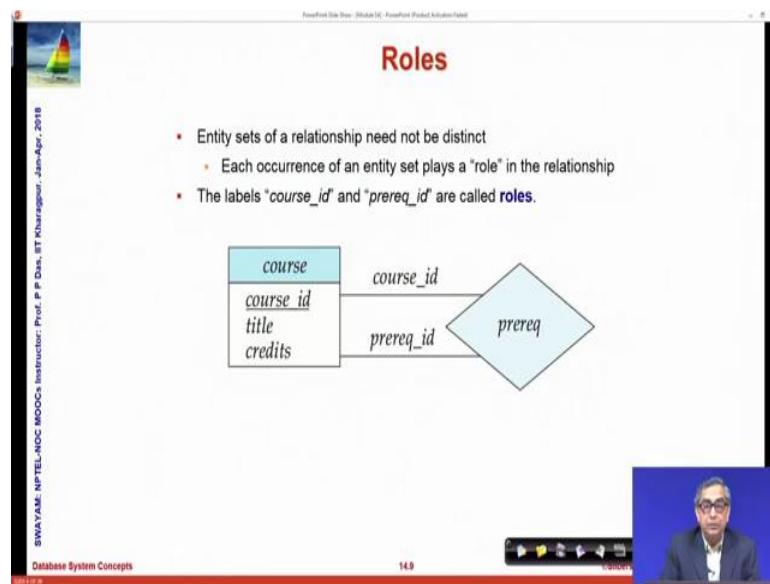
A relationship between two entity sets is represented by a diamond, and 2 connecting lines to the 2 entity sets. So, here it says that advisor is a relationship between entity set instructor, and entity set student. Trying to convey the real-world situation that students are advised by the instructors or students have instructors and so on.

(Refer Slide Time: 02:23)



As we had mentioned that relationships could also have attributes. So, if the advisor relationship has an attribute date then it will be tagged to the adviser relationship with the attribute coming as a within a rectangle and attached to the name of the relationship by a dotted line. So, this shows that advisor is the relationship between instructor. And student and the adviser relationship has an attribute date.

(Refer Slide Time: 02:58)



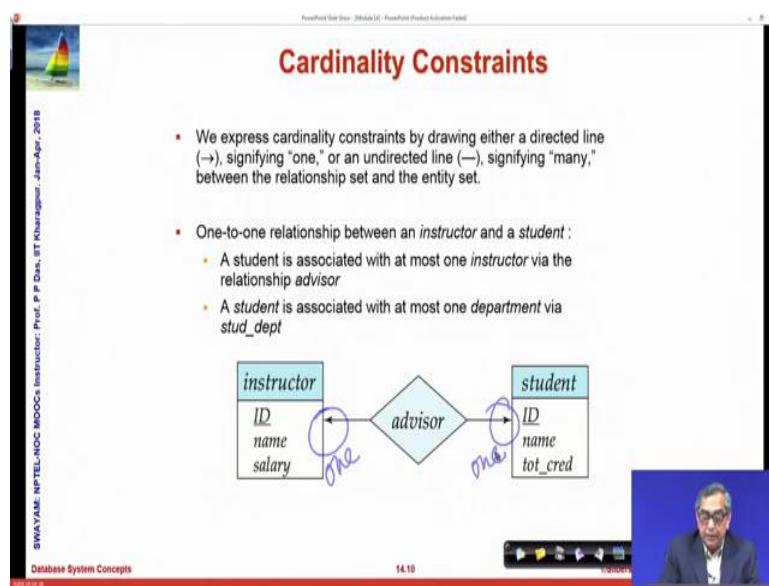
It is possible that the relationship that hold between 2 entity sets can be can use entity sets which are same. That is, it is possible that a set is related to itself.

So, as an example we show, the entity set course which has a relationship prereq, prerequisite which takes a course id, and relates it to another course id called the prerequisite id. Because obviously, if a course has a prerequisite. Then that prerequisite itself is another course id which must occur in this table itself.

So, in this case, if you can see that unlike the earlier case the prereq id is not actually a field of this relation course. So, we say these are rules. So, we say the rule that prereq relate from the course relation to itself are course id and prereq id; where in the actual table both of them relate to course id, but prereq will pair them to show which course has what prerequisite. And we often need this kind of; we have seen similar instances of this while we treated dealt with the recursive queries in databases.

The source destination of aligns problem that we discussed has similar kind of relationship structure. So, you can think about a relationship flies from the set of source to this set of destination, and basically this; these 2 sets the places source places in the destination place are necessarily the same set the same relation. There could be a constraint on the cardinality.

(Refer Slide Time: 05:07)



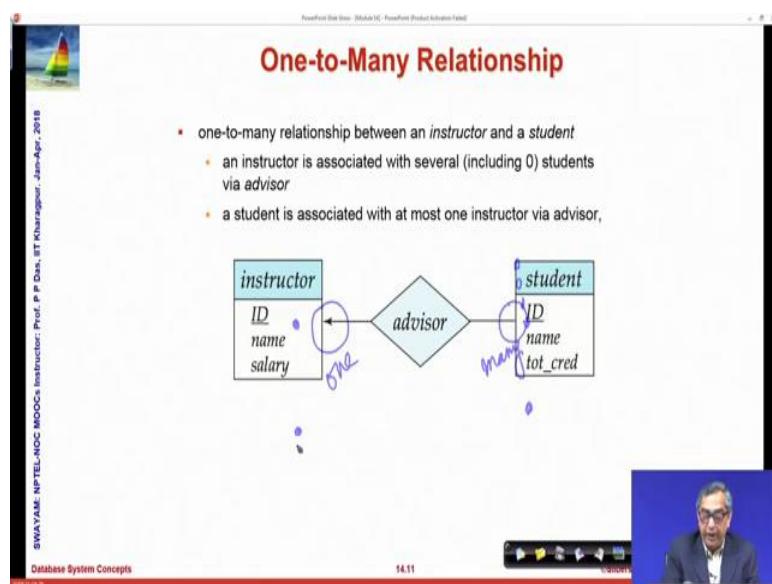
So, the line that links the relationship diamond with the rectangles of the relations rectangles of the entity sets. Those lines could have an arrow at the end or may not have an arrow at the end. So, if it has got an arrow, then it means that suppose it has an arrow it means 1. And if it does not have a arrow if it is simple then it means many. So, using

this rotation we can designate one to one to many these all different kinds of cardinalities that we had discussed.

So, for example, if we are showing this arrow on both hands, both ends of this relationship advisor then, it means that it is a one to one relationship because there is an arrow here. So, this is 1. There is an arrow here. So, this is 1. There is an arrow here. So, this is 1; which means that a student is associated at most with at most one instructor. And it also means that an instructor is associated with at most one student.

This may not be a reality this usually is not the reality, but this we are just showing this as an example. So, if the student instructor relationship advisor relationship is one to one, then this is how we will denote it.

(Refer Slide Time: 06:54)



If it is one to many; so, which side is 1, this side is 1 and this side is many. So, it is one to many from instructor to student; which says that every student has at most one instructor. And an instructor may have several students. It could be null also it could be none also.

So, for an instructor here there are many students, but for a student here there are only at most one instructor. So, it is one to many and this is how we designate.

(Refer Slide Time: 07:35)

The slide is titled "Many-to-One Relationships". It contains a bulleted list and a diagram. The list describes a many-to-one relationship between a student and an instructor, stating that an instructor is associated with at most one student via advisor, and a student is associated with several (including 0) instructors via advisor. The diagram shows two tables, "instructor" and "student", connected by a diamond-shaped relationship named "advisor". The "instructor" table has attributes ID, name, and salary. The "student" table has attributes ID, name, and tot\_cred.

A similar thing will happen, if I read the same relation in the other direction. So, instructor to student was one to many. So, student instructor also drawn in the same way, because this is this is the one side, this is the one side and this is the many side.

So, if the situation is the same. So, if we read from the student instructor, then it is also designates the many to one relationship.

(Refer Slide Time: 08:05)

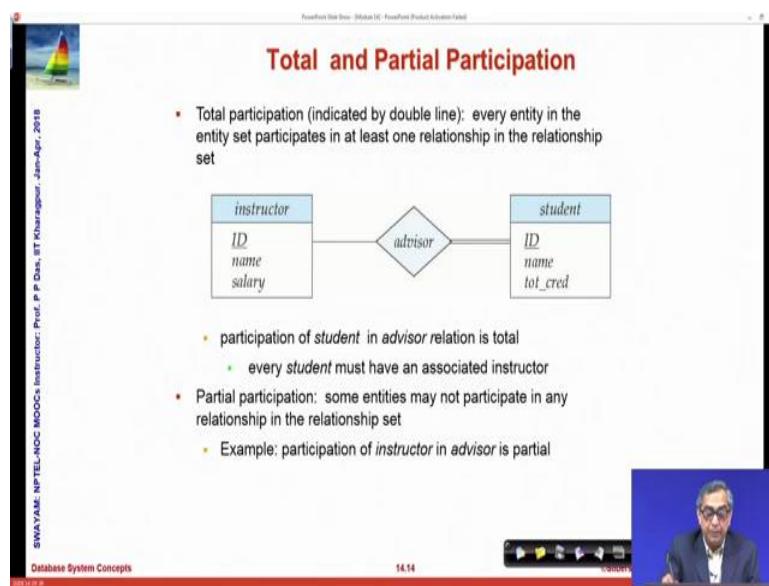
The slide is titled "Many-to-Many Relationship". It contains a bulleted list and a diagram. The list states that an instructor is associated with several (possibly 0) students via advisor, and a student is associated with several (possibly 0) instructors via advisor. The diagram shows two tables, "instructor" and "student", connected by a diamond-shaped relationship named "advisor". There are no arrows on the lines connecting the tables to the diamond, indicating a many-to-many relationship.

And finally, we can have a many to many relationship, where there is no arrow at either end which means that an instructor is associated with; several possibly none no student

why the advisor relation and the student may also have several instructors by the advisor relationship.

Now, we can you can certainly figure out that in the particular case of student instructor scenario of providing advice, one to one as well as many to many are not the usual real world scenarios, but these are we have just using to show you how to model this usual scenario would be from instructor to student it is one to many relationship.

(Refer Slide Time: 08:50)



A relationship could be total or it could be partial. If you if one side of the relationship, or whichever side of the relationship is total, then we draw a double line. So, you can you can see here in the diagram we are drawing a double line, which means that in the advisor relationship the involvement of the student is total, which means that every student must feature in the advisor relationship. Or in other words every student must have an advisor. But it is partial on the instructor side because every instructor may not have a student.

So, this double life shows that reality. Some entities may not participate in any relationship is a partial.

(Refer Slide Time: 09:59)

**Notation for Expressing More Complex Constraints**

- A line may have an associated minimum and maximum cardinality, shown in the form  $l..h$ , where  $l$  is the minimum and  $h$  the maximum cardinality
  - A minimum value of 1 indicates total participation.
  - A maximum value of 1 indicates that the entity participates in at most one relationship
  - A maximum value of \* indicates no limit.

```

    graph LR
      instructor[instructor] -- "0..*" -->|advisor| student[student]
      student -- "1..1" -->|advisor| instructor
  
```

Instructor can advise 0 or more students. A student must have 1 advisor; cannot have multiple advisors

SWAYAM-NPTEL-MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018  
Database System Concepts

14.15

Now, these constraints the cardinality constraints can be made more precise by actually using numbers. You can actually say on the 2 sides of the relationship, that at the minimum how many entities should relate, and at the maximum how many entities can relate. For example, if we are saying that we are on the right-hand side here, if you see we are saying that it is maximum minimum is one maximum is one which what does it say it says that every student the minimum is one. So, every student must feature in the advisor relationship.

So, in real world, every student must have a an advisor, must have an instructor. It says maximum is one; which says that every student can have at most one instructor. So, this one to one one, dot dot one says that every student must have at least one instructor, every student must have at most one instructor. So, together it says that every student must have exactly one instructor.

Whereas if I if we see on this side, it says that 0 dot, dot, star, star stands for no limit. It can be anything any number. So, the minimum is 0 which means that an instructor may not have a student. And star says that if the instructor can have any number of student. Naturally 0, 1, 2, 3, 4, 2, or 200. So, any instructor can advise any number of students.

So, these kind of precise number constraints can be put in addition to the one to many, or one to one, many to many kind of notations in the diagram. So, when we do that we have the precise cardinality of the complex relations that exist.

(Refer Slide Time: 12:10)

Notation to Express Entity with Complex Attributes

```

instructor
ID
name
first_name
middle_initial
last_name
address
street
street_number
street_name
apt_number
city
state
zip
{ phone_number }
date_of_birth
age()

```

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts

14.16

Prof. P. P. Das

Next, we take a look into the handling of the complex attributes. The first you remember that, the first kind of complex attribute is one which is composite. Say, name which has first name middle name, initial middle initials and last name.

(Refer Slide Time: 12:26)

Expressing Weak Entity Sets

- In E-R diagrams, a weak entity set is depicted via a double rectangle
- We underline the discriminator of a weak entity set with a dashed line
- The relationship set connecting the weak entity set to the identifying strong entity set is depicted by a double diamond
- Primary key for section – (course\_id, sec\_id, semester, year)

```

course
course_id
title
credits
sec_course
section
sec_id
semester
year

```

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts

14.17

Prof. P. P. Das

So, when we have that, then the way we represent is at the actual name of the attribute is at the outermost level. And it is composites are written with certain shift on the left. So, these all say that these are composites of name. So, if this says that street city state zip are composites of address, and further indentation say, that these are composites of street. So, this is how graphically we show that how complex attributes feature.

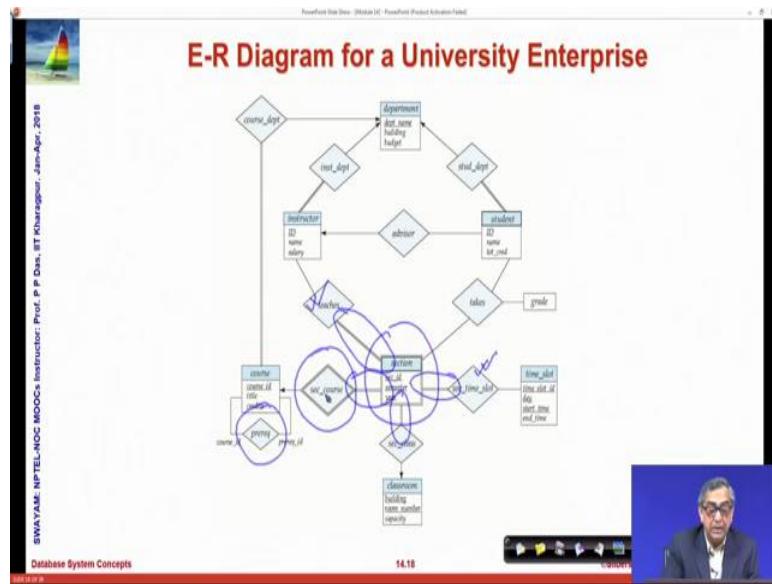
Now, let us go back to discussing the weak entity sets. In the E-R diagram, a weak entity set is represented by a double rectangle. You remember the section is a weak entity set. And why is it so? Because the same course may have 2 different, I am sorry, 2 different courses may have the same section id semester and year that is 2 courses 2 or more courses may run sections by the same name in the same semester and the year. So, a section cannot be uniquely identified by these 3 attributes. It needs a relationship with the identifying entity set course to be specific the course id so that the entities here in can be uniquely specified.

So, since this has happened. So, we designate that by putting this double rectangle around the weak entity set section. We underlined the discriminator of a weak entity set with dashed line. So, you remember these are the discriminators. Because given the identifying attribute in the identifying set. These are the attributes which distinguish different tuples of section. So, they are not shown with solid underline, they are shown as dotted underline, dashed underlined. So, that you can make out that this is a weak entity set and these are the discriminators.

The relationship set connecting the weak entity set to the identifying strong entity set, is also so, this the moment you have weak entity set. You know that there has to be a relationship to the strong entity set which identifies it. So, that relationship set course which say, course id against this minds that is designated with a double diamond. So, that you know that this is the identifying relationship between a weak entity set and the corresponding strong entity set. And once that happens in the primary key becomes the discriminators of section the weak entity set, and the primary key of the identifying a strong entity set the course. So, that forms our final primary key for this entity set section.

My new course id is not a part of this relation, but it actually plays the role through this section id as a key for the section relation without which the section entities in the section cannot be uniquely identified.

(Refer Slide Time: 16:28)



So, having said that this is the E-R diagram of the university enterprise. Some of the points that you could take a look at; this is the weak entity set we have just seen. This is the relationship to the identifying strong entity set. This is a prerequisite multi role relationship. This we can see is a total involvement. So, worry why is it a total involvement, because every section must have at least one teacher. So, there cannot be a section which does not feature in the teacher's relationship. Similarly, every section must get a timeslot where the classes for that section is held.

So, every section must feature in the sec timeslot. So, these are similarly, it must get a classroom. So, these are all different total involvements. That we total roles that we can see we can see some of that elsewhere as well. For example, you can see it here, we can see it here, because in between instructor and the department the inst department relationship. Certainly, every instructor must have a department. So, it is total, but it is not the same for the department, every department will not have instructors.

So, these is how if we can go through carefully. And for example, this is another which is total, which means that every course need a department you cannot run a course which does not have a department. So, this is how we can see that how the E-R diagram that the first conceptual level diagram of a very simple university enterprise is being designed following the notions and symbols of E-R model that we have already developed.

Next comes the part where, from this model which is primarily diagram based we have to really go to the relational schema, which is names of relations and attributes which is pretty much a straightforward job.

(Refer Slide Time: 18:41)

Reduction to Relation Schemas

- Entity sets and relationship sets can be expressed uniformly as *relation schemas* that represent the contents of the database
- A database which conforms to an E-R diagram can be represented by a collection of schemas
- For each entity set and relationship set there is a unique schema that is assigned the name of the corresponding entity set or relationship set
- Each schema has a number of columns (generally corresponding to attributes), which have unique names

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Date, IIT Kharagpur - June-April, 2018  
Database System Concepts

14.20

So, entity sets and relationship sets have to be represented in terms of relational schema. What is the beauty of the E-R model? And the relational schema is that that, when you reduce the entity relationship model to relational schema both entity sets and relationships sets. Both of them turn out to be relational schemas. And so, that the database finally, can be represented simply as a set of schemas each one of which must have a set of identifying primary key.

(Refer Slide Time: 19:20)

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

## Representing Entity Sets

- A strong entity set reduces to a schema with the same attributes  
 $\text{student}(ID, name, tot\_cred)$
- A weak entity set becomes a table that includes a column for the primary key of the identifying strong entity set  
 $\text{section}(\text{course\_id}, \text{sec\_id}, \text{sem}, \text{year})$

```

    graph LR
        course[course  
course_id  
title  
credits] -- sec_course --> section[section  
sec_id  
semester  
year]
    
```

Database System Concepts 14.21

So, let us look into that. So, on the strong entity set, that reduces to schema with the same attribute as student. So, student has id name and total credit so, which we saw earlier. Now this gets converted to a schema with id being the primary key. The other case of weak entity set, section which had the three discriminators, and through set course relationship was identified from the strong entity set. Course borrows the primary key of the course to be defined in terms of this relational schema.

(Refer Slide Time: 20:05)

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

## Representing Relationship Sets

- A many-to-many relationship set is represented as a schema with attributes for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set.
- Example: schema for relationship set *advisor*

```

    graph LR
        instructor[instructor  
ID  
name  
salary] -- advisor --> student[student  
ID  
name  
tot_cred]
        advisor((advisor = (s.ID, i.ID)))
    
```

Database System Concepts 14.22

One moment; this borrows the primary key from here and becomes.

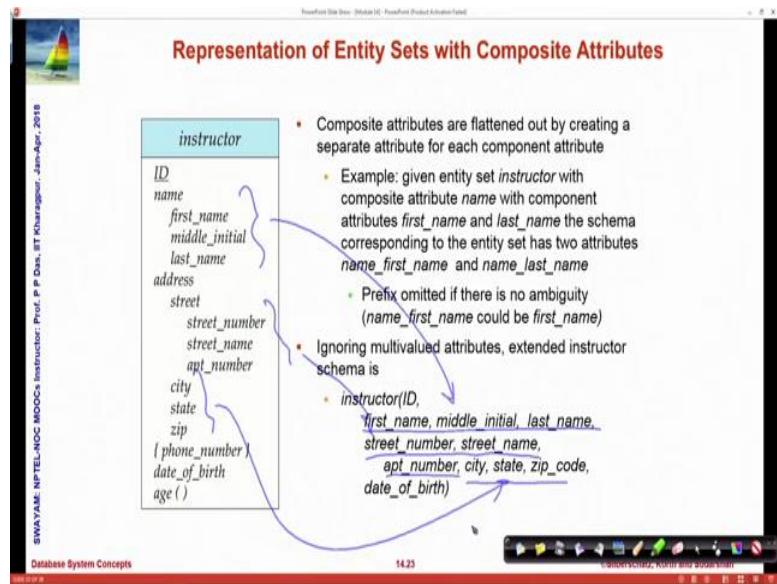
(Refer Slide Time: 20:10)

So, you can see that in the E-R model, the section did not have course id as an attribute, but while we reduce this to the relational schema through this sec course relationship, we have borrowed this primary key from course. The primary key of course, the course id and added that to section to make it a complete relational schema.

Next comes the representation of relationships. So, we are showing a relationship advisor so, which relates instructors to students. So, naturally every instructor is identified by id every student is identified by id. Since both of the attributes have the same name id. We are calling them as s underscore id and for the student and I underscore id for the instructor. So, the advisor relation is basically a pairing of these two ids which gives rise to a relationship which looks like this relationship schema which looks like this. So, it can in general say that if we have a relationship in the E-R model, which we want to represent in the schema then we will take, we will create a schema which has the primary key of both the sets, and put them together. And if the names clash we will just change the name with the name of the relation, and that will give us the schema for the relationship in this case the advisor.

So, you have seen how to represent entity sets, weak entity sets and relationships.

(Refer Slide Time: 22:03)



Ah let us look at how do we deal with composite attributes. Because so far, we had assumed that the relational schema has attributes, and every attribute has a domain. The type from where its values come. So, if I have a composite attribute, where every attribute has a set of components. Then the easiest way to handle this is to what is known as flatten the composite attribute.

So, flattening basically is for example, if I take a name it has 3 components. So, each one of them I can call by given new name, name underscore first name, name underscore middle initial name underscore last name. By prefixing with the attribute name, I make the names of these components necessarily unique. Now after I do the prefixing I might figure out that actually prefixing is not required first name itself is a unique. Because it does not occur anywhere else if it is then I can I may drop the prefix name. But in general, I can take the attribute name prefix on the component, and just flatten them out make them all attributes each separate attribute.

So, here when we flatten out we will have first name, middle, initial, last name as you can see these flattened out from here. Then we have street number, street name, apartment number, flattened out from the street. Subsequently, we have city state zip flattened out from here. So, all of them flattened out has become separate attributes. And flattening is a very straightforward mechanism by which you can convert complex composite attributes into the regular schema design getting to little bit of issue if you have a multi valued attribute.

(Refer Slide Time: 24:02)

The screenshot shows a PowerPoint slide with the title "Representation of Entity Sets with Multivalued Attributes". The slide content includes a bulleted list:

- A multivalued attribute  $M$  of an entity  $E$  is represented by a separate schema  $EM$
- Schema  $EM$  has attributes corresponding to the primary key of  $E$  and an attribute corresponding to multivalued attribute  $M$
- Example: Multivalued attribute  $phone\_number$  of  $instructor$  is represented by a schema:  
 $inst\_phone = (ID, phone\_number)$
- Each value of the multivalued attribute maps to a separate tuple of the relation on schema  $EM$ 
  - For example, an  $instructor$  entity with primary key 22222 and phone numbers 456-7890 and 123-4567 maps to two tuples: (22222, 456-7890) and (22222, 123-4567)

The video player at the bottom shows a man speaking, with the caption "Database system Concepts" and the time "14:24".

Multivalued attribute is one where one attribute may have multiple values at the same time. And the example we talked about is a phone number.

I may have multiple phone numbers. So, certainly against an attribute I can keep only one value. So, if my attribute is multiple value. Then the basic idea is to use a separate schema to maintain this multiple values. For example, if I have to maintain multiple phone numbers of an instructor. I may have a separate I may decide to have a separate relation which relates the key of the instructor relation, and the attribute that I want to maintain multiply.

So, in this relationship in this relation inst underscore phone against the same id I can have different phone numbers. So, there will be different records which match on the id, but do not match on the phone number which gives me the different values that the phone number can take. And then this inst phone in conjunction with the instructor relation will actually denote the multi valued phone number attribute. So, this is just an example showing that for one primary key of an instructor 1 to 2 2 2 2 and there are two phone numbers. So, this will basically mean you have two tuples in the new relation.

(Refer Slide Time: 25:37)

**Redundancy of Schemas**

- Many-to-one and one-to-many relationship sets that are total on the many-side can be represented by adding an extra attribute to the "many" side, containing the primary key of the "one" side
- Example: Instead of creating a schema for relationship set *inst\_dept*, add an attribute *dept\_name* to the schema arising from entity set *instructor*

So, with that we can handle multiple multivalued attributes also. Some of the relationships that we may have modeled, which we have done in doing the database E-R schema could have redundancy. For example, take a case here we have the instructor. And we have the student advisor relation is incidental here, and we have department. So, we want to say that the instructors belong to certain departments. Every instructor belongs to one department which is the totality of the relationship here.

Similarly, every student belongs to a department, totality of the relationship on this side. And inst dept inst\_dept in that context is a relationship which is between instructor and department. Similarly , so, we can there is certain redundancy in this, because we can get make this simpler if we just take the primary key of this relation and put in here. If we do that then basically this become redundant these are no more required. So, all that you are saying is the instructor has a dept name field which says which department does it belong to.

So, if there is a choice between whether you will keep such relationships or you will actually reduce the redundancy in the schema, and involve the primary key of the other relation into your primary table which is instructor or the student here. So, instead of creating a schema for relationship set inst department. You will simply add a department name. So, mind you this is at the at the E-R model level you did have a separate relationship, but while you reduce it to your relationship relational schema you are reducing that relationship by including the dept name as an attribute in instructor.

(Refer Slide Time: 27:53)

Redundancy of Schemas (Cont.)

- The schema corresponding to a relationship set linking a weak entity set to its identifying strong entity set is redundant.
- Example: The section schema already contains the attributes that would appear in the sec\_course schema

```
graph LR; course[course  
course_id  
title  
credits] --- sec_course((sec_course)); section[section  
sec_id  
semester  
year] --- sec_course; sec_course --> course; sec_course --> section;
```

So, that is called the reduction of schema which is often used. for one to one relationship. So, this is this, was this is good if you have many to one relation. Because this was possible, because every instructor has one department. So, if you just include the department name with the instructor. Or every student has one department ah. So, it is possible that way, but if you have a one to one relationship, then naturally you can do the similar reduction by I by choosing the either side as the many. You know, because because the the unique side has to come on the many side. The unique side here. This is the unique side here. Because every instructor has a unique department and this is the many side here.

So, the unique side has to attribute has to come in here. The unique side primary key has to come in here. So, instead of so, you can apply the same principle to a one to one relationship by treating any one of them as a many side, and add the extra attribute on the other side to get rid of this additional schema. The schema corresponding to a relationship set linking a weak entity set to it is underlying strong entity set is certainly redundant, we have already seen this. So, this (Refer Time: 29:13) is made redundant by including the primary key of the identifying relation, identifying entity set in the weak entity set. So, that is another reduction of schema that can be done.

(Refer Slide Time: 29:31)

The screenshot shows a Microsoft PowerPoint slide titled "Module Summary" in red font at the top center. Below the title, there is a bulleted list of two items:

- Illustrated E-R Diagram notation for E-R Models
- Discussed translation of E-R Models to Relational Schemas

On the left side of the slide, there is a vertical watermark or logo for "SWAYAM-NPTEL-MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018". At the bottom left, it says "Database System Concepts". At the bottom right, it shows the time as "14.28" and the slide number as "10 of 10". The bottom right corner also has a note: "Under construction, poster and about section". The overall background of the slide is white.

So, to summarize we have in this module illustrated the entity relationship diagram, which are very nice ways of graphically representing what we see in the real world. So, it has graphical representation of entity sets, attributes the key attributes primary key attributes the weak entity sets, and the relationships along with the cardinality information.

And then we have shown that using certain reduction rules how we can easily reduce this entity relationship diagram or entity relationship model into the traditional relational schema. And we have seen that both the entity sets as well as the relationship sets become relational schemas.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 15**  
**Entity-Relationship Model/3**

Welcome to module 15 of Database Management Systems. We have been discussing about entity relationship model and this is the third and the closing module on this topic.

(Refer Slide Time: 00:37)

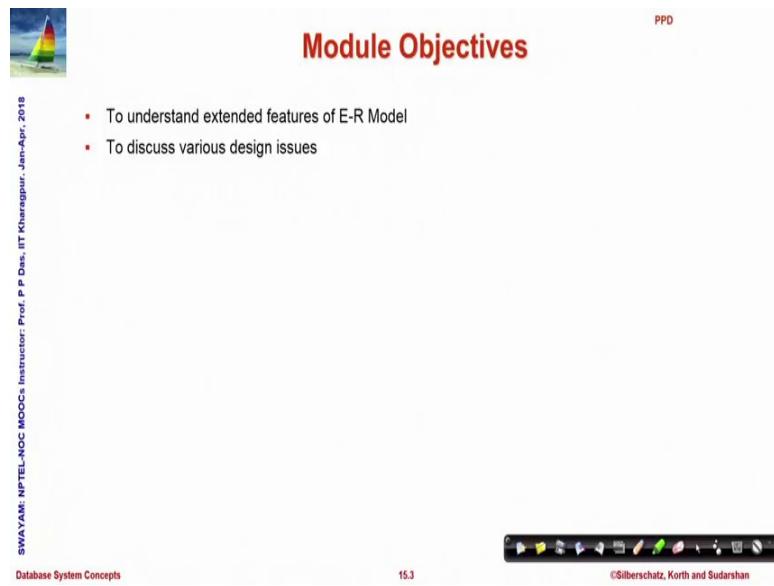
The slide has a white background with a decorative header featuring a sailboat icon and the text "Module Recap" in red. In the top right corner, there is a small "PPD" label. On the left side, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur, Jan-Apr, 2018". At the bottom, there is a footer with icons for navigation, a copyright notice: "©Silberschatz, Korth and Sudarshan", and page numbers: "15.2" and "317".

**Module Recap**

- E-R Diagram
- E-R Model to Relational Schema

So, in the last module we have discussed about E-R diagram and we have also seen how E-R model can be converted to a relational schema.

(Refer Slide Time: 00:47)

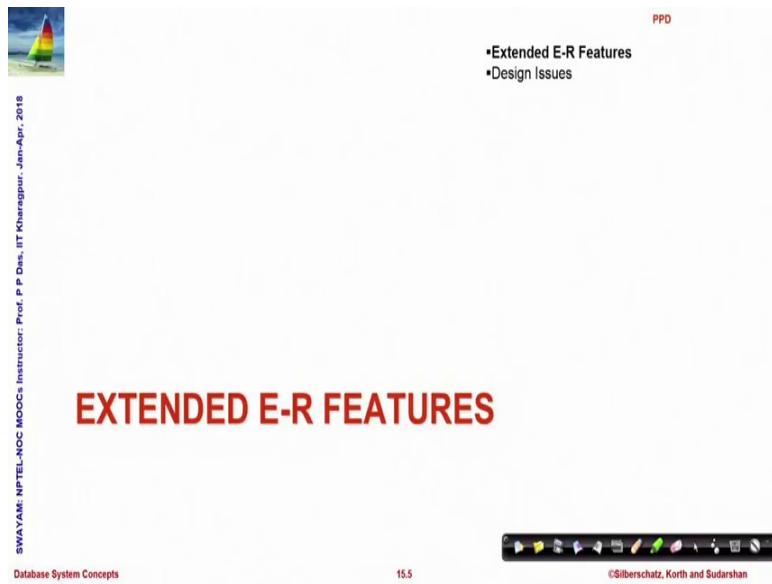


The slide title is "Module Objectives". It features a small sailboat icon in the top left corner and the letters "PPD" in the top right corner. A vertical copyright notice on the left side reads: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018". The main content is a bulleted list: "To understand extended features of E-R Model" and "To discuss various design issues". At the bottom, there is a navigation bar with icons for back, forward, search, and other presentation controls, along with the text "Database System Concepts" and "15.3". The copyright notice at the bottom right is "©Silberschatz, Korth and Sudarshan".

In this module we will try to go through a few extended features of E-R model, try to show some of the more complicated situations how they can be modeled in the E-R model and along with that we will discuss a variety of design issues that will follow.

So, these are the outline.

(Refer Slide Time: 01:19)



The slide title is "EXTENDED E-R FEATURES". It features a small sailboat icon in the top left corner and the letters "PPD" in the top right corner. A vertical copyright notice on the left side reads: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018". The main content is a bulleted list: "•Extended E-R Features" and "•Design Issues". At the bottom, there is a navigation bar with icons for back, forward, search, and other presentation controls, along with the text "Database System Concepts" and "15.5". The copyright notice at the bottom right is "©Silberschatz, Korth and Sudarshan".

So, we start with extended entity relationship features.

(Refer Slide Time: 01:23)

The slide has a header 'Non-binary Relationship Sets' and a subtitle 'SWAYAM: NPTEL-NOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. It includes a vertical sidebar with the text 'Database System Concepts'. The main content shows an E-R diagram with three entity sets: 'instructor', 'project', and 'student'. The 'instructor' set has attributes 'ID', 'name', and 'salary'. The 'student' set has attributes 'ID', 'name', and 'tot\_cred'. A ternary relationship 'proj\_guide' connects the three sets. The 'project' set is shown above the relationship, and the 'instructor' and 'student' sets are below it. The slide also features a video feed of a professor on the left and a navigation bar at the bottom right.

The first that we note is so, far in the entity relationship model we have talked about relationships between two entity sets. So, we have talked about the student attending courses or instructors advising students and so, on. So, such relationships are naturally called binary, but it is possible that more than two entity sets, let us say three entity sets could be involved in the same relation and we show an example here where we have three entity sets instructor, student and project.

So, the project entity set is a list of projects being done by the students or to be done by the students. So, the relationship project guide is a relationship between the guide who is an instructor, the student who will do the project and the project that has to be performed. So, all three together define this relationship; so, in such cases it is possible in E-R model that we can represent it conveniently as a non binary relationship.

Now this is an E-R diagram this is called a ternary relationship R.

(Refer Slide Time: 03:02)

The slide has a header 'Cardinality Constraints on Ternary Relationship' with a sailboat icon. On the left, vertical text reads: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018'. Below the header is a bulleted list: '• We allow at most one arrow out of a ternary (or greater degree) relationship to indicate a cardinality constraint'. To the right is a diagram showing two entity sets, E<sub>1</sub> and E<sub>2</sub>, represented by boxes. A diamond-shaped arrow connects them, with arrows pointing from E<sub>1</sub> to the diamond and from the diamond to E<sub>2</sub>. At the bottom left is a video frame of a professor, and at the bottom right are navigation icons.

Now we have talked about cardinality constraints on binary relationship one to one, many to one one to many and many to many. And we specified that if we have a binary relationship say this is one entity set and this is another entity set and we have a relationship. So, if we just connect them it means a many to many relation, but if we have an arrow on one side entity set then it means on the arrow side its one.

So, this is from entity set E 1 to E 2 this is one to many; we could have arrow at both ends and; that means, one to one. Now the question is how will that work out what will be the meaning of arrow in terms of a ternary relationship.

(Refer Slide Time: 03:55)

**Cardinality Constraints on Ternary Relationship**

- We allow at most one arrow out of a ternary (or greater degree) relationship to indicate a cardinality constraint
- For example, an arrow from *proj\_guide* to *instructor* indicates each student has at most one guide for a project
- If there is more than one arrow, there are two ways of defining the meaning.
  - For example, a ternary relationship  $R$  between  $A$ ,  $B$  and  $C$  with arrows to  $B$  and  $C$  could mean
    - Each  $A$  entity is associated with a unique entity from  $B$  and  $C$  or
    - Each pair of entities from  $(A, B)$  is associated with a unique  $C$  entity, and each pair  $(A, C)$  is associated with a unique  $B$
  - Each alternative has been used in different formalisms
  - To avoid confusion we outlaw more than one arrow

SWAYAM: NPTEL-NOC Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr., 2018  
Database System Concepts

15.7 ©Silberschatz, Korth and Sudarshan

Now, in the case of ternary relationship or this would generalize to relationships of higher degree whether where more than three entity sets may be involved. We put a restriction that we will allow at most one arrow out of the ternary relationship. So, we could have only if we if we look at if we look at say this relationship then we could have an arrow only at this end.

But multiple arrows are not allowed and the reason is certainly to keep the semantics of the cardinality meaningful. For example, if we have a ternary relationship between  $A$   $B$  and  $C$  let us say this is  $A$  this is  $B$  and this is  $C$  and we have a ternary relationship between them. And let us say if we have a more than one arrow; there for example, suppose then we have say an arrow to  $B$  and an arrow to  $C$  the question is how should we interpret that?

Should we interpret that an entity of entity set  $A$  is associated with unique entity from  $B$  and  $C$  together or should we associate, should we interpret that the entities formed by the pair  $A$   $B$  and the entity formed by the pair  $A$   $C$  are uniquely related. So, there is multiplicity of interpretation; if we allow more than one arrow in case of eternity or higher degree relationship. So, what will follow for simplicity in this course and that is what is followed often in practice; is in case of a ternary or higher order relationship only one arrow will be allowed in that in it in that relationship.

(Refer Slide Time: 06:20)

The slide has a title 'Specialization' in red at the top right. On the left, there is a small sailboat icon. To the right of the title is a bulleted list:

- Top-down design process; we designate sub-groupings within an entity set that are distinctive from other entities in the set

Below the list is a hand-drawn diagram. It shows two rectangular boxes, one above the other. The top box contains a small sketch of a person's head and shoulders. An arrow points from the bottom box to the top box. To the right of the boxes, the text 'B is A' is written in blue ink. At the bottom of the slide, there is a navigation bar with icons for back, forward, search, and other presentation controls.

Now, let us talk about specialization those of you who have some background of object oriented systems would be familiar with the notion of specialization and generalization in object oriented system. So, we say that if we have a certain concept say we have a concept called person and then we say that a student is a person; what we mean is a student is a specialization of person. And person is a generalization of student and in that process student inherits all the attributes of person, but in addition the student may have some specialized attributes.

So, what it means that if you look from the perspective of such specialization; say if I draw like this; this is an entity set A and this entity set B and to mark the specialization we show an arrowhead with a blank triangle at that end. So, if we by this what we mean is B is a A. So, B inherits all the properties of A, but can have some more properties. So, if you look at all the entities that A may have you will find that a subgroup of the entities in the entity set A have some additional common properties.

So, if A is set of persons and B is a set of students then A may have entities which represent people who are not students; who are employees, who could be retired and so, on, but there will be a number of entities. So, who have the commonality of being student they are enrolled in certain course of certain university and so on.

So, in terms of the E-R diagram E-R model what we do is we try to look at the entity A and move top down. So, that whenever we find a group of entities which have certain

commonality; we move them into a lower separate, specialized entity set and relate these two entity sets to the specialization relation.

(Refer Slide Time: 08:54)

**Specialization**

- Top-down design process; we designate sub-groupings within an entity set that are distinctive from other entities in the set
- These sub-groupings become lower-level entity sets that have attributes or participate in relationships that do not apply to the higher-level entity set
- Depicted by a triangle component labeled ISA (e.g., *instructor* "is a" *person*)
- **Attribute inheritance** – a lower-level entity set inherits all the attributes and relationship participation of the higher-level entity set to which it is linked

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts

15.8 ©Silberschatz, Korth and Sudarshan

So, these sub groups form lower level entity sets and as I said that it is designated in a certain way. And as in the object oriented system the lower level entity set inherits all the attributes and relationships of participation of the higher level entity set. So, here is an example.

(Refer Slide Time: 09:16)

**Specialization Example**

- Overlapping – *employee* and *student*
- Disjoint – *instructor* and *secretary*
- Total and partial

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts

15.9 ©Silberschatz, Korth and Sudarshan

Can say the person at the so, called root of this hierarchy of specialization; it has a set of properties ID, name, street, city and employee is another relationship; another entity set which is a specialization of person relation person entity set.

So, employee inherits all attributes ID name street and city, but in addition the commonality of the entities in the employee entity set is a fact that all of them have a salary attribute. A similar entity set student is a specialization of person where the again all attributes are inherited, but there is a common attribute called total credits which is common for all the students, but is not available or common for the persons in general.

And as you can see that it could be hierarchical it could go further down employee could be specialized into instructor and secretary. Again by the rule of specialization instructor will inherit all attributes of employee which means that it will inherit attributes of person; which imply has inherited plus the employee specific attributes salary. So, it will inherit all five of those attributes and then it adds another attribute which is specific for the instructor which says a rank which is another specific attribute that you have.

Now, when we specialize a certain entity set into two or more entity sets like we specialized person in employee and student; then there could be different situations that could exist for example, certain entity may be a member of both employee as well as student. If that happens then we say that their overlapping entity sets or they could be disjoint where no member of instructor would be a member of the secretary and vice versa.

So, we say that this disjointness tell us that no instructor can be a secretary and no secretary can be an instructor. Whereas, overlapping specialized sets denote that well an employee may or may not be a student, but it is possible that some employee is also a student and vice versa and that is how we will represent this. And we will see that when we specialized the specialized entity could be total or they could be partial.

(Refer Slide Time: 12:06)

The slide is titled "Representing Specialization via Schemas". It features a logo of a sailboat on the left and a sidebar with course details: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018". The main content area contains a table showing schema representation and a list of drawbacks.

| schema   | attributes             |
|----------|------------------------|
| person   | ID, name, street, city |
| student  | ID, tot_cred           |
| employee | ID, salary             |

**Method 1:**

- Form a schema for the higher-level entity
- Form a schema for each lower-level entity set, include primary key of higher-level entity set and local attributes

**Drawback:**

- Getting information about, an *employee* requires accessing two relations, the one corresponding to the low-level schema and the one corresponding to the high-level schema

We will talk about that totality and partiality little later; let us look at how do we represent this information in the relational schema because as we have seen that whenever we have a E-R diagram; it is important to find out what is the relational schema that will be corresponding to that E-R diagram or E-R model. So, here we could do this in two ways one that we are showing here is form a schema for the higher level entity. So, form a schema for the person as you can see here that person is described in terms of four attributes and this form a schema for each of the lower level entity set where you include the primary key of the higher level entity set.

So, when you are forming the schema of person of student which is a specialization of person you include the ID which is the key of the higher level entity set person. And along with that you include the so, called local attributes or attributes which are specific to this lower level entity set in this case total credit similar thing happens with employee.

Now this representation is in a way optimized because you are representing the information only once when it is needed, but the drawback is if you have to find out information about say employee; then you will not only have to access the employee entity set or the corresponding relation in the relational schema, but you will also have to access the parent or higher level entity set to get the attribute values which are inherited. And if you have a multi level hierarchy as we have shown this could involve accessing multiple relations to find information about a single entity in an entity set.

So, this is an in terms of data representation this is an optimized representation, but it has the overhead of having to access multiple entity sets to get information about certain entities.

(Refer Slide Time: 14:22)

The slide features a sailboat icon in the top left corner. The title 'Representing Specialization as Schemas (Cont.)' is centered at the top. On the left, there is a vertical column of text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Dass, IIT Kharagpur - Jan-Apr. 2018'. Below this is a small video thumbnail showing a man in a suit. The main content area contains a bulleted list under 'Method 2' and a table illustrating the schema representation.

- Method 2:
  - Form a schema for each entity set with all local and inherited attributes

| schema   | attributes                       |
|----------|----------------------------------|
| person   | ID, name, street, city           |
| student  | ID, name, street, city, tot_cred |
| employee | ID, name, street, city, salary   |

▪ Drawback: *name, street and city* may be stored redundantly for people who are both students and employees

15.11 ©Silberschatz, Korth and Sudarshan

An alternate scheme would be that based on the hierarchy of specialization, you assume all attributes as they are inherited and then represent every entity set in full. So, when you the representation of person does not change, but when you represent student now earlier you are just having ID and total credit; now in you include all entities that are inherited.

Similarly, you do the same thing; so, you have the all entities of the parent to all attributes of the parent entity set as well as the local attribute of that specific entity set. Now this naturally makes it easy to extract information from a for a single entity set, but at the same time, you are storing the same data redundantly for people who are having overlapped representation. So, if we have as we know student and employee overlapped. So, the same entity will happen in student as well as in employee; so, it will the information of the common attributes name street city etcetera they will occur in both these tables in the design.

So, these are two methods and now we have just given you the relative advantages and its advantages of the same and based on a particular situation you will have to choose what is a good method to represent.

(Refer Slide Time: 15:51)

**Generalization**

- A **bottom-up design process** – combine a number of entity sets that share the same features into a higher-level entity set.

```
graph TD; Student[Student] --> UGStd[UG Std.]; Student --> PGStd[PG Std.]
```

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts 15.12 ©Silberschatz, Korth and Sudarshan

You can look at now you know from the object based system that if you have a specialization hierarchy you can think of it as a generalization hierarchy also.

The generalization hierarchy goes in a bottom up manner. So, instead of actually starting with an entity set and finding out subsets of entities which have greater commonality between them and putting them as specialized, you could actually group them in terms of finding out what they share and create a higher level entities. For example, the way I am saying is let us say that I have one entity set which say UG student and have another entity set in the same university which say PG student.

So, there are both of them are students naturally they are disjoint a person cannot be UG as well as PG student at the same time. And once you represent that you find that well there are a lot of information which are common between these two entity sets like the student roll number, name and so on so, forth. So, you could choose that well you instead of having them as two separate entity sets, you could extract out the common attributes and put them at a higher level entity. So, all that you are doing is instead of going top down you are going bottom up in the whole approach.

(Refer Slide Time: 17:21)

The slide features a title 'Generalization' in red at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list of points:

- A bottom-up design process – combine a number of entity sets that share the same features into a higher-level entity set.
- Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way
- The terms specialization and generalization are used interchangeably

On the left margin, vertical text reads: SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018. At the bottom left is the text 'Database System Concepts'. In the bottom right corner, it says '©Silberschatz, Korth and Sudarshan' and shows the slide number '15.12'.

So, if you do that then there you can easily see that specialization and generalization are inverse of each other and they are used interchangeably in terms of the relational entity relationship design.

(Refer Slide Time: 17:35)

The slide title is 'Design Constraints on a Specialization/Generalization' in red at the top right. It includes a small sailboat icon on the left. The main content is a bulleted list of constraints:

- **Completeness constraint** -- specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within a generalization
  - **total**: an entity must belong to one of the lower-level entity sets
  - **partial**: an entity need not belong to one of the lower-level entity sets
- Partial generalization is the default. We can specify total generalization in an ER diagram by adding the keyword **total** in the diagram and drawing a dashed line from the keyword to the corresponding hollow arrow-head to which it applies (for a total generalization), or to the set of hollow arrow-heads to which it applies (for an overlapping generalization).
- The **student** generalization is total: All student entities must be either graduate or undergraduate. Because the higher-level entity set arrived at through generalization is generally composed of only those entities in the lower-level entity sets, the completeness constraint for a generalized higher-level entity set is usually total.

On the right side, there is an Entity-Relationship (ER) diagram showing the hierarchy. It starts with a 'person' entity at the top level, which generalizes into 'employee' and 'student'. The 'employee' entity has attributes 'salary' and 'rank'. The 'student' entity has attributes 'tot\_credits' and 'hours\_per\_week'. The 'instructor' and 'secretary' entities are shown as specializations of 'employee', indicated by hollow arrows pointing from 'employee' to both.

Vertical text on the left margin: SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018. Bottom text: Database System Concepts. Slide number: 15.13. Copyright: ©Silberschatz, Korth and Sudarshan.

The other constraint that you can identify, you should identify is the constraint of completeness which say that if I have a entity set say a person. And then we have specializations of employee and student; the question is for a higher level entity set is it

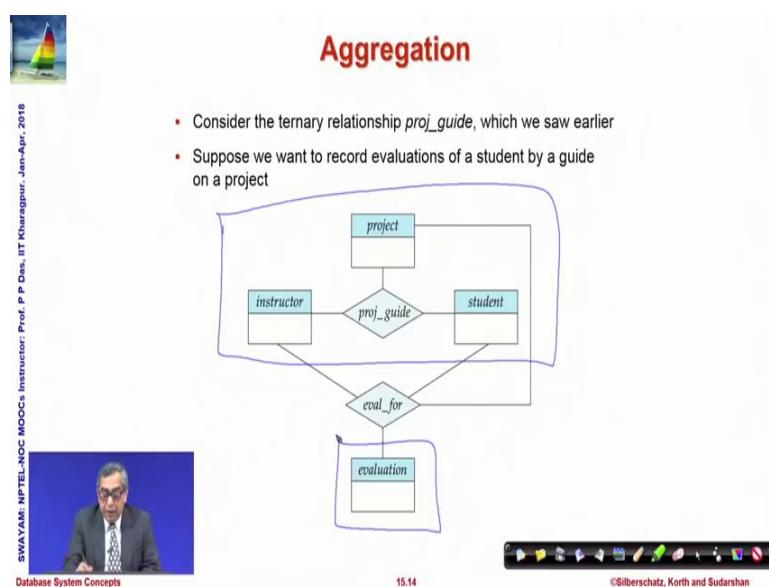
necessarily that every entity will be represented in one of the or more than one of the lower level entity sets.

If that is guaranteed that an entity must belong to one of the lower level entity set we say that it is a complete specialization. But if it is that a high level entity is may or may not be featuring in a entity set, which is at a lower level then we will say it is a partially specialized hierarchy. So, the both of these are possible depending on different situation that we have. So, by default we assume partial specialization and. So, if we want to say a certain specialization is total we will have to write the keyword total by the side of the arrow head that is representing the specialization hierarchy.

You can say that the example I talked of in uniting unite undergraduate and graduate or postgraduate students into the entity set of students, gives you a hierarchy which is total because every entity in the entity set student must be either a UG student or a PG student; it is not possible that I have a student who is neither a UG student nor a PG student. So, every high entity at the higher level entity set student must feature in one of these two specializations.

So, therefore they are necessarily total in the relationship. So, this is the completeness constraint that you can think of.

(Refer Slide Time: 19:52)



Moving on let us talk about another feature which is known as aggregation; the situation is like this we have already talked about this part of the diagram which is a ternary relationship which relates project instructor and student.

Now, let us say once the project progresses you would need to add evaluation to that. So, there is another entity set which represents evaluation and how we are grading or putting marks and so, on. So, naturally the evaluation of a student will be dependent on the project, the student and the supervisor and that will relate to the evaluation. So, evaluation eval for the relationship is necessarily a relationship between four entities or four entity sets so, to say.

(Refer Slide Time: 20:47)

**Aggregation (Cont.)**

- Relationship sets `eval_for` and `proj_guide` represent overlapping information
  - Every `eval_for` relationship corresponds to a `proj_guide` relationship
  - However, some `proj_guide` relationships may not correspond to any `eval_for` relationships
    - So we cannot discard the `proj_guide` relationship
- Eliminate this redundancy via aggregation
  - Treat relationship as an abstract entity
  - Allows relationships between relationships
  - Abstraction of relationship into new entity

SWAYAM, NPTEL-NOC, MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jam-Apri- 2018  
Database System Concepts

15.15 ©Silberschatz, Korth and Sudarshan

Now, the question is how do we represent this information? The relationship sets eval for project guide the two that we saw if we just want to recall once more. The project guide involves three of the relation entity sets and the eval for relates to four of the entity sets. Now every eval for relationship corresponds to a project guide relationship; that is if I have an entity in eval for relationship, I will have a corresponding entity in the project guide relationship which specifies the student project and the instructor.

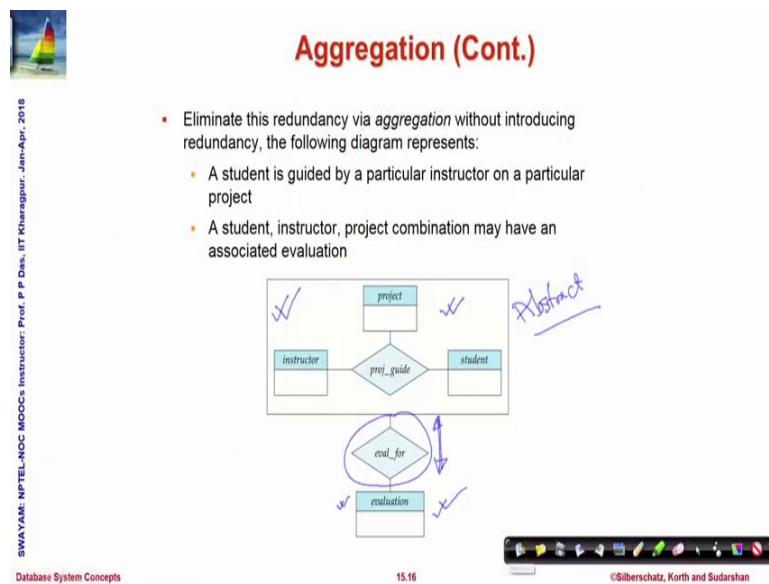
So, but it is other way it is possible that some project guide relationship may not correspond to any eval relationship; that is it is possible that there is a allotted project by a student with a particular instructor which has yet not been evaluated; the evaluation process is not complete or the time has not come.

So, if we have to represent the information only for the eval for relationship; we will get partial information because it is possible that some entities in eval for does not have the eval for information do not feature there, but need to be preserved because I need to remember the project guide, instructor, the student and the project. So, we need to keep both duplicating the information.

So, we can use aggregation to eliminate this duplication of information or redundancy of information. So, what we do is we treat the first relationship; the project get relationship as if it is an abstract entity and then you allow relationship between two relationships; this is something we did not do before relationship so, far has always been between entity sets.

So, what you can see that project guide relationship itself as if it is an virtual entity; it is an abstract entity and then you allow the relationship between the project guide and the eval for relationship which relates to the evaluation entity set this really this shows I mean I will just show you in the diagram.

(Refer Slide Time: 23:08)



So, this is how it will now work out to be. So, this is the abstract project guide entity set which is an abstract entity set because it is actually relationship. And that relates to eval for which on the other site has the evaluation. So, what will happen is a student is guided by a particular instructor in a particular project will feature in this abstract entity set; which relates the three entity sets project student and instructor. And if it has an

evaluation then the; this through this relationship it will be represented and the evaluation value will exist.

So, we know that if a project is evaluated then it certainly have a corresponding entity in the abstract entity set project guide, but the reverse may not be true I may have an entity in the project guide entity set which does not have an evaluation. So, by using this aggregation model; I can represent the information of this situation model this situation more accurately than I could do otherwise.

(Refer Slide Time: 24:30)

**Representing Aggregation via Schemas**

- To represent aggregation, create a schema containing
  - Primary key of the aggregated relationship,
  - The primary key of the associated entity set
  - Any descriptive attributes
- In our example:
  - The schema eval\_for is:  
 $\text{eval\_for}(\text{s\_ID}, \text{project\_id}, \text{i\_ID}, \text{evaluation\_id})$
  - The schema proj\_guide is redundant

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts

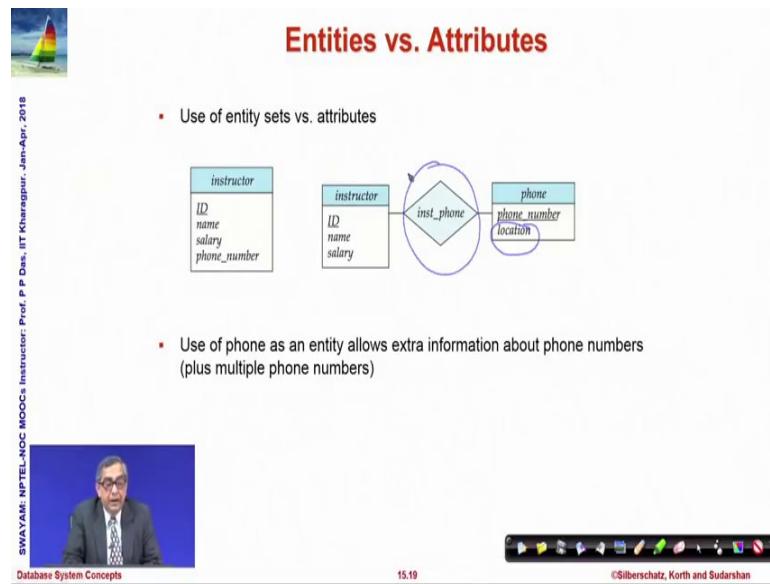
15.17 ©Silberschatz, Korth and Sudarshan

So, this can be represented again how to represent this in terms of the schema? So, what we do we represent the aggregation we create a schema containing the primary key of the aggregated relationship. The primary key of the associated entity set and all the other descriptive attributes and put them together.

So, in our example the schema would be eval four and that schema will have these are entities these are attributes of the aggregated or abstract entity set which is coming from the student project and the instructor entities and this is for the evaluation ID. So, we put this together; so, now, you can see that all of these are related representing who is the guide of which student in what project and if this exists then this gives you the evaluation.

So, naturally once this has been represented; the project guide schema by itself becomes redundant and therefore, it can be removed. So, this is a process through which we come to the decision of actually having this schema to represent all the required information. Naturally if the evaluation is not done then the evaluation ID for eval for will not exist and that will be a null showing that it is not present right now ok.

(Refer Slide Time: 26:25)



Now, given these basic features as well as the extended features let me talk about a few design issues which will be required to see what kind of information that that the different challenges that we have seen so, far. For example, we have seen the case of multivalued attributes.

So, and the way we can represent that is using that multivalued attribute as a separate entity set like the phone number which also has the advantage of having its own added information. For example, once we do this then not only I can have against the same instructor ID; I can have multiple phone numbers, but I can have location for each one of these phone numbers. And I make use of this relation relationship that I create which allow me to represent this multi valued attribute.

(Refer Slide Time: 27:30)

The slide is titled "Entities vs. Relationship sets". It features a diagram illustrating relationships between entities. At the top left is a small sailboat icon. On the left edge, vertical text reads: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018". The main diagram shows three entities: "section", "registration", and "student". "section" has attributes "sec\_id", "semester", and "year". "student" has attributes "ID", "name", and "tot\_cred". Two relationship sets, "section\_reg" and "student\_reg", connect "section" and "registration", and "registration" and "student" respectively. The "registration" entity is represented by a rectangle with three dashed lines inside, labeled with ellipses (...).

**Use of entity sets vs. relationship sets**  
Possible guideline is to designate a relationship set to describe an action that occurs between entities

```
graph LR; section[entity section] --> section_reg{relationship section_reg}; section_reg --- registration[entity registration]; registration --- student_reg{relationship student_reg}; student_reg --- student[entity student]
```

**Placement of relationship attributes**  
For example, attribute date as attribute of advisor or as attribute of student

So, this is a common technique that will be used frequently in such cases you can have entities versus relationship for example, if we have inform we need to keep information about registration how students register to different sections; then we could represent registration as an entity set and have different relationships of section registration which specify how registration is related to section and student reg; which specify how do the station is related to students to represent that kind of information.

We can have placement of relationship attributes also attribute date we have talked about as an attribute of adviser to designate as when that particular instructor became adviser of a student is a common situation that we have already seen.

(Refer Slide Time: 28:25)

**Binary Vs. Non-Binary Relationships**

- Although it is possible to replace any non-binary ( $n$ -ary, for  $n > 2$ ) relationship set by a number of distinct binary relationship sets, a  $n$ -ary relationship set shows more clearly that several entities participate in a single relationship
- Some relationships that appear to be non-binary may be better represented using binary relationships
  - For example, a ternary relationship *parents*, relating a child to his/her father and mother, is best replaced by two binary relationships, *father* and *mother*
    - Using two binary relationships allows partial information (e.g., only mother being known)
  - But there are some relationships that are naturally non-binary
    - Example: *proj\_guide*

SWAYAM-NPTEL-NOC Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts

15.21 ©Silberschatz, Korth and Sudarshan

There is also question of the choice being made between the binary and non binary relationship ternary or higher degree. Now as it turns out that it is possible that you could represent ternary relationships directly or you can decompose that. For example, a ternary relationship can be decomposed in terms of two binary relationship; for example, let us say if we talk about persons then person every person has parents; so, he or she has a father and a mother.

Now, if we represent this as a ternary relationship then the one difficulty that we have that a person must have both the father and mother to be represented there. For example, if we can come to a situation where only the mother is known, the father is not known I will not be able to represent that because it will always have to come as a triplet of three persons thel the person under consideration her father and her mother, but if I represent the person and the father in one relationship; the person and the mother in another relationship, then I can take care of the situation where when one of the parents are known; I can still represent this.

So, there are certain tradeoffs which can be done between the choice of binary and non binary relationships, but; obviously, there are certain relationships which are inherently non binary for example, the project guide example we have seen. The project guide information cannot be decomposed without certain loss of information to be represented

by say the instructor and the project and another relationship between the student and the project it really that does not represent the same information.

(Refer Slide Time: 30:20)

**Converting Non-Binary Relationships to Binary Form**

- In general, any non-binary relationship can be represented using binary relationships by creating an artificial entity set.
- Replace  $R$  between entity sets  $A$ ,  $B$  and  $C$  by an entity set  $E$ , and three relationship sets:
  1.  $R_A$ , relating  $E$  and  $A$
  2.  $R_B$ , relating  $E$  and  $B$
  3.  $R_C$ , relating  $E$  and  $C$
- Create an identifying attribute for  $E$  and add any attributes of  $R$  to  $E$
- For each relationship  $(a_i, b_i, c_i)$  in  $R$ , create
  1. a new entity  $e_i$  in the entity set  $E$
  2. add  $(e_i, a_i)$  to  $R_A$
  3. add  $(e_i, b_i)$  to  $R_B$
  4. add  $(e_i, c_i)$  to  $R_C$

(a)

(b)

15.22 ©Silberschatz, Korth and Sudarshan

So, in general you can convert a non binary relationship by in the binary form by doing this. So, this is a ternary relationship being shown and for doing that these are the three entity sets involving the ternary relationship and to make decompose into a ternary relationship; what we do is into binary relationships we inject a new entity artificial entity set  $E$  and then we define three different relations between them.

So, which individually relates to the entity sets  $A$ ,  $B$  and  $C$ . So, this is a standard decomposition and you can easily understand that  $A$ ,  $B$  and  $C$  in our earlier example could all be persons and  $R_A$  could mean that father of  $R_B$  could mean mother of and so, on. So, I can do it in decompose it in this manner and represent that.

(Refer Slide Time: 31:24)

The slide has a header 'Converting Non-Binary Relationships (Cont.)' with a small sailboat icon. The main content is a bulleted list:

- Also need to translate constraints
  - Translating all constraints may not be possible
  - There may be instances in the translated schema that cannot correspond to any instance of  $R$ 
    - Exercise: add constraints to the relationships  $R_A$ ,  $R_B$  and  $R_C$  to ensure that a newly created entity corresponds to exactly one entity in each of entity sets  $A$ ,  $B$  and  $C$
  - We can avoid creating an identifying attribute by making  $E$  a weak entity set (described shortly) identified by the three relationship sets

On the left margin, there is vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018'. At the bottom, it says 'Database System Concepts'.

At the bottom right, there is a navigation bar with icons and the text '15.23 ©Silberschatz, Korth and Sudarshan'.

Now while we do this decomposition we will also have to remember in that; we need to translate all constraints that are present for the ternary relationship. And often times it may become difficult to translate all constraints, it may not be possible and there may be instances in the translated schema that cannot correspond to an instance of the original relationship.

So, we will have to avoid we can we will have to take care of this situation by identifying attributes. And making use of the weak entity sets which we have already seen in our earlier discussions.

(Refer Slide Time: 32:09)

The slide has a header 'E-R Design Decisions' in red. On the left, there is a small sailboat icon and some vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. The main content is a bulleted list of design decisions:

- The use of an attribute or entity set to represent an object
- Whether a real-world concept is best expressed by an entity set or a relationship set
- The use of a ternary relationship versus a pair of binary relationships
- The use of a strong or weak entity set
- The use of specialization/generalization – contributes to modularity in the design
- The use of aggregation – can treat the aggregate entity set as a single unit without concern for the details of its internal structure

At the bottom, there is a video frame showing a man speaking, the text 'Database System Concepts', the number '15.24', and a navigation bar.

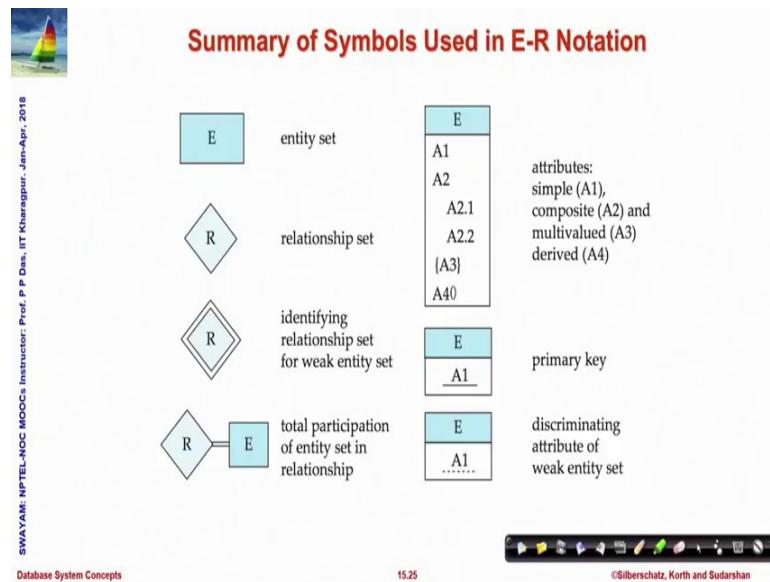
So, if we summarize the discussions on the E-R design decisions; we see that the first decision that we need to take in case of design is the use of an attribute or entity set to represent the object.

So, that is the first modeling that what is the concept and what are the attributes or what is the representing entity set for the object that we are trying to deal with instructor, student project and so, on. And we will also have to see whether in the real world this actually is an entity set or it is a relationship set that it is not a concept by itself, but is a concept which relates two or more entity sets and thereby becomes a set of representation.

The use of ternary relationship versus a pair of binary relationship; this trade off will have to be weighed as a design consideration; we have to look into the use of strong or weak entity set. So, we will have to identify the weak entity sets and see if they should be represented through the identifying relation as against a strong entity set. We have to identify the specialization generalization situation where so, that we can get more specific information and create appropriate modularity in the design.

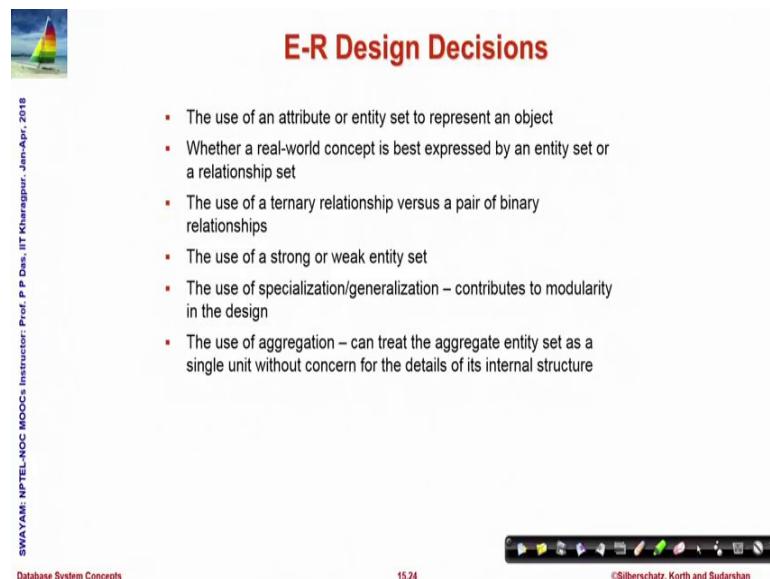
We have to look at aggregation which where we can aggregate entity sets bound by a relationship and create an abstract single unit which can play a role of an independent entity set in the whole design.

(Refer Slide Time: 34:00)



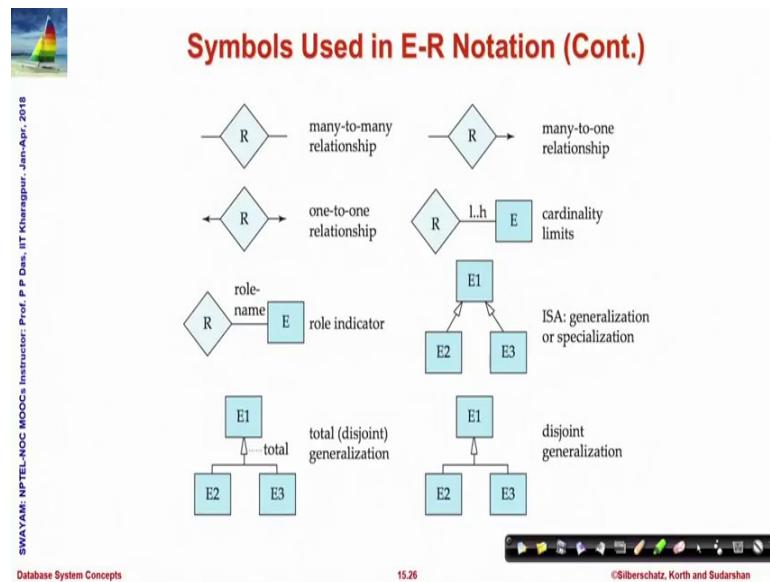
So, this is the basic. So, these are the basic design decisions that you need to make.

(Refer Slide Time: 34:02)



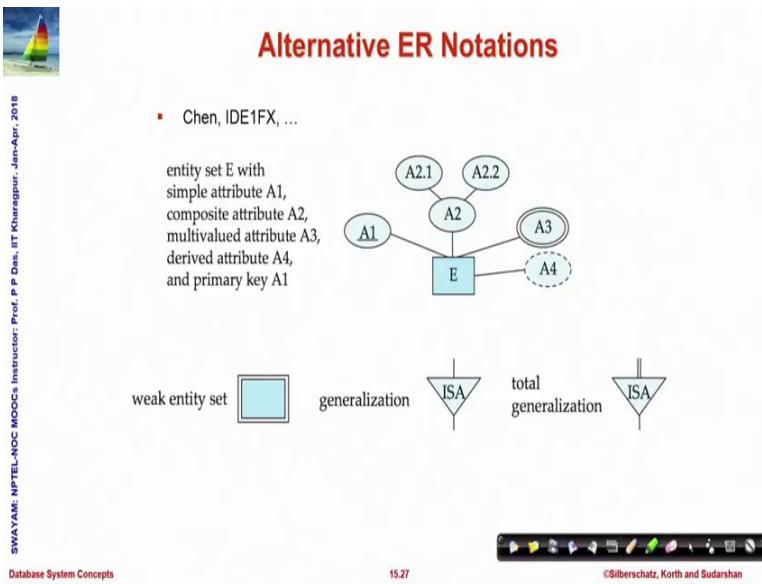
And we will certainly come up with lot more of design decisions as we go along. And before I close in the presentation I have summarized the different symbols that are used in the E-R notation. So, I will not these have already been discussed in depth. So, I will not go through them one by one, but I have put them as a list in the couple of slides.

(Refer Slide Time: 34:30)



This is a next slide in that which will be a quick reference for you while you are initially doing the E-R diagram so, that you know exactly which symbol to pick up for what situation.

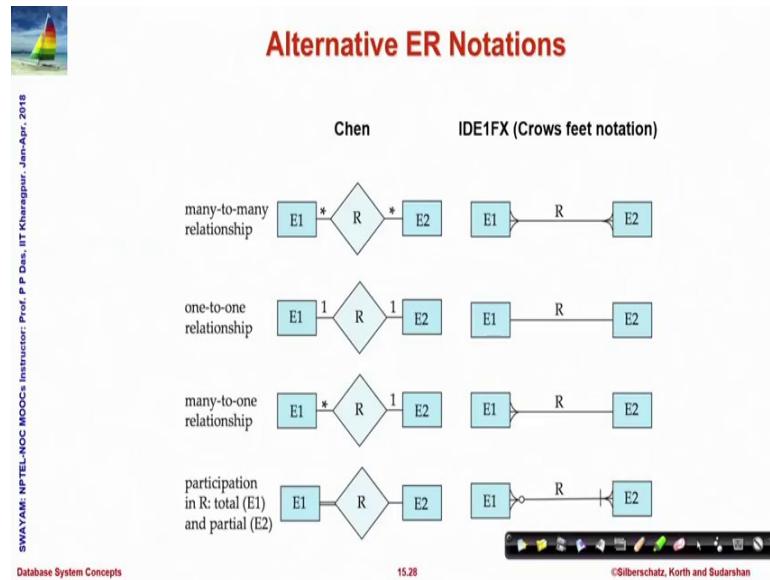
(Refer Slide Time: 34:42)



And at the end also there are few slides which show you that the E-R notation itself is not a unique one.

There are multiple ways to represent similar things for example, this is one which is showing you different composite attributes, the generalization, relationship is shown differently.

(Refer Slide Time: 35:04)



So, there are; these are all different styles of showing the constraints that that apply to a particular relationship. And we will I mean we have included this not because we will use these alternate notations, but I have put them because it is possible that you come across some E-R diagram where these notations are used and if you come across and you are not able to identify then please refer to this slides.

(Refer Slide Time: 35:36)

**Module Summary**

- Discussed the extended features of E-R Model
- Deliberated on various design issues

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts      15.29      ©Silberschatz, Korth and Sudarshan

And you will be able to recognize what is what is corresponding symbol that you already know.

So, in this module we have discussed the extended features of E-R model and we have deliberated on certain design issues. And we will close our discussion on the entity relationship model here and move on to discuss the actual relational design.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 16**  
**Relational Database Design**

Welcome to Module 16 of Database Management Systems till the last module which closed with the third week.

(Refer Slide Time: 00:31)

The slide has a header "Week 03 Recap" in red, a small sailboat icon, and a "PPD" watermark. It lists topics covered in Week 3 across four modules:

- Module 11: Advanced SQL**
  - Accessing SQL From a Programming Language
  - Functions and Procedural Constructs
  - Triggers
- Module 12: Formal Relational Query Languages**
  - Relational Algebra
  - Tuple Relational Calculus (Overview only)
  - Domain Relational Calculus (Overview only)
  - Equivalence of Algebra and Calculus
- Module 13: Entity-Relationship Model/1**
  - Design Process
  - E-R Model
- Module 14: Entity-Relationship Model/2**
  - E-R Diagram
  - E-R Model to Relational Schema
- Module 15: Entity-Relationship Model/3**
  - Extended E-R Features
  - Design Issues

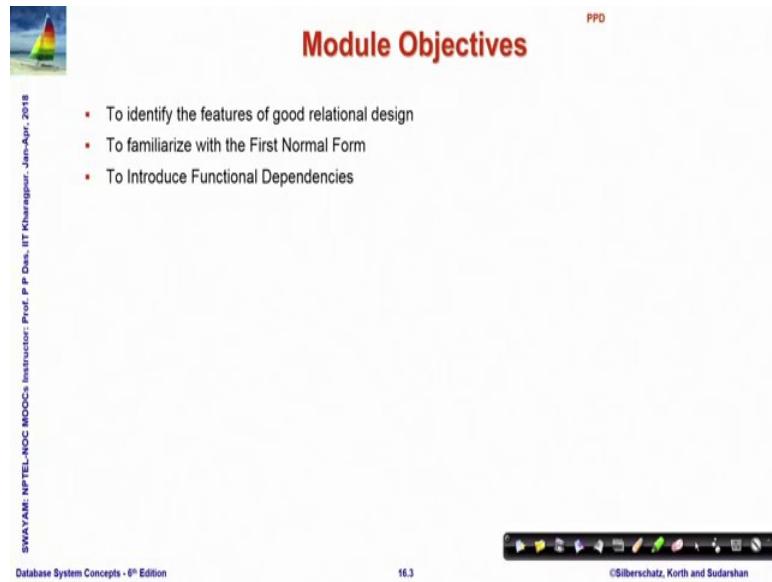
Vertical text on the left: SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Bottom footer: Database System Concepts - 8<sup>th</sup> Edition, 16.2, ©Silberschatz, Korth and Sudarshan

Specifically in the third week, we talked about certain advanced features of SQL and the formal query language in terms of relational and algebra and calculi and then, we talked in a depth in terms of the entity relationship model, the first basic conceptual level representation of the real world that we can do in terms of designing a system.

Now, our next task would be to take it to more proper complete relational database design and this will have a lot of theory at different levels that we need to understand. We will slowly develop that and this discussion will span five modules that is we will take the whole week to complete.

(Refer Slide Time: 01:25)



**Module Objectives**

PPD

- To identify the features of good relational design
- To familiarize with the First Normal Form
- To Introduce Functional Dependencies

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Doshi, IIT Kharagpur - Jan-Apr - 2018

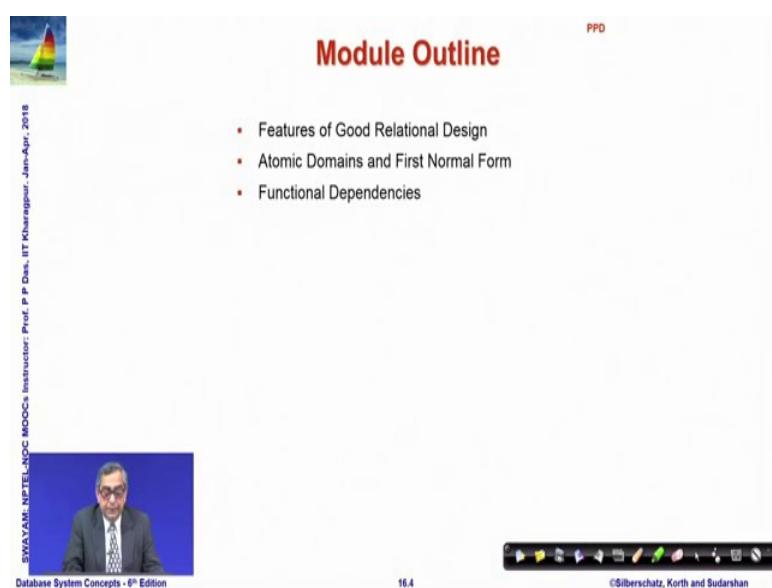
Database System Concepts - 8<sup>th</sup> Edition

16.3

©Silberschatz, Korth and Sudarshan

So, the objective of the current module, the first of the Relational Design Module is to identify features of good relational design having done the ER model. We have ER model we do the ER model, we have the entity sets relationships, we convert them to schema. We have seen how to do that and immediately we have some design, but the question is, is it a good design. So, we will discuss about what are the features of a good design and then, we will introduce the formal definition of what is First Normal Form and we will introduce a very critical concept of relational database design, the Functional Dependencies.

(Refer Slide Time: 02:07)



**Module Outline**

PPD

- Features of Good Relational Design
- Atomic Domains and First Normal Form
- Functional Dependencies

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Doshi, IIT Kharagpur - Jan-Apr - 2018



Database System Concepts - 8<sup>th</sup> Edition

16.4

©Silberschatz, Korth and Sudarshan

These are the module outline for that.

(Refer Slide Time: 02:10)

The slide features a small sailboat icon in the top left corner. In the top right, the text 'PPD' is written above a bulleted list: '•Features of Good Relational Design', '•Atomic Domains and First Normal Form', and '•Functional Dependencies'. The main title 'FEATURES OF GOOD RELATIONAL DESIGN' is centered in large red capital letters. Below the title, the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr., 2018' is visible. At the bottom, there is a navigation bar with icons and the text 'Database System Concepts - 8<sup>th</sup> Edition', '16.5', and '©Silberschatz, Korth and Sudarshan'.

So, to start with the features of good relational design, let us take an example.

(Refer Slide Time: 02:13)

The slide shows a table with columns: ID, name, salary, dept\_name, building, and budget. Handwritten annotations include a blue arrow pointing to the 'dept\_name' column, a blue circle around the 'dept\_name' column header, and a blue bracket spanning the 'dept\_name' and 'building' columns. To the right of the table, handwritten notes say 'Redundancy' with an arrow pointing to the 'dept\_name' column, 'Anomaly' with an arrow pointing to the 'building' column, and 'Update insertion deletion' with an arrow pointing to the 'dept\_name' column. The slide also includes the title 'Combine Schemas?' in red, the text 'Suppose we combine instructor and department into inst\_dept', '(No connection to relationship set inst\_dept)', and the footer information 'Database System Concepts - 8<sup>th</sup> Edition', '16.6', and '©Silberschatz, Korth and Sudarshan'.

Suppose we have seen the instructor relation, instructor entity set as a relation. You have seen the department relation. Now, let us consider that if these two were not two separate relations, if they were all kept in a common relation that is all the attributes are kept in the common relation, so earlier if you recall that your instructor relation was this and your department relation was this much. So, if we keep everything together, of course we

are calling it `inst_dept`, but please keep in mind this is not the same `inst_dept` that we discussed in terms of the ER model. This is just putting these two together.

Now, the question is if you look into this data carefully, for example if you look into this particular row, if you look into this particular row and if you look into this particular row, these are rows of instructors who all belong to computer science. Now, earlier we were representing the information of instructor only in this part. So, we just knew that it is computer science and we represent the information of department in this part. So, given a department name say computer science, we knew, where is it located, the building and what budget it has. Now, when we are combined, we will see that naturally since computer science is located in the Taylor building, we know that it has a budget of say 100,000. So, all of these records will have this information repeated.

So, this is not a very good situation. This is not a good situation because this kind of situation is typically in database known as redundancy, that is you have the same data in multiple places. So, what is the consequence of redundancy? For example, there could be different kinds of anomaly when you have redundancy. What is an anomaly? An anomaly is the possibility of certain data getting inconsistent. For example, let us say Computer Science department moves from Taylor building to Painter building. Now, what will happen if it moves to painter building? Then, I will need to remove this, make it a painter, make this value painter. I have to also do this, make this painter. I have to also do this, make this painter. So, if I have a change, then I will have to make the change at multiple entries. Think about the earlier situation where I just had these three in my department relation, then naturally computer science had only one row and therefore, this change, this update could be done at only one place.

So, it is not only that if while doing this in case of this redundancy, I have to do this multiple times. It also has the difficulty that if I forget to update any one of them or more of them, then I have inconsistent data. Similarly, if I want to insert a new value, I will have to do that for all this redundant information. If I have to delete say for some reason let say the university decides to wind up the Physics department, then I have to delete all these rows which have physics as an entry and the consequence of that is the department is deleted, but as a consequence of that I will delete the whole row and therefore, I will not only remove the department, but I will also remove the corresponding instructor who was enrolled for that department.

So, this kind of redundancy can lead to different kinds of anomalies in a database design. On the other hand, if you look at, well why am I complicating the whole situation? We have already had a good design in terms of where these anomalies were, not their departments were separate instructor was separate. In that case, the situation is that to answer some of the queries, I may have to do a very expensive join operation. For example, if I want to know if Einstein wants to know what is the budget of his department that cannot be found out from the earlier instructor database, instructor relation which had only these fields.

So, I have to pick up Einstein from here, do a join based on the department name, dept\_name with the department table department relation and then only, I will be able to find out that an Einstein belongs to Physics. Physics has a budget of 70000. So, Einstein's department has a budget 70000. So, there is a tradeoff between how much data information if you make redundant and lead to different anomalous situations or how much data you optimize in the representation, but get into the possible situation of having a higher cost in terms of answering your queries.

(Refer Slide Time: 07:49)

**Combine Schemas?**

- Suppose we combine *instructor* and *department* into *inst\_dept*
  - (No connection to relationship set *inst\_dept*)
- Result is possible repetition of information (*building* and *budget* against *dept\_name*)

| ID    | name       | salary | dept_name  | building | budget |
|-------|------------|--------|------------|----------|--------|
| 22222 | Einstein   | 95000  | Physics    | Watson   | 70000  |
| 12121 | Wu         | 90000  | Finance    | Painter  | 120000 |
| 32343 | El Said    | 60000  | History    | Painter  | 50000  |
| 45565 | Katz       | 75000  | Comp. Sci. | Taylor   | 100000 |
| 98345 | Kim        | 80000  | Elec. Eng. | Taylor   | 85000  |
| 76766 | Crick      | 72000  | Biology    | Watson   | 90000  |
| 10101 | Srinivasan | 65000  | Comp. Sci. | Taylor   | 100000 |
| 58583 | Califieri  | 62000  | History    | Painter  | 50000  |
| 83821 | Brandt     | 92000  | Comp. Sci. | Taylor   | 100000 |
| 15151 | Mozart     | 40000  | Music      | Packard  | 80000  |
| 33456 | Gold       | 87000  | Physics    | Watson   | 70000  |
| 76543 | Singh      | 80000  | Finance    | Painter  | 120000 |

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition      16.6      ©Silberschatz, Korth and Sudarshan

So, this is one of the core design issues that we will start with. So, let us look into some more.

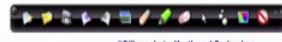
(Refer Slide Time: 07:54)



## A Combined Schema Without Repetition

SWAYAM-NPTEL-NCOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018

- Consider combining relations
  - $\text{sec\_class(sec\_id, building, room\_number)}$  and
  - $\text{section(course\_id, sec\_id, semester, year)}$into one relation
  - $\text{section(course\_id, sec\_id, semester, year, building, room\_number)}$
- No repetition in this case



16.7

©Silberschatz, Korth and Sudarshan

Of these examples, let us say we look into another combined combination of schema. Suppose section is a relation which have the sections of a course which give the section id, semester, year and say section class is another relation which tell me for a section id, what is the building and room number where it is located. So, if we have this kind of relations combined into a common relation, then I have all of these coming from the section and this and these coming from the section class, but we can see that there is no repetition or redundant information in this case.

So, it is note that the combining schemas is necessarily always bad in terms of repetition or in terms of redundancy. So, different situations will have to be assessed.

(Refer Slide Time: 08:57)



## What About Smaller Schemas?

SWAYAM-NPTEL-NCOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018

- Suppose we had started with  $\text{inst\_dept}$ . How would we know to split up (**decompose**) it into  $\text{instructor}$  and  $\text{department}$ ?
- Write a rule "if there were a schema  $(\text{dept\_name}, \text{building}, \text{budget})$ , then  $\text{dept\_name}$  would be a candidate key"
- Denote as a **functional dependency**:  
 $\text{dept\_name} \rightarrow \text{building, budget}$
- In  $\text{inst\_dept}$ , because  $\text{dept\_name}$  is not a candidate key, the building and budget of a department may have to be repeated.
  - This indicates the need to decompose  $\text{inst\_dept}$
- Not all decompositions are good. Suppose we decompose  $\text{employee}(ID, name, street, city, salary)$  into
  - $\text{employee1}(ID, name)$
  - $\text{employee2}(name, street, city, salary)$
- The next slide shows how we lose information -- we cannot reconstruct the original  $\text{employee}$  relation -- and so, this is a **lossy decomposition**.



Database System Concepts - 8<sup>th</sup> Edition

16.8

©Silberschatz, Korth and Sudarshan

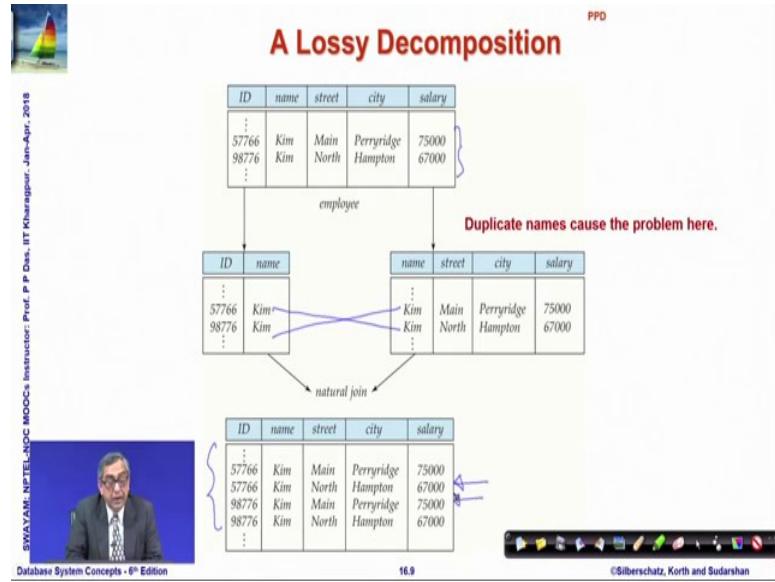
So, if we want to look at the other side that if we just as I said that if we make the schema smaller, so that we avoid redundancy and then, what we see that 12 from the combined `inst_dept` relationship that we saw. So, let me just show you once more. So, this is if we look at the `inst_dept`, then in this we can we know that from the earlier information about the department relationship that department name is a key, is a primary key of the relation which has department name, building and budget. What is the consequence of being a primary key? If it is a primary key, then no two records can match on the department name and be different in terms of the building and the budget.

If two records are there which have the same department name, they must be identical. So, they are distinguishable completely by that. So, let us see what is the consequence of this. So, we are saying that we write it as a rule that if there is a schema department, name, building, budget, then department name would be a candidate key and we write this observation that if two records match on the department name, they must match on the building and budget and very loosely, we will come to the formal definition. Very loosely we call this the functional dependency. We say that the building and budget is functionally dependent on the department name and that is a situation where we can split this `inst_dept` and create a smaller relationship because department name is not a candidate key in the `inst_dept`. It does not decide the records of `inst_dept` uniquely.

So, since it does not, so when the values of this key, this attribute department name is duplicated or triplicated, the values of the building and budget are repeated and we have the redundancy. So, this is a situation, very common situation which is indicative of the fact that we need a decomposition into smaller, but at the same time we can also observe, I mean let us take a different example. If we are thinking that decomposition is the panacea of solving these kind of redundancy and related problems, then let us try to see a different relationship `employee` which has id, name, street, city, salary and we want to make it smaller and want to make two relations id and name and name, city street, salary.

So, if we do that, then how do we get the salary for a particular id? We will naturally have to join these two relations in terms of the common attribute name. We have seen that in the query and the question is when I do this join, do I get back the original information or I lose some information.

(Refer Slide Time: 12:42)



Look at an example. So, here is an example of the combined instance and I have two different ids, but incidentally the names are same. The names of these two distinct employees are same. So, when I decompose, I get this relation which shows id and name. I get this relation which against the name shows this, but when I try to join them by natural join, I not only get the combination of this with this which is what I need, but I also get this combination. So, if I say this is what I get as well in terms of natural join, this is what I get as well in terms of the natural join which are really not there in the original relation.

So, you can see that in the natural join, I get four records, I get four rows whereas, in the original one I had only two rows. So, I get some entries which are actually erroneous. These are not there in the database. So, this is when this happens. We say that we have loss of information and such joins are said to be lossy joins. So, when we decompose, we need to make sure that our joins are lossless in nature; otherwise that is not a good design.

(Refer Slide Time: 14:08)



### Example of Lossless-Join Decomposition

SWAYAM NPTEL-NC MOOCs Instructor: Prof. P. Desai, IIT Kharagpur - Jan-Apr. 2018

- Lossless join decomposition

- Decomposition of  $R = (A, B, C)$

$$R_1 = (A, B) \quad R_2 = (B, C)$$

| A        | B | C |
|----------|---|---|
| $\alpha$ | 1 | A |
| $\beta$  | 2 | B |

$r$

| A        | B |
|----------|---|
| $\alpha$ | 1 |
| $\beta$  | 2 |

$\Pi_{A,B}(r)$

| B | C |
|---|---|
| 1 | A |
| 2 | B |

$\Pi_{B,C}(r)$

$\Pi_A(r) \bowtie \Pi_B(r)$

| A        | B | C |
|----------|---|---|
| $\alpha$ | 1 | A |
| $\beta$  | 2 | B |



So, you can see this is again a hypothetical example which shows three attributes in relation having three attributes. You have decomposed it into two relations having two attributes each and we have shown an instance and in this case, it shows that when I take the join, the original information I am sorry, wait.

When I take the join, the original information is completely retrieved. I get back the same table and when that happens, I say that the join is lossless. So, what we need to understand is on one side there is a need to decompose relations into smaller relations to reduce redundancy and while we do that, we will also have to keep this in mind that the smaller relations must be composable through certain natural join procedure to the original relation, and I must get back that original relation, otherwise I have a lossy join which is not acceptable. Also, the decomposition will have the costs of doing natural join every time I want to answer those queries.

(Refer Slide Time: 15:35)



PPD

- Features of Good Relational Design
- Atomic Domains and First Normal Form
- Functional Dependencies

## ATOMIC DOMAINS AND FIRST NORMAL FORM

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

16.11

©Silberschatz, Korth and Sudarshan

The next that we look at is the way the relationships are categorized as First Normal Form.

(Refer Slide Time: 15:45)



### First Normal Form (1NF)

PPD

- Domain is **atomic** if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes
    - Identification numbers like CS101 that can be broken up into parts
- A relational schema R is in **first normal form** if
  - the domains of all attributes of R are atomic
  - the value of each attribute contains only a single value from that domain
- Non-atomic values complicate storage and encourage redundant (repeated) storage of data
  - Example: Set of accounts stored with each customer, and set of owners stored with each account
  - We assume all relations are in first normal form

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

16.12

©Silberschatz, Korth and Sudarshan

We consider that the domains of attributes are atomic if they are indivisible. So, anything that is a number, string and so on is considered to be atomic and we say a relational schema is in its First Normal Form if the domains of all attributes are atomic and all attributes single valued, there is no multi valued attribute. If these conditions are satisfied, then we will say that every relate that relational schema is in its First Normal Form. So, we will slowly understand the purpose of defining such normal forms, but let us initially understand the definition. So, if we have attributes which are composite in

nature, naturally my relationship, my relational schema is not in First Normal Form if we have attributes which are multiple valued, it is not so.

(Refer Slide Time: 16:44)

The slide has a decorative header with a sailboat icon and the title 'First Normal Form (Cont'd)' in red. On the left, there is vertical text: 'SWAYAM-NETELNOC-MOOCs', 'Instructor: Prof. P P Das', 'Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018', and 'Database System Concepts - 8th Edition'. The main content is a bulleted list:

- Atomicity is actually a property of how the elements of the domain are used
  - Example: Strings would normally be considered indivisible
  - Suppose that students are given roll numbers which are strings of the form CS0012 or EE1127
  - If the first two characters are extracted to find the department, the domain of roll numbers is not atomic
  - Doing so is a bad idea: leads to encoding of information in application program rather than in the database

At the bottom, there is a navigation bar with icons for back, forward, search, and other presentation controls. The page number '16.13' is at the bottom center, and the copyright notice '©Silberschatz, Korth and Sudarshan' is at the bottom right.

So, if we say that we have possible values are like this, then if we just treat them as strings, then the corresponding relational schema is in First Normal Form, but if we say that from this string we can extract the first two characters which is CS which tells me what is a department. The next four characters gives me a number, the serial number of the particular student in the role. Then I am not actually using an atomic domain because my domain needs to be interpreted separately than just being a value. So, these are not parts of what can be a First Normal Form.

(Refer Slide Time: 17:28)



## First Normal Form (Cont'd)

PPD

- The following is not in 1NF

Customer

| Customer ID | First Name | Surname | Telephone Number                     |
|-------------|------------|---------|--------------------------------------|
| 123         | Pooja      | Singh   | 555-861-2025 192-122-1111            |
| 456         | San        | Zhang   | (555) 403-1659 Ext. 53; 182-929-2929 |
| 789         | John       | Doe     | 555-808-9633                         |

- A telephone number is composite
- Telephone number is multi-valued

Source: [https://en.wikipedia.org/wiki/First\\_normal\\_form](https://en.wikipedia.org/wiki/First_normal_form)

Database System Concepts - 8<sup>th</sup> Edition

16.14

©Silberschatz, Korth and Sudarshan



So, I have given some examples of what is not and what is First Normal Form. So, this is an example where at the telephone number field exists and there can be multiple telephone numbers. So, this is not in First Normal Form because the telephone number itself is composite because it has different components and also, you can have multiple telephone number. So, this relation is not in the First Normal Form.

(Refer Slide Time: 17:54)



## First Normal Form (Cont'd)

PPD

- Consider:

Customer

| Customer ID | First Name | Surname | Telephone Number1      | Telephone Number2 |
|-------------|------------|---------|------------------------|-------------------|
| 123         | Pooja      | Singh   | 555-861-2025           | 192-122-1111      |
| 456         | San        | Zhang   | (555) 403-1659 Ext. 53 | 182-929-2929      |
| 789         | John       | Doe     | 555-808-9633           |                   |

- Is in 1NF if telephone number is not considered composite
- However, conceptually, we have two attributes for the same concept
  - Arbitrary and meaningless ordering of attributes
  - How to search telephone numbers
  - Why only two numbers?

Source: [https://en.wikipedia.org/wiki/First\\_normal\\_form](https://en.wikipedia.org/wiki/First_normal_form)

Database System Concepts - 8<sup>th</sup> Edition

16.15

©Silberschatz, Korth and Sudarshan



What you can do? You can separate out these phone numbers into two different attributes; Telephone number 1 and 2. Even then it is not exactly in First Normal Form because you do not know in which order they should be handled. If you have to search for a telephone number, then you will have to search multiple attributes which are

conceptually same and then, the question is why only two attributes. Cannot anybody have 3 phone numbers, 7 phone numbers and so on. So, this is really not a good option.

(Refer Slide Time: 18:26)

**First Normal Form (Cont'd)**

- Is the following in 1NF?

| Customer    |            |         |                        |
|-------------|------------|---------|------------------------|
| Customer ID | First Name | Surname | Telephone Number       |
| 123         | Pooja      | Singh   | 555-861-2025           |
| 123         | Pooja      | Singh   | 192-122-1111           |
| 456         | San        | Zhang   | 182-929-2929           |
| 456         | San        | Zhang   | (555) 403-1659 Ext. 53 |
| 789         | John       | Doe     | 555-808-9633           |

- Duplicated information
- ID is no more the key. Key is (ID, Telephone Number)

Source: [https://en.wikipedia.org/wiki/First\\_normal\\_form](https://en.wikipedia.org/wiki/First_normal_form)

Database System Concepts - 8<sup>th</sup> Edition

16.16

©Silberschatz, Korth and Sudarshan

So, the other way could be that for every telephone number, you introduce a separate row. Once you do that you already know you have redundancy and you have possibilities of varied kinds of anomalies that could happen.

(Refer Slide Time: 18:40)

**First Normal Form (Cont'd)**

- Better to have 2 relations:

| Customer Name |            |         | Customer Telephone Number |                        |
|---------------|------------|---------|---------------------------|------------------------|
| Customer ID   | First Name | Surname | Customer ID               | Telephone Number       |
| 123           | Pooja      | Singh   | 123                       | 555-861-2025           |
| 456           | San        | Zhang   | 123                       | 192-122-1111           |
| 789           | John       | Doe     | 456                       | (555) 403-1659 Ext. 53 |
|               |            |         | 456                       | 182-929-2929           |
|               |            |         | 789                       | 555-808-9633           |

- One-to-Many relationship between parent and child relations
- Incidentally, satisfies 2NF and 3NF

Source: [https://en.wikipedia.org/wiki/First\\_normal\\_form](https://en.wikipedia.org/wiki/First_normal_form)

Database System Concepts - 8<sup>th</sup> Edition

16.17

©Silberschatz, Korth and Sudarshan

So, one way it could be achieved is we follow the principle that we had seen in ER modelling that this multivalued dependency can be represented in terms of a separate

relation where against the customer id we just keep the telephone number. So, we can keep multiple of them and we take that out from the customer name. So, one to many relationship between the parent and the child, between the customer name and telephone number, every customer may have more than one telephone number is possible and that makes it 1 NF relation, First Normal Form relation and we will later on see that it also is 2 NF and 3 NF, but that is a future story.

(Refer Slide Time: 19:29)

The slide features a sailboat icon in the top left corner. In the top right, the letters 'PPD' are displayed above a bulleted list of topics: '•Features of Good Relational Design', '•Atomic Domains and First Normal Form', and '•Functional Dependencies'. The main title 'FUNCTIONAL DEPENDENCIES' is centered in large red capital letters. At the bottom, there is a navigation bar with icons for back, forward, search, and other presentation controls. The vertical footer on the left contains the text 'SWAYAM: NPTEL-NOC MOOCs', 'Instructor: Prof. P. P. Deshpande, IIT Kharagpur', and 'Jan-Apr. 2018'. The footer also includes the text 'Database System Concepts - 6<sup>th</sup> Edition' at the bottom.

Now, finally we come to the core of what the mathematical formulation which dictates much of the data base, relational database design is known as functional dependencies.

(Refer Slide Time: 19:47)



## Goal — Devise a Theory for the Following

SWAYAM NPTEL-NCX MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018

- Decide whether a particular relation  $R$  is in "good" form.
- In the case that a relation  $R$  is not in "good" form, decompose it into a set of relations  $\{R_1, R_2, \dots, R_n\}$  such that
  - each relation is in good form ✓
  - the decomposition is a lossless-join decomposition

$$\begin{aligned} R_i &= \text{set of attributes} \\ R &= \bigcup_i R_i \end{aligned}$$

Database System Concepts - 8<sup>th</sup> Edition

16.19

©Silberschatz, Korth and Sudarshan



I just talked about little bit of that while talking about department name building and budget. Now, to decide whether a particular relation is good or rather a particular relational scheme is good, we need to check against certain measures and if it is not good, we need to decompose it into a set of relations such that these conditions satisfy that every, each one of these,  $\{R_1, R_2, \dots, R_n\}$ . So, I mean if you have you now got rusted, then it is basically  $R_i$  is a set of attributes because it is a relational schema. A relational schema is a set of attributes.

So, naturally  $R$  will be the union of all of these,  $R_i$  the total set of attributes. So, instead of keeping all the information into one relation in one table, we are basically decomposing it into  $n$  different schemas.

(Refer Slide Time: 20:42)



## Goal — Devise a Theory for the Following

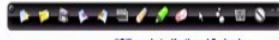
SWAYAM-NPTEL-NOOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018

- Decide whether a particular relation  $R$  is in "good" form.
- In the case that a relation  $R$  is not in "good" form, decompose it into a set of relations ( $R_1, R_2, \dots, R_n$ ) such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- Our theory is based on:
  - functional dependencies
  - multivalued dependencies

Database System Concepts - 8<sup>th</sup> Edition

16.19

©Silberschatz, Korth and Sudarshan



So, what we need to guarantee is each one of these relation  $R_1, R_2, \dots, R_n$  is in good form. How do I get back the original relation? Original relation that was represented by all that attributes enough is to take a lossless join. This would take a join and that this decomposition must give me a lossless join. So, to ensure that; we make use of two key ideas more foundationally; functional dependencies and then, multivalued dependencies.

(Refer Slide Time: 21:20)



## Functional Dependencies

SWAYAM-NPTEL-NOOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018

- Constraints on the set of legal relations
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes
- A functional dependency is a generalization of the notion of a key



16.20

©Silberschatz, Korth and Sudarshan



A functional dependency is a constraint on the set of legal relation. So, mind you it is a constraint on the schema and once that constraint is defined, it must hold for all relations that the schema satisfied. So, here we need that the value of certain set of attributes uniquely determine the value of another set of attributes. So, I know the value of three

attributes, I should be able to say that the values of the other four attributes would be fixed. So, you have already seen this notion in terms of key or super key. You have seen that similar type of concept exists where we said a key is a set of attributes, so that if the values of two rows are identical over these set of attributes, then the two peoples, the two rows must be totally identical.

So, key is something which does a similar thing as a functional dependency, but is more specific. Functional dependencies are generalization.

(Refer Slide Time: 22:30)

The slide has a title 'Functional Dependencies (Cont.)' in red at the top right. On the left, there is a small logo of a sailboat on water. The main content area contains the following text and a handwritten note:

- Let  $R$  be a relation schema
- $\alpha \subseteq R$  and  $\beta \subseteq R$
- The **functional dependency**

$\alpha \rightarrow \beta$

**holds on  $R$**  if and only if for any legal relations  $r(R)$ , whenever any two tuples  $t_1$  and  $t_2$  of  $r$  agree on the attributes  $\alpha$ , they also agree on the attributes  $\beta$ . That is,

$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

A handwritten note in blue ink shows a diagram with two sets of brackets: one above labeled  $\alpha$  and another below labeled  $\beta$ , with an arrow pointing from the first to the second.

At the bottom, there is footer text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr., 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '16.21', and '©Silberschatz, Korth and Sudarshan'.

So, let us formally define that let  $R$  be a relational schema which means that it is a set of attributes and let us say  $\alpha, \beta \subseteq R$ , then we write this and note this notation.  $\alpha$  is a set of attributes;  $\beta$  is another set of attributes. Both are subset of the same  $R$  and we say  $\alpha \rightarrow \beta$  that is if I know the value of a tuple over the attributes of  $\alpha$ , then the values of that tuple over the attributes of  $\beta$  would be fixed or in other words, they say that if I have two tuples  $t_1$  and  $t_2$  and their values over the set of  $\alpha$  attributes are same, then necessarily their values over the set of  $\beta$  attributes must be same and mind you this is something which is a design constraint. It is not just an incidental property. It is not just the fact that a particular instance of a schema satisfies this, but when you say this is a functional dependency, we need all possible past, present and future instances of the schema must satisfy this.

(Refer Slide Time: 24:03)



## Functional Dependencies (Cont.)

- Let  $R$  be a relation schema  
 $\alpha \subseteq R$  and  $\beta \subseteq R$
- The **functional dependency**  
 $\alpha \rightarrow \beta$   
holds on  $R$  if and only if for any legal relations  $r(R)$ , whenever any two tuples  $t_1$  and  $t_2$  of  $r$  agree on the attributes  $\alpha$ , they also agree on the attributes  $\beta$ . That is,  
 $t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$
- Example: Consider  $r(A,B)$  with the following instance of  $r$ .

|   |   |
|---|---|
| 1 | 4 |
| 1 | 5 |
| 3 | 7 |

- On this instance,  $A \rightarrow B$  does NOT hold, but  $B \rightarrow A$  does hold.



So, consider this if you take a relation, a schema with an instance as given here between two attributes **A** and **B**, then we can say at least given this instance not we still do not know what happens in the whole schema for all instances, but on this instance we can say that  $A \rightarrow B$  does not hold because between the first and the second record, the value of **A** is same one, but the value of **B** are different 4 and 5, but we can certainly say that on this instance at least  $B \rightarrow A$  holds because whenever the value if we take any two tuples, their value over **B** does not at all match. If they does not match, then naturally there is no question of what happens to the value of the tuple over the set of attributes **A**. So, we will say that  $B \rightarrow A$  holds in this instance.

(Refer Slide Time: 25:01)



## Functional Dependencies (Cont.)

- $K$  is a superkey for relation schema  $R$  if and only if  $K \rightarrow R$
- $K$  is a candidate key for  $R$  if and only if
  - $K \rightarrow R$ , and
  - for no  $\alpha \subset K$ ,  $\alpha \rightarrow R$
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:  
 $inst\_dept (ID, name, salary, dept\_name, building, budget)$ .

We expect these functional dependencies to hold:

$$dept\_name \rightarrow building$$

and  $ID \rightarrow building$

but would not expect the following to hold:

$$dept\_name \rightarrow salary$$



So, given this definition of functional dependency, now we can have a formal definition of what the super key is. Super key is naturally a subset of attributes which → the whole set and a candidate key is a super key which is minimal which means that  $k$  is a candidate key. If the two conditions have to satisfy this condition say there is a super key that it → all the attributes and the other condition says minimality that there is no subset  $\alpha \subset k$ , such that  $\alpha \rightarrow R$  if there exists a subset  $\alpha \subset k$ , the proper subset  $\alpha \subset k$ . So, that  $\alpha \rightarrow R$ , then  $k$  would not be a candidate key. We will have to check for  $\alpha$ . So, these two; what we had stated earlier in qualitative terms and now mathematically established. So, we can say that there are different functional dependencies. For example, inst\_dept combined relation if we look at, then we know that **department name** → **building** functionally.

So, these are functional dependencies that must hold, but certainly we would not expect department name to functionally determine salary. That would be too much, right. So, functional dependencies are facts about the real world that we try to understand from the real world and then, represent in terms of the functional dependency formulation in the database.

(Refer Slide Time: 26:41)

The slide has a header 'Use of Functional Dependencies' with a sailboat icon. The main content is a bulleted list under the heading 'We use functional dependencies to:'.

- We use functional dependencies to:
  - test relations to see if they are legal under a given set of functional dependencies.
    - If a relation  $r$  is legal under a set  $F$  of functional dependencies, we say that  $r$  **satisfies**  $F$
  - specify constraints on the set of legal relations
    - We say that  $F$  **holds on**  $R$  if all legal relations on  $R$  satisfy the set of functional dependencies  $F$
- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances
  - For example, a specific instance of *instructor* may, by chance, satisfy  $name \rightarrow ID$

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Date, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

16.23

©Silberschatz, Korth and Sudarshan

So, we can use functional dependencies to test relations if they are valid under the set of functional dependencies. So, there could be multiple functional dependencies in the set and if a relation we are using  $r$  here just to remind you that a relation means that a

particular instance is legal under a set of functional dependencies. We will say that  $r$  satisfies that and if we have that it holds  $F$  will be satisfied by all possible instances of a relational schema  $R$ , then we say  $F$  holds on  $R$ .

So, a relation satisfies a functional set of functional dependencies and a relational schema for a relational schema, the functional dependency set of functional dependencies holds on that schema which means that for all possible past, present and future instances relations, the relations will satisfy the functional dependencies. So, we have for example  $\text{id}$ . We know  $\text{id} \rightarrow \text{name}$  that if the id is distinct, then the name has to be distinct, but we may find that instance where  $\text{name} \rightarrow \text{id}$ . So, we can say that  $\text{name} \rightarrow \text{id}$  is satisfied by a particular instance where it so happens that there is no two rows where the name is identical, but we cannot, may not be able to infer that as this dependency holding on the relational scheme as a whole because tomorrow we can get another entry, so that two rows might match on the name, but could still be distinct entries not matching on  $\text{id}$ .

(Refer Slide Time: 28:46)

The slide features a title 'Functional Dependencies (Cont.)' in red at the top right. On the left, there is a small logo of a sailboat on water. The main content area contains a bulleted list of points about trivial functional dependencies:

- A functional dependency is **trivial** if it is satisfied by all instances of a relation
  - Example:
    - $ID, name \rightarrow ID$
    - $name \rightarrow name$
  - In general,  $\alpha \rightarrow \beta$  is trivial if  $\beta \subseteq \alpha$

At the bottom of the slide, there is footer text: 'SWAYAM: NPTEL-NOCO Instructor: Prof. P. Das, IIT Kharagpur - Jam-Apr- 2018', 'Database System Concepts - 6<sup>th</sup> Edition', '10.24', and '©Silberschatz, Korth and Sudarshan'.

So, that is how this will have to be looked at in specificity. We say that a functional dependency is trivial if the left hand side is a superset of the right hand side. So, if I have a bigger set of attributes on the left hand side  $ID$  and  $name$ , then obviously  $\text{ID, name} \rightarrow \text{ID}$ ,  $\text{ID, name} \rightarrow \text{name}$ ,  $\text{name} \rightarrow \text{name}$ . So, if you just think about because in a functional dependency the left hand side attributes that tuples have to match on the left hand side attribute and if they do, then they must match on the right hand side attribute.

So, if the right hand side set of attributes is a subset of the left hand side, then obviously the functional dependency will be vacuously true and these are called trivial dependencies.

(Refer Slide Time: 29:33)

## Functional Dependencies (Cont.)

**PPD**

- Functional dependencies are:

| StudentID | Semester | Lecture           | TA    |
|-----------|----------|-------------------|-------|
| 1234      | 6        | Numerical Methods | John  |
| 1221      | 4        | Numerical Methods | Smith |
| 1234      | 6        | Visual Computing  | Bob   |
| 1201      | 2        | Numerical Methods | Peter |
| 1201      | 2        | Physics II        | Simon |

- $\text{StudentID} \rightarrow \text{Semester}$
- $\{\text{StudentID}, \text{Lecture}\} \rightarrow \text{TA}$
- $\{\text{StudentID}, \text{Lecture}\} \rightarrow \{\text{TA}, \text{Semester}\}$

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018  
 Database System Concepts - 8<sup>th</sup> Edition      16.25      ©Silberschatz, Korth and Sudarshan

So, in the next couple of slides, I have shown few examples of functional dependencies of different tables. Here **StudentID → Semester** which mean that we are trying to model that a student cannot be at the same time in two semesters, then **(Student ID, lecture) → TA** and so on and you can see for this particular relation **(Student ID, lecture)** also happens to be the candidate key.

(Refer Slide Time: 30:05)

The slide title is "Functional Dependencies (Cont.)". It features a small sailboat icon in the top left corner and the letters "PPD" in the top right corner. The slide content includes a table of employee data:

| Employee ID | Employee Name | Department ID | Department Name |
|-------------|---------------|---------------|-----------------|
| 0001        | John Doe      | 1             | Human Resources |
| 0002        | Jane Doe      | 2             | Marketing       |
| 0003        | John Smith    | 1             | Human Resources |
| 0004        | Jane Goodall  | 3             | Sales           |

Below the table, three functional dependencies are listed:

- $\text{Employee ID} \rightarrow \text{Employee Name}$
- $\text{Employee ID} \rightarrow \text{Department ID}$
- $\text{Department ID} \rightarrow \text{Department Name}$

On the left side of the slide, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr., 2018". At the bottom, it says "Database System Concepts - 8<sup>th</sup> Edition". On the right, it shows slide number 16.26 and copyright information: "©Silberschatz, Korth and Sudarshan".

These are another example. So, these are just go through them, try to convince yourself that these functional dependencies are very genuinely real world situations that can be modeled in this way.

(Refer Slide Time: 30:19)

The slide title is "Closure of a Set of Functional Dependencies". It features a small sailboat icon in the top left corner. The slide content is a list of points:

- Given a set  $F$  of functional dependencies, there are certain other functional dependencies that are logically implied by  $F$ 
  - For example: If  $A \rightarrow B$  and  $B \rightarrow C$ , then we can infer that  $A \rightarrow C$
- The set of all functional dependencies logically implied by  $F$  is the **closure** of  $F$
- We denote the closure of  $F$  by  $F^+$
- $F^+$  is a superset of  $F$ 
  - $F = \{A \rightarrow B, B \rightarrow C\}$
  - $F^+ = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$

On the left side of the slide, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr., 2018". At the bottom, it says "Database System Concepts - 8<sup>th</sup> Edition". On the right, it shows slide number 16.27 and copyright information: "©Silberschatz, Korth and Sudarshan".

Given a set of functional dependencies, we can actually compute a closure. For example, if  $A \rightarrow B$  and  $B \rightarrow C$ , then we can infer that  $A$  functionally determined because if two peoples match on,  $A \rightarrow B$  says that they match on B. Now, if  $B \rightarrow C$  also holds, then if they match on B, they match on C. So, if the match on A, then necessarily they may have

to match on C. So, this is called the logical implication of a set of functional dependencies and we will see more of this later, but if we take all functional dependencies of a given set F, that are logically implied from this set F. We said that is a closure set and we represent that by  $F^+$ .

So,  $F^+$  necessarily is a superset of F. So, here in that above example, this is F and this is  $F^+$ .

(Refer Slide Time: 31:00)

The slide is titled "Module Summary" in red. To the left of the title is a small image of a sailboat on water. Below the title is a bulleted list of three items:

- Identified the features of good relational design
- Familiarized with the First Normal Form
- Introduced the notion of Functional Dependencies

On the far left, there is vertical text that reads: "SWAYAM: NPTEL-NOC's MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018". At the bottom left, it says "Database System Concepts - 8th Edition". In the center bottom, it shows the time "16.28". On the right side, there is a copyright notice "©Silberschatz, Korth and Sudarshan".

So, we will continue more on the theory of functional dependencies, but let us conclude this module by summarizing that we have identified the features of good relational designs tradeoff between decomposition and lossless join properties that we need. We are familiarized with the First Normal Form and atomic domains and we have introduced the notion of functional dependencies on which we will build up more and try to get zeroing very concrete strategies for good results.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 17**  
**Relational Database Design (Contd.)**

Welcome to the Module 17 of Database Management Systems. From the last module, we are discussing Relational Database Design. So, this is second in the series of five modules which we will discuss this.

(Refer Slide Time: 00:31)

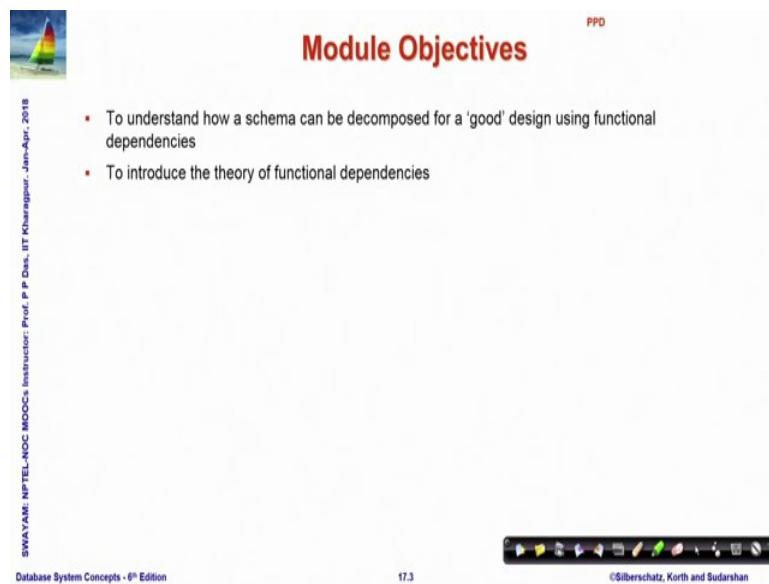
The slide has a header 'Module Recap' in red. On the left, there is a small image of a sailboat on water. On the right, there is a small logo with the letters 'PPD'. The main content is a bulleted list:

- Features of Good Relational Design
- Atomic Domains and First Normal Form
- Functional Dependencies

At the bottom, there is footer text: 'SWAYAM: NPTEL-NOCO's INOCOCC Course', 'Prof. P. P. Das, IIT Kharagpur', 'Instructor: Prof. P. P. Das, IIT Kharagpur', 'Database System Concepts - 8<sup>th</sup> Edition', '17.2', and '©Silberschatz, Korth and Sudarshan'.

We have already seen basic features of good relational design. We have studied about first normal form, atomic domains and got introduced to functional dependencies.

(Refer Slide Time: 00:43)



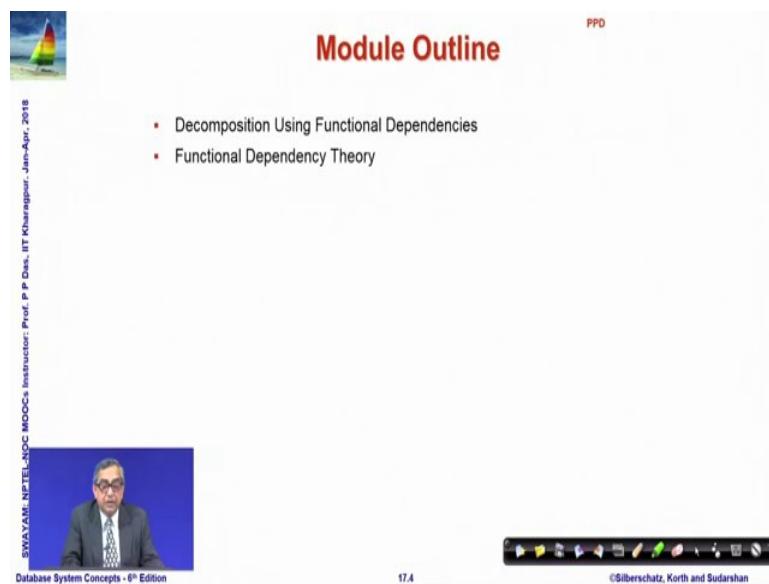
This slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. On the far left, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs", "Instructor: Prof. P. P. Deshpande, IIT Kharagpur", and "Date: Apr. 2018". At the bottom left, it says "Database System Concepts - 8<sup>th</sup> Edition". The slide contains a bulleted list of objectives:

- To understand how a schema can be decomposed for a 'good' design using functional dependencies
- To introduce the theory of functional dependencies

The bottom right corner includes the copyright notice "©Silberschatz, Korth and Sudarshan" and a set of standard presentation navigation icons.

So, we will develop further on that to see how decompositions into good design can be done by making use of the notion of functional dependencies and we will more formally introduce the theory of functional dependencies.

(Refer Slide Time: 00:57)



This slide is titled "Module Outline" in red at the top right. It features a small sailboat icon in the top left corner. On the far left, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs", "Instructor: Prof. P. P. Deshpande, IIT Kharagpur", and "Date: Apr. 2018". At the bottom left, it says "Database System Concepts - 8<sup>th</sup> Edition". The slide contains a bulleted list of topics:

- Decomposition Using Functional Dependencies
- Functional Dependency Theory

The bottom right corner includes the copyright notice "©Silberschatz, Korth and Sudarshan" and a set of standard presentation navigation icons. A video frame showing a man speaking is positioned in the lower-left area of the slide.

So, that is all that we discuss in this.

(Refer Slide Time: 01:00)

The slide features a small sailboat icon in the top left corner. In the top right, there's a list of topics under the heading 'PPD': 'Decomposition Using Functional Dependencies', 'Functional Dependency Theory', and 'Functional Dependencies'. The main title 'DECOMPOSITION USING FUNCTIONAL DEPENDENCIES' is centered in large red capital letters. Below the title is a navigation bar with icons for back, forward, search, and other presentation controls. The footer contains the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr., 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '17.5', and '©Silberschatz, Korth and Sudarshan'.

So, decomposition using functional dependencies is the first thing that we look at.

(Refer Slide Time: 01:04)

The slide has a sailboat icon in the top left. The title 'Boyce-Codd Normal Form' is centered in red capital letters. Below the title, a definition states: 'A relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies in F+ of the form  $\alpha \rightarrow \beta$ , where  $\alpha \subseteq R$  and  $\beta \subseteq R$ , at least one of the following holds:'. Two bullet points follow: '•  $\alpha \rightarrow \beta$  is trivial (i.e.,  $\beta \subseteq \alpha$ )' with a checkmark, and '•  $\alpha$  is a superkey for R' with a checkmark. To the right, there's a diagram showing two boxes labeled E1 and E2 connected by a diamond-shaped relationship arrow pointing from E1 to E2. The footer includes the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr., 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '17.6', and '©Silberschatz, Korth and Sudarshan'.

The first normal form of relations which were studied, we look at its Boyce-Codd Normal Form. So, normal forms are kind of set of properties which is satisfied by a relational schema and if they are satisfied, then we have certain guarantees in terms of what can or cannot happen in that relational schema design. So, Boyce-Codd is a simplest kind of beyond 1NF is a simplest kind of normal form and a relational schema is said to be in Boyce-Codd normal form if with respect to a set of functional

dependencies, all functional dependencies in the closure. So, in respect of F, we compute  $F^+$  which is a closure and if I have a dependency  $\alpha \rightarrow \beta$ , then naturally  $\alpha \subseteq R, \beta \subseteq R$ , but what is important is every functional dependency in the closure set must either be trivial that is right hand side  $\subseteq$  the left hand side or the left hand side set  $\alpha$  must be super key. So, only those kind of functional dependencies are possible. No other functional dependencies are possible. If that is satisfied by the relational schema R, then it is said to be in the Boyce-Codd normal form.

(Refer Slide Time: 02:39)

**Boyce-Codd Normal Form**

A relation schema  $R$  is in BCNF with respect to a set  $F$  of functional dependencies if for all functional dependencies in  $F^+$  of the form

$$\alpha \rightarrow \beta$$

where  $\alpha \subseteq R$  and  $\beta \subseteq R$ , at least one of the following holds:

- $\alpha \rightarrow \beta$  is trivial (i.e.,  $\beta \subseteq \alpha$ )
- $\alpha$  is a superkey for  $R$

Example schema *not* in BCNF:

*instr\_dept* (ID, name, salary, dept\_name, building, budget)

because dept\_name → building, budget holds on *instr\_dept*, but dept\_name is not a superkey

Navigation icons: back, forward, search, etc.

SWAYAM: NPTEL-MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr 2018

Database System Concepts - 8<sup>th</sup> Edition

17.6

©Silberschatz, Korth and Sudarshan

So, if we look at *inst\_dept* schema of the combined relations we saw last time, then we will know that certainly this is not in Boyce-Codd Normal Form because this functional dependency holds in this schema where it is neither a trivial dependency and nor department name is a super key. So, this is not in BCNF.

(Refer Slide Time: 03:10)

**Decomposing a Schema into BCNF**

- Suppose we have a schema  $R$  and a non-trivial dependency  $\alpha \rightarrow \beta$  causes a violation of BCNF
- We decompose  $R$  into:
  - $(\alpha \cup \beta)$
  - $(R - (\beta - \alpha))$
- In our example,
  - $\alpha = \text{dept\_name}$
  - $\beta = \text{building, budget}$
  - $\text{dept\_name} \rightarrow \text{building, budget}$

*inst\_dept is replaced by*

- $(\alpha \cup \beta) = (\text{dept\_name, building, budget})$  ✓  $R_1$
- $\text{dept\_name} \rightarrow \text{building, budget}$  ✓
- $(R - (\beta - \alpha)) = (\text{id, name, dept\_name})$  ✓  $R_2$
- $\text{id} \rightarrow \text{name, salary, dept\_name}$  ✓

So, if a relational scheme is not in BCNF, then the question naturally is can I make it into BCNF so then that process is the process of decomposition. So, what you do? You divide the set of attributes into two or more sets of attributes. So, here let us say that we have a relational schema which has a non-trivial dependency,  $\alpha \rightarrow \beta$ , where  $\alpha$  is not a super key. So, with respect to this functional dependency, the relational schema is not in BCNF, then we can decompose  $R$  by two sets. One is  $\alpha \cup \beta$  take the  $\cup$  of these two attribute sets and remove  $\beta - \alpha$  from  $R$ , take the difference of  $\beta - \alpha$  and remove that from  $R$ . The resulting pair of relations, relational schemas will be in Boyce-Codd Normal Form with respect to this particular functional dependency.

So, let us see an example. So, if  $\alpha$  is department name  $\beta$  is **(building, budget)**, we have department name  $\rightarrow$  building budget. So,  $\alpha \rightarrow \beta$  and we have already seen that it does not hold. It is not satisfied by the **inst\_dept**. So, you replace it by taking  $\alpha \cup \beta$ . So,  $\alpha \cup \beta$  is this set of relational, this relational schema and you do  $R - (\beta - \alpha)$ . Naturally if this is  $\beta$  and  $\alpha$ , then  $\beta - \alpha$  is necessary building budget because if the department does not occur in  $\beta$ .

So, this set is building budget and if I remove it from  $R$  which means that id, name, salary and department name are retained, but building and budget gets removed. So, I get another relational schema which has these four names and it holds the functional dependency id determinant. So, even now if I look into this schema  $R_1$  and this schema

R2, there are different dependencies that hold on R1 and with respect to that dependency R1 is in BCNF because department name is the super key, is the primary key and with respect to this dependency, R2 is in BCNF because id is a key.

So, I can see that the original combined relational schema was not in BCNF with respect to this functional dependency, but when I do this decomposition, I get two schemas which are each in BCNF normal form. So, this is the basic process and we will see depending on the normal form and different notions of functional dependencies, we will see how these conversions can be done, but this is a basic approach of converting a schema into a normal form.

(Refer Slide Time: 06:24)

**BCNF and Dependency Preservation**

- Constraints, including functional dependencies, are costly to check in practice unless they pertain to only one relation
- If it is sufficient to test only those dependencies on each individual relation of a decomposition in order to ensure that *all* functional dependencies hold, then that decomposition is *dependency preserving*.
- Because it is not always possible to achieve both BCNF and dependency preservation, we consider a weaker normal form, known as *third normal form*.

SWAYAM-NETTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
17.8 ©Silberschatz, Korth and Sudarshan

Now, the question is if the constraints including the functional dependencies if we look at, then functional dependencies will have to be checked on different instance. Now, in general it is difficult to check a functional dependency  $\alpha$  determining  $\beta$  if the attributes  $\alpha$  and the attributes of  $\beta$  or the attributes of  $\beta$  are distributed between multiple relations because naturally how do I check if they are true, how do I check that two tuples which match on  $\alpha$  is indeed matching on  $\beta$  unless I perform a costly join operation. So, the objective is to be able to come to designs where it is sufficient to test only those dependencies on individual relations of the decomposition and with that I must be able to ensure that all functional dependencies hold. So, it is a very interesting situation.

So, we are saying that we will decompose, get into a number of relational schema. Every schema will have a number of dependencies, functional dependencies and those functional dependencies if they involve only the attributes of that relational schema, they can be tested very easily and if these functional dependencies together mean ensure that all functional dependencies hold that is if the closure of this set of functional dependencies is same as the closure of the earlier set, the original set, then we say that the decomposition that we have achieved is dependency preserving because I can actually effectively compute.

This is dependency preserving because I can effectively compute whether every dependency is satisfied by checking on every individual relation, but the unfortunate part of the reality is that it is not always possible to achieve a Boyce-Codd Normal Form Decomposition which also preserves the dependencies. See if there are in some cases will be able to do like the example we saw just now the instructor and department, but it is not always possible. So, we usually need another weaker form, normal form which is known as a third normal form and we will subsequently look into those.

(Refer Slide Time: 09:09)

The slide has a decorative header featuring a sailboat on water. The title 'Third Normal Form' is centered in red. The content is organized into two columns. The left column contains a vertical footer with text: 'SWAYAM-NPTEL-NOCS Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018'. The right column contains a bulleted list of conditions for a relation to be in 3NF:

- A relation schema  $R$  is in **third normal form (3NF)** if for all:  
 $\alpha \rightarrow \beta$  in  $F^+$   
at least one of the following holds:
  - $\alpha \rightarrow \beta$  is trivial (i.e.,  $\beta \subseteq \alpha$ )
  - $\alpha$  is a superkey for  $R$
  - Each attribute  $A$  in  $\beta - \alpha$  is contained in a candidate key for  $R$   
(**NOTE:** each attribute may be in a different candidate key)
- If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must hold)
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later)

At the bottom, there is a video frame showing a man in a suit, a navigation bar with icons, and footer text: 'Database System Concepts - 6<sup>th</sup> Edition', '17.9', and '©Silberschatz, Korth and Sudarshan'.

A third normal form is again a relational schema is there and for all attribute, for all dependencies that belong to the closure of the functional dependencies, this following conditions must hold either  $\alpha \rightarrow \beta$  is trivial which is a condition which BCNF or  $\alpha$  is a super key of R which is also a condition that we say in BCNF or each attribute in  $\beta - \alpha$

that is right hand side difference the left hand side is contained in a candidate key for R. It is not very obvious as to why we need that. That will unfold slowly. This is the condition we did not have in BCNF. So, naturally you can see that based on the first two conditions, you can always say that if a relational schema is in BCNF, it necessarily is in 3NF, but not the reverse.

There could be some schema which is in 3 NF because of the third condition where there exists a functional dependency. So, that  $\beta - \alpha$  is contained in a candidate key for R, but it is not in the BCNF form and also you can note that the attributes of that are contained in  $\beta - \alpha$  must be in some candidate key, not necessarily in the same candidate key. If they exist in some candidate key, then itself 3NF condition will get satisfied. So, if a relation is in BCNF, it is in 3NF. We have already seen that. So, third condition minimally relaxes BCNF to ensure that we have a dependency preservation. We will see this more later. So, I am just introducing the concept of a relaxed normal form here.

(Refer Slide Time: 11:11)

The slide has a header 'Goals of Normalization' in red. On the left is a small sailboat icon. The main content is a bulleted list:

- Let  $R$  be a relation scheme with a set  $F$  of functional dependencies
- Decide whether a relation scheme  $R$  is in "good" form
- In the case that a relation scheme  $R$  is not in "good" form, decompose it into a set of relation schemes  $\{R_1, R_2, \dots, R_n\}$  such that
  - each relation scheme is in good form
  - the decomposition is a lossless-join decomposition
  - Preferably, the decomposition should be dependency preserving

At the bottom, there is a video frame showing a man speaking, the text 'Database System Concepts - 8<sup>th</sup> Edition', the page number '17.10', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, what is a goal of this normalization is if to summarize let  $R$  be a relational scheme,  $F$  is a set of functional dependencies, we need to decide whether the relational scheme  $R$  is in a good form which means that it should not have unnecessary redundancy. It should be impossible to acquire information by doing lossless join. So, in case it is not in good form. We can convert it by decomposition into  $N$  relational schema, such that each schema is in good form. The decomposition has a lossless join, so that I can get back the

original relation from this and preferably the decomposition should preserve the dependencies. So, that is what we will target henceforth.

(Refer Slide Time: 12:08)

The slide has a title 'How good is BCNF?' in red at the top right. To the left is a small sailboat icon. Below the title is a bulleted list:

- There are database schemas in BCNF that do not seem to be sufficiently normalized
- Consider a relation

*inst\_info (ID, child\_name, phone)*

- where an instructor may have more than one phone and can have multiple children

Below the list is a table with three columns: ID, child\_name, and phone. The data is as follows:

| ID    | child_name | phone          |
|-------|------------|----------------|
| 99999 | David      | 512-555-1234 ✓ |
| 99999 | David      | 512-555-4321 ✗ |
| 99999 | William    | 512-555-1234 ✓ |
| 99999 | Willian    | 512-555-4321 ✗ |

inst\_info

At the bottom left is a video player interface showing a person speaking. The bottom right shows the slide number 17.11 and the copyright notice ©Silberschatz, Korth and Sudarshan.

So, when we do that let us quickly evaluate as to we have seen BCNF. So, how good really BCNF is. So, if I have something in BCNF should I really be very happy always. So, let us look at a relational schema. This is an information relating the idea of a person, the name of the child and the phone number and naturally the person, the instructor may have more than one phone and may have multiple children. So, this is a possible instance that you can see though all of these belong to the same instructor. He has naturally you can see that two children and there are this is here, this is here.

So, this is here and this is here. So, there are two different phone numbers. So, naturally you have four possible combinations that you need to look at.

(Refer Slide Time: 13:08)

The slide has a header 'How good is BCNF? (Cont.)' in red. On the left is a small sailboat icon. The right side contains a video player window showing a man in a suit speaking. Below the video player are some navigation icons. The footer includes text: 'SYAYAM-NETTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '17.12', and '©Silberschatz, Korth and Sudarshan'.

- There are no non-trivial functional dependencies and therefore the relation is in BCNF
- Insertion anomalies – i.e., if we add a phone 981-992-3443 to 99999, we need to add two tuples
  - (99999, David, 981-992-3443)
  - (99999, William, 981-992-3443)

So, now there is no non-trivial functional dependency in this relation. So, since there is no non-trivial functional dependency, this relation naturally is in BCNF form because that is the existence of non trivial dependency is what makes a schema not conform to the BCNF form. So, there is no such. So, this is in BCNF form and now, if you look at, but what did we see the key thing that we saw if we just go back, the key thing that we saw that there is ample redundancy of data, the same data is entered multiple times.

So, the consequence of that could be insertion anomaly. If we want to add a phone number to the same instructor, then we need to add two tuple because the instructor also has two children. If the instructor and three children will need to add three and unless this is maintained always, then we will have difficulty.

(Refer Slide Time: 14:15)

The slide features a small sailboat icon in the top-left corner. The title 'How good is BCNF? (Cont.)' is centered at the top in red. On the left, there is vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018'. Below the title, a bulleted list says: 'Therefore, it is better to decompose *inst\_info* into:'. Two tables are shown side-by-side. The first table, 'inst\_child', has columns 'ID' and 'child\_name', with data: ID 99999, child\_name David; ID 99999, child\_name David; ID 99999, child\_name William; ID 99999, child\_name Willian. The second table, 'inst\_phone', has columns 'ID' and 'phone', with data: ID 99999, phone 512-555-1234; ID 99999, phone 512-555-4321; ID 99999, phone 512-555-1234; ID 99999, phone 512-555-4321. A note below the tables states: 'This suggests the need for higher normal forms, such as Fourth Normal Form (4NF)'. At the bottom, there is a navigation bar with icons for back, forward, search, etc., and the text 'Database System Concepts - 8<sup>th</sup> Edition', '17.13', and '©Silberschatz, Korth and Sudarshan'.

So, the redundancy consequences anomaly that we are getting into, so it could be better to decompose this to say that I make this orthogonal; I keep the child information with id and I keep the phone number information in the id separately. So, if I do that, then I can decompose it in this manner and if I decompose that, this have just shown that if you are dividing that table in two parts. So, naturally these are not required, neither are these required. So, these are the entries that I get and you can convince yourself that you can actually do a lossless join to get back the information.

So, BCNF not necessarily give you good designs and we will see later on that there are other normal forms which can be used to improve on BCNF.

(Refer Slide Time: 15:05)

The slide has a header 'PPD' and a sidebar with the text: '•Decomposition Using Functional Dependencies •Functional Dependency Theory'. The main title 'FUNCTIONAL DEPENDENCY THEORY' is in large red capital letters. The footer includes 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '17.14', and '©Silberschatz, Korth and Sudarshan'.

Now, let us formally get into how do we convert decomposed relation into a third normal form and how we assess that we need to understand more of the functional dependencies.

(Refer Slide Time: 15:20)

The slide has a header 'Functional-Dependency Theory' and a sidebar with the text: '•We now consider the formal theory that tells us which functional dependencies are implied logically by a given set of functional dependencies •We then develop algorithms to generate lossless decompositions into BCNF and 3NF •We then develop algorithms to test if a decomposition is dependency-preserving'. The footer includes 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '17.15', and '©Silberschatz, Korth and Sudarshan'.

So, we will consider now a little bit of formal theory on them and then, develop algorithms that can generate lossless join decomposition into BCNF and 3 NF and we will also create algorithm to test if decomposition preserves the dependency.

(Refer Slide Time: 15:39)

Closure of a Set of Functional Dependencies

- Given a set  $F$  of functional dependencies, there are certain other functional dependencies that are logically implied by  $F$ 
  - For e.g.: If  $A \rightarrow B$  and  $B \rightarrow C$ , then we can infer that  $A \rightarrow C$
- The set of all functional dependencies logically implied by  $F$  is the closure of  $F$
- We denote the closure of  $F$  by  $F^*$

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr., 2018  
Database System Concepts - 8<sup>th</sup> Edition  
17.16 ©Silberschatz, Korth and Sudarshan

So, just quickly to recap we have already introduced the closure set of a functional dependencies. It is all dependencies that are logically implied by it. Now, the question certainly is given a set how do I compute this closure of a set?

(Refer Slide Time: 15:57)

Closure of a Set of Functional Dependencies

- We can find  $F^*$ , the closure of  $F$ , by repeatedly applying **Armstrong's Axioms**:
  - if  $\beta \sqsubseteq \alpha$ , then  $\alpha \rightarrow \beta$  (reflexivity)
  - if  $\alpha \rightarrow \beta$ , then  $\gamma \alpha \rightarrow \gamma \beta$  (augmentation)
  - if  $\alpha \rightarrow \beta$ , and  $\beta \rightarrow \gamma$ , then  $\alpha \rightarrow \gamma$  (transitivity)
- These rules are
  - sound (generate only functional dependencies that actually hold), and
  - complete (generate all functional dependencies that hold)

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr., 2018  
Database System Concepts - 8<sup>th</sup> Edition  
17.17 ©Silberschatz, Korth and Sudarshan

So, to do this we make use of three rules known by Armstrong's Axiom named after the person who first observed them. So, the first rule is reflexivity which says that if  $\beta \sqsubseteq \alpha$ , then  $\alpha \rightarrow \beta$ . Always  $\alpha \rightarrow \beta$ . So, this is basically reflexivity you can see is a different

way of saying specifying about trivial dependencies. Next comes important thing augmentation which says that if  $\alpha \rightarrow \beta$ , then  $\gamma \alpha$  where  $\gamma$  is some set of attributes in R.

Then,  $\gamma \alpha \rightarrow \gamma \beta$  which is very easy to see because  $\alpha \rightarrow \beta$  means two tuples who match on  $\alpha$  will necessarily match on  $\beta$ . Now, if that happens and whatever is  $\gamma$  if two tuples match on  $\gamma$  and  $\alpha$ , then certainly they will match and  $\gamma$  and  $\beta$  because  $\alpha \rightarrow \beta$  tells me that they will match on  $\beta$  and  $\gamma$  is the same set of attributes. So, augmentation also is easy. Then, we have transitivity which we earlier saw also if  $\alpha \rightarrow \beta$  and  $\beta \rightarrow \gamma$ , then obviously  $\alpha \rightarrow \gamma$ .

So, these are the foundational rules observed which can be made used to compute the closure of the set of functional dependencies. Now, these rules as they say this is more for you know understanding the theory better. These rules are sound as well as complete. Soundness mean that if I use these rules repeatedly in a set of dependencies, then it generates functional dependencies all of which actually hold. So, it will never generate a functional dependency which is not correct, which will not hold and the second it is complete which means that if I keep on using these rules, then all functional dependencies that can at all hold will eventually get generated.

(Refer Slide Time: 18:06)


**Example**

- $R = (A, B, C, G, H, I)$
- $F = \{ A \rightarrow B$   
 $A \rightarrow C \checkmark$   
 $CG \rightarrow H$   
 $CG \rightarrow I \checkmark$   
 $B \rightarrow H \}$
- Some members of  $F^*$ 
  - $A \rightarrow H$ 
    - by transitivity from  $A \rightarrow B$  and  $B \rightarrow H$
  - $AG \rightarrow I \checkmark$ 
    - by augmenting  $A \rightarrow C$  with G, to get  $AG \rightarrow CG$  and then transitivity with  $CG \rightarrow I$
  - $CG \rightarrow HI$ 
    - by augmenting  $CG \rightarrow I$  to infer  $CG \rightarrow CGI$ , and augmenting of  $CG \rightarrow H$  to infer  $CGI \rightarrow HI$ , and then transitivity

SWAYAM: NPTEL-NOCO MOOCs Instructor: Prof. P. P. Desai, Jai Kharasgari - Jan-Apr. 2018
Database System Concepts - 6<sup>th</sup> Edition
17.18
©Silberschatz, Korth and Sudarshan



So, that is a very strong result and that is what leads to say the following example. So, when we are trying to compute the functional, the closure of the function set of functional dependencies here. So, there are six attributes in the set. There are six

different functional dependencies and we identify some members of the closure. For example, we can see that  $A \rightarrow B$  and  $B \rightarrow H$ . So, transitivity clearly shows that  $A \rightarrow H$ , very clear. So, in the closure that must be there.

Similarly, we can see that  $A \rightarrow C$ . Now, if we augment it with G, that is put G on both sides, then  $AG \rightarrow CG$  and we know that  $CG \rightarrow I$ . So, if we combine these two by transitivity, then we can get a new functional dependency which has  $AG \rightarrow I$ . So, in this manner you can do the next one also and you can try to infer several other functional dependencies that can be inferred by different applications of the Armstrong's axiom, the three rules in any multiple different ways.

(Refer Slide Time: 19:36)

**Procedure for Computing  $F^+$**

- To compute the closure of a set of functional dependencies  $F$ :

```

 $F^+ = F$ 
repeat
    for each functional dependency  $f$  in  $F^+$ 
        apply reflexivity and augmentation rules on  $f$ 
        add the resulting functional dependencies to  $F^+$ 
    for each pair of functional dependencies  $f_1$  and  $f_2$  in  $F^+$ 
        if  $f_1$  and  $f_2$  can be combined using transitivity
            then add the resulting functional dependency to  $F^+$ 
    until  $F^+$  does not change any further

```

**NOTE:** We shall see an alternative procedure for this task later

SWAYAM-NPTEL-NOOC Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition      17.19      ©Silberschatz, Korth and Sudarshan

So, to get the closure what we need to do is, now very simple is certainly we will have a repetitive algorithm to get the closure. The first algorithm will start with the set of functional dependencies that we have. So, the closure must include the given set of functional dependencies. So,  $F^+$  must have  $F$ . So, let us start with initial value of  $F^+$  as  $F$ , then for every functional dependency is  $F^+$ . This is what we keep on repeating. Look at the outer loop, every functional dependency that we have  $F^+$  now will apply reflexivity and augmentation and add the resulting functional dependency in  $F^+$ . It is possible that the same functional dependency gets generated and added multiple times does not matter.  $F^+$  is a set. It will naturally eliminate duplicates.

Then, for each pair of functional dependencies because reflexivity and augmentation applies to one functional dependency only, but transitivity applies to two functional dependencies. So, for every pair of functional dependencies, we check whether they can be combined by transitivity. If they do, then the transitive closure of the transitive functional dependency that arise out of that is also added to  $F^+$  and mind you more and more functional dependencies you add, there are more and more opportunities to apply the Armstrong's Axiom rules and newer functional dependencies will continue to get added, but eventually you reach a point where  $F^+$  does not change any further and when that is achieved, we know that the functional, the closure of the functional dependencies have been obtained and that is our final set.

(Refer Slide Time: 21:32)

**Closure of Functional Dependencies (Cont.)**

- Additional rules:
  - If  $\alpha \rightarrow \beta$  holds and  $\alpha \rightarrow \gamma$  holds, then  $\alpha \rightarrow \beta\gamma$  holds (**union**)
  - If  $\alpha \rightarrow \beta\gamma$  holds, then  $\alpha \rightarrow \beta$  holds and  $\alpha \rightarrow \gamma$  holds (**decomposition**)
  - If  $\alpha \rightarrow \beta$  holds and  $\gamma \beta \rightarrow \delta$  holds, then  $\alpha\gamma \rightarrow \delta$  holds (**pseudotransitivity**)

The above rules can be inferred from Armstrong's axioms

SWAYAM-NPTEL-NOOC Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts - 6<sup>th</sup> Edition  
17.20 ©Silberschatz, Korth and Sudarshan

We can also observe that based on the rules of Armstrong, the Armstrong's Axioms we can also generate lot of derived rules. Some of those are shown here. For example, if  $A \rightarrow$ , if  $\alpha \rightarrow \beta$  holds and if  $\alpha \rightarrow \gamma$ , that also holds, then  $\alpha \rightarrow \beta$  and  $\gamma$  together. This is called the **union** set. So, if there are two functional dependencies which are the same left hand side set of attribute, then we can take the **union** of their right hand side attributes and that functional dependency will hold obviously, it is trivial to prove this.

If  $\alpha \rightarrow \beta\gamma$ , then  $\alpha \rightarrow \beta$  holds and  $\alpha \rightarrow \gamma$  holds. This is called decomposition. So, kind of the other side of the **U** which also is trivial because  $\alpha \rightarrow \beta\gamma$  says if two tuples match on  $\alpha$ , they match on  $\beta$  as well as  $\gamma$  attributes. So, obviously you take the first part, you get  $\alpha$

→  $\beta$ . You take the second part of the observation, you get  $\alpha \rightarrow \gamma$ . So, that is a composition rule. The third is interesting. It is called the pseudo transitivity which says that  $\alpha \rightarrow \beta$  if that holds and  $\gamma \beta \rightarrow \delta$  if that holds, then  $\alpha \gamma \rightarrow \delta$  which is not difficult to get because if this holds, then I can augment  $\gamma$  on both sides. I get  $\beta \gamma$  and then, I have given  $\beta \gamma \rightarrow \delta$ . So, if I combine these two in terms of transitivity, I get  $\alpha \gamma \rightarrow \delta$ .

So, this is called pseudo transitivity because here you are adding another attribute in the transitivity. So, often times it becomes easier to make use of these additional rules to quickly get to the closer set.

(Refer Slide Time: 23:45)

**Closure of Attribute Sets**

- Given a set of attributes  $\alpha$ , define the **closure** of  $\alpha$  under  $F$  (denoted by  $\alpha^+$ ) as the set of attributes that are functionally determined by  $\alpha$  under  $F$
- Algorithm to compute  $\alpha^+$ , the closure of  $\alpha$  under  $F$

```

result := α; ✓
while (changes to result) do
  for each β → γ in F do
    begin
      if β ⊆ result then result := result ∪ γ
    end
  
```

The hand-drawn diagram shows a Venn diagram with three overlapping circles labeled  $\alpha$ ,  $\beta$ , and  $\gamma$ . Arrows point from  $\alpha$  to  $\beta$  and from  $\beta$  to  $\gamma$ . A large arrow points from the union of  $\alpha$  and  $\beta$  to the intersection of  $\alpha$  and  $\beta$ , representing the step where new attributes are added to the result set.

So, given a set of attributes we also compute the closure of a set of attributes. This is a second concept we have seen how to give the set of functional dependencies, how to compute the closure of the functional dependencies. Now, we are given a set of attributes and we want to define the closure of this set of functional, this set of attributes under the set of functional dependencies and as the closure of functional dependencies  $F$  is denoted by  $F^+$ , the closure of a set of attributes  $\alpha$  under  $F$  is denoted by  $\alpha^+$ . So, this set of closure attributes of  $\alpha$  is a set of attributes that are functionally determined by  $\alpha$  under  $F$ . So, all set of attributes that functionally determined by  $\alpha$  under the set of functional dependencies is member of  $\alpha^+$ .

So, the following simple algorithm can compute the closure naturally. Initially let us say the result is the final closure set. So, initially we can say that result can be initialized with

$\alpha$  because certainly the whole of  $\alpha$  would necessarily belong to  $\alpha^+$  by the reflexivity condition, then for each functional dependency  $\beta \rightarrow \gamma$ , we check if  $\beta$  is a subset the result. If  $\beta$  is a subset the current set of attributes that form result which mean that  $\alpha \rightarrow \beta$ , it will have to because result is the set of all attributes that  $\alpha$  functionally determines. So, if  $\beta$  is a subset the result, then necessarily  $\alpha \rightarrow \beta$  is a consequence of this and we know that this is their  $\beta \rightarrow \gamma$  combined by transitivity.

So, I know  $\alpha \rightarrow \gamma$ . If function  $\alpha \rightarrow \gamma$ , then it must get into the result and this is exactly what the statement is saying that take result and add  $\alpha$ , add  $\gamma$ , the set of attributes  $\gamma$  to the result. How long should you do that? Naturally you will do that as long as over a full iteration of functional dependencies in F, if there is no change to the result, then you know that all future iterations will have no change. So, you reach a fixed point and you declare that the closure of the set of attributes have been obtained.

(Refer Slide Time: 26:44)

**Example of Attribute Set Closure**

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
- $(AG)^*$

1.  $result = AG$
2.  $result = ABCG$  ( $A \rightarrow C$  and  $A \rightarrow B$ )
3.  $result = ABCGH$  ( $CG \rightarrow H$  and  $CG \subseteq AGBC$ )
4.  $result = ABCGHI$  ( $CG \rightarrow I$  and  $CG \subseteq AGBCH$ )

$AG \rightarrow ABCGHI$

SWAYAM NPTEL MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jam-Apr- 2018  
 Database System Concepts - 8<sup>th</sup> Edition

17.22 ©Silberschatz, Korth and Sudarshan

Now, this closure information is very interesting and we just show an example here based on the same set of attributes and same set of functional dependencies.

So, we are trying to find the closure of the set of attributes AG. So,  $AG^+$  initially it will be AG. Now, since  $A \rightarrow C$ , so given that I can say that C will get included in this set in the same iteration. If I look at  $A \rightarrow B$ , so B will get included in this set. So, after this first iterative loop I will have the result as ABCG. If ABCG is there and I am looking at the next iteration, then  $CG \rightarrow H$ . So, H comes into the set because CG is a subset that I

comes into the set because  $CG \rightarrow I$  and at this point, it eventually ends in this case. In this particular example, you can see that all attributes have got included. So, you can see that it immediately gives you another information as a byproduct of the closure that closure of AG is all attributes which mean that AG is a key. It has to be a key because  **$AG \rightarrow ABCGHI$** .

So, what is the meaning of  $AG^+$  being this? So, if the meaning of this is  **$AG \rightarrow ABCGHI$** , right, so we will see that this closure set has a lot of valuable information in this.

(Refer Slide Time: 28:31)

**Example of Attribute Set Closure**

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B$   
 $A \rightarrow C$   
 $CG \rightarrow H$   
 $CG \rightarrow I$   
 $B \rightarrow H\}$
- $(AG)^*$ 
  1.  $result = AG$
  2.  $result = ABCG$  ( $A \rightarrow C$  and  $A \rightarrow B$ )
  3.  $result = ABCGH$  ( $CG \rightarrow H$  and  $CG \subseteq AGB$ )
  4.  $result = ABCGHI$  ( $CG \rightarrow I$  and  $CG \subseteq AGBCH$ )
- Is AG a candidate key?
  1. Is AG a super key?
    1. Does  $AG \rightarrow R? == ls(AG)^* \sqsupseteq R$
    2. Is any subset of AG a superkey?
      1. Does  $A \rightarrow R? == ls(A)^* \sqsupseteq R$
      2. Does  $G \rightarrow R? == ls(G)^* \sqsupseteq R$

So, we can say that AG is a candidate key and because of this, we can also check whether AG is a super key or not. All that we need to do is drop some member from AG, we drop G and check whether  $A \rightarrow R$  which means we check whether  $A^+ == R$  or not. We check we drop A from AG and check whether  $G \rightarrow R$  which means  $G^+ == R$  and by that we can easily determine whether the set of attributes is a key or not.

(Refer Slide Time: 29:14)

The slide has a header 'Uses of Attribute Closure' with a sailboat icon. The text discusses several uses of the attribute closure algorithm, including testing for superkeys, testing functional dependencies, and computing closure of F. It also mentions that attribute closure can be used to find closures of subsets of attributes.

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P.P. Desai, IIT Kanpur Date: June-Apr., 2018

There are several uses of the attribute closure algorithm:

- Testing for superkey:
  - To test if  $\alpha$  is a superkey, we compute  $\alpha^+$  and check if  $\alpha^+$  contains all attributes of  $R$ .
- Testing functional dependencies
  - To check if a functional dependency  $\alpha \rightarrow \beta$  holds (or, in other words, is in  $F^*$ ), just check if  $\beta \subseteq \alpha^+$ .
  - That is, we compute  $\alpha^+$  by using attribute closure, and then check if it contains  $\beta$ .
  - Is a simple and cheap test, and very useful
- Computing closure of  $F$ 
  - For each  $\gamma \subseteq R$ , we find the closure  $\gamma^+$ , and for each  $S \subseteq \gamma^+$ , we output a functional dependency  $\gamma \rightarrow S$ .

Database System Concepts - 8<sup>th</sup> Edition 17.23 ©Silberschatz, Korth and Sudarshan

So, there are several ways. The attribute closure can be used as we have just seen. It helps you determine whether something is a super key. We can check for testing functional dependencies because if we have to check whether a functional dependency  $\alpha \rightarrow \beta$  hold, all that we will have to do is to compute the closure of the set of attributes  $\alpha$  that is  $\alpha^+$  and check whether  $\beta$  is a subset that. If it is, then certainly holds. If it is not, then it does not hold. So, it is simple and useful test that can be made use of.

So, it can also be used in computing the closure of  $F$  that for example, for every subset  $\gamma$  of  $R$ , if we find  $\gamma^+$  that is a closure of the set of attributes of  $\gamma$  and then, for each subset  $\gamma^+$ , we know that there is a functional dependency  $\gamma \rightarrow S$  which is just is the same statement being made in you know or in different forms and the closure of attributes is a very nice concept which help you play around in this multiple ways and we will see subsequently many of the algorithms for normalization.

(Refer Slide Time: 30:35)

**Module Summary**

- Discussed issues in 'good' design in the context of functional dependencies
- Introduced the theory of functional dependencies

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

17.24

©Silberschatz, Korth and Sudarshan

How they make effective use of this closure set, notion of both closure of functional dependencies and in very practical implementation algorithms, the closure of attributes. So, to summarize this module, we have discussed issues further issues in the good design in the context of functional dependencies and in the process, we have also extended the theory of functional dependencies and we will continue it this in the next module to get more insight into the algorithms that actually work with the functional dependencies.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 18**  
**Relational Database Design (Contd.)**

Welcome to module 18 of Database Management Systems. We have been discussing about relational database design. This is a part 3 of that.

(Refer Slide Time: 00:30)

**Module Recap**

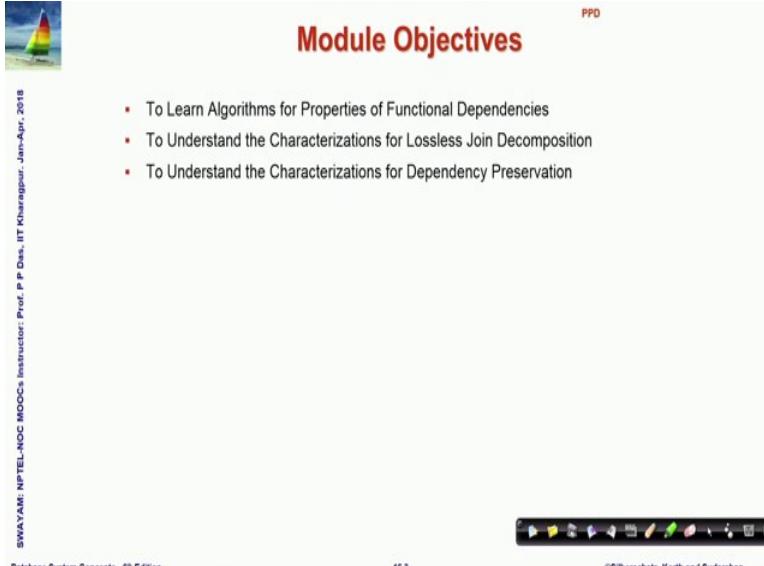
- Decomposition Using Functional Dependencies
- Functional Dependency Theory

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition      16.2      ©Silberschatz, Korth and Sudarshan

In the last module, we discussed about the Notion of functional dependency and decomposition based on that in an elementary level and certain bit of its theory.

(Refer Slide Time: 00:39)



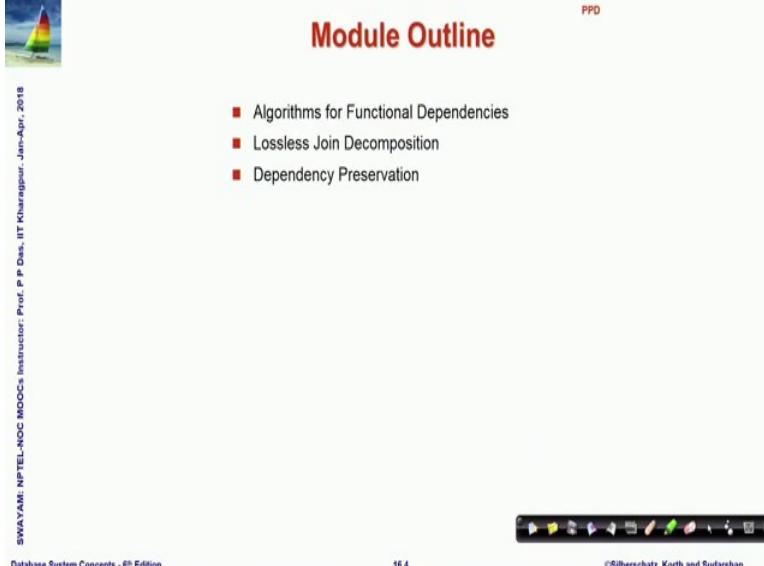
This slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. On the far left, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs", "Instructor: Prof. P. P. Deshpande, IIT Kharagpur", and "2018". At the bottom left is the text "Database System Concepts - 8<sup>th</sup> Edition". The center contains a bulleted list of objectives:

- To Learn Algorithms for Properties of Functional Dependencies
- To Understand the Characterizations for Lossless Join Decomposition
- To Understand the Characterizations for Dependency Preservation

The bottom right shows the copyright notice "©Silberschatz, Korth and Sudarshan". A navigation bar with various icons is at the very bottom.

In this current module, we learnt different algorithms that use the functional dependencies and can make conclusions about the design or make changes to the design. We will also try to understand the characterization for lossless, join decomposition and the notion of dependency preservation.

(Refer Slide Time: 01:06)



This slide is titled "Module Outline" in red at the top right. It features a small sailboat icon in the top left corner. On the far left, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs", "Instructor: Prof. P. P. Deshpande, IIT Kharagpur", and "2018". At the bottom left is the text "Database System Concepts - 8<sup>th</sup> Edition". The center contains a bulleted list of topics:

- Algorithms for Functional Dependencies
- Lossless Join Decomposition
- Dependency Preservation

The bottom right shows the copyright notice "©Silberschatz, Korth and Sudarshan". A navigation bar with various icons is at the very bottom.

Therefore, this module will have these three topics algorithms for functional dependencies, lossless join decomposition and dependency preservation.

(Refer Slide Time: 01:15)

The slide has a header 'Example of Attribute Set Closure' with a sailboat icon. On the left, there's a vertical sidebar with text: 'SWAYAM-NPTEL-NOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr. 2018'. Below this is a video frame showing a man with glasses and a blue shirt. The main content area contains the following text:

- $R = \{A, B, C, G, H, I\}$
- $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
- $(AG)^*$ 
  1.  $result = AG$
  2.  $result = ABCG$  ( $A \rightarrow C$  and  $A \rightarrow B$ )
  3.  $result = ABCGH$  ( $CG \rightarrow H$  and  $CG \subseteq AGBC$ )
  4.  $result = ABCGHI$  ( $CG \rightarrow I$  and  $CG \subseteq AGBCH$ )
- Is AG a candidate key?
  1. Is AG a super key?
    1. Does  $AG \rightarrow R? == Is(AG)^* \supseteq R$
    2. Is any subset of AG a superkey?
      1. Does  $A \rightarrow R? == Is(A)^* \supseteq R$
      2. Does  $G \rightarrow R? == Is(G)^* \supseteq R$

At the bottom right are navigation icons and the text '©Silberschatz, Korth and Sudarshan'.

So, first we start with the algorithms and I quickly reproduce what we had ended in the last module in terms of computing the closure of a set of attributes. So, if we have a relation having these attributes and a set of functional dependencies, then for a given subset of attributes, in this case AG we can iteratively compute the closure set when no further changes can be done, and with using that we can make different conclusions.

For example, if our question is whether AG can be a candidate key, we would first like to check whether it is a super key that is whether its closure has all the attributes of R and we would like to check if we have taken a subset of AG. If we take just as a attribute A or attribute G whether the closure of that will actually work as a key or not.

(Refer Slide Time: 02:12)

The slide has a header 'Uses of Attribute Closure' with a sailboat icon. The text discusses several uses of the attribute closure algorithm, including testing for superkeys, functional dependencies, and computing closures. It also includes a video player showing a speaker and navigation icons.

There are several uses of the attribute closure algorithm:

- Testing for superkey:
  - To test if  $\alpha$  is a superkey, we compute  $\alpha^+$  and check if  $\alpha^+$  contains all attributes of  $R$ .
- Testing functional dependencies
  - To check if a functional dependency  $\alpha \rightarrow \beta$  holds (or, in other words, is in  $F^*$ ), just check if  $\beta \subseteq \alpha^+$ .
  - That is, we compute  $\alpha^+$  by using attribute closure, and then check if it contains  $\beta$ .
  - Is a simple and cheap test, and very useful
- Computing closure of  $F$ 
  - For each  $\gamma \subseteq R$ , we find the closure  $\gamma^+$ , and for each  $S \subseteq \gamma^+$ , we output a functional dependency  $\gamma \rightarrow S$ .

SWAYAM/NIIT-NOOCs Instructor: Prof. P.P. Desai, IIT Kharagpur Date: Jan-Apr. 2018

16.7 ©Silberschatz, Korth and Sudarshan

So, this algorithm of attribute closure turns out to be a very powerful one, where as we have just seen it can be used for checking super keys, the candidate keys, primary, non-primary attributes and so on. It can be used for checking functional dependencies. For example, let us suppose that if we have to check that whether if a particular functional dependency  $\alpha \rightarrow \beta$  holds, then rather in other words whether  $\alpha \rightarrow \beta$  is in the closure of the set of functional dependencies  $F$ , then all that we need to do is to compute  $\alpha^+$  that is a closure of the set of attributes on the left hand side of the dependency and check if  $\beta$  is a subset of that. If  $\beta$  is a subset of that, then I know that  $\alpha \rightarrow \beta$  actually holds.

So, in this manner it can also be used to compute the closure of the whole set of functional dependencies  $F$ . So, if I mean at least at A, rudimentary level we can think of that. If we take any subset of the set of attributes and find the closure and then, all attributes that belong to that closure set are actually functionally dependent and therefore, those functional dependencies will exist.

(Refer Slide Time: 03:42)

The slide has a title 'Canonical Cover' in red at the top right. On the left is a small logo of a sailboat on water. The main content area contains a bulleted list of points about functional dependencies:

- Sets of functional dependencies may have redundant dependencies that can be inferred from the others
  - For example:  $A \rightarrow C$  is redundant in:  $\{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$
  - Parts of a functional dependency may be redundant
    - E.g.: on RHS:  $\{A \rightarrow B, B \rightarrow C, A \rightarrow CD\}$  can be simplified to  $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$ 
      - In the forward: (1)  $A \rightarrow CD \rightarrow A \rightarrow C$  and  $A \rightarrow D$  (2)  $A \rightarrow B, B \rightarrow C \rightarrow A \rightarrow C$
      - In the reverse: (1)  $A \rightarrow B, B \rightarrow C \rightarrow A \rightarrow C$  (2)  $A \rightarrow C, A \rightarrow D \rightarrow A \rightarrow CD$
    - E.g.: on LHS:  $\{A \rightarrow B, B \rightarrow C, AC \rightarrow D\}$  can be simplified to  $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$ 
      - In the forward: (1)  $A \rightarrow B, B \rightarrow C \rightarrow A \rightarrow C \rightarrow A \rightarrow AC$  (2)  $A \rightarrow AC, AC \rightarrow D \rightarrow A \rightarrow D$
      - In the reverse:  $A \rightarrow D \rightarrow AC \rightarrow D$
  - Intuitively, a canonical cover of  $F$  is a "minimal" set of functional dependencies equivalent to  $F$ , having no redundant dependencies or redundant parts of dependencies

At the bottom left is the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kanpur Date: Jan-Apr., 2018'. At the bottom center is 'Database System Concepts - 8<sup>th</sup> Edition' and '16.8'. At the bottom right is '©Silberschatz, Korth and Sudarshan'.

Now, we move forward from there and talk about what is known as a canonical cover. A set of functional dependencies may have a number of redundant dependencies also. So, we need to understand that because there are lot of dependencies which can be inferred from a certain set of dependencies, for example if you look into this set, you will easily understand that in this whole set if I actually have just this, we will be able to by transitivity, we will be able to conclude about  $A \rightarrow C$ . So, in that way  $AC$ ,  $A \rightarrow C$  is a redundant dependency.

So, here I am just showing you some examples. For example, say I have a set of functional dependencies as this set and I want to know whether I can replace it by a simpler set here where this particular attribute on the right hand side of this dependency may be extraneous. So, if I have to do that, then what we need to perform is, we need to show that given the set of functional dependencies, the original set whether this can imply this set that is from this set of functional dependencies, whether I can logically conclude the simplified set.

So, using the rules we will need to do that I have worked that out here under the forward scheme and we would also need to establish that if I have the simplified set, then can I go to the original set that was given. So, if the simplified set also logically  $\rightarrow$  the original set, then we can say that these are in a way equivalent and therefore, I would like to use a simpler set.

So, there is another example following here where I have another set given, where if we look into this, I would like to check whether I can get rid of this C on the left hand side and as it stands, we can actually do that and here in this whole process, I have shown it in terms of using the Armstrong's Axioms how you can prove this, but what we can do to systematize this whole process, we can again make use of the notion of closure of attributes and compute whether these two sets are equivalent, whether simplification can be done. So, we will say a cover is canonical. If it is in a sense minimal and still equivalent to the original set of dependencies and we will formally introduce what is minimal.

(Refer Slide Time: 06:44)

**Canonical Cover: RHS**

- $\{A \rightarrow B, B \rightarrow C, A \rightarrow CD\} \xrightarrow{\quad} \{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$ 
  - (1)  $A \rightarrow CD \xrightarrow{\quad} A \rightarrow C$  and  $A \rightarrow D$  (2)  $A \rightarrow B, B \rightarrow C \xrightarrow{\quad} A \rightarrow C$
  - $A^+ = ABCD$
  
- $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\} \xrightarrow{\quad} \{A \rightarrow B, B \rightarrow C, A \rightarrow CD\}$ 
  - $A \rightarrow B, B \rightarrow C \xrightarrow{\quad} A \rightarrow C$
  - $A \rightarrow C, A \rightarrow D \xrightarrow{\quad} A \rightarrow CD$
  - $A^+ = ABCD$

SWAYAM: NPTEL-NOCOs Instructor: Prof. P. P. Das, IIT Kharagpur. Date: Apr. 2018

Database System Concepts - 6<sup>th</sup> Edition      16.9      ©Silberschatz, Korth and Sudarshan

Before that let us just look at the same examples again. So, we are trying to show the forward direction in the first case and the reverse direction in the first case, but the only difference that I wanted to highlight is in terms of showing that you do not need to really explore on the Armstrong's Axioms, but what you can do is, you can simply take the left hand side attribute and compute its closure and see whether the right hand side is included. That is basically testing for whether the given functional dependency is actually implied.

(Refer Slide Time: 07:19)

The slide has a header 'PPD' and a title 'Canonical Cover: LHS'. It features a small sailboat icon in the top left. The main content is a list of functional dependency transformations:

- $\{A \rightarrow B, B \rightarrow C, AC \rightarrow D\} \rightarrow \{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$ 
  - $A \rightarrow B, B \rightarrow C \rightarrow A \rightarrow C \rightarrow A \rightarrow AC$
  - $A \rightarrow AC, AC \rightarrow D \rightarrow A \rightarrow D$
  - $A^+ = ABCD$
- $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\} \rightarrow \{A \rightarrow B, B \rightarrow C, AC \rightarrow D\}$ 
  - $A \rightarrow D \rightarrow AC \rightarrow D$
  - $AC^+ = ABCD$

On the left edge of the slide, there is vertical text: 'SWAYAM-NIITEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018 Date: 16.10 ©Silberschatz, Korth and Sudarshan'.

At the bottom right, there is a toolbar with icons for navigation and editing.

Similar things can be done to simplify the left hand side also. So, this is the other example I showed and I am just showing you that how you conclude this based on the closure of attributes algorithm.

(Refer Slide Time: 07:32)

The slide has a header 'PPD' and a title 'Extraneous Attributes'. It features a small sailboat icon in the top left. The main content is a list of definitions for extraneous attributes:

- Consider a set  $F$  of functional dependencies and the functional dependency  $\alpha \rightarrow \beta$  in  $F$ .
  - Attribute  $A$  is **extraneous** in  $\alpha$  if  $A \in \alpha$  and  $F$  logically implies  $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$ .
  - Attribute  $A$  is **extraneous** in  $\beta$  if  $A \in \beta$  and the set of functional dependencies  $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$  logically implies  $F$ .

On the right side of the slide, there are handwritten notes in red ink:  
 $\alpha \rightarrow \beta$   
 $A \in \beta$

On the left edge of the slide, there is vertical text: 'SWAYAM-NIITEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018 Date: 16.10 ©Silberschatz, Korth and Sudarshan'.

At the bottom right, there is a toolbar with icons for navigation and editing.

So, now I can formally define these possible removals. So, if I can remove an attribute as I have shown I can remove it from the right hand side or I can remove it from the left hand side. So, if an attribute can be removed, then it is called extraneous. So, if I have a functional dependency, let us say  $\alpha \rightarrow \beta$  and I have an attribute  $A \in \alpha$ , then we can check

whether it is possible to remove A from  $\alpha$ . So, to test that what we do is, we form a new set by removing the original functional dependency and adding the new functional dependency where the left hand side does not have that A and if F logically  $\rightarrow$  this, then certainly we can conclude that A on the left hand side of the functional dependency was extraneous.

Similar thing can be done for checking if there is an extraneous attribute on the right hand side of a dependency and in this case, naturally what we will need to do is, we will need to work out the simpler set and then check whether F is implied by that because as you can understand that if you are making the left hand, if you are removing an attribute from the left hand side, then you are making your precondition softer.

So, you need to see whether that is implied by the original set and on the other hand, if you are removing something on the right hand side, then you are making your consequence simpler. So, you need to understand whether that set  $\rightarrow$  the original set.

(Refer Slide Time: 09:27)

The slide has a header 'Extraneous Attributes' with a small sailboat icon. The content is organized into sections:

- Consider a set  $F$  of functional dependencies and the functional dependency  $\alpha \rightarrow \beta$  in  $F$ .
  - Attribute A is **extraneous** in  $\alpha$  if  $A \in \alpha$  and  $F$  logically implies  $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$ .
  - Attribute A is **extraneous** in  $\beta$  if  $A \in \beta$  and the set of functional dependencies  $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$  logically implies  $F$ .
- Note:* Implication in the opposite direction is trivial in each of the cases above, since a "stronger" functional dependency always implies a weaker one
- Example: Given  $F = \{A \rightarrow C, AB \rightarrow C\}$ 
  - B is extraneous in  $AB \rightarrow C$  because  $(A \rightarrow C, AB \rightarrow C)$  logically implies  $A \rightarrow C$  (i.e. the result of dropping B from  $AB \rightarrow C$ ).
  - $A^+ = AC$  in  $\{A \rightarrow C, AB \rightarrow C\}$
- Example: Given  $F = \{A \rightarrow C, AB \rightarrow CD\}$ 
  - C is extraneous in  $AB \rightarrow CD$  since  $AB \rightarrow C$  can be inferred even after deleting C
  - $AB^+ = ABCD$  in  $\{A \rightarrow C, AB \rightarrow D\}$

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P P Date, IIT Kanpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition      16.11      ©Silberschatz, Korth and Sudarshan

So, if you look into that and obviously, the other directions of this implication is not necessary to be proven because that will automatically follow because in the first case when I am removing an attribute, extraneous attribute from the left hand side of a functional dependency, naturally the set that I get that will always imply the original set because it is always possible to add additional attributes on the left hand side and so on. So, here are some examples worked out. So, here where I show that given a set AC and

$\mathbf{AB} \rightarrow \mathbf{C}$ , B is actually extraneous because as you can see if I remove B, then I get  $\mathbf{A} \rightarrow \mathbf{C}$  which is originally already there in the set. You can establish that by computing the closure of the attribute set.

Another example where you are trying to see an extraneous attribute on the right hand side; so in this example, C on the right hand side of the set  $\mathbf{AB} \rightarrow \mathbf{CD}$  is extraneous because it can be inferred even after because  $\mathbf{AB} \rightarrow \mathbf{C}$  can be inferred even after deleting this C from the right hand side.

(Refer Slide Time: 10:42)

The slide has a title 'Testing if an Attribute is Extraneous' in red at the top right. On the left is a small logo of a sailboat on water. The main content is a bulleted list of steps:

- Consider a set  $F$  of functional dependencies and the functional dependency  $\alpha \rightarrow \beta$  in  $F$ .
- To test if attribute  $A \in \alpha$  is extraneous in  $\alpha$ 
  1. Compute  $((\alpha - A)^*)^*$  using the dependencies in  $F$
  2. Check that  $((\alpha - A)^*)^*$  contains  $\beta$ ; if it does,  $A$  is extraneous in  $\alpha$
- To test if attribute  $A \in \beta$  is extraneous in  $\beta$ 
  1. Compute  $\alpha^*$  using only the dependencies in  $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ ,
  2. Check that  $\alpha^*$  contains  $A$ ; if it does,  $A$  is extraneous in  $\beta$

At the bottom left is vertical text: 'SWAYAM: NPTEL-NOCs; Instructor: Prof. P. P. Deshpande, IIT Kanpur'; at the bottom center: 'Database System Concepts - 8<sup>th</sup> Edition' and '16.12'; at the bottom right: '©Silberschatz, Korth and Sudarshan'. There is also a decorative footer bar with various icons.

So, these are using this notion. We can formalize a test for whether an attribute is extraneous. So, this is the formal steps of the step are given here, but I am sure you have already understood through the example.

(Refer Slide Time: 10:58)

The slide has a header 'Canonical Cover' in red. In the top right corner, there is a small logo with the letters 'PPD'. On the left side, there is a small image of a sailboat on water. The main content area contains the following text:

- A **canonical cover** for  $F$  is a set of dependencies  $F_c$  such that
  - $F$  logically implies all dependencies in  $F_c$ , and
  - $F_c$  logically implies all dependencies in  $F$ , and
  - No functional dependency in  $F_c$  contains an extraneous attribute, and
  - Each left side of functional dependency in  $F_c$  is unique
- To compute a canonical cover for  $F$ :  
repeat
  - Use the union rule to replace any dependencies in  $F$   
 $\alpha_1 \rightarrow \beta_1$  and  $\alpha_1 \rightarrow \beta_2$  with  $\alpha_1 \rightarrow \beta_1 \beta_2$
  - Find a functional dependency  $\alpha \rightarrow \beta$  with an extraneous attribute either in  $\alpha$  or in  $\beta$   
/\* Note: test for extraneous attributes done using  $F_c$ , not  $F^*$ /
  - If an extraneous attribute is found, delete it from  $\alpha \rightarrow \beta$until  $F$  does not change
- Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

At the bottom of the slide, there is a navigation bar with icons for back, forward, search, and other presentation controls. The footer contains the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr- 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '16.13', and '©Silberschatz, Korth and Sudarshan'.

So, given this a canonical cover of a set of functional dependencies  $F$ , it is denoted by  $F_c$  will mean that it is a set which is equivalent to  $F$  which means  $F$  will logically imply all dependencies in  $F_c$  and  $F_c$  will logically imply all dependencies in  $F$ .

No functional dependency in  $F_c$  will contain any extraneous attribute. So, all of them will be required attributes and each left hand side of the functional dependency in  $F_c$  must be unique. So, it is a minimal set of functional dependencies. Please note on these two core points. A cover is canonical if it is a minimal set and it is an irreducible set.

So, neither you can remove any dependency nor you can remove any extraneous attribute from this dependency set. So, here is the algorithm. So, I am not going through the steps of the algorithm. You can go through that and convince yourself that it indeed computes the canonical cover and practice more on that.

(Refer Slide Time: 12:07)

The slide has a header 'Computing a Canonical Cover' with a sailboat icon. On the left, there's a vertical sidebar with text: 'SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kanpur Date: Jan-Apr., 2018'. The main content area contains a bulleted list of steps for computing a canonical cover:

- $R = (A, B, C)$
- $F = \{A \rightarrow BC$ 
  - $B \rightarrow C$
  - $A \rightarrow B$
  - $AB \rightarrow C\}$
- Combine  $A \rightarrow BC$  and  $A \rightarrow B$  into  $A \rightarrow BC$ 
  - Set is now  $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$
- $A$  is extraneous in  $AB \rightarrow C$ 
  - Check if the result of deleting  $A$  from  $AB \rightarrow C$  is implied by the other dependencies
    - Yes: in fact,  $B \rightarrow C$  is already present!
  - Set is now  $\{A \rightarrow BC, B \rightarrow C\}$
- $C$  is extraneous in  $A \rightarrow BC$ 
  - Check if  $A \rightarrow C$  is logically implied by  $A \rightarrow B$  and the other dependencies
    - Yes: using transitivity on  $A \rightarrow B$  and  $B \rightarrow C$ .
    - Can use attribute closure of  $A$  in more complex cases

The canonical cover is: 
$$\begin{array}{l} A \rightarrow B \\ B \rightarrow C \end{array}$$

Navigation icons and copyright information at the bottom right: ©Silberschatz, Korth and Sudarshan

So, here I have shown an example where we want to compute the canonical cover here. So, first since all left hand sides have to be unique, so first we combine two, these two into in terms of  $A \rightarrow BC$ . So, it becomes a simpler set. So,  $A \rightarrow B$  is removed, then I would check for  $A$  being extraneous in  $AB \rightarrow C$  and we find that it indeed is extraneous.

So, because  $B \rightarrow C$  is already there, you can do the formal test in terms of the closure. So, the set gets even simpler. I will check if  $C$  is extraneous in  $A \rightarrow BC$ . I find that it indeed is and again you can use transitivity to get here or can use attribute closure and finally, I get that the set of the original set  $F$  is covered by a canonical set where just you have  $A \rightarrow B$  and  $B \rightarrow C$ .

So, this set is logically implied by the original set and this set can logically imply the original set and we will often use the canonical cover for simplicity and for ease of application.

(Refer Slide Time: 13:32)

The slide has a header 'Equivalence of Sets of Functional Dependencies' with a sailboat icon. It includes a sidebar with course information: SWAYAM: NPTEL-NOC MOOCs, Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr. 2018, and a footer 'PPD'. The main content discusses the equivalence of sets F and G of functional dependencies, defining it as  $F^+ = G^+$ . It lists four cases based on whether F covers G or G covers F:

| Condition  | CASES      |               |               |               |
|------------|------------|---------------|---------------|---------------|
|            | F Covers G | True          | True          | False         |
| G Covers F | True       | False         | True          | False         |
| Result     | $F=G$      | $F \supset G$ | $G \supset F$ | No Comparison |

A photograph of a professor is visible in the bottom left corner.

Naturally this is strongly using the underlying concept of equivalence of two sets of functional dependencies F and G. They are equivalent if their closures are equal or in other words, if F covers G and G covers F, that is F logically  $\rightarrow$  G and G logically  $\rightarrow$  F. So, this table shows you at different conditions where you can conclude whether F and G are equivalent sets of functional dependencies. So, they will have to, both covers have to be true for the sets to be equivalent.

(Refer Slide Time: 14:09)

The slide has a header 'Practice Problems on Functional Dependencies' with a sailboat icon. It includes a sidebar with course information: SWAYAM: NPTEL-NOC MOOCs, Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr. 2018, and a footer 'PPD'. The main content asks to find implied functional dependencies from a set of given ones:

- Find if a given functional dependency is implied from a set of Functional Dependencies:
  - For:  $A \rightarrow BC$ ,  $CD \rightarrow E$ ,  $E \rightarrow C$ ,  $D \rightarrow AEH$ ,  $ABH \rightarrow BD$ ,  $DH \rightarrow BC$ 
    - Check:  $BCD \rightarrow H$
    - Check:  $AED \rightarrow C$
  - For:  $AB \rightarrow CD$ ,  $AF \rightarrow D$ ,  $DE \rightarrow F$ ,  $C \rightarrow G$ ,  $F \rightarrow E$ ,  $G \rightarrow A$ 
    - Check:  $CF \rightarrow DF$
    - Check:  $BG \rightarrow E$
    - Check:  $AF \rightarrow G$
    - Check:  $AB \rightarrow EF$
  - For:  $A \rightarrow BC$ ,  $B \rightarrow E$ ,  $CD \rightarrow EF$ 
    - Check:  $AD \rightarrow F$

A photograph of a professor is visible in the bottom left corner.

Next what I have done is, we have put a number of practice problems for various kind of things that you can do with functional dependencies. The first set of problems. Find in first set of problems you have to find if a given functional dependency is implied from a set of functional dependencies. So, there are three problems where three sets of functional dependencies are given and you are given to check one or more functional dependencies if it is implied from that set. So, use the attribute closure and the algorithm that we have discussed to practice these problems and become master of that.

(Refer Slide Time: 14:54)

The slide has a header 'Practice Problems on Functional Dependencies' with a sailboat icon. On the right, it says 'PPD'. The left margin contains vertical text: 'SWAYAM NPTEL-NOCO MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jain-Apr-2018'. The main content area has a bullet point '■ Find Super Key using Functional Dependencies:' followed by two numbered examples. Below this is a video player showing a man speaking, with the source 'Source: http://www.edugrabs.com/how-to-find-super-key-from-functional-dependencies' and a timestamp '16.18'. The bottom right corner says '©Silberschatz, Korth and Sudarshan'.

You can also check if you can find candidate key using the functional dependencies. The sets are given. Your task would be to find the candidate keys.

You can also use the algorithms to find super keys for a given set of functional dependencies. So, do practice these problems.

 Practice Problems on Functional Dependencies PPD

SWAYAM: NPTEL-NOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr - 2018

■ Find Prime and Non Prime Attributes using Functional Dependencies:

1. R(ABCDEF) having FDs {AB→C, C→D, D→E, F→B, E→F}
2. R(ABCDEF) having FDs {AB → C, C → DE, E → F, C → B}
3. R(ABCDEFGHIJ) having FDs {AB → C, A → DE, B → F, F → GH, D → IJ}
4. R(ABDLPT) having FDs {B → PT, A → D, T → L}
5. R(ABCDEFGH) having FDs {E → G, AB → C, AC → B, AD → E, B → D, BC → A}
6. R(ABCDE) having FDs {A → BC, CD → E, B → D, E → A}
7. R(ABCDEH) having FDs {A → B, BC → D, E → C, D → A}

- Prime Attributes – Attribute set that belongs to any candidate key are called Prime Attributes
  - It is union of all the candidate key attribute: {CK1 ∪ CK2 ∪ CK3 ∪ .....}
  - If Prime attribute determined by other attribute set, then more than one candidate key is possible.
  - For example, If A is Candidate Key, and X→A, then, X is also Candidate Key .
- Non Prime Attribute – Attribute set does not belongs to any candidate key are called Non Prime Attributes

Source: <http://www.edugrabs.com/prime-and-non-prime-attributes/>

Database System Concepts - 8<sup>th</sup> Edition 16.19 ©Silberschatz, Korth and Sudarshan

You can find prime and non-prime attributes using functional dependencies. Prime attributes are attributes that belong to any candidate key, not necessarily the same candidate key. All attributes that belong to some candidate key, you take a set together and you call them as a prime attribute and non prime attributes are those that do not belong to any candidate key at all. So, here your task is to find the prime and non-prime attributes using the sets of functional dependencies given.

(Refer Slide Time: 15:50)

 Practice Problems on Functional Dependencies PPD

SWAYAM: NPTEL-NOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr - 2018

■ Check the Equivalence of a Pair of Sets of Functional Dependencies:

1. Consider the two sets F and G with their FDs as below :
  1. F : A → C, AC → D, E → AD, E → H
  2. G: A → CD, E → AH
2. Consider the two sets P and Q with their FDs as below :
  1. P : A → B, AB → C, D → ACE
  2. Q : A → BC, D → AE

Source: <http://www.edugrabs.com/equivalence-of-sets-of-functional-dependencies/>

Database System Concepts - 8<sup>th</sup> Edition 16.20 ©Silberschatz, Korth and Sudarshan

You can check for equivalents for a pair of sets of functional dependencies. There are couple of problems given on that. So, please try them out.

(Refer Slide Time: 16:01)

**Practice Problems on Functional Dependencies**

PPD

■ Find the Minimal Cover or Irreducible Sets or Canonical Cover of a Set of Functional Dependencies:

1.  $AB \rightarrow CD, BC \rightarrow D$
2.  $ABCD \rightarrow E, E \rightarrow D, AC \rightarrow D, A \rightarrow B$

Source: <http://www.edugrabs.com/questions-on-minimal-cover/>

SWAYAM-NIETEL-NOC-MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur Date: 2018

16.21 ©Silberschatz, Korth and Sudarshan

For here for the different sets, you have to compute the minimal cover or the irreducible set or canonical cover of the set of functional dependencies.

So, please practice on this problem, so that you become comfortable with using this algorithms for dealing easily with the functional dependency sets of functional dependencies, individual functional dependencies and so on. So, after this week is closed and your assignments are also done, then we will publish the solutions for these practice problems as well next let me take up a little characterization of the concept that we had introduced earlier in terms of the lossless join decomposition.

(Refer Slide Time: 16:48)

The slide has a title 'Lossless-join Decomposition' at the top right. On the left, there is a small logo of a sailboat on water. The main content area contains a bulleted list of points about lossless join decomposition. A callout box provides conditions for identifying lossless or lossy decomposition.

**Lossless-join Decomposition**

- For the case of  $R = (R_1, R_2)$ , we require that for all possible relations  $r$  on schema  $R$   
 $r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$
- A decomposition of  $R$  into  $R_1$  and  $R_2$  is lossless join if at least one of the following dependencies is in  $F^+$ :
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$
- The above functional dependencies are a sufficient condition for lossless join decomposition; the dependencies are a necessary condition only if all constraints are functional dependencies

*To Identify whether a decomposition is lossy or lossless, it must satisfy the following conditions :*

- $R_1 \cup R_2 = R$
- $R_1 \cap R_2 \neq \emptyset$  and
- $R_1 \cap R_2 \rightarrow R_1$  or  $R_1 \cap R_2 \rightarrow R_2$

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition 16.23 ©Silberschatz, Korth and Sudarshan

So, in the lossless join decomposition, the problem is say that you have a relational scheme  $R$  and you are trying to divide that into two relational schemes  $R_1$  and  $R_2$ . So, both  $R_1$  and  $R_2$  are having a set of attributes and  $R$  naturally has a set of attributes which is a union of the attributes of  $R_1$  and  $R_2$ , then is it possible that if I take a relation, project it on the attributes of  $R_1$  and on the attributes of  $R_2$ , the two relations that we get. If I take a natural join of that, do I get back  $R$ ?

If I do, then I say that I have a lossless join. If I do not, then I have lost some information due to this projection and re-computation of the original relation based using the natural join. This requirement of lossless join decomposition is determined if at least one of the following dependencies exist in the closure set of  $F$  which this is saying that if I do  $\mathbf{R1} \cap \mathbf{R2}$ , that is A attributes which are common. You will recall that when we do natural join, it is this set of attributes which take part because these set of attributes will help you  $\Pi_{R1}(r) \ \Pi_{R2}(r)$ .

So, if  $\mathbf{R1} \cap \mathbf{R2} \rightarrow \mathbf{R1}$  or  $\mathbf{R1} \cap \mathbf{R2} \rightarrow \mathbf{R2}$ , that is if the intersection set of attributes is a super key either in  $R_1$  or in  $R_2$  or both then, we say that the join will be a lossless join. Note that this is a sufficient condition which means that there could be some instances where this property is not satisfied yet the join is lossless, but we need guarantees for our design. So, we make use of the fact that if one of these conditions are satisfied, then it is a sufficient condition to say that the join will must, join will necessarily be lossless.

(Refer Slide Time: 19:10)

**Example**

- Consider Supplier\_Parts schema: Supplier\_Parts(S#, Sname, City, P#, Qty) ✓
- Having dependencies: S# → Sname, S# → City, (S#, P#) → Qty ✓
- Decompose as: Supplier(S#, Sname, City, Qty) / Parts(P#, Qty)
- Take Natural Join to reconstruct: Supplier ⚡ Parts

| S# | Sname | City   | P#  | Qty | S# | Sname | City   | Qty | P#  | Qty | S# | Sname | City   | P#  | Qty |
|----|-------|--------|-----|-----|----|-------|--------|-----|-----|-----|----|-------|--------|-----|-----|
| 3  | Smith | London | 301 | 20  | 3  | Smith | London | 20  | 301 | 20  | 3  | Smith | London | 301 | 20  |
| 5  | Nick  | NY     | 500 | 50  | 5  | Nick  | NY     | 50  | 500 | 50  | 5  | Nick  | NY     | 500 | 50  |
| 2  | Steve | Boston | 20  | 10  | 2  | Steve | Boston | 10  | 20  | 10  | 5  | Nick  | NY     | 20  | 10  |
| 5  | Nick  | NY     | 400 | 40  | 5  | Nick  | NY     | 40  | 400 | 40  | 2  | Steve | Boston | 20  | 10  |
| 5  | Nick  | NY     | 301 | 10  | 5  | Nick  | NY     | 10  | 301 | 10  | 5  | Nick  | NY     | 400 | 40  |

We get extra tuples! Join is Lossy!

Common attribute Qty is not a superkey in Supplier or in Parts

Does not preserve (S#, P#) → Qty

Source: <http://www.edugrabs.com/lossy-join-decomposition/>

So, here I give you a quick example to show the idea. So, we have a supplier relationship here which has five attributes. Here is an instance of that and we know that these are the dependencies that hold the **supplier number → the supplier name** and the **(supplier city, supplier number, product number) → quantity** and we decompose them in this manner, we put a supplier relationship where we have the number, name, city and quantity of supplier and then, we have parts relation where we just have a product name and the quantity.

So, this is the projected supplier relation instance. This is a projected parts relation instance and then, we take a natural join to reconstruct. So, we are taking a natural join to reconstruct and we get this relationship. Now, our desire was that we must get back the original relation, but if you compare, you will find that this is not the case here. We have one tuple here and we have another tuple here. I have specifically highlighted them in red which were not there in the original relation.

They have come in because when I did the join naturally, the join had to be performed on this common attribute quantity and based on that value. So, Nick, 5 Nick NY, then we have 10, 5 Nick NY 10, this entry and we have two entries of 10 and 10 here. So, the combination of this with this where the product number is 20 is actually not present in the original instance of the relation and that is what shows up here a similar one exists here.

So, we get extra tuples and mind you though we are actually getting extra tuple, we will say that this join is lossy because if you get extra tuple, then you are losing information, you are losing correctness. So, being lossy is actually losing correctness. So, even though we have more tuples, we say that this is a lossy join and you can now go back and analyze it. The common attribute QTY is not a super key either in this or in this. So,  $R1 \cap R2 \rightarrow R1$  or  $R1 \cap R2 \rightarrow R2$  does not hold. So, it does not and in addition it also does not preserve this functional dependency because these are not, no more determined.

Now, let us see it. So, we saw a case where the join decomposition that we did and then, the subsequent join that we performed did not prove to be a lossless join. We lost information. So, let us take a look as to can we actually do a decomposition which will be lossless where we will not lose information.

(Refer Slide Time: 22:29)

**Example**

- Consider Supplier\_Parts schema: Supplier\_Parts(S#, Sname, City, P#, Qty)
- Having dependencies:  $S\# \rightarrow Sname$ ,  $S\# \rightarrow City$ ,  $(S\#, P\#) \rightarrow Qty$
- Decompose as: Supplier(S#, Sname, City) Parts(S#, P#, Qty)
- Take Natural Join to reconstruct: Supplier  $\bowtie$  Parts

| S# | Sname | City   | P#  | Qty | S# | Sname | City   | S# | Sname | City   | P#  | Qty |
|----|-------|--------|-----|-----|----|-------|--------|----|-------|--------|-----|-----|
| 3  | Smith | London | 301 | 20  | 3  | Smith | London | 3  | Smith | London | 301 | 20  |
| 5  | Nick  | NY     | 500 | 50  | 5  | Nick  | NY     | 5  | Nick  | NY     | 500 | 50  |
| 2  | Steve | Boston | 20  | 10  | 2  | Steve | Boston | 2  | Steve | Boston | 20  | 10  |
| 5  | Nick  | NY     | 400 | 40  | 5  | Nick  | NY     | 5  | Nick  | NY     | 400 | 40  |
| 5  | Nick  | NY     | 301 | 10  | 5  | Nick  | NY     | 5  | Nick  | NY     | 301 | 10  |

SWAYAM: NPTEL-NOCO Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr-2018

| S# | Sname | City   | P#  | Qty | S# | Sname | City   | S# | Sname | City   | P#  | Qty |
|----|-------|--------|-----|-----|----|-------|--------|----|-------|--------|-----|-----|
| 3  | Smith | London | 301 | 20  | 3  | Smith | London | 3  | Smith | London | 301 | 20  |
| 5  | Nick  | NY     | 500 | 50  | 5  | Nick  | NY     | 5  | Nick  | NY     | 500 | 50  |
| 2  | Steve | Boston | 20  | 10  | 2  | Steve | Boston | 2  | Steve | Boston | 20  | 10  |
| 5  | Nick  | NY     | 400 | 40  | 5  | Nick  | NY     | 5  | Nick  | NY     | 400 | 40  |
| 5  | Nick  | NY     | 301 | 10  | 5  | Nick  | NY     | 5  | Nick  | NY     | 301 | 10  |

Source: <http://www.edugrabs.com/desirable-properties-of-decomposition/lossless>

So, I take the same example the supplier, but the decomposition, the same set of dependencies also, but the decomposition is different. Now, we have name number, name and city in one supplier relation and supplier name, number, product number and quantity in the other parts relation and then, we again go back and perform the join.

Now, we find that from the original relation, this was the original relation, this is the projected supplier relation, these are projected parts relation and this is the natural join of these two relations. So, this is the natural join of these two relations and we find that they exactly match with the original relation.

So, we have not lost any information we get it back. So, we say that the join is lossless and the reason we could guarantee that is because if you look into the set of functional dependencies, you will find that S number, the supplier number is a key in the supplier relationship because  $S\# \rightarrow S \text{ name}$ ,  $S \text{ city}$ . So,  $R_1 \cap R_2 \rightarrow R_1$  is true here and therefore, it actually gives you a lossless join.

It also preserves all the dependencies because if you look into these dependencies, you can check for this dependency. In this relation, you can check for this dependency also in this relation and you can check for this dependency in also in the parts relation which is something which we were not able to do in the last decomposition that we have.

(Refer Slide Time: 24:33)

**Example**

- $R = (A, B, C)$   
 $F = \{A \rightarrow B, B \rightarrow C\}$ 
  - Can be decomposed in two different ways
- $R_1 = (A, B), R_2 = (B, C)$ 
  - Lossless-join decomposition:  
 $R_1 \cap R_2 = \{B\}$  and  $B \rightarrow BC$
  - Dependency preserving
- $R_1 = (A, B), R_2 = (A, C)$ 
  - Lossless-join decomposition:  
 $R_1 \cap R_2 = \{A\}$  and  $A \rightarrow AB$
  - Not dependency preserving  
(cannot check  $B \rightarrow C$  without computing  $R_1 \bowtie R_2$ )

SWAYAM/NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr. 2018

Dat... 16.26 ©Silberschatz, Korth and Sudarshan

So, naturally this is a type of decomposition that we will prefer. So, here we have I have given some more examples which you can practice and I show that given a very simple schema having three attributes and two dependencies, one decomposition into AB and BC is lossless join decomposition whereas, the other one AB and AC is a lossy decomposition.

(Refer Slide Time: 24:57)

The slide features a sailboat icon in the top left corner. The title 'Practice Problems on Lossless Join' is centered at the top in red. In the top right corner, there is a small 'PPD' logo. On the left side, there is vertical text: 'SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Des., IIT Kharagpur - Jan-Apr- 2018'. A video thumbnail of a man speaking is in the center-left. Below it, a list of practice problems is provided:

- Check if the decomposition of R into D is lossless:
  1. R(ABC): F = {A → B, A → C}. D = R<sub>1</sub>(AB), R<sub>2</sub>(BC)
  2. R(ABCDEF): F = {A → B, B → C, C → D, E → F}. D = R<sub>1</sub>(AB), R<sub>2</sub>(BCD), R<sub>3</sub>(DEF)
  3. R(ABCDEF): F = {A → B, C → DE, AC → F}. D = R<sub>1</sub>(BE), R<sub>2</sub>(ACDEF)
  4. R(ABCDEG): F = {AB → C, AC → B, AD → E, B → D, BC → A, E → G}
    1. D<sub>1</sub> = R<sub>1</sub>(AB), R<sub>2</sub>(BC), R<sub>3</sub>(ABDE), R<sub>4</sub>(EG)
    2. D<sub>2</sub> = R<sub>1</sub>(ABC), R<sub>2</sub>(ACDE), R<sub>3</sub>(ADG)
  5. R(ABCDEFGHIJ): F = {AB → C, B → F, D → IJ, A → DE, F → GH}
    1. D<sub>1</sub> = R<sub>1</sub>(ABC), R<sub>2</sub>(ADE), R<sub>3</sub>(BF), R<sub>4</sub>(FGH), R<sub>5</sub>(DIJ)
    2. D<sub>2</sub> = R<sub>1</sub>(ABCDE), R<sub>2</sub>(BFGH), R<sub>3</sub>(DIJ)
    3. D<sub>3</sub> = R<sub>1</sub>(ABCD), R<sub>2</sub>(DE), R<sub>3</sub>(BF), R<sub>4</sub>(FGH), R<sub>5</sub>(DIJ)

Source: <http://www.edugrabs.com/questions-on-lossless-join/>

16.27 ©Silberschatz, Korth and Sudarshan

I have given a number of practice problems on lossless join, so that you can practice and become master of these kind of algorithm. Finally, let me quickly go over the dependency preservation concept.

(Refer Slide Time: 25:17)

The slide features a sailboat icon in the top left corner. The title 'Dependency Preservation' is centered at the top in red. In the top right corner, there is a small 'PPD' logo. On the left side, there is vertical text: 'SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Des., IIT Kharagpur - Jan-Apr- 2018'. A video thumbnail of a man speaking is in the center-left. Below it, a list of points is provided:

- Let  $F_i$  be the set of dependencies  $F^+$  that include only attributes in  $R_i$ 
  - A decomposition is **dependency preserving**, if
$$(F_1 \cup F_2 \cup \dots \cup F_n)^+ = F^+$$
  - If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive

Let R be the original relational schema having FD set F. Let  $R_1$  and  $R_2$  having FD set  $F_1$  and  $F_2$  respectively, are the decomposed sub-relations of R. The decomposition of R is said to be preserving if

- $F_1 \cup F_2 \equiv F$  (Decomposition Preserving Dependency)
- If  $F_1 \cup F_2 \subset F$  (Decomposition NOT Preserving Dependency) and
- $F_1 \cup F_2 \supset F$  (this is not possible)

Source: <http://www.edugrabs.com/questions-on-lossless-join/>

16.29 ©Silberschatz, Korth and Sudarshan

Dependency preservation is if you have a relation which you have decomposed into n different relations, so if you decompose a relation into a number of relations, then naturally all functional dependencies you cannot check on all the relations because a

dependency may involve attributes all of which may not be present in a particular decomposed relation that you have. It may be distributed amongst different.

So, when you do this decomposition, for every relation you get a new set of subset of functional dependencies. So, the decomposed relation  $R_i$  the  $i^{\text{th}}$  relation will have a set of dependencies  $F_i$  which is a subset of the original set  $F$  and involves only the attributes which exist in  $R_i$ . So, the decomposition will be said to be dependency preserving if I can take the union of all these functional dependencies, what is projected on  $R_1$ , on  $R_2$  and  $R_n$ ,  $F_1, F_2, F_n$ . If we can take union of and if we take  $F$ , they must be equivalent sets which we know the requirement.

So, equivalence mean that their covers will have to be equal. If it is not, then some there will be at least one dependency which you will not be able to check in any one of the projected relations and to be able to check that, you will have to compute the natural join and that is as we know is a very expensive process and we would not be able to do that on a regular basis.

(Refer Slide Time: 27:12)

The slide has a title 'Testing for Dependency Preservation' in red. To the left is a small logo of a sailboat on water. On the right is a video frame showing a man speaking. The slide contains the following text:

To check if a dependency  $\alpha \rightarrow \beta$  is preserved in a decomposition of  $R$  into  $R_1, R_2, \dots, R_n$  we apply the following test (with attribute closure done with respect to  $F$ )

- $\text{result} = \alpha$
- **while** (changes to  $\text{result}$ ) **do**
- for each**  $R_i$  **in the decomposition**
- $t = (\text{result} \cap R_i)^* \cap R_i$
- $\text{result} = \text{result} \cup t$
- If  $\text{result}$  contains all attributes in  $\beta$ , then the functional dependency  $\alpha \rightarrow \beta$  is preserved.
- We apply the test on all dependencies in  $F$  to check if a decomposition is dependency preserving
- This procedure takes polynomial time, instead of the exponential time required to compute  $F^*$  and  $(F_1 \cup F_2 \cup \dots \cup F_n)^*$

At the bottom, it says 'SWAYAM NPTEL-NOC's Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr- 2018'. There is also a footer with 'Database System Concepts - 8<sup>th</sup> Edition', '16.30', and '©Silberschatz, Korth and Sudarshan'.

So, here I have written down the algorithm to test if a decomposition actually preserves the dependency or not. So, I will not go through the steps. I will leave that for you to understand, but what I will do, I will just show you a simple set of worked out example and reason on that.

(Refer Slide Time: 27:34)

The slide has a title 'Example' in red at the top right. On the left, there is a small logo of a sailboat. The main content area contains the following text and tables:

- R(ABCDEF):
- F = {A→BCD, A→EF, BC→AD, BC→E, BC→F, B→F, D→E}
- D = {ABCD, BF, DE}
- On projections:

| ABCD (R1)          | BF (R2) | DE (R3) |
|--------------------|---------|---------|
| A → BCD<br>BC → AD | B → F   | D → E   |

Below the table, there is a list of dependencies and their preservation status:

- Need to check for: A→BCD, A→EF, BC→AD, BC→E, BC→F, B→F, D→E
- (BC)+/F1 = ABCD, (ABCD)+/F2 = ABCDF, (ABCD)+/F3 = ABCDEF. Preserves BC→E, BC→F
- (A)+/F1 = ABCD, (ABCD)+/F2 = ABCDF, (ABCD)+/F3 = ABCDEF. Preserves A→EF

At the bottom left, there is a video feed of a professor. The bottom right shows a toolbar with various icons.

So, I show you two different methods of doing this. So, here we have a set of attributes given the dependencies that work in that and a particular decomposition. So, given the set of attributes and the decomposition if we project, now if we project the set of functional dependencies and these are the sets that we get. So, on R1, we have two dependencies on R2, we have three dependence, one dependency and R3 we have one dependency again.

So, if we now think about the U of these and the closure for that, then we can see that these four dependencies which occur here and therefore, I have struck them off in this set. These four dependencies can be checked directly on the projected relations. So, that leaves us with three dependencies in the original set which cannot be checked on any one of R1, R2 or R3. For example, if you consider **BC** → **E**, then B exist on R1 and C also exist on R1, but E is not there. So, you cannot check that dependency on R1, you cannot check that on R2 because C and E do not exist and you cannot check them on R3, check it on R3 because none of them actually exist.

So, what we will need for the dependency preservation to hold is the dependencies which are already existing four dependencies that are struck off if they collectively can logically imply these dependencies, so that they can be checked. Then, we will be able to say that this is dependency preserving. So, what you do is something very simple. You want to say, you want to check whether this is preserved. So, we start with the left hand

side and compute the closure. The only difference you compute the closure first with the set of functional dependencies projected on R1, that is F1, the set closure set that you get, you take that and compute its closure with respect to the second set of functional dependencies F2.

The closure that you get, you take that and you compute the closure with respect to the third set of functional dependencies which is on R3 and that is your final closure set. So, this closure set includes the right hand side attribute E. So, we can conclude that  $\text{BC} \rightarrow \text{E}$  and that relationship will be preserved because we have starting from BC. We have seen that in every projected relation what all implied functional dependencies that can be checked which is what the meaning of the closure set of attributes R and since that set eventually has E, we will know that this can be, this will be preserved.

This set also has F. So, the other one will also be preserved. So, this is preserved, this is preserved to check whether this dependency is preserved. We need to again repeat the process and find whether EF belongs to the final closure set which it does and therefore, we conclude that this decomposition is dependency preserving.

(Refer Slide Time: 31:04)

**Example**

PPD

SWAYAM: NPTEL-NOOC MOOCs Instructor: Prof. P. P. Doshi, IIT Kanpur - Jan-Apr - 2018

- R(ABCDEF): F = {A→BCD, A→EF, BC→AD, BC→E, BC→F, B→F, D→E}. D = {ABCD, BF, DE}
- On projections:
 

|           |         |         |
|-----------|---------|---------|
| ABCD (R1) | BF (R2) | DE (R3) |
|-----------|---------|---------|

 A → B, A → C, A → D, BC → A, BC → D, B → F, D → E
- Infer reverse FD's:
  - B+F = BF: B → A cannot be inferred
  - C+F = C: C → A cannot be inferred
  - D+F = DE: D → A and D → BC cannot be inferred
  - A+F = ABCDEF: A → BC can be inferred, but it is equal to A → B and A → C
  - F+F = F: F → B cannot be inferred
  - E+F = E: E → D cannot be inferred
- Need to check for: A→BCD, A→EF, BC→AD, BC→E, BC→F, B→F, D→E
  - (BC)+E = ABCDEF. Preserves BC→E, BC→F ✓ ✓
  - (A)+F = ABCDEF. Preserves A→EF ✓

With the same example I will just show you a little different way of ah doing the same exercise. I have not written down the algorithm for this in longhand, but the example should be quite illustrative. So, we are what you do when you project, you check if some dependency has multiple attributes on the left hand, on the right hand side, then you

write them in a separately decomposed manner. So,  $A \rightarrow BCD$  is written in terms of three dependencies.  $A \rightarrow B$ ,  $B \rightarrow C$  and  $C \rightarrow D$ . So, you make sure that all dependencies are written in a form where the right hand side has a single attribute, then you compute what is known as the reverse functional dependencies that is you take the right hand side and compute whether the right hand side can imply the left hand side.

So, I will just show you one. So, in case the right hand side here is B, you have AB on the right hand side. So, you compute the closure with respect to F. The original set, not the projected set of D and you get BF. So, you know that this inverse, this reverse functional dependency which is  $AB \rightarrow A$  which is the reverse dependency cannot be inferred and you do this for each of the right hand side single attribute and check if some, if the reverse dependencies can be inferred or not.

The interesting case occurs here where if you try to do the closure of A, you actually find that  $A \rightarrow BC$  which is a reverse of this functional dependency can be inferred, but you do not consider that as a violation because it is you already have  $A \rightarrow B$  and  $A \rightarrow C$ . So, that logically implying that  $A \rightarrow BC$ . So, it is not a new violation that is getting imposed.

So, with this your test for reverse functional dependencies is passed and then, you finally check for whether the three dependencies which are not part of the projected set of dependencies, you take the closure of the left hand side with respect to in this case. Again, the original set of functional dependencies, not the projected one and check if the right hand side belongs there. If they do, then combined with these two strategies you say that the set of functional dependencies are preserved under this decomposition.

So, this is the process to follow. You can follow any one of the two approaches to solve.

(Refer Slide Time: 34:01)

The slide features a sailboat icon in the top left corner. The title 'Practice Problems on Dependency Preservation' is centered at the top in red. Below the title, a bullet point says 'Check whether the decomposition of R into D is preserving dependency:' followed by five numbered questions. The footer contains the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr- 2018', 'Source: http://www.edugrabs.com/question-on-dependency-preserving-deco...', 'Database System Concepts - 8<sup>th</sup> Edition', '16.33', and '©Silberschatz, Korth and Sudarshan'.

So, I have given some practice problems on dependency preservation which you should practice on to.

(Refer Slide Time: 34:06)

The slide features a sailboat icon in the top left corner. The title 'Module Summary' is centered at the top in red. Below the title, there is a bulleted list of three items. The footer contains the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr- 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '16.34', and '©Silberschatz, Korth and Sudarshan'.

Summarize we have studied the algorithms for properties of functional dependencies and we have understood the characterization and determination algorithm for lossless join decomposition and for dependency preservation in a decomposition. In the coming module, we will make use of these and discuss about how to improve these designs of relational schemas through the use of different normal forms.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 19**  
**Relational Database Design (Contd.)**

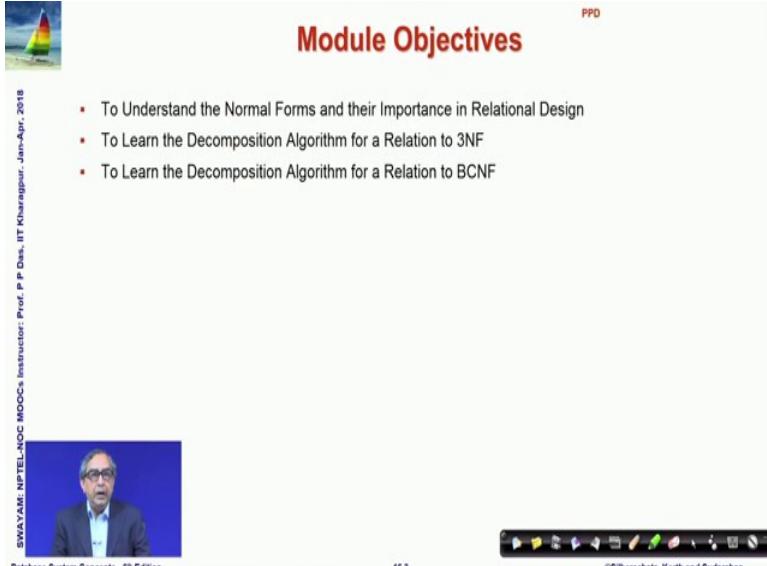
Welcome to module 19 of Database Management Systems; we have been discussing relational database design and this is the fourth part; fourth module in that series.

(Refer Slide Time: 00:38)

The slide has a green header bar with the text "Module Recap" in red. In the top right corner, there is a small "PPD" logo. On the left side, there is a small image of a sailboat on water. The main content area contains a bulleted list of three items: "Algorithms for Functional Dependencies", "Lossless Join Decomposition", and "Dependency Preservation". At the bottom of the slide, there is a footer bar with several icons (including arrows, a magnifying glass, and a search icon) and some text. The footer also includes the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr., 2018", "Database System Concepts - 8<sup>th</sup> Edition", "16.2", and "©Silberschatz, Korth and Sudarshan".

In the last module, we have discussed about algorithms for functional dependencies lossless joint decomposition and dependency preservation. So, based on this foundational algorithms and concepts.

(Refer Slide Time: 00:51)

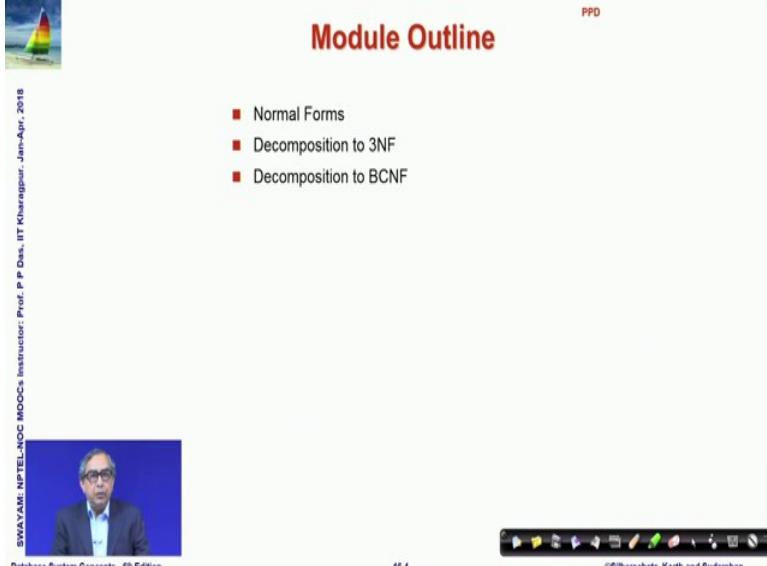


The slide is titled "Module Objectives" in red. It features a small sailboat icon in the top left corner and a video player window showing a man speaking in the bottom left. The footer includes the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr., 2018", "Database System Concepts - 8<sup>th</sup> Edition", "16.3", and "©Silberschatz, Korth and Sudarshan".

- To Understand the Normal Forms and their Importance in Relational Design
- To Learn the Decomposition Algorithm for a Relation to 3NF
- To Learn the Decomposition Algorithm for a Relation to BCNF

We will in today's module get into understanding the core design aspects of relational databases; that is a normal forms and how important they are in terms of the relational design. We would specifically learn about decomposition of a relational schema into the third normal form and into Boyce Codd BCNF form.

(Refer Slide Time: 01:20)



The slide is titled "Module Outline" in red. It features a small sailboat icon in the top left corner and a video player window showing a man speaking in the bottom left. The footer includes the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr., 2018", "Database System Concepts - 8<sup>th</sup> Edition", "16.4", and "©Silberschatz, Korth and Sudarshan".

- Normal Forms
- Decomposition to 3NF
- Decomposition to BCNF

So, our topics will be the three normal forms decomposition of 3 NF and into BCNF.

(Refer Slide Time: 01:27)

The slide features a small sailboat icon in the top left corner. In the top right, the text "PPD" is written vertically. Below it, a bulleted list includes "Normal Forms", "Decomposition to 3NF", and "Decomposition to BCNF". The main title "NORMAL FORMS" is centered in large red capital letters. Below the title is a video frame showing a man with glasses and a blue shirt. The video frame has a vertical caption "SWAYAM: NPTEL NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr. 2018". At the bottom of the slide, there is a navigation bar with icons and the text "Database System Concepts - 8<sup>th</sup> Edition", "16.5", and "©Silberschatz, Korth and Sudarshan".

So, starting with the normal forms.

(Refer Slide Time: 01:29)

The slide features a small sailboat icon in the top left corner. The main title "Normalization or Schema Refinement" is centered in large red capital letters. Below the title is a bulleted list of points about normalization:

- Normalization or Schema Refinement is a technique of organizing the data in the database
- A systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics
  - Insertion Anomaly
  - Update Anomaly
  - Deletion Anomaly
- Most common technique for the Schema Refinement is decomposition.
  - Goal of Normalization: Eliminate Redundancy
- Redundancy refers to repetition of same data or duplicate copies of same data stored in different locations
- Normalization is used for mainly two purpose:
  - Eliminating redundant (useless) data
  - Ensuring data dependencies make sense, that is, data is logically stored

At the bottom of the slide, there is a navigation bar with icons and the text "Database System Concepts - 8<sup>th</sup> Edition", "16.6", and "©Silberschatz, Korth and Sudarshan".

So, normal forms or normalization of a schema is a technique of refinement to organize the data in the database. So, the question naturally arises as to why do we need to do this refinement after we have done a design based on possibly the E-R diagram based approach that we had talked of we had identified the entities and we had identified the attributes for the entities their relationships; then why do we need to normalize?

The answer to this question lies in the fact that a design for a relational schema may give rise to a variety of anomalies in terms of the data. These are typically three anomalies which concerns us most the insertion, the update and the deletion anomaly. So, the anomalies happen when there is redundancy in the data in terms of the schema. And whether there will be redundant data and how much what kind of redundant data would be there depends on the design of the database schema depends on the design of the normal form that we are using for it.

But if we have redundancy then there is potential for anomalies and therefore, we want to reduce the redundancy and get rid of this anomaly.

(Refer Slide Time: 03:00)

**Anomalies**

PPD

SWAYAM-NPTEL-NOOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur

**1. Update Anomaly:** Employee 519 is shown as having different addresses on different records

| Employees' Skills |                    |                 |
|-------------------|--------------------|-----------------|
| Employee ID       | Employee Address   | Skill           |
| 426               | 87 Sycamore Grove  | Typing          |
| 426               | 87 Sycamore Grove  | Shorthand       |
| 519               | 94 Chestnut Street | Public Speaking |
| 519               | 96 Walnut Avenue   | Carpentry       |

**2. Insertion Anomaly:** Until the new faculty member, Dr. Newsome, is assigned to teach at least one course, his details cannot be recorded

| Faculty ID | Faculty Name   | Faculty Hire Date | Course Code |
|------------|----------------|-------------------|-------------|
| 389        | Dr. Giddens    | 10-Feb-1985       | ENG-206     |
| 407        | Dr. Saperstein | 19-Apr-1999       | CMP-101     |
| 407        | Dr. Saperstein | 19-Apr-1999       | CMP-201     |
| 424        | Dr. Newsome    | 29-Mar-2007       | ?           |

**3. Deletion Anomaly:** All information about Dr. Giddens is lost if he temporarily ceases to be assigned to any courses.

| Faculty ID | Faculty Name   | Faculty Hire Date | Course Code |
|------------|----------------|-------------------|-------------|
| 389        | Dr. Giddens    | 10-Feb-1985       | ENG-206     |
| 407        | Dr. Saperstein | 19-Apr-1999       | CMP-101     |
| 407        | Dr. Saperstein | 19-Apr-1999       | CMP-201     |

Resolution: Decompose the Schema

1. Update: (ID, Address), (ID, Skill)
2. Insert: (ID, Name, Hire Date), (ID, Code)
3. Delete: (ID, Name, Hire Date), (ID, Code)

Database System Concepts - 8<sup>th</sup> Edition

16.7

©Silberschatz, Korth and Sudarshan

So, we will quickly take a look into the anomalies that are that we are talking of first one is called an update anomaly. So, we are showing you a snapshot of an instance of a database which has three attributes and you can look at the row having two entries the last two rows for employee code 519 and there are two different addresses in these two different rows. So, if we know that the employee will have a unique address or in other words if **employee ID → employee address** address then this situation is not possible.

So, but when we try to update then it is for example, the employees address has changed. And while making that change this change will need to be incorporated in all the records having the same ID. And if because of some coding error or something we miss out to

update any of the address fields then we will have a difficulty and that difficulty is having inconsistent address data as in this case.

So, this is known as update anomaly similarly I could have an insertion anomaly which I am illustrating here in terms of another database schema which has four attributes. And we have faculty ID, name, the hiring date and the course name naturally given the faculty ID the faculty name and hire date should be unique. Now suppose a new faculty joins and as soon as the faculty joins he or she may not have an assigned course.

So, if we want to enter that record here we will not be able to do that because we do not have any value for the course code. So, either we use a null value or we cannot actually enter this value; this kind of situation is known as an insertion anomaly. Similarly I could have a deletion anomaly in the same table we are showing that in the table the first highlighted row; the for faculty ID 389 if that faculty stops taking any course for the time being.

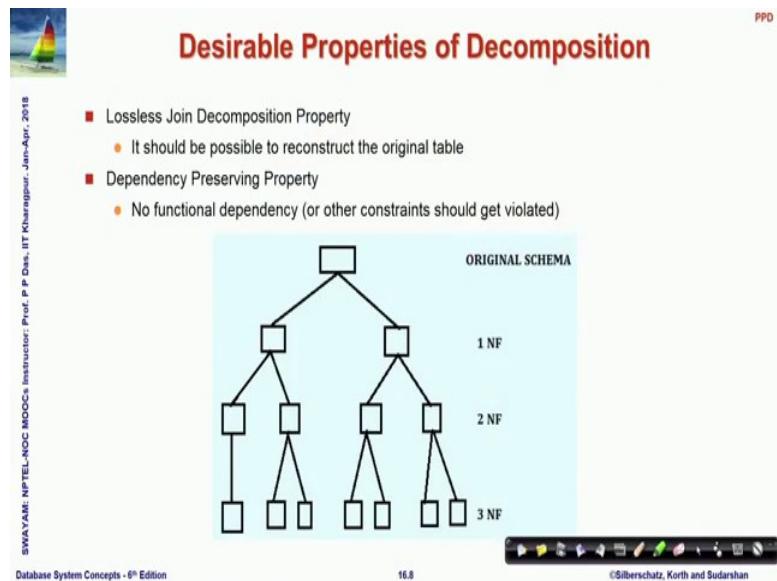
So, the association between 389 and the corresponding course code will be removed and once you remove that you remove this whole record in the process you actually lose the whole of the faculty information the ID, name and hire date. So, these are difficulties in these relational schemas and that lead to a whole lot of problems.

So, the resolution for this lie in terms of decomposing the schema that instead of having one relation, I will decompose this set of attributes into multiple different relations. So, for example, the update anomaly can be removed if we have two different tables; one that maintains ID with address and one that maintains ID with skill. So, in that case what will happen if the for every ID the address will not be repeated.

So, if the address is updated; it will be updated only at one place and it will not feature in the other table. Similarly, to avoid insert or delete anomaly the other table schema can be split into ID, name and hire date as one table and ID and code, rows code as another table. And you can you can easily understand that if this is split in this way then you cannot have an insert anomaly because you can insert a new faculty without assigning a course to him because that will feature in as a separate record in a different table similarly in the same way the deletion anomaly also disappears.

So, these anomalies are resultant of the redundant data that we are having and can be removed by taking care of the process of decomposition.

(Refer Slide Time: 07:03)



Now, when we decompose then we would desire certain properties to be held and we talked about this loosely earlier as well. We would require the lossless join decomposition property that it should be possible to take any instance of the two or more decomposed relations and join them by natural join using common set of attributes and get back the original instance of the relation if that does not happen then the relationship is lossy we have discussed it at length in the last module. At the same time we would want that all functional dependencies that hold must be; can must be testable in the decomposed set of relation.

So, all functional dependencies when they are projected in terms of the decomposed set of relations; they must be testable within them. So, that to test for a dependency I do not need to carry out a join this is a point we discussed in the last module as well. So, based on that once you start with the original schema, you can check for what are the different possibilities or sources of redundancy define constraints based on that and step by step; you could convert a schema into a one normal form have more constraints put onto it convert it into two normal form have further constraints decompose it into third normal form and so, on.

(Refer Slide Time: 08:34)

PPD

Normalization and Normal Forms

- A normal form specifies a set of conditions that the relational schema must satisfy in terms of its constraints – they offer varied levels of guarantee for the design
- Normalization rules are divided into various normal forms. Most common normal forms are:
  - First Normal Form (1 NF)
  - Second Normal Form (2 NF)
  - Third Normal Form (3 NF)
- Informally, a relational database relation is often described as "normalized" if it meets third normal form. Most 3NF relations are free of insertion, update, and deletion anomalies

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

16.9

©Silberschatz, Korth and Sudarshan

So, normalization is a process through which we do this kind of decomposition and make sure that once a relational schema is expressed in terms of a normal form; it satisfies a given set of properties that that normal form should adhere to. And the common normal forms are 1 NF, 2 NF and 3 NF and loosely speaking when we say if a database schema is normalized; we normal usually mean that it is in the 3 NF form a third normal form. And most third number form relations are free of insert, delete or update anomalies. So, that they are a good positive in the design.

(Refer Slide Time: 09:12)

PPD

Normalization and Normal Forms

- Additional Normal Forms
  - Elementary Key Normal Form (EKNF)
  - Boyce-codd Normal Form (BCNF)
  - Multivalued Dependencies And Fourth Normal Form (4 NF)
  - Essential Tuple Normal Form (ETNF)
  - Join Dependencies And Fifth Normal Form (5 NF)
  - Sixth Normal Form (6NF)
  - Domain/Key Normal Form (DKNF)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018



Database System Concepts - 8<sup>th</sup> Edition

16.10

©Silberschatz, Korth and Sudarshan

Of course, these are not the only normal forms as you can see there is a whole lot of lists of variety of normal forms; we will not study all of them we will study further in the next module the other two highlighted ones.

(Refer Slide Time: 09:26)

**First Normal Form (1 NF)**

- A relation is in first Normal Form if and only if all underlying domains contain atomic values only
- In other words, a relation doesn't have multivalued attributes (MVA)
- Example:
  - **STUDENT(Sid, Sname, Cname)**

| Students                       |       |        | Students                 |       |       |
|--------------------------------|-------|--------|--------------------------|-------|-------|
| SID                            | Sname | Cname  | SID                      | Sname | Cname |
| S1                             | A     | C,C++  | S1                       | A     | C     |
| S2                             | B     | C++,DB | S1                       | A     | C++   |
| S3                             | A     | DB     | S2                       | B     | C++   |
| <b>SID : Primary Key</b>       |       |        | S2                       | B     | DB    |
| <b>MVA exists → Not in 1NF</b> |       |        | S3                       | A     | DB    |
|                                |       |        | <b>SID : Primary Key</b> |       |       |

**No MVA → In 1NF**

Source: <http://www.edugrabs.com/normal-forms/#fn>

Database System Concepts - 6<sup>th</sup> Edition      16.11      ©Silberschatz, Korth and Sudarshan

But first let us get started with the first normal form which we had talked about earlier as well; that first normal form is one where the multivalued attributes are not allowed. So, if you think about a think about a relationship where you have a student relationship between student her name and the courses taken by the student then since the students take multiple courses; the C name in this case can take multiple values. So, we do not allow that we expand them into different rows and that once we have done that we say that relation is in the one normal form.

(Refer Slide Time: 10:02)

**First Normal Form (1 NF): Possible Redundancy**

■ Example:

- Supplier(SID, Status, City, PID, Qty)

| Supplier: |        |        |     |     |
|-----------|--------|--------|-----|-----|
| SID       | Status | City   | PID | Qty |
| S1        | 30     | Delhi  | P1  | 100 |
| S1        | 30     | Delhi  | P2  | 125 |
| S1        | 30     | Delhi  | P3  | 200 |
| S1        | 30     | Delhi  | P4  | 130 |
| S2        | 10     | Karnal | P1  | 115 |
| S2        | 10     | Karnal | P2  | 250 |
| S3        | 40     | Rohtak | P1  | 245 |
| S4        | 30     | Delhi  | P4  | 300 |
| S4        | 30     | Delhi  | P5  | 315 |

Key : (SID, PID)

Drawbacks:

- Deletion Anomaly – If we delete the tuple <S3,40,Rohtak,P1,245>, then we lose the information about S3 that S3 lives in Rohtak.
- Insertion Anomaly – We cannot insert a Supplier S5 located in Karnal, until S5 supplies at least one part.
- Update Anomaly – If Supplier S1 moves from Delhi to Kanpur, then it is difficult to update all the tuples containing (S1, Delhi) as SID and City respectively.

Normal Forms are the methods of reducing redundancy. However, sometimes 1 NF increases redundancy. It does not make any efforts in order to decrease redundancy.

Source: <http://www.edugrabs.com/normal-forms/>

Database System Concepts - 8<sup>th</sup> Edition

16.12

©Silberschatz, Korth and Sudarshan

But one normal form may give rise to a variety of different redundancies and therefore, anomalies. So, this is another instance; in fact, the earlier instances that you saw all of them were also in one normal form, but they had deletion insertion and update anomaly. So, here is another example where we are illustrating that.

(Refer Slide Time: 10:26)

**First Normal Form (1 NF): Possible Redundancy**

■ When LHS is not a Superkey :

- Let  $X \rightarrow Y$  is a non trivial FD over R with X is not a superkey of R, then redundancy exist between X and Y attribute set.
- Hence in order to identify the redundancy, we need not to look at the actual data, it can be identified by given functional dependency.
- Example :  $X \rightarrow Y$  and X is not a Candidate Key  
 $\Rightarrow X$  can duplicate  
 $\Rightarrow$  corresponding Y value would duplicate also.

| X | Y |
|---|---|
| 1 | 3 |
| 1 | 3 |
| 2 | 3 |
| 2 | 3 |
| 4 | 6 |

■ When LHS is a Superkey :

- If  $X \rightarrow Y$  is a non trivial FD over R with X is a superkey of R, then redundancy does not exist between X and Y attribute set.
- Example :  $X \rightarrow Y$  and X is a Candidate Key  
 $\Rightarrow X$  cannot duplicate  
 $\Rightarrow$  corresponding Y value may or may not duplicate.

| X | Y |
|---|---|
| 1 | 4 |
| 2 | 6 |
| 3 | 4 |

Source: <http://www.edugrabs.com/normal-forms/#nf>

Database System Concepts - 8<sup>th</sup> Edition

16.13

©Silberschatz, Korth and Sudarshan

So, it is a possible that if I have a functional dependency  $X \rightarrow Y$  which is nontrivial functional dependency over the set of attributes and X is not a super key; then there exists a redundancy between X and Y attribute set. So, on the left we have shown an

instance of this relationship is only on the X and Y attributes and you can see since X is not a key; I can have two rows having the value 1 in X.

And since the value is 1 in X; the value Y will be same for these two rows and we have redundancy of that please all. Please remember that X is not a super key; so, there are other attributes which actually form the super key and therefore, such instances are possible.

Whereas if you look at the right column where the left hand side X is a super key then such instances will not happen.

(Refer Slide Time: 11:22)

The slide has a header 'Second Normal Form (2 NF)' in red. On the left is a small sailboat icon. The right side has a 'PPD' watermark. The main content area contains a bulleted list under a red square icon:

- Relation R is in Second Normal Form (2NF) only iff :
  - R should be in 1NF and
  - R should not contain any *Partial Dependency*

A box labeled 'Partial Dependency:' contains the following text:

Let R be a relational Schema and X, Y, A be the attribute sets over R where  
X: Any Candidate Key, Y: Proper Subset of Candidate Key, and A: Non Key Attribute

If  $Y \rightarrow A$  exists in R, then R is not in 2 NF.

$(Y \rightarrow A)$  is a Partial dependency only if

- Y: Proper subset of Candidate Key
- A: Non Prime Attribute

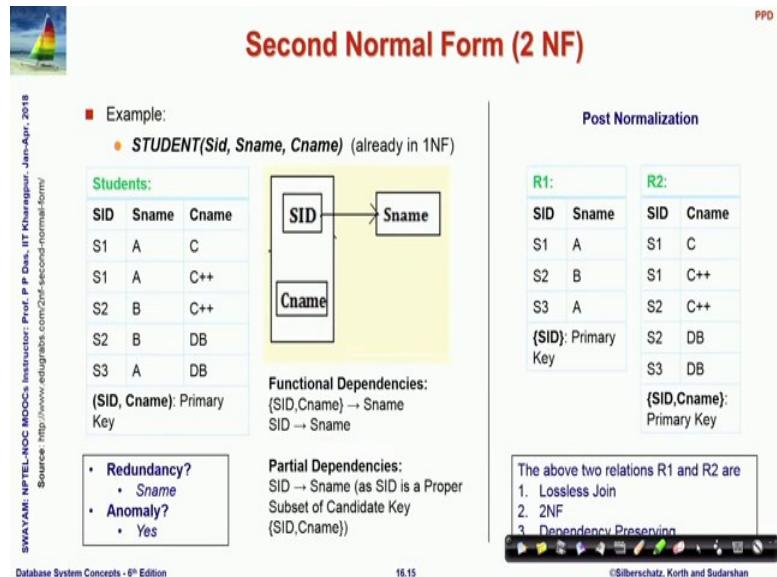
At the bottom, it says 'Source: http://www.edugrabs.com/2nf-second-normal-form' and shows a navigation bar with icons like back, forward, search, etc. The footer includes 'SWAYAM: NPTEL-NOC's Initiative', 'Prof. P P Dass, IIT Kharagpur - Jan-Apr, 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '16.14', and '©Silberschatz, Korth and Sudarshan'.

Moving on the second normal form which is obviously, a relation is in second normal form if it is in first normal form and it does not have any partial dependency. So, what is the partial dependency? I have given the definition here partial dependency why functionally determines A if that can hold in the set of functional dependency then if I have that Y is a proper subset of a candidate key and A is a nonprime attribute in nonprime attribute is one which one nonprime attribute we defined in the last module is an attribute which does not feature in any of the candidate keys.

So, if Y is a proper subset of a candidate key which functionally determines a nonprime attribute; then this is known as a partial dependency and if there is partial dependency

then the relationship is not in second normal form. So, second normal form will require that the relation is in 1 NF and there is no partial dependency.

(Refer Slide Time: 12:25)



So, here I were showing an example where on the left you can see that SID and Cname together forms a key and  $SID \rightarrow Sname$ . So,  $(SID, Cname) \rightarrow Sname$  naturally  $SID \rightarrow Sname$  is a partial dependency because the left hand side  $SID \subseteq$  candidate key  $\{SID, Cname\}$ . And **Sname** is not featuring in any candidate key. So, Sname is actually a nonprime attribute and the result of that as you can see in the first two rows or in the third and fourth row you can see that Sname is repeated.

So, there is redundancy and therefore, consequently we will have anomalies that we have talked of, but we can normalize we can decompose this into two separate relations R1 and R2 as I am showing on the right; where you associate SID and Sname in one table and SID and Cname in other table. Naturally then the dependency that the partial dependency that you had disappears because  $SID \rightarrow Sname$  in R1; now becomes is not a partial dependency because in that table SID becomes a primary key. So, it does not qualify as a partial dependency.

So, R1 and R2 both are in second normal form and you will get rid of the redundancy that you saw and this decomposition ensures that it has a list lossless join incidentally; this is we have not guaranteed that it is in second normal form and it has also the dependency preservation.

(Refer Slide Time: 14:04)

**Second Normal Form (2 NF): Possible Redundancy**

**Example:** Supplier(SID, Status, City, PID, Qty)

| Supplier: |        |        |     |     |
|-----------|--------|--------|-----|-----|
| SID       | Status | City   | PID | Qty |
| S1        | 30     | Delhi  | P1  | 100 |
| S1        | 30     | Delhi  | P2  | 125 |
| S1        | 30     | Delhi  | P3  | 200 |
| S1        | 30     | Delhi  | P4  | 130 |
| S2        | 10     | Kamal  | P1  | 115 |
| S2        | 10     | Kamal  | P2  | 250 |
| S3        | 40     | Rohtak | P1  | 245 |
| S4        | 30     | Delhi  | P4  | 300 |
| S4        | 30     | Delhi  | P5  | 315 |

Key : (SID, PID)

**Partial Dependencies:**  
SID → Status  
SID → City

**Post Normalization**

| Sup_City :          | Sup_Qty :       |
|---------------------|-----------------|
| SID   Status   City | SID   PID   Qty |

**FDD of Sup\_City :**

```

graph TD
    SID --> Status
    SID --> City
    Status --> City

```

**FDD of Sup.Qty :**

```

graph TD
    Qty --> SID
    Qty --> PID

```

**Drawbacks:**

- **Deletion Anomaly** – If we delete a tuple in Sup\_City, then we not only lose the information about a supplier, but also lose the status value of a particular city.
- **Insertion Anomaly** – We cannot insert a City and its status until a supplier supplies at least one part.
- **Updation Anomaly** – If the status value for a city is changed, then we will face the problem of searching every tuple for that city.

Source: <http://www.edugrabs.com/2nf-second-normal-form/>

Database System Concepts - 8<sup>th</sup> Edition

16.16

©Silberschatz, Korth and Sudarshan

But it is possible again in second normal form a relation could be in second normal form yet it could have some possible redundancies. So, there is a design instance that I am showing with the supplier ID, SID the status key which are functionally determined by SID and the product and quantity values.

So, that in the table supplier SID and PID together form say key whereas, and as that happens you can clearly see that there is a lot of redundancy that you can see in terms of the status happening and which will cause you different anomalies to occur. So, if I normalize in the second normal form on the right then I will have a supplier city say with the three attributes SID, status and city and another supplier quantity which has SID, PID and quantity naturally in this there is no partial dependency anymore.

Earlier we had **SID → status** as a partial dependency because SID is a proper was a proper subset of the primary key which is SID CID, but after I normalize this dependency does not exist, but yet there will be redundancy in this relationship and there the status will continue to be redundant.

(Refer Slide Time: 15:30)

**Second Normal Form (2 NF): Possible Redundancy**

In the **Sup\_City** relation :

- City → Status**
- Non Key Attribute → Non Key Attribute

In the **STUDENT** relation:

- SID → Cname**
- Proper Subset of 1 CK → Proper Subset of other CK

Diagram illustrating redundancy:

- (i) Two Non Key Attributes connected by a bidirectional arrow.
- (ii) A Proper Subset of one Candidate Key connected by a bidirectional arrow to a Proper Subset of other Candidate Key.

Source: <http://www.edugrabs.com/2nf-second-normal-form>

Database System Concepts - 8<sup>th</sup> Edition

16.17

©Silberschatz, Korth and Sudarshan

And for that reason we have to move on to the next type of normal form. So, this I am just explaining here as to what are the possible redundancy sources of possible redundancy that you can have in 2 NF.

(Refer Slide Time: 15:43)

**Third Normal Form (3 NF)**

Let  $R$  be the relational schema.

■ [E. F. Codd, 1971]  $R$  is in 3NF only if:

- $R$  should be in 2NF
- $R$  should not contain transitive dependencies (OR, Every non-prime attribute of  $R$  is non-transitively dependent on every key of  $R$ )

■ [Carlo Zaniolo, 1982] Alternately,  $R$  is in 3NF iff for each of its functional dependencies  $X \rightarrow A$ , at least one of the following conditions holds:

- $X$  contains  $A$  (that is,  $A$  is a subset of  $X$ , meaning  $X \rightarrow A$  is trivial functional dependency), or
- $X$  is a superkey, or
- Every element of  $A-X$ , the set difference between  $A$  and  $X$ , is a *prime attribute* (i.e., each attribute in  $A-X$  is contained in some candidate key)

■ [Simple Statement] A relational schema  $R$  is in 3NF if for every FD  $X \rightarrow A$  associated with  $R$  either

- $A \subseteq X$  (i.e., the FD is trivial) or
- $X$  is a superkey of  $R$  or
- $A$  is part of some key (not just superkey!)

Source: <http://www.edugrabs.com/3nf-third-normal-form>

In the 3 NF; third normal form what you define is your relation first of all has to be in 2 NF. So, we are looking at the first definition these are there are three forms of definitions given all of them are actually equivalent, you do not have to worry about why and how they are equivalent slowly you will start understanding.

But we take it in three different forms because each form of the definition allow us to understand certain aspect of the three normal form. So, the first thing which is true for everything is it has to be in the second normal form and it should not contain any transitive dependency which means that I should not if I have  $X \rightarrow Y$  and  $Y \rightarrow Z$ ; then I should not have  $X \rightarrow Z$  which can be inferred transitively as you know through the angstrom axiom.

Alternately, there was an alternate definition given later on by Zanilo and I have stated a simpler simplified version of that at the bottom. So, we will say that a relational schema is in 3 NF if for every functional dependency  $X \rightarrow A$  that holds on this schema either it is a trivial dependency which is  $A$  is a subset of  $X$  or  $X$  is a super key.

So, this is kind of the condition also as you had seen earlier this also is a condition to be in Boyce Codd normal form. So, you can easily understand the 3 NF is a any relation which is in 3 NF is also in the Boyce Codd normal form, but we add a fourth third condition where you say that we will say this is in 3 NF; even if the first two conditions are not satisfied, but  $a$  is a part of some key just note the wording is a part of some key not just the super key..

So, if  $A$  is a part of some key then and the first two conditions are also not are not satisfied even then we will say that the relation is in third normal form. So, to check for a relation to be in third normal form; we will actually check for whether any one of the three conditions hold.

(Refer Slide Time: 18:00)

The slide features a small sailboat icon in the top-left corner. The title 'Third Normal Form (3 NF)' is centered at the top in a red box. The main content consists of several bullet points explaining transitive dependency:

- A **transitive dependency** is a functional dependency which holds by virtue of transitivity. A transitive dependency can occur only in a relation that has three or more attributes.
- Let A, B, and C designate three distinct attributes (or distinct collections of attributes) in the relation. Suppose all three of the following conditions hold:
  - $A \rightarrow B$
  - It is not the case that  $B \rightarrow A$
  - $B \rightarrow C$
- Then the functional dependency  $A \rightarrow C$  (which follows from 1 and 3 by the axiom of transitivity) is a transitive dependency

At the bottom left, vertical text reads: SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018. At the bottom right, it says: Database System Concepts - 8<sup>th</sup> Edition, 16.19, ©Silberschatz, Korth and Sudarshan.

So, this is a definition of transitive dependency which I have just loosely told you. So, I will skip over this.

(Refer Slide Time: 18:09)

The slide features a small sailboat icon in the top-left corner. The title 'Third Normal Form (3 NF)' is centered at the top in a red box. The main content consists of several bullet points explaining transitive dependency, followed by a table of book data:

- Example of **transitive dependency**
- The functional dependency  $\{Book\} \rightarrow \{\text{Author Nationality}\}$  applies; that is, if we know the book, we know the author's nationality. Furthermore:
  - $\{Book\} \rightarrow \{\text{Author}\}$
  - $\{\text{Author}\}$  does not  $\rightarrow \{Book\}$
  - $\{\text{Author}\} \rightarrow \{\text{Author Nationality}\}$
- Therefore  $\{Book\} \rightarrow \{\text{Author Nationality}\}$  is a transitive dependency.
- Transitive dependency occurred because a non-key attribute (**Author**) was determining another non-key attribute (**Author Nationality**).

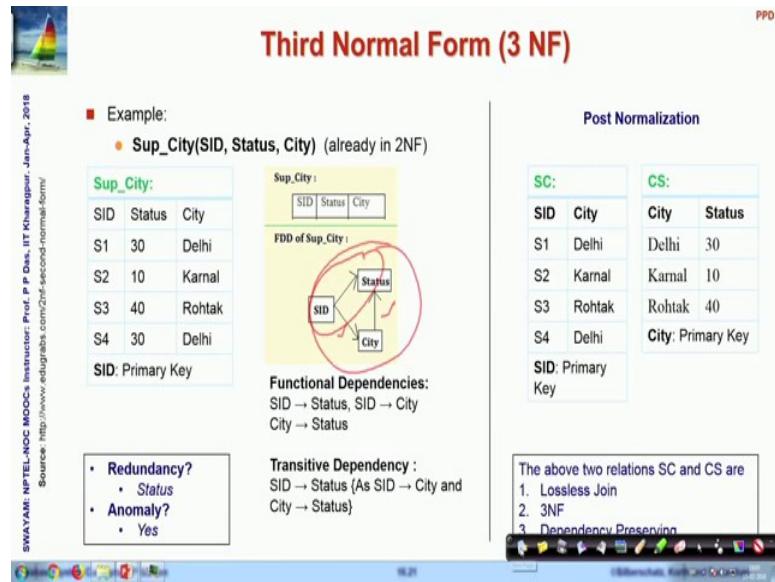
| Book                                  | Genre                   | Author       | Author Nationality |
|---------------------------------------|-------------------------|--------------|--------------------|
| Twenty Thousand Leagues Under the Sea | Science Fiction         | Jules Verne  | French             |
| Journey to the Center of the Earth    | Science Fiction         | Jules Verne  | French             |
| Leaves of Grass                       | Poetry                  | Walt Whitman | American           |
| Anna Karenina                         | Literary Fiction        | Leo Tolstoy  | Russian            |
| A Confession                          | Religious Autobiography | Leo Tolstoy  | Russian            |

At the bottom left, vertical text reads: SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018. At the bottom right, it says: Database System Concepts - 8<sup>th</sup> Edition, 16.20, ©Silberschatz, Korth and Sudarshan.

There is given another example of a very different kind of a relationship book, genre author and author nationality as you can understand. Given the book you know the author there is a functional dependency given the author do you know the author nationality and the, but author does not actually determine the book because the author may have written multiple books. But given that **book → author** and **author → author**

**nationality** we have that **book → author nationality** and therefore, we have redundancy possibility of redundancy in here which is a transitive redundancy due to this transitive dependency that we have.

(Refer Slide Time: 18:48)



So, here is a the earlier example where you can as you can see clearly in this diagram you can if you note this diagram you can see that **SID → city** and **city → status**. So, this is it this is the transitive dependency that **SID → status**.

So, if that happens and status becomes redundant and therefore, there could be anomalies. And we can easily normalize by making them into SID and city and city and status. And in that naturally that that redundancy goes away because you have no more the transitive dependency in the relationship; you only have **SID → the city** which is a primary key in SC and **city → status** which is the primary key in the CS.

(Refer Slide Time: 19:50)

The slide has a header 'Third Normal Form (3 NF)' with a sailboat icon. On the left, vertical text reads 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018'. The main content is a bulleted list:

- Example
  - Relation `dept_advisor`:
- `dept_advisor(s_ID, i_ID, dept_name)`
- $F = \{s\_ID, dept\_name \rightarrow i\_ID, i\_ID \rightarrow dept\_name\}$
- Two candidate keys: `s_ID, dept_name`, and `i_ID, s_ID`
- $R$  is in 3NF
  - $s\_ID, dept\_name \rightarrow i\_ID$ 
    - `s_ID, dept_name` is a superkey
  - $i\_ID \rightarrow dept\_name$ 
    - `dept_name` is contained in a candidate key

A box contains the definition: 'A relational schema  $R$  is in 3NF if for every FD  $X \rightarrow A$  associated with  $R$  either

- $A \subseteq X$  (i.e., the FD is trivial) or
- $X$  is a superkey of  $R$  or
- $A$  is part of some key (not just superkey!)

'.

At the bottom, it says 'Database System Concepts - 8<sup>th</sup> Edition' and '16.22 ©Silberschatz, Korth and Sudarshan'.

So, there are these other examples that you can go through where we have taken the example of a student ID, i\_ID and the department name and shown that what kind of problems, you might get into in this. In this case you can see that the relationship actually is in the third normal form because there are two candidate keys and. So, this s\_ID and department name is a super key and this relationship is in the third normal form. Because **i\_ID → department name** is contained in a candidate key. So, that is the it is in 3 NF due to the third condition that we have had shown.

(Refer Slide Time: 20:41)

The slide has a header 'Redundancy in 3NF' with a sailboat icon. On the left, vertical text reads 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018'. The main content is a bulleted list:

- There is some redundancy in this schema
- Example of problems due to redundancy in 3NF ( $J: s\_ID, L: i\_ID, K: dept\_name$ )
  - $R = (J, L, K)$   
 $F = \{JK \rightarrow L, L \rightarrow K\}$

|       | $J$   | $L$   | $K$ |
|-------|-------|-------|-----|
| $j_1$ | $l_1$ | $k_1$ |     |
| $j_2$ | $l_1$ | $k_1$ |     |
| $j_3$ | $l_1$ | $k_1$ |     |
| null  | $l_2$ | $k_2$ |     |

A bulleted list continues:

- repetition of information (e.g., the relationship  $l_1, k_1$ )
  - $(i\_ID, dept\_name)$
- need to use null values (e.g., to represent the relationship  $l_2, k_2$  where there is no corresponding value for  $J$ ).
  - $(i\_ID, dept\_name)$  if there is no separate relation mapping instructors to departments

At the bottom, it says 'Database System Concepts - 8<sup>th</sup> Edition' and '16.23 ©Silberschatz, Korth and Sudarshan'.

So, when you, but this is a where you can there is some redundancy in this schema that you can observe. So, this is just constructed and you have been because of this redundancy you have been able to we have had to use null values in this case.

(Refer Slide Time: 21:02)



## Third Normal Form (3 NF): Possible Redundancy

PPD

■ A table is automatically in 3NF if one of the following hold :

- (i) If relation consists of two attributes.
- (ii) If 2NF table consists of only one non key attributes

■ If  $X \rightarrow A$  is a dependency, then the table is in the 3NF, if one of the following conditions exists:

- If X is a superkey
- If X is a part of superkey

■ If  $X \rightarrow A$  is a dependency, then the table is said to be NOT in 3NF if the following:

- If X is a proper subset of some key (partial dependency)
- If X is not a proper subset of key (non key)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr., 2018

Source: <http://www.edugrabs.com/2nf-second-normal-form/>



Database System Concepts - 6<sup>th</sup> Edition      16.24      ©Silberschatz, Korth and Sudarshan

So, in a third normal form there is possible redundancy coming in and these are the different cases that we have to check through.

(Refer Slide Time: 21:13)



- Normal Forms
- Decomposition to 3NF
- Decomposition to BCNF

## DECOMPOSITION TO 3NF

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 6<sup>th</sup> Edition      16.25      ©Silberschatz, Korth and Sudarshan



So, next what ; so, we have seen the different normal forms first normal form no multivalued attribute then the second normal form no partial dependency then the third normal form where you do not have any transitive dependency. So, all these are cascading definitions. So, in third normal form you have no multivalued attribute, no partial dependency and no transitive dependency.

So, now what will take a look into is how if I am given a relational schema and if it is violating any one or more of this condition. So, that the schema is not in the three normal form, third normal form then how can we decompose it into the third normal form?

(Refer Slide Time: 21:55)

The slide has a title 'Third Normal Form: Motivation' in red. To the left of the title is a small image of a sailboat on water. On the right side of the slide, there is a vertical sidebar with text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018'. Below the title is a list of bullet points:

- There are some situations where
  - BCNF is not dependency preserving, and
  - Efficient checking for FD violation on updates is important
- Solution: define a weaker normal form, called Third Normal Form (3NF)
  - Allows some redundancy (with resultant problems; as seen above)
  - But functional dependencies can be checked on individual relations without computing a join
  - **There is always a lossless-join, dependency-preserving decomposition into 3NF**

At the bottom of the slide, there is a video player interface showing a thumbnail of a person speaking, the text 'Database System Concepts - 8<sup>th</sup> Edition', the time '16.26', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, the question naturally is certainly is can it always be done is the basic question that can I always decompose a schema into third normal form the answer is yes, you can and that is always a lossless join and dependency preserving decomposition into third normal form which is of great value.

Because that is we said is that desirable properties of our decomposition and if you recall our discussions in the earlier part of the relational design modules, then you would recall that Boyce Codd normal form also we had discussed at the early stages. And that gives you a decomposition which is lossless join, but it does not guarantee preservation of the dependencies where third normal form does that.

(Refer Slide Time: 22:49)

The slide features a small sailboat icon in the top left corner. The title 'Testing for 3NF' is centered at the top in a red font. Below the title is a bulleted list of optimization points:

- Optimization: Need to check only FDs in  $F$ , need not check all FDs in  $F^*$ .
- Use attribute closure to check for each dependency  $\alpha \rightarrow \beta$ , if  $\alpha$  is a superkey.
- If  $\alpha$  is not a superkey, we have to verify if each attribute in  $\beta$  is contained in a candidate key of  $R$ 
  - this test is rather more expensive, since it involves finding candidate keys
  - testing for 3NF has been shown to be NP-hard
- Interestingly, decomposition into third normal form (described shortly) can be done in polynomial time

On the left side of the slide, there is vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018'. At the bottom left is the text 'Database System Concepts - 8<sup>th</sup> Edition'. The bottom right shows a navigation bar with icons and the text '©Silberschatz, Korth and Sudarshan'.

So, naturally there are different algorithms first the question is can you test if a relationship is in third normal form; I will not go into the details of that and the computer science result here is testing for third normal form is an NP hard problem. So, there is no known polynomial time algorithm for that, but the interesting thing is the actually that decomposition can be done in very simply in polynomial time.

(Refer Slide Time: 23:17)

The slide features a small sailboat icon in the top left corner. The title '3NF Decomposition Algorithm' is centered at the top in a red font. Below the title is a bulleted list of steps:

- Given: relation  $R$ , set  $F$  of functional dependencies
- Find: decomposition of  $R$  into a set of 3NF relations  $R_i$
- Algorithm:
  - Eliminate redundant FDs, resulting in a canonical cover  $F_c$  of  $F$
  - Create a relation  $R_i = XY$  for each FD  $X \rightarrow Y$  in  $F_c$
  - If the key  $K$  of  $R$  does not occur in any relation  $R_i$ , create one more relation  $R_i = K$

On the left side of the slide, there is vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018'. At the bottom left is the text 'Database System Concepts - 8<sup>th</sup> Edition'. The bottom right shows a navigation bar with icons and the text '©Silberschatz, Korth and Sudarshan'.

So, what do you have what is the decomposition algorithm very written in very simple terms you want to you have given a relation  $R$  and a set of functional dependencies that

hold on you. So, you first compute a canonical cover you know what is a canonical cover. So, you compute a canonical cover you eliminate extraneous attributes eliminate redundant FDs and you have the canonical cover  $F_c$  from  $F$  then you create for every functional dependency  $X \rightarrow Y$  that exists in the canonical cover.

You compute you make a relation say the  $i^{\text{th}}$  relation taking  $X \cup Y$ . So, you call it the relation  $XY$  and you do that for all the functional dependencies in the cover. And after that if you find that the key does not occur in any one of these decomposed relations as generated, then you generate one separate relation to represent the key.

(Refer Slide Time: 24:19)

**3NF Decomposition Algorithm (Formal)**

```

Let  $F_c$  be a canonical cover for  $F$ ;
 $i := 0$ ;
for each functional dependency  $\alpha \rightarrow \beta$  in  $F_c$  do
    if none of the schemas  $R_j$ ,  $1 \leq j \leq i$  contains  $\alpha \beta$ 
        then begin
             $i := i + 1$ ;
             $R_i := \alpha \beta$ 
        end
    if none of the schemas  $R_j$ ,  $1 \leq j \leq i$  contains a candidate key for  $R$ 
        then begin
             $i := i + 1$ ;
             $R_i :=$  any candidate key for  $R$ ;
        end
    /* Optionally, remove redundant relations */
repeat
    if any schema  $R_j$  is contained in another schema  $R_k$ 
        then /* delete  $R_j$  */
             $R_j = R_k$ ;
             $i := i - 1$ ;
return  $(R_1, R_2, \dots, R_i)$ 

```

SVAYAM: NPTEL NOC Instructor: Prof. P. P. Deshpande, IIT Kanpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

16.29

©Silberschatz, Korth and Sudarshan

That is a very simple algorithm and I just wrote it in simple hand. So, that you can understand it easily, but here is the formal algorithm. So, if you are interested to rigor I mean in the rigor of how 3 NF decomposition will happen here is the algorithm, but I will not go through these in steps.

(Refer Slide Time: 24:37)

The slide has a title '3NF Decomposition Algorithm' at the top right. On the left, there is a small sailboat icon. Below the title, there is a bulleted list of points:

- Above algorithm ensures:
  - Each relation schema  $R_i$  is in 3NF
  - Decomposition is d
    - Dependency preserving and
    - Lossless-join

On the left side of the slide, there is vertical text: 'SWAYAM: NPTEL NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr- 2018'. At the bottom left, it says 'Database System Concepts - 8<sup>th</sup> Edition'. In the bottom right corner, it says '©Silberschatz, Korth and Sudarshan'.

So, that ensures that each relation  $R_i$  that I have decomposed and generated is actually in third normal form and this decomposition is dependency preserving and is lossless join we are not proving that but we are just using that result.

(Refer Slide Time: 24:54)

The slide has a title 'Example of 3NF Decomposition' at the top right. On the left, there is a small sailboat icon. Below the title, there is a bulleted list of points:

- Relation schema:  
 $cust\_banker\_branch = (customer\_id, employee\_id, branch\_name, type)$

Below the list, there is a large screenshot of a computer screen showing a database diagram or code editor. The code appears to be a SQL query or a series of statements related to the 'cust\_banker\_branch' schema. The text is mostly illegible due to the high contrast between the black background and the white text.

So, here is an example of a schema; so, we have a `customer_banker_branch`. So, these are the four attributes and these are the different functional dependencies that exist. Now naturally given this first thing you will have to do is first thing you have to do is to look at the different to look at taking the canonical cover the minimal cover.

So, if you compute try to compute the minimal cover; you will find that branch name actually is extraneous in the first dependency. So, you can remove that and there is nothing else.

(Refer Slide Time: 25:36)

**Example of 3NF Decomposition**

- The **for** loop generates following 3NF schema:  
 $(customer\_id, employee\_id, type)$   
 $(employee\_id, branch\_name)$   
 $(customer\_id, branch\_name, employee\_id)$ 
  - Observe that  $(customer\_id, employee\_id, type)$  contains a candidate key of the original schema, so no further relation schema needs be added
- At end of for loop, detect and delete schemas, such as  $(employee\_id, branch\_name)$ , which are subsets of other schemas
  - result will not depend on the order in which FDs are considered
- The resultant simplified 3NF schema is:  
 $(customer\_id, employee\_id, type)$   
 $(customer\_id, branch\_name, employee\_id)$

SWAYAM: NPTEL-NOC/MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition      16.32      ©Silberschatz, Korth and Sudarshan

So, your canonical cover turns out to be this set of dependencies and then you go over and for each one of them. So, you take each one the first one is **(customer ID, employee ID) → type**. So, for that you generate a schema customer ID, employee ID and type again you take the second functional dependency **employee ID → branch name**. So, create employee ID and branch name as a different schema and in this way you will generate three decomposed schema in the third normal form.

Now, once you have done that then you find that your if you look into the original key it was customer ID and employee ID and you find that here in the third; second and the third you already have that. So, you do not need to add a separate relation for accommodating the key and also the third relation. So, we can now declare that no further key needs to be added and we have the final 3 NF decomposition..

So, at the end of the fault detect and delete. So, this is this is a stated in terms of the detailed algorithm, but this is you can say that the employee ID and branch name the second relation in the decomposition is actually a subset of the third relation. So, you can remove that as well. So, you will be left with only two relations in this decompose

schema which both of which are in third normal form and this decomposition is guaranteed you lossless join and dependency preservation.

(Refer Slide Time: 27:17)

**Practice Problem for 3NF Decomposition: 1**

- $R = ABCDEFGH$
- FDs = { $A \rightarrow B$ ,  $ABCD \rightarrow E$ ,  $EF \rightarrow GH$ ,  $ACDF \rightarrow EG$ }

Solution is given in the next slide (hidden from presentation – check after you have solved)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr- 2018

Database System Concepts - 8<sup>th</sup> Edition      16.33      ©Silberschatz, Korth and Sudarshan

So, I have given some practice problems for you I have also given the solution, but the solution is not in the current run of the presentation; you will get see them in the presentation as hidden slides. So, you first try solving them and once you have solved them then you look at the solution in the slide.

(Refer Slide Time: 27:37)

**Practice Problem for 3NF Decomposition: 2**

- $R = CSJDPQV$
- FDs = { $C \rightarrow CSJDPQV$ ,  $SD \rightarrow P$ ,  $JP \rightarrow C$ ,  $J \rightarrow S$ }

Solution is given in the next slide (hidden from presentation – check after you have solved)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr- 2018

Database System Concepts - 8<sup>th</sup> Edition      16.35      ©Silberschatz, Korth and Sudarshan

So, there are two problems; so, this is a second one and you can solve them in that way.

(Refer Slide Time: 27:40)

PPD

- Normal Forms
- Decomposition to 3NF
- Decomposition to BCNF

## DECOMPOSITION TO BCNF

SWAYAM: NPTEL NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

16.37

©Silberschatz, Korth and Sudarshan

Next is the we will quickly recap on the decomposition of BCNF Boyce Codd normal form which we had seen earlier.

(Refer Slide Time: 27:49)

## Testing for BCNF

- To check if a non-trivial dependency  $\alpha \rightarrow \beta$  causes a violation of BCNF
  1. compute  $\alpha^+$  (the attribute closure of  $\alpha$ ), and
  2. verify that it includes all attributes of  $R$ , that is, it is a superkey of  $R$ .
- **Simplified test:** To check if a relation schema  $R$  is in BCNF, it suffices to check only the dependencies in the given set  $F$  for violation of BCNF, rather than checking all dependencies in  $F^+$ .
  - If none of the dependencies in  $F$  causes a violation of BCNF, then none of the dependencies in  $F^+$  will cause a violation of BCNF either.
- However, **simplified test using only  $F$  is incorrect when testing a relation in a decomposition of  $R$** 
  - Consider  $R = (A, B, C, D, E)$ , with  $F = \{ A \rightarrow B, BC \rightarrow D \}$ 
    - Decompose  $R$  into  $R_1 = (A, B)$  and  $R_2 = (A, C, D, E)$
    - Neither of the dependencies in  $F$  contain only attributes from  $(A, C, D, E)$  so we might be misled into thinking  $R_2$  satisfies BCNF.
    - In fact, dependency  $AC \rightarrow D$  in  $F^+$  shows  $R_2$  is not in BCNF.

SWAYAM: NPTEL NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr. 2018

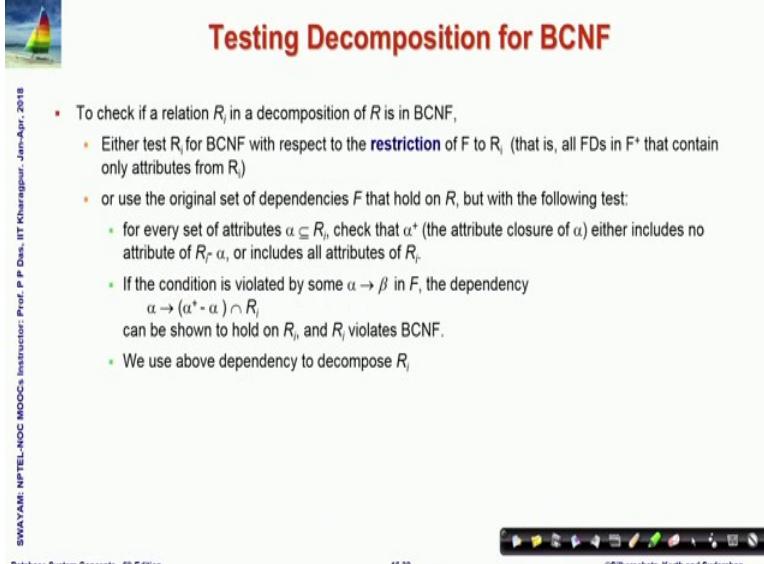
Database System Concepts - 8<sup>th</sup> Edition

16.38

©Silberschatz, Korth and Sudarshan

And we know that the Boyce Codd normal form guarantees that there will have be every dependency that exists must be either trivial or the left hand side must be a super key. So, using the algorithms, you can test for the Boyce Codd normal form which is described here I am not going through in steps.

(Refer Slide Time: 28:08)



The slide title is "Testing Decomposition for BCNF". It features a small sailboat icon in the top left corner. On the right side, there is a decorative footer bar with various icons. The slide content is as follows:

To check if a relation  $R_i$  in a decomposition of  $R$  is in BCNF,

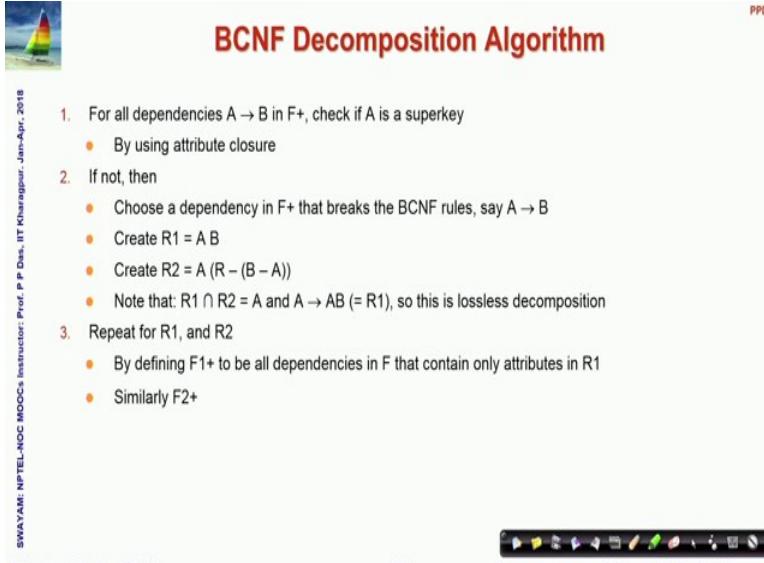
- Either test  $R_i$  for BCNF with respect to the **restriction** of  $F$  to  $R_i$  (that is, all FDs in  $F^+$  that contain only attributes from  $R_i$ )
- or use the original set of dependencies  $F$  that hold on  $R$ , but with the following test:
  - for every set of attributes  $\alpha \subseteq R_i$ , check that  $\alpha^+$  (the attribute closure of  $\alpha$ ) either includes no attribute of  $R_i - \alpha$ , or includes all attributes of  $R_i$ .
  - If the condition is violated by some  $\alpha \rightarrow \beta$  in  $F$ , the dependency  $\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$  can be shown to hold on  $R_i$ , and  $R_i$  violates BCNF.
- We use above dependency to decompose  $R_i$ .

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Desai, IIT Kharagpur - Jan-Apr- 2018

Database System Concepts - 8<sup>th</sup> Edition 16.39 ©Silberschatz, Korth and Sudarshan

And here is the more detailed formal algorithm to find determine whether a Boyce Codd normal form is in a decomposed form is in Boyce Codd.

(Refer Slide Time: 28:18)



The slide title is "BCNF Decomposition Algorithm". It features a small sailboat icon in the top left corner. On the right side, there is a decorative footer bar with various icons. The slide content is as follows:

- For all dependencies  $A \rightarrow B$  in  $F^+$ , check if  $A$  is a superkey
  - By using attribute closure
- If not, then
  - Choose a dependency in  $F^+$  that breaks the BCNF rules, say  $A \rightarrow B$
  - Create  $R_1 = A B$
  - Create  $R_2 = A (R - (B - A))$
  - Note that:  $R_1 \cap R_2 = A$  and  $A \rightarrow AB (= R_1)$ , so this is lossless decomposition
- Repeat for  $R_1$ , and  $R_2$ 
  - By defining  $F_{1+}$  to be all dependencies in  $F$  that contain only attributes in  $R_1$
  - Similarly  $F_{2+}$

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Desai, IIT Kharagpur - Jan-Apr- 2018

Database System Concepts - 8<sup>th</sup> Edition 16.40 ©Silberschatz, Korth and Sudarshan

So, I will just quickly recap on the algorithm to do that naturally for all dependencies you first determine the super key and check if  $A \rightarrow B$  is a super key or not if it and that you can easily do using attribute cover. If it is not a super key then you choose a dependency  $A \rightarrow B$  which violates and you form by Boyce Codd goes in every step it decomposes one relation into two separate relation.

So, one that you take by taking U of the attributes of A and B and the other where you take out B minus A; these attributes this difference attributes you take out from R and then you add A and make the other relationship. Naturally in between these two A is a common attribute and since and that will determine A B because  $A \rightarrow B$ .

So,  $A \rightarrow A B$  that is whole of R1. So, naturally the lossless join is guaranteed and you repeat that keep on doing that for the resultant relations that you have got. keep on decomposing them till you finally, close and you have no more violating dependency and you will have a decomposition into Boyce Codd normal form.

(Refer Slide Time: 29:39)

**BCNF Decomposition Algorithm**

```

result := {R};
done := false;
compute F+;
while (not done) do
    if (there is a schema Rj in result that is not in BCNF)
        then begin
            let α → β be a nontrivial functional dependency that
            holds on Rj such that α → Rj is not in F+,
            and α ∩ β = ∅;
            result := (result - Rj) ∪ (Rj - β) ∪ (α, β);
            end
        else done := true;

```

Note: each  $R_j$  is in BCNF, and decomposition is lossless-join.

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Desai, IIT Kanpur - Jan-Apr-2018  
Database System Concepts - 8<sup>th</sup> Edition  
16.41 ©Silberschatz, Korth and Sudarshan

Here is the formal algorithm again for you to go by steps if you are interested.

(Refer Slide Time: 29:46)

The slide features a sailboat icon in the top left corner. The title 'Example of BCNF Decomposition' is centered at the top in red. To the right of the title is a list of bullet points. On the left side, there is vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr- 2018'. In the bottom left corner, there is a small video thumbnail showing a man speaking. The bottom right corner contains the copyright notice '©Silberschatz, Korth and Sudarshan'.

- $R = (A, B, C)$   
 $F = \{A \rightarrow B$   
 $B \rightarrow C\}$   
Key = {A}
- $R$  is not in BCNF ( $B \rightarrow C$  but  $B$  is not superkey)
- Decomposition
  - $R_1 = (B, C)$
  - $R_2 = (A, B)$

Otherwise you know how to do this; again I have shown another example here which is showing that how to decompose in BCNF. So, you should practice this that is why I have work them out in steps here. So, here  $A \rightarrow B$ ;  $B \rightarrow C$  naturally A is the key; R is not in BCNF because  $B \rightarrow C$  is a functional dependency where B is not a super key.

(Refer Slide Time: 30:15)

The slide features a sailboat icon in the top left corner. The title 'Example of BCNF Decomposition' is centered at the top in red. To the right of the title is a list of bullet points. On the left side, there is vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr- 2018'. In the bottom left corner, there is a small video thumbnail showing a man speaking. The bottom right corner contains the copyright notice '©Silberschatz, Korth and Sudarshan'.

- class (course\_id, title, dept\_name, credits, sec\_id, semester, year, building, room\_number, capacity, time\_slot\_id)
- Functional dependencies:
  - course\_id  $\rightarrow$  title, dept\_name, credits
  - building, room\_number  $\rightarrow$  capacity
  - course\_id, sec\_id, semester, year  $\rightarrow$  building, room\_number, time\_slot\_id
- A candidate key {course\_id, sec\_id, semester, year}.
- BCNF Decomposition:
  - course\_id  $\rightarrow$  title, dept\_name, credits holds
    - but course\_id is not a superkey.
  - We replace class by:
    - course(course\_id, title, dept\_name, credits)
    - class-1 (course\_id, sec\_id, semester, year, building, room\_number, capacity, time\_slot\_id)

So, you can decompose them in terms of. So, you can decompose in terms of BC as one relation and AB as another relation. Here is another example a more detailed one of a class relationship which has a whole set of attributes and these functional dependencies

and based on that the candidate key is course ID, section ID, semester and year and you can proceed with the BCNF decomposition; taking the first functional dependency that holds, but the left hand side the course ID is not a super key. So, you will replace it by a one relation; which is say new course relation and a new class relation which is the remaining attributes.

(Refer Slide Time: 31:06)

The slide has a title 'BCNF Decomposition (Cont.)' in red at the top right. On the left is a small logo of a sailboat. The main content is a bulleted list of points:

- course is in BCNF
  - How do we know this?
- building, room\_number → capacity holds on class-1(course\_id, sec\_id, semester, year, building, room\_number, capacity, time\_slot\_id)
  - but {building, room\_number} is not a superkey for class-1.
  - We replace class-1 by:
    - classroom (building, room\_number, capacity)
    - section (course\_id, sec\_id, semester, year, building, room\_number, time\_slot\_id)
- classroom and section are in BCNF.

At the bottom left is a small video thumbnail showing a person speaking. The footer contains text: 'SWAYAM/NPTEL/NOC Instructor: Prof. P P Desai, IIT Kanpur', 'Database System Concepts - 8<sup>th</sup> Edition', '16.44', and '©Silberschatz, Korth and Sudarshan'.

And then you get convinced that course is in BCNF, but the other one the class is not because **(building,room number)→ capacity** where building room number together is not a super key. So, you split it again and you replace class 1 in terms of two new relations class room and section and both of them are in BCNF and you are done with this.

(Refer Slide Time: 31:31)

The slide has a red header 'BCNF and Dependency Preservation'. On the left, there is a vertical sidebar with the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018'. The main content area contains the following text:

- It is not always possible to get a BCNF decomposition that is dependency preserving
- $R = (J, K, L)$   
 $F = \{JK \rightarrow L$   
 $L \rightarrow K\}$   
Two candidate keys = JK and JL
- $R$  is not in BCNF
- Any decomposition of  $R$  will fail to preserve  
 $JK \rightarrow L$   
This implies that testing for  $JK \rightarrow L$  requires a join

At the bottom, there is footer text: 'Database System Concepts - 6<sup>th</sup> Edition', '16.45', and '©Silberschatz, Korth and Sudarshan'.

But BCNF as I would again warning you BCNF does not preserve dependence it gives you lossless join, but it does not preserve the dependencies. So, it is not always possible to decompose into BCNF with dependency preservation. So, here is an example which we saw little earlier and there are two candidate keys R is not in BCNF, you can clearly see and any decomposition will fail **JK → L** and that will require a join. So, this will not preserve the dependencies in terms of the decomposition.

(Refer Slide Time: 32:06)

The slide has a red header 'Practice Problem for BCNF Decomposition'. On the left, there is a vertical sidebar with the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018'. The main content area contains the following text:

- $R = ABCDE, F = \{A \rightarrow B, BC \rightarrow D\}$
- $R = ABCDE, F = \{A \rightarrow B, BC \rightarrow D\}$
- $R = ABCDEH, F = \{A \rightarrow BC, E \rightarrow HA\}$
- $R = CSJDPQV, F = \{C \rightarrow CSJDPQV, SD \rightarrow P, JP \rightarrow C, J \rightarrow S\}$
- $R = ABCD, F = \{C \rightarrow D, C \rightarrow A, B \rightarrow C\}$

At the bottom, there is footer text: 'Database System Concepts - 6<sup>th</sup> Edition', '16.46', and '©Silberschatz, Korth and Sudarshan'.

Again, I have given a set of practice problems here which we you should try and get confident in terms of the Boyce Codd from normal form normalization.

(Refer Slide Time: 32:19)

## Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF such that:
  - the decomposition is lossless
  - the dependencies are preserved
- It is always possible to decompose a relation into a set of relations that are in BCNF such that:
  - the decomposition is lossless
  - it may not be possible to preserve dependencies.

| S# | 3NF                                                                                                  | BCNF                                                            |
|----|------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------|
| 1. | It concentrates on Primary Key                                                                       | It concentrates on Candidate Key.                               |
| 2. | Redundancy is high as compared to BCNF                                                               | 0% redundancy                                                   |
| 3. | It may preserve all the dependencies                                                                 | It may not preserve the dependencies.                           |
| 4. | A dependency $X \rightarrow Y$ is allowed in 3NF if $X$ is a super key or $Y$ is a part of some key. | A dependency $X \rightarrow Y$ is allowed if $X$ is a super key |

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018  
 Database System Concepts - 8th Edition  
 16.47  
 ©Silberschatz, Korth and Sudarshan

Now, it is always possible to decompose a relation into a set of relation in 3 NF; if the decomposition is lossless and the dependencies are preserved. Whereas, in case of BCNF it is not possible; so, here is a table which summarizes the relative comparison between Boyce Codd and third normal form because they are the common once Boyce Codd naturally is more strict it gives you lesser dependent lesser redundancies, but it cannot guarantee that your dependencies will be preserved. So, more often we will accept 3 NF as an acceptable normalized decomposition with some redundancy still existing it is possible and we cannot get rid of them.

(Refer Slide Time: 33:09)

**Module Summary**

- Studied the Normal Forms and their Importance in Relational Design – how progressive increase of constraints can minimize redundancy in a schema
- Learnt how to decompose a schema into 3NF while preserving dependency and lossless join
- Learnt how to decompose a schema into BCNF with lossless join

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: 2018

Database System Concepts - 8<sup>th</sup> Edition      16.48      ©Silberschatz, Korth and Sudarshan

So, we have studied about the normal forms and their importance and how progressively we can increase the constraints to minimize redundancy in the schema and learned how to decompose a schema into third normal form and also in the Boyce Codd normal form.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 20**  
**Relational Database Design (Contd.)**

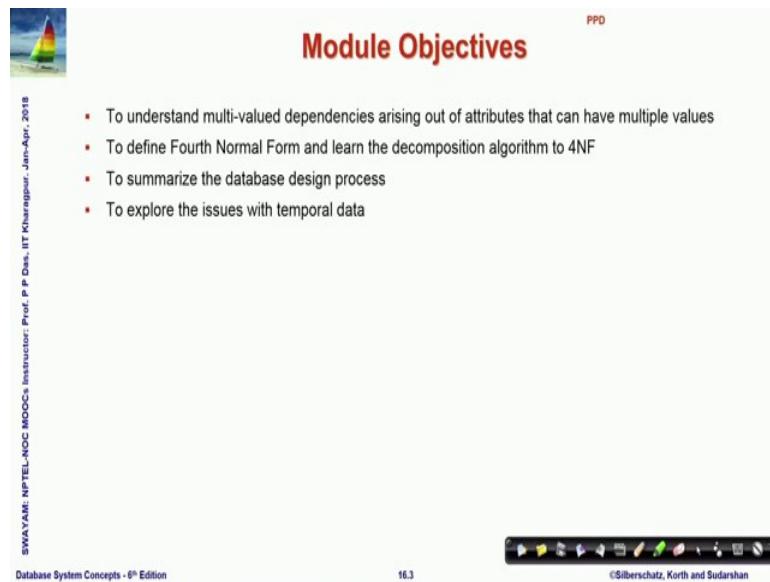
Welcome to module 20 of Database Management Systems. We have been discussing about relational database design, since the last 4 modules and this will be the concluding part of relational database design.

(Refer Slide Time: 00:33)

The slide is titled "Module Recap" in red at the top right. It features a small sailboat icon in the top left corner. On the far left, there is vertical text: "SWAYAM: NPTEL-NOCO's MOOCs", "Instructor: Prof. P. P. Das", "Module: Database System Concepts - 8<sup>th</sup> Edition", and "Date: Jan-Apr., 2018". In the top right corner, it says "PPD". At the bottom right, it says "©Silberschatz, Korth and Sudarshan". The main content area contains a bulleted list of three items: "■ Normal Forms", "■ Decomposition to 3NF", and "■ Decomposition to BCNF". A navigation bar with various icons is at the very bottom.

In the last module, we have seen some very key concepts of relational design, that of normal forms Third and Boyce Codd normal form specifically and how to decompose into them? And how do we get benefit in terms of doing this kind of decomposition? In removing the anomalies by reducing the redundancy in the design.

(Refer Slide Time: 00:59)

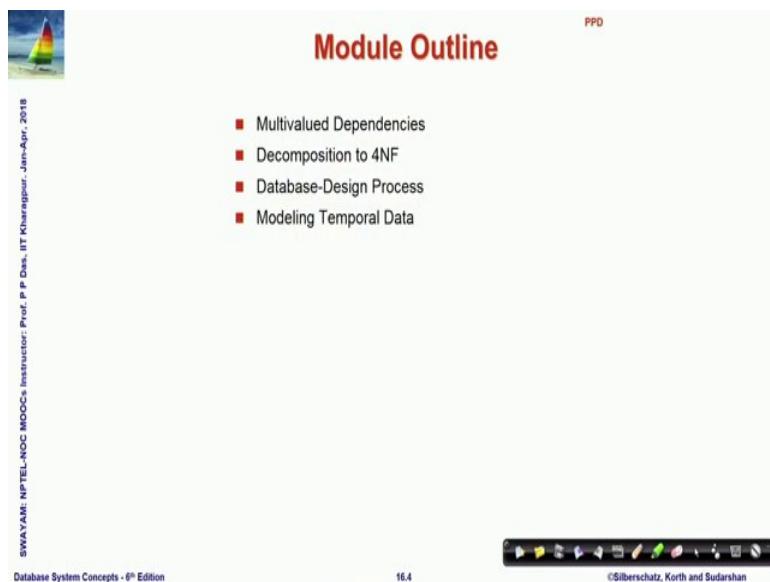


This slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. The background is white with a thin grey border. On the left side, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018". At the bottom left is the text "Database System Concepts - 8<sup>th</sup> Edition". In the bottom right corner is the copyright notice "©Silberschatz, Korth and Sudarshan". A navigation bar with various icons is located at the bottom.

- To understand multi-valued dependencies arising out of attributes that can have multiple values
- To define Fourth Normal Form and learn the decomposition algorithm to 4NF
- To summarize the database design process
- To explore the issues with temporal data

In view of that in the background of that, in this module will we would try to understand a new kind of data dependency, an additional kind of data dependency, which is called multivalued dependency. Which can occur when an attribute can have can take multiple possible values, which we had eliminated in the first normal form together and based on that, we will define 4th normal form and decomposition into 4 NF and then we will summarize this whole set of discussions of relational database design, and talk little bit about what happens, when you have temporal data in your system.

(Refer Slide Time: 01:39)



This slide is titled "Module Outline" in red at the top right. It features a small sailboat icon in the top left corner. The background is white with a thin grey border. On the left side, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018". At the bottom left is the text "Database System Concepts - 8<sup>th</sup> Edition". In the bottom right corner is the copyright notice "©Silberschatz, Korth and Sudarshan". A navigation bar with various icons is located at the bottom.

- Multivalued Dependencies
- Decomposition to 4NF
- Database-Design Process
- Modeling Temporal Data

(Refer Slide Time: 01:47)

**Multivalued Dependency**

PPD

■ Persons(Man, Phones, Dog\_Like)

| Person :  |           |              | Meaning of the tuples                                   |
|-----------|-----------|--------------|---------------------------------------------------------|
| Man(M)    | Phones(P) | Dogs_Like(D) | Man M have phones P, and likes the dogs D.              |
| M1        | P1/P2     | D1/D2        | M1 have phones P1 and P2, and likes the dogs D1 and D2. |
| M2        | P3        | D2           | M2 have phones P3, and likes the dog D2.                |
| Key : MPD |           |              |                                                         |

There are no non trivial FDs because all attributes are combined forming Candidate Key i.e. MDP. In the above relation, two multivalued dependencies exists –

- Man  $\rightarrow\!\!\!\rightarrow$  Phones
- Man  $\rightarrow\!\!\!\rightarrow$  Dogs\_Like

A man's phone are independent of the dogs they like. But after converting the above relation in Single Valued Attribute, each of a man's phones appears with each of the dogs they like in all combinations.

Source: <http://www.edugrabs.com/multivalued-dependency-mvd/>

Post 1NF Normalization

| Man(M) | Phones(P) | Dogs_Likes(D) |
|--------|-----------|---------------|
| M1     | P1        | D1            |
| M1     | P2        | D2            |
| M2     | P3        | D2            |
| M1     | P1        | D2            |
| M1     | P2        | D1            |

SWAYAM-NPTEL-NOOC-MOOCs; Instructor: Prof. P. P. Deshpande, IIT Kharagpur

Database System Concepts - 8<sup>th</sup> Edition

16.6

©Silberschatz, Korth and Sudarshan

So, these are the outline points, based on our objectives and we start with the discussion of multivalued dependency. Consider a situation like this. So, here we are trying to represent an individual instead of persons with three attributes, man which is an may be id or name of that person, phones and dog like. So, the idea is that the here persons and they can have one, two, three any number of phones, which is true for all of us and then a person may have any number of dogs that he or she likes.

So, both of these phones and dog like D, P and D attributes can take multiple values and ah. So, if we if we if you look at the 1 NF normalized form here. So, in 1 NF what we do? We create separate rows for them. So, we have created separate rows for M 1 against P 1 and P 1 or P 2 in phones and similarly for D 1 and D 2. So, once we have done that then we have here, we can see I have highlighted with yellow, you can see the different redundancies that are arising.

Because, since I have phones and dog liking attributes. So, it is possible that if phone takes 2 values and dog like takes 2 values, then actually 4 different combinations of them are possible. But in reality, it may be in reality it may be actually these 2 are true, that M 1 has phone P 1 and likes dog D 1 M 1 has phone P 2 and likes dog D 2, but you could also have such redundant tuples coming in, because there they are now valid.

So, this is the situation which we try to try to capture, in terms of what you see here multivalued dependencies, where which is row shown in terms of double arrows as you

can see here. So, **man** →→ **phones**, **man** →→ **dog likes**. So, there are two different multi valued dependencies in this case. So, this multi valued dependency adds a new source of redundancy in our data, and that is very real in various models of our system.

(Refer Slide Time: 04:19)

**Multivalued Dependency**

If two or more independent relations are kept in a single relation, then Multivalued Dependency is possible. For example, Let there are two relations :

- *Student(SID, Sname) where (SID → Sname)*
- *Course(CID, Cname) where (CID → Cname)*

There is no relation defined between Student and Course. If we kept them in a single relation named *Student\_Course*, then MVD will exists because of *m:n Cardinality*

If two or more MVDs exist in a relation, then while converting into SVAs, MVD exists.

| Student: |       | Course: |       | SID | Sname | CID | Cname |
|----------|-------|---------|-------|-----|-------|-----|-------|
| SID      | Sname | CID     | Cname | S1  | A     | C1  | C     |
| S1       | A     | C1      | C     | S1  | A     | C2  | B     |
| S2       | B     | C2      | B     | S2  | B     | C1  | C     |
|          |       |         |       | S2  | B     | C2  | B     |

2 MVDs exist:  
 1. SID →→ CID  
 2. SID →→ Cname

Source: <http://www.edugrabs.com/multivalued-dependency-mvd/>

Database System Concepts - 8<sup>th</sup> Edition

16.7

©Silberschatz, Korth and Sudarshan

So, let us move on. So, this is just another example, we have two different relations Student give the student id and name, Courses giving course id and name and the corresponding functional dependencies in them, you can see two instances of that. But if we as such, there is no relationship between student and course, but if we choose to keep them in a single relation, say *Student\_Course* which I have shown on bottom right here. Then there will be you can see lot of redundancies coming in, because since I they can be in terms of all different combinations. So, S 1 has in name A may be taking course C 1 having name C, but again the S 1 having name A could be taking course C 2 having name B, you just do not know which one is correct.

So, you can see that here again you have 2 multiple value dependencies, one where **SID** →→ **CID** and **SID** →→ **Cname**. So, these are the two different multiple values that you can, find against the SID and this is ah. So, if two or more multi valued dependencies exist in a relation, then while we convert the we convert multivalued attributes into single valued attributes, then the multi value dependency will show up. So, that is the basic problem that we would like to address.

(Refer Slide Time: 05:59)

**Multivalued Dependencies**

- Suppose we record names of children, and phone numbers for instructors:
  - *inst\_child(ID, child\_name)*
  - *inst\_phone(ID, phone\_number)*
- If we were to combine these schemas to get
  - *inst\_info(ID, child\_name, phone\_number)*
- Example data:
  - (99999, David, 512-555-1234)
  - (99999, David, 512-555-4321)
  - (99999, William, 512-555-1234)
  - (99999, William, 512-555-4321)
- This relation is in BCNF
  - Why?

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kanpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

16.8

©Silberschatz, Korth and Sudarshan

This is another example of two relations, where the ID and child are together, when ID and phone\_number are together. So, naturally if I combine them into a single relation, you have a set of possibilities of multiple different tuples. Because given an ID there could be multiple children, there given an id there could be multiple phone numbers. Mind you, this relation of isn't info is still in Boyce Codd normal form, because there is no dependence there is no functional dependency that holds on this relation. So, the key of this relation is the union of all the three attributes and therefore, that being the key and no functional dependency holding on it, naturally vacuously makes it Boyce Codd normal form, but you can still see that there are redundancy in that is data.

(Refer Slide Time: 06:48)

**Multivalued Dependencies (MVDs)**

PPD

Let  $R$  be a relation schema and let  $\alpha \subseteq R$  and  $\beta \subseteq R$ . The **multivalued dependency**  $\alpha \rightarrow\!\!\! \rightarrow \beta$  holds on  $R$  if in any legal relation  $r(R)$ , for all pairs for tuples  $t_1$  and  $t_2$  in  $r$  such that  $t_1[\alpha] = t_2[\alpha]$ , there exist tuples  $t_3$  and  $t_4$  in  $r$  such that:

$$\begin{aligned} t_1[\alpha] &= t_2[\alpha] = t_3[\alpha] = t_4[\alpha] \\ t_3[\beta] &= t_1[\beta] \\ t_3[R - \beta] &= t_2[R - \beta] \\ t_4[\beta] &= t_2[\beta] \\ t_4[R - \beta] &= t_1[R - \beta] \end{aligned}$$

Example: A relation of university courses, the books recommended for the course, and the lecturers who will be teaching the course:

- course  $\rightarrow\!\!\! \rightarrow$  book
- course  $\rightarrow\!\!\! \rightarrow$  lecturer

Test: course  $\rightarrow\!\!\! \rightarrow$  book

|     | Course       | Book        | Lecturer | Tuples |
|-----|--------------|-------------|----------|--------|
| AHA | Silberschatz | John D.     |          | t1     |
| AHA | Nederpelt    | William M.  |          | t2     |
| AHA | Silberschatz | William M.  |          | t3     |
| AHA | Nederpelt    | John D.     |          | t4     |
| AHA | Silberschatz | Christian G |          |        |
| AHA | Nederpelt    | Christian G |          |        |
| OSO | Silberschatz | John D.     |          |        |
| OSO | Silberschatz | William M.  |          |        |

So now, let us define multivalued dependency in a formal way and. So, we say that  $\alpha \rightarrow\!\!\! \rightarrow \beta$ , naturally  $\alpha$  and  $\beta$  both have to be subsets of the given set of attributes. When we say that? When there are for all pairs of tuples  $t_1$  and  $t_2$  such that they match on the fields of  $\alpha$ , this till this point it looks like functional dependencies. There exists two more tuples  $t_3$  and  $t_4$  such that this condition sold, what are the conditions? Look, carefully here we say that all of them match on the  $\alpha$  attributes which is fine, then you say that  $t_3$  matches with  $t_1$  in the  $\beta$  attributes and  $t_3$  matches on the remaining attributes with  $t_2$ . Similarly,  $t_4$  matches with  $t_2$  in the  $\beta$  attributes and  $t_4$  matches with  $t_1$  on the remaining attributes.

So, let us look at an example, gets confusing. So, here is course book and lecturer ah. So, it is a relationship of university courses known naturally, every course has multiple recommended books and every course has been taken by multiple different lecturers from time to time. So, course can have multiple books. So, there is a multivalued dependency here, it can be taught by multiple lectures.

So, there is a multivalued dependency here and therefore, I can have an instance of this particular relation and I am just showing you, how to test for the multivalued dependency **course  $\rightarrow\!\!\! \rightarrow$  book**. So, these are the two, four tuples I have marked  $t_1, t_2, t_3, t_4$  if you look into the first condition. So, this is your  $\alpha$  I am checking for. So, this is  $\alpha$  this is  $\beta$ . So, this is  $\beta$  and this is ah. So, to say **R  $- (\beta - \alpha)$  ok**.

So, the first condition that all these tuples will have to match on  $\alpha$  yes, they do, all four of them have AHA here. So, that is fine take at the second condition  $t_3$  on  $\beta$  is Silberschatz and  $t_1$  on  $\beta$  is also Silberschatz. So, they match and  $t_3$  on the remaining attributes remaining attributes are, if I take out  $\beta$  if I take out book it is AHA it is course and the lecturer that is remaining. Now it already matches on the course. So, I do not have to check for that, but. So, I can just check for whether it matches on lecturer, between some checking for this rule, whether  $t_3$  and  $t_2$  match yes  $t_3$  and  $t_2$  match, they have the same name for the lecturer.

Look at the next one which is  $t_4$  and  $t_2$  match on  $\beta$ ,  $t_4$  and  $t_2$  match on  $\beta$  yes, they have the same name of the book, and whether  $t_4$  and  $t_1$  match on the lecturer, this rule  $t_4$  and  $t_1$  match on the lecturer this rule. So, it also satisfies. So, I can say that this relation has holds the multivalued dependency course multi determining book. In a similar way you can you can mark your  $t_1$ ,  $t_2$ ,  $t_3$ ,  $t_4$  on this and check for course multi determining the lecturer, actually we will we will soon state that; if **course  $\rightarrow\!\!\!\rightarrow$  book**, then it is trivial that course will also multi determine lecturer.

(Refer Slide Time: 10:36)


**Example**

SWAYAM: NPTEL-NOC's Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

- Let  $R$  be a relation schema with a set of attributes that are partitioned into 3 nonempty subsets.  $Y, Z, W$
- We say that  $Y \rightarrow\!\!\!\rightarrow Z$  ( $Y$  **multidetermines**  $Z$ ) if and only if for all possible relations  $r(R)$ 

$$\langle y_1, z_1, w_1 \rangle \in r \text{ and } \langle y_1, z_2, w_2 \rangle \in r$$

then

$$\langle y_1, z_1, w_2 \rangle \in r \text{ and } \langle y_1, z_2, w_1 \rangle \in r$$
- Note that since the behavior of  $Z$  and  $W$  are identical it follows that  
 $Y \rightarrow\!\!\!\rightarrow Z$  if  $Y \rightarrow\!\!\!\rightarrow W$





Database System Concepts - 6<sup>th</sup> Edition
16.10
©Silberschatz, Korth and Sudarshan

So, this is just to tell you if you have 3 non-empty sets of attributes  $Y$ ,  $Z$  and  $W$  and then we say,  $Y \rightarrow\!\!\!\rightarrow Z$ , if and only if there are these are the possible relations. That I can have  $Y_1$  and  $Z_1$   $W_1$  in a relation and  $Y_1$  and  $Z_2$ ,  $W_2$  in the relation, then I can have

Y 1, Z 1 with W 2 and Y 1, Z 2 with W 1, that is you can basically take the cross of these to other 2 attributes and those are r tuples, possible tuples in your relation and ah.

So, you can you can naturally if you read it in little in a different way, then you can observe that since the behavior of Z and W are identical they are switchable. So, if  $Y \rightarrow\!\!\! \rightarrow Z$ , then you can you have ZY multi determining W and vice versa. So, this is; what is a core observation in terms of the multi value dependencies

(Refer Slide Time: 11:40)

**Example (Cont.)**

- In our example:  
 $ID \rightarrow\!\!\! \rightarrow child\_name$   
 $ID \rightarrow\!\!\! \rightarrow phone\_number$
- The above formal definition is supposed to formalize the notion that given a particular value of Y ( $ID$ ) it has associated with it a set of values of Z ( $child\_name$ ) and a set of values of W ( $phone\_number$ ), and these two sets are in some sense independent of each other.
- Note:
  - If  $Y \rightarrow Z$  then  $Y \rightarrow\!\!\! \rightarrow Z$
  - Indeed we have (in above notation)  $Z_1 = Z_2$   
The claim follows.

SWATAM: NPTEL/NOC INDOCS Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
16.11 ©Silberschatz, Korth and Sudarshan

So, this is in terms of our example, you can now clearly understand that  $ID \rightarrow\!\!\! \rightarrow child\_name$ ,  $ID \rightarrow\!\!\! \rightarrow phone\_number$ , in the earlier example that we took and ah. So, we can also note that if there is a functional dependency,  $Y \rightarrow Z$  then; obviously,  $Y \rightarrow\!\!\! \rightarrow Z$ , that is that is just quite obvious.

(Refer Slide Time: 12:06)

The slide has a header 'Use of Multivalued Dependencies' with a sailboat icon. On the left, there is a vertical sidebar with text: 'SWAYAM: NPTEL-NCOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018'. Below the sidebar is a video frame showing a man speaking. The main content area contains a bulleted list:

- We use multivalued dependencies in two ways:
  1. To test relations to **determine** whether they are legal under a given set of functional and multivalued dependencies
  2. To specify **constraints** on the set of legal relations. We shall thus concern ourselves *only* with relations that satisfy a given set of functional and multivalued dependencies.
- If a relation  $r$  fails to satisfy a given multivalued dependency, we can construct a relations  $r'$  that does satisfy the multivalued dependency by adding tuples to  $r$ .

At the bottom, there is footer text: 'Database System Concepts - 8<sup>th</sup> Edition', '16.12', and '©Silberschatz, Korth and Sudarshan'.

So, we have to we can make use of multi value dependency to specify, further constraints to remove redundancies and defining what is legal in a relation. And if a relation fails to satisfy a given multivalued dependency, then we can construct a relation R primed, that does satisfy the multi valued dependency by adding tuples to that r right.

(Refer Slide Time: 12:32)

The slide has a header 'Theory of MVDs' with a sailboat icon. On the left, there is a vertical sidebar with text: 'SWAYAM: NPTEL-NCOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018'. Below the sidebar is a table:

| Name               | Rule                                                                                                                                                       |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| C- Complementation | : If $X \rightarrow\!\!\rightarrow Y$ , then $X \rightarrow\!\!\rightarrow \{R - (X \cup Y)\}$ .                                                           |
| A- Augmentation    | : If $X \rightarrow\!\!\rightarrow Y$ and $W \supseteq Z$ , then $WX \rightarrow\!\!\rightarrow YZ$ .                                                      |
| T- Transitivity    | : If $X \rightarrow\!\!\rightarrow Y$ and $Y \rightarrow\!\!\rightarrow Z$ , then $X \rightarrow\!\!\rightarrow (Z - Y)$ .                                 |
| Replication        | : If $X \rightarrow Y$ , then $X \rightarrow\!\!\rightarrow Y$ but the reverse is not true.                                                                |
| Coalescence        | : If $X \rightarrow\!\!\rightarrow Y$ and there is a $W$ such that $W \cap Y$ is empty, $W \rightarrow Z$ , and $Y \supseteq Z$ , then $X \rightarrow Z$ . |

Below the table, there is a bulleted list:

- A MVD  $X \rightarrow\!\!\rightarrow Y$  in  $R$  is called a trivial MVD if
  - $Y$  is a subset of  $X$  ( $X \supseteq Y$ ) or
  - $X \cup Y = R$ . Otherwise, it is a non trivial MVD and we have to repeat values redundantly in the tuples.

At the bottom, there is footer text: 'Source: <http://www.edugrabs.com/multivalued-dependency-mvd/>', 'Database System Concepts - 8<sup>th</sup> Edition', '16.13', and '©Silberschatz, Korth and Sudarshan'.

Now, once having defined the notion of multi valued dependency, we next proceed to check, how do we reason about that. So, I would remind you about functional

dependencies, and the different rules of ah functional dependencies Armstrong's rules, that we had introduced the all of these of augmentation transitivity and all that.

So, in terms of functional dependencies we have three rules, commonly called the cat rules. Which purely involve the functional dependencies, first is a complementation which is a kind which we have just discussed shown, that if  $X \rightarrow\!\!\!\rightarrow Y$ , then  $X \rightarrow\!\!\!\rightarrow R - (X \cup Y) \rightarrow\!\!\!\rightarrow$  the remaining set of attributes.

Augmentation that is I can augment any multivalued dependency with left and putting attributes on the left and right-hand side, as long as I put all attributes that I put on the right-hand side, I put them on the left-hand side. I may put more attributes on the left-hand side, but all attributes that I put on the right-hand side here  $Z$  must be a subset of the attributes that I put on the left-hand side, that augmentation is possible. Transitivity is manifesting in a little different way, if  $X \rightarrow\!\!\!\rightarrow Y$  and  $Y \rightarrow\!\!\!\rightarrow Z$  then,  $X \rightarrow\!\!\!\rightarrow Z - Y$ .

So, these are the these are the 3 rules which are basically these three are rules that, involve only multi valued dependencies and the other two rules, actually involve the relationship between multi value dependency the replication rule and the coalescence rule, which are between the multi value dependency and the functional dependency. We are not going deeper into that further, or trying to take specific examples and show how they work.

I just want you to know that such rules exist through which, you can define similar algorithms for multivalued dependency also, as we did for functional dependency like as you can understand the most critical algorithm to define would be the algorithm of closure, which can again be used in the situation where I have functional as well as multivalued dependency.

So, just we will keep that, in little bit advance space of this course. So, just know that such things exist, but we are not going into the details of that. Finally, for a multivalued dependency where,  $X \rightarrow\!\!\!\rightarrow Y$  we call that MVD to be trivial. If either  $Y \subseteq X$  which is the notion we used for functional dependencies or there is a second condition here, that the  $X \cup Y$  that left hand right hand side gives you the whole set of attributes, otherwise it is a non-trivial multivalued dependency and we have to repeat the values. So, these are the two conditions, if they satisfy then we know that we have a trivial multi value dependency and we do not want to deal with that.

(Refer Slide Time: 15:40)

The slide has a title 'Theory of MVDs' in red at the top right. On the left is a small sailboat icon. The main content is a bulleted list:

- From the definition of multivalued dependency, we can derive the following rule:
  - If  $\alpha \rightarrow \beta$ , then  $\alpha \rightarrow\rightarrow \beta$
- That is, every functional dependency is also a multivalued dependency
- The **closure**  $D^*$  of  $D$  is the set of all functional and multivalued dependencies logically implied by  $D$ .
  - We can compute  $D^*$  from  $D$ , using the formal definitions of functional dependencies and multivalued dependencies.
  - We can manage with such reasoning for very simple multivalued dependencies, which seem to be most common in practice
  - For complex dependencies, it is better to reason about sets of dependencies using a system of inference rules

At the bottom left is vertical text: SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018. At the bottom center is 'Database System Concepts - 8<sup>th</sup> Edition' and '16.14'. At the bottom right is '©Silberschatz, Korth and Sudarshan'.

So, there is little bit of references to the theory given here, I have mentioned that there are closure algorithms ah. So, that given a set of dependencies I will now generalize and say dependencies, which means that there could be functional dependencies, as well as multivalued dependencies. Given a set of dependencies you can define a closure of all of these functional and multivalued dependencies together, that are implied by the given set and we can have all those parallel definitions of closure of the dependencies, the minimal cover, canonical cover and so on.

So, I just want you to note that these things have been defined and the existent theory, but will be beyond the current course that we are pursuing. So, it is now that we have is we have seen an another additional source of redundancy in our data, in terms of multiple values and in terms of the multi value dependency that hold.

(Refer Slide Time: 16:50)

The slide has a header 'Fourth Normal Form' in red. On the left is a small sailboat icon. The main content is a bulleted list:

- A relation schema  $R$  is in **4NF** with respect to a set  $D$  of functional and multivalued dependencies if for all multivalued dependencies in  $D^*$  of the form  $\alpha \rightarrow\!\!\rightarrow \beta$ , where  $\alpha \subseteq R$  and  $\beta \subseteq R$ , at least one of the following hold:
  - $\alpha \rightarrow\!\!\rightarrow \beta$  is trivial (i.e.,  $\beta \subseteq \alpha$  or  $\alpha \cup \beta = R$ )
  - $\alpha$  is a superkey for schema  $R$
- If a relation is in 4NF it is in BCNF

At the bottom left is a video thumbnail showing a man speaking, with the text 'SWAYAM-NPTEL NOC Instructor: Prof. P.P. Desai, IIT Kanpur- Jan-Apr- 2018'. At the bottom right are navigation icons and the text 'Database System Concepts - 8<sup>th</sup> Edition 16.16 ©Silberschatz, Korth and Sudarshan'.

So, we would now like to look into if such dependencies exist, then how do you decompose a relation to satisfy that the redundancy caused by such dependencies are not affecting us. So, such a normal form it is beyond the third normal form is called to be said to be a 4th normal form or 4 NF. Where you say that a relation is in 4 NF if, every multi valued dependency  $\alpha \rightarrow\!\!\rightarrow \beta$ , in the closure of the set of dependencies is either trivial, trivial means that left hand side is a subset of the right-hand side or the union of the left and right-hand side gives you the whole set of attributes.

So, it is either trivial every dependency is either trivial, or a left-hand side is a superset of the schema  $R$ , you can very well relate that this is just a little twist on the definition of the Boyce Codd normal form, where the second condition was identical and only thing in the first condition instead of MVD, you had a functional dependency. So, when we have this, we say we are a relation would be in the in the 4th normal form. Naturally, if a relation is in 4th normal form, it is trivial that it will be in the Boyce Codd normal form, but the reverse will not be necessarily true.

(Refer Slide Time: 18:19)

The slide has a title 'Restriction of Multivalued Dependencies' at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list of points:

- The restriction of  $D$  to  $R_i$  is the set  $D_i$  consisting of
  - All functional dependencies in  $D^+$  that include only attributes of  $R_i$
  - All multivalued dependencies of the form
$$\alpha \twoheadrightarrow (\beta \cap R_i)$$
where  $\alpha \subseteq R_i$  and  $\alpha \twoheadrightarrow \beta$  is in  $D^+$

At the bottom left, there is a small video thumbnail showing a person speaking. The bottom right corner shows the text '©Silberschatz, Korth and Sudarshan'.

So, again the same set of concepts that, if I have a set of dependencies and you have a decomposed relation then smaller relation, then I can project that set of dependencies, in terms of a particular subset of the attributes and here is the condition that is given.

(Refer Slide Time: 18:43)

The slide has a title '4NF Decomposition Algorithm' at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a numbered list of steps:

- For all dependencies  $A \twoheadrightarrow B$  in  $D^+$ , check if  $A$  is a superkey
  - By using attribute closure
- If not, then
  - Choose a dependency in  $F^+$  that breaks the 4NF rules, say  $A \twoheadrightarrow B$
  - Create  $R_1 = A B$
  - Create  $R_2 = A (R - (B - A))$
  - Note that:  $R_1 \cap R_2 = A$  and  $A \twoheadrightarrow AB (= R_1)$ , so this is lossless decomposition
- Repeat for  $R_1$ , and  $R_2$ 
  - By defining  $D_{1+}$  to be all dependencies in  $F$  that contain only attributes in  $R_1$
  - Similarly  $D_{2+}$

At the bottom left, there is a small video thumbnail showing a person speaking. The bottom right corner shows the text '©Silberschatz, Korth and Sudarshan'.

So, the decomposition algorithm into 4 NF is exactly like the decomposition algorithm of the Boyce Codd Normal form BCNF. Only difference being that, now you may be doing this crucial step of 2A decomposition, for every multivalued dependency also earlier we were doing this only for the functional dependency.

So, now if there is any offending multivalued dependency, which is not satisfying the 4 NF form? We can decompose the relation in terms of R 1 and R 2, as in here which is exactly like the Boyce Codd normal form and then the rest of it is simple. If ah if it is you know by this another important point, that you that you must note is in this process you actually guarantee lossless join.

So, this also continues to be in lossless join, with every decomposition and then you keep on repeating till all dependencies in F, in your set has been dealt with the attributes in R 1 and have converted them into the 4 NF form. So, you have a total 4 NF decomposition happening.

(Refer Slide Time: 20:04)

The slide features a small sailboat icon in the top left corner. The title '4NF Decomposition Algorithm' is centered at the top in red. On the left edge, there is vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr- 2018'. The main content is a pseudocode algorithm:

```
result := {R};  
done := false;  
compute D+;  
Let D_i denote the restriction of D+ to R_i;  
while (not done)  
  if (there is a schema R_i in result that is not in 4NF) then  
    begin  
      let α →→ β be a nontrivial multivalued dependency that holds  
      on R_i such that α → R_i is not in D_i, and α ∩ β = ∅;  
      result := (result - R_i) ∪ (R_i - β) ∪ (α, β);  
    end  
  else done := true;  
Note: each R_i is in 4NF, and decomposition is lossless-join
```

At the bottom, there is footer text: 'Database System Concepts - 8<sup>th</sup> Edition', '16.19', and '©Silberschatz, Korth and Sudarshan'. A standard presentation toolbar is visible at the very bottom.

(Refer Slide Time: 20:13)

**Example of 4NF Decomposition**

The diagram illustrates the decomposition of a relation into three 4NF relations. On the left, the original relation is shown with attributes Man(M), Phones(P), Dogs\_Likes(D), and Address(A). Functional dependencies FD1 (Man →→ Phones), FD2 (Man →→ Dogs\_Likes), and FD3 (Man → Address) are listed. It is noted that the key is MPD (Multi-Valued Dependent). On the right, the relation is decomposed into three relations: Person\_Phones (Man, Phones), Person\_DogsLikes (Man, Dogs\_Likes), and Person\_Address (Man, Address). The dependencies FD1, FD2, and FD3 are mapped to these relations. A note on the right states that while the individual relations are in 4NF, the overall MVD is not due to the shared superkey Man.

| Man(M) | Phones(P) | Dogs_Likes(D) | Address(A)          |
|--------|-----------|---------------|---------------------|
| M1     | P1        | D1            | 49-ABC,Bhiwani(HR.) |
| M1     | P2        | D2            | 49-ABC,Bhiwani(HR.) |
| M2     | P3        | D2            | 36-XYZ,Rohtak(HR.)  |
| M1     | P1        | D2            | 49-ABC,Bhiwani(HR.) |
| M1     | P2        | D1            | 49-ABC,Bhiwani(HR.) |

Source: http://www.edugrabs.com/4nf-normal-form/  
SWAYAMI: NPTEL-NOOC Instructor: Prof. P. P. Doshi, IIT Kanpur - Jan-Apr-2018

Ah let us take a this here, is the like before here is a formal algorithm for those who would be interested, to formally study the steps. Ah here I am just showing examples of 4 NF decomposition. So, we started this discussion with a person relational scheme, having man, phone and dog likes MPD, I have added I have just modified and I have added another attribute address. So, that in addition to the multi value dependencies, I can also have a functional dependency. So, we have two multivalued dependencies like before, man multi determining phones and man multi determining dog like, but now we have a functional dependency man determining address the key continues to be MPD.

So, all of these dependencies will violate the 4 NF, because none of them satisfy the either of the condition, that none of them are trivial and on for none of them left hand side is a super key because key is MPD. So, you can see that in on instances of this, relational schema you will have multiple redundant records, in the actual instance. So, on the right we normalize we normalize by taking FD 1; **man U phones** that gives you the first relation and then the rest of it. Then again you split based on FD 2, you have the second relation in the decomposition man and dog like and the third one gets generated as a byproduct of that, which is man and address.

And you have three relations now, which together represent the original relation each one of them is in 4th normal form. Actually, what happens is, when you when you have decomposed then, FD1 in this has become a relation where, the multivalued dependency **man →→ phones** can be checked in terms of a functional dependency itself, and that that is what gives you the multi value dependency. And since it is multivalued so, man and

phones together continues to form the key, similarly in the second one the man and dog like is the key. Because you just have the multivalued dependency and given the same man, you will have multiple dogs whom he or she likes, but in the third one in the person address where you have man and address you have only man as the key, because man is a functional dependency that holds.

(Refer Slide Time: 23:01)

**Example of 4NF Decomposition**

- $R = (A, B, C, G, H, I)$
- $F = \{ A \rightarrow\!\!\!\rightarrow B, B \rightarrow\!\!\!\rightarrow HI, CG \rightarrow\!\!\!\rightarrow HI \}$
- $R$  is not in 4NF since  $A \rightarrow\!\!\!\rightarrow B$  and  $A$  is not a superkey for  $R$
- Decomposition
  - a)  $R_1 = (A, B)$  ( $R_1$  is in 4NF)
  - b)  $R_2 = (A, C, G, H, I)$  ( $R_2$  is not in 4NF, decompose into  $R_3$  and  $R_4$ )
  - c)  $R_3 = (C, G, H)$  ( $R_3$  is in 4NF)
  - d)  $R_4 = (A, C, G, I)$  ( $R_4$  is not in 4NF, decompose into  $R_5$  and  $R_6$ )
    - $A \rightarrow\!\!\!\rightarrow B$  and  $B \rightarrow\!\!\!\rightarrow HI \Rightarrow A \rightarrow\!\!\!\rightarrow HI$ , (MVD transitivity), and
    - and hence  $A \rightarrow\!\!\!\rightarrow I$  (MVD restriction to  $R_4$ )
  - e)  $R_5 = (A, I)$  ( $R_5$  is in 4NF)
  - f)  $R_6 = (A, C, G)$  ( $R_6$  is in 4NF)

SWAYAM: NPTEL-NOC: Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018  
Database System Concepts - 8<sup>th</sup> Edition  
16.21  
©Silberschatz, Korth and Sudarshan

So, this is a simple illustration of decomposition into 4 NF, here is a little more elaborate one, again this is a we have I have worked through the steps. So, there are three multivalued dependencies and you can see that,  $A \rightarrow\!\!\!\rightarrow B$  is not does not is not who does not hold the condition of 4 NF. So, you have to decompose, you decompose get  $R_1$  which is in 4 NF and the remaining  $R_2$  which is also not in 4 NF. So, decompose in  $R_3$  which is in 4 NF and  $R_4$ , we can  $R_4$  is not in 4 NF you decompose  $R_4$  into  $R_5$  and  $R_6$  and work through that, and you will be able to see that  $R_5$  is in 4 NF and  $R_6$  also is in 4 NF, which gives a complete multivalued decomposition of this whole set.

Naturally with that, we will conclude our discussion on the decomposition process, there are there would be some more aspects to look at and there is lot of more normal forms that exist. But this is for all practical purposes; a database is normalized, when it is represented in terms of the third normal form. And I have discussed still I have discussed the 4th normal form, because in some places people prefer to represent also in 4th normal form.

So, that they guarantee that they have even less redundancy in the data, but leaving that, let us quickly take a round in terms of the what we have done so far and what is a basic overall design process that we should be following.

(Refer Slide Time: 24:43)

The slide has a title 'Design Goals' in red at the top right. On the left is a small sailboat icon. The main content is a bulleted list of goals for relational database design:

- Goal for a relational database design is:
  - BCNF / 4NF
  - Lossless join
  - Dependency preservation
- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF
- Interestingly, SQL does not provide a direct way of specifying functional dependencies other than superkeys.  
Can specify FDs using assertions, but they are expensive to test, (and currently not supported by any of the widely used databases!)
- Even if we had a dependency preserving decomposition, using SQL we would not be able to efficiently test a functional dependency whose left hand side is not a key

At the bottom left is vertical text: 'SWAYAM: NPTEL-NOCOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018'. At the bottom center: 'Database System Concepts - 8<sup>th</sup> Edition' and '16.23'. At the bottom right: '©Silberschatz, Korth and Sudarshan' and a set of icons.

So, again to remind you the goal for our design is to have a relational database which is in BCNF or 3 NF has a lossless join due to the decomposition, and dependency preservation. If we cannot achieve that, I am sorry earlier what I meant is BCNF or 4 NF not BCNF and 3 NF. So, the idea would be I have a decomposition in BCNF and 4 NF lossless join and dependency preservation which may not be achievable. If I cannot achieve that then I have to sacrifice either, the lack of dependency preservation. So, dependencies will have to be checked using natural join or, I will allow a little bit of redundancy and use the third normal form where I have the guarantee.

Now, at this point you must wonder and note that SQL, the language in which we are doing the creation and update and the query processing. That SQL does not provide any directory of specifying or checking any dependency, other than the functional other than the functional dependency that checks the super key. Super key is the only functional dependency that SQL would check, no other functional dependency or multivalued dependency and other type dependencies can be specified or checked in SQL. You can do that using assertions, in the while discussing SQL I were talked about assertions we can do that using assertions, but that too is very expensive to test. So, it is

not usually supported by any of the databases, which are widely used because that slows down your every process very, very much.

So, you can understand that in terms of your design goals, you have to do a very good job to make sure that, your functional and multivalued dependencies are accurately expressed in the design and accordingly the schemas are normalized in the proper ways satisfying BCNF or 4 NF or 3 NF normal forms. But because, while you will actually have instances there will not be a practical way, to see if you are violating any one or more of these ah rules of dependencies that you have set.

(Refer Slide Time: 27:09)

The slide has a header 'Further Normal Forms' in red. On the left is a small logo of a sailboat on water. The main content is a bulleted list:

- Further NFs
  - Elementary Key Normal Form (EKNF)
  - Essential Tuple Normal Form (ETNF)
  - Join Dependencies And Fifth Normal Form (5 NF)
  - Sixth Normal Form (6NF)
  - Domain/Key Normal Form (DKNF)
- **Join dependencies** generalize multivalued dependencies
  - lead to **project-join normal form (PJNF)** (also called **fifth normal form**)
  - A class of even more general constraints, leads to a normal form called **domain-key normal form**.
  - Problem with these generalized constraints: are hard to reason with, and no set of sound and complete set of inference rules exists.
  - Hence rarely used

Small text at the bottom left: SWAYAM: NPTEL-NOC MOOCs; Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition

Small text at the bottom right: 16.24 ©Silberschatz, Korth and Sudarshan

So, as I mentioned there are actually these are not the only forms, there are various other normal forms as well and fifth normal form, 6 normal form and so on, but it is very rarely these are very rarely used. It is not easy to decompose into these normal forms and by this decomposition does not give you enough returns in terms of the reduction of redundancy and removal of anomalies, that people often would have motivation to do them, but you should know that such normal forms exist.

(Refer Slide Time: 27:44)

The slide has a header 'Overall Database Design Process' with a sailboat icon. On the left, there is a vertical sidebar with text: 'SWAYAM: NPTEL NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018'. The main content area contains a bulleted list:

- We have assumed schema  $R$  is given
  - $R$  could have been generated when converting E-R diagram to a set of tables
  - $R$  could have been a single relation containing *all* attributes that are of interest (called **universal relation**)
  - Normalization breaks  $R$  into smaller relations
  - $R$  could have been the result of some ad hoc design of relations, which we then test/convert to normal form

At the bottom left is a video thumbnail of a professor, and at the bottom right are navigation icons and the text 'Database System Concepts - 8<sup>th</sup> Edition 16.25 ©Silberschatz, Korth and Sudarshan'.

So, in the overall process if we look, at I mean what we have been doing is there are several tracks that we could be taking one possible thing is, the whole set of attributes have been generated while we have converted or relation has been generated. When you have converted the entity relationship diagram, the UML or the ER diagram into a set of tables, that is how we got our set of attributes or the relational schema  $R$ , it is also possible that we just started with a single relation containing all attributes, which is called the universal relation? And then normalization will break them into smaller relations. It could have been or could have been the result of some adds of design of relations also, and then you convert them.

(Refer Slide Time: 28:33)

The slide has a header 'ER Model and Normalization' with a sailboat icon. The text discusses normalization rules and provides an example of a functional dependency between department\_name and building. It also notes that most relationships are binary. The footer includes the title 'Database System Concepts - 8<sup>th</sup> Edition', the date '2018', and the author's name '©Silberschatz, Korth and Sudarshan'.

SWAYAM-NPTEL NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018

## ER Model and Normalization

- When an E-R diagram is carefully designed, identifying all entities correctly, the tables generated from the E-R diagram should not need further normalization
- However, in a real (imperfect) design, there can be functional dependencies from non-key attributes of an entity to other attributes of the entity
  - Example: an *employee* entity with attributes *department\_name* and *building*, and a functional dependency  $department\_name \rightarrow building$
  - Good design would have made department an entity
- Functional dependencies from non-key attributes of a relationship set possible, but rare --- most relationships are binary

Database System Concepts - 8<sup>th</sup> Edition

16.26

©Silberschatz, Korth and Sudarshan

So, there are possible all different possible tracks that can happen. So, if we have taken the ER model track, then frankly speaking if the ER model is carefully designed, then every entity defined in that ER model will have only the dependency; which are the determining super key.

So, just recall the employee department building kind of situation we discussed earlier. So, an employee entity has attributes department name and building, and there is a functional dependency from department\_name to building. So, what it means that in the entity relationship diagram itself we didn't do a good job. If we had done a good job then we would have identified that the department itself is an entity and therefore, would not feature as an attribute on the employee. So, it would have been I mean right there, we would have if we had called it as a separate entity, then that is equivalent of what we are doing now taking the relation and then breaking it down through decomposition.

So, functional dependencies from non-key attributes of a relationship are possible ah, but are rare. So, mostly the relationships are binary, and if you do a careful design of the ER model then many of these deep exercise of normalization you will not have to go through.

(Refer Slide Time: 29:55)

The slide has a title 'Denormalization for Performance' at the top right. On the left is a small logo of a sailboat on water. The main content is a bulleted list of points:

- May want to use non-normalized schema for performance
- For example, displaying *prereqs* along with *course\_id*, and *title* requires join of *course* with *prereq*
  - *Course(course\_id, title, ...)*
  - *Prerequisite(course\_id, prereq)*
- Alternative 1: Use denormalized relation containing attributes of *course* as well as *prereq* with all above attributes: *Course(course\_id, title, prereq, ...)*
  - faster lookup
  - extra space and extra execution time for updates
  - extra coding work for programmer and possibility of error in extra code
- Alternative 2: use a materialized view defined as  
*Course*  $\bowtie$  *Prerequisite*
  - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors

At the bottom left is vertical text: SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur. At the bottom center is the text: Database System Concepts - 8<sup>th</sup> Edition, 16.27. At the bottom right is the copyright notice: ©Silberschatz, Korth and Sudarshan.

It should also be kept in your view, that at the times when you want to de normalize want to use denormalized relations, because if you have normalized and the only way to get back the original view is to perform join. So, if we of course, if you have prerequisites and if you want to say, view or print prerequisites with that title and course id naturally you will have to take a join with the course, which is expensive.

So, one option could be first alternative could be that, you use a denormalized relation, where the course prerequisite is actually included in the course and you know that will have you know violations of some of the normal forms. Because, there are there are functional dependencies between them, but that will certainly lead you to first a look up, because you have them in the same table you do not need to perform join. But you need extra space, exact extra execution time for update, because you have redundant data you have redundancy while programming on that coding on that, because of this redundancy there could be possibility of error, because any of these anomalies can happen and your code will have to now take care of that.

So, it does help in certain way in terms of getting a better efficiency, but if there is a there is a cost to pay in a different way also the other alternative could be you can have a materialized view, which is actually the join in course of prerequisite. In terms of performance it has a same benefit or the costs as you say, but only thing is you will not need to do that extra coding. So, it is better from that perspective.

So, always keep the issue of denormalization in view, and we do a careful design that if it is very frequent that, you will have to compute a join then, you might want to sacrifice some of the redundancy some of the you know possibilities of having anomaly, and still have a you know denormalized design in your database.

(Refer Slide Time: 31:59)

The slide has a header 'Other Design Issues' with a sailboat icon. The main content lists issues with earnings tables and company\_year tables. A sidebar on the left contains course information: SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Date, IIT Kanpur; Date: Jan-Apr., 2018. The footer includes the book title 'Database System Concepts - 8<sup>th</sup> Edition', page number 16.28, and copyright information: ©Silberschatz, Korth and Sudarshan.

- Some aspects of database design are not caught by normalization
- Examples of bad database design, to be avoided:
  - Instead of *earnings (company\_id, year, amount )*, use
    - *earnings\_2004, earnings\_2005, earnings\_2006*, etc., all on the schema (*company\_id, earnings*).
    - Above are in BCNF, but make querying across years difficult and needs new table each year
  - *company\_year (company\_id, earnings\_2004, earnings\_2005, earnings\_2006)*
    - Also in BCNF, but also makes querying across years difficult and requires new attribute each year.
    - Is an example of a **crosstab**, where values for one attribute become column names
    - Used in spreadsheets, and in data analysis tools

There are several other issues of design, which do not get captured in what we have designed. For example, let say very regularly we are we have returns, income tax returns to submit and we will be maintain income tax return tax your sales tax return and on all that, and you maintain your accounts book of transactions debit credit accounted and so on. Now naturally, these are all bound in terms of one-year effectivity.

So, when they come when in the next year comes, then you need a separate you know set of records to be done for that year. So, how do you. So, if you if you have such a table where you along with the company idea of year and amount and then how do you take care of this situation, because one way could be that you have all of these you take out year, from the attribute and you have separate table in every year. So, you will have to create new table and remember their name. So, if queries which run across year will be difficult to do.

The other way could be that, you every New Year you start renaming you know you do a year where your earnings from different years are shown on different columns. So, you are basically every year you have the result in terms of a different attribute. So, that also

is not a very good solution for a database it is something which is with the spreadsheets will often use, but in terms of data which has a certain format and needs to be you know redefined from scratch, at a different time frame in a different way, then you will come across these issues.

(Refer Slide Time: 33:56)

**Modeling Temporal Data**

- **Temporal data** have an association time interval during which the data are *valid*.
- A **snapshot** is the value of the data at a particular point in time
- Several proposals to extend ER model by adding valid time to
  - attributes, e.g., address of an instructor at different points in time
  - entities, e.g., time duration when a student entity exists
  - relationships, e.g., time during which an instructor was associated with a student as an advisor.
- But no accepted standard
- Adding a temporal component results in functional dependencies like  
 $ID \rightarrow street, city$   
not to hold, because the address varies over time
- A **temporal functional dependency**  $X \rightarrow Y$  holds on schema  $R$  if the functional dependency  $X \rightarrow Y$  holds on all snapshots for all legal instances  $r(R)$ .

SWAYAM: NPTEL-NOC MOOCs; Instructor: Prof. P P Das, IIT Kanpur - Jan-Apr - 2018

Database System Concepts - 8<sup>th</sup> Edition

16.30

©Silberschatz, Korth and Sudarshan

Ah let me close with just pointing out that, if we have a one kind of data that we have not looked at which are temporally in nature, that is all that we have said is the attributes and their values. So, if we put a value to an attribute then that value is taken to be the truth for now, and for the past and for the features. So, if that value changes, then you completely erase that in the database.

So, for example, today I stay at a certain address, tomorrow I may take up a different quarter my address has changed. So, in our design if there is against my employee id there is an address given, then once I change my quarter my address will change it will not be possible to recollect, what address I resided in say 2017.

So, temporal data of such kind, temporal data I am sorry just of such kind have an association with an interval. So, a snapshot often does not solve the problem. So, you have to decide, how you do that? Whether you can you would like to put some attributes, which specify the timestamp or you would like to I mean really have multivalued attributes, denoting the different time frames where they are they may have taken effect, there is no accepted standard. And the fact that, if you if you know that it

keeps on changing with time then your original dependencies might get affected they will change as well. So, these are the things that you will have to take care ah.

(Refer Slide Time: 35:30)

The slide has a title 'Modeling Temporal Data (Cont.)' in red. To the left is a small sailboat icon. The main content is a bulleted list:

- In practice, database designers may add start and end time attributes to relations
  - E.g., `course(course_id, course_title)` is replaced by  
`course(course_id, course_title, start, end)`
  - Constraint: no two tuples can have overlapping valid times
    - Hard to enforce efficiently
- Foreign key references may be to current version of data, or to data at a point in time
  - E.g., student transcript should refer to course information at the time the course was taken

On the left edge of the slide, there is vertical text: 'SWAYAM: NPTEL/NOC MOOCs Instructor: Prof. P P Das, IIT Kanpur - Jan-Apr. 2018'. At the bottom left is the text 'Database System Concepts - 8<sup>th</sup> Edition'. At the bottom right is the text '©Silberschatz, Korth and Sudarshan'. The bottom of the slide shows a video player interface with a play button and other controls.

This is another style, that many a times when you have to say that ok. This course with this title existed from this semester, a different semester the title may have changed. So, you can put a start and end attribute with which specifies what is the time for which the remaining attributes made sense, a good design, but these also have issues because, if you do this kind of temporal intervals, then how do you make sure that between 2 records the intervals are not overlapped. So, you are not saying that at the same time this course had them X this of course, also had name Y. So, they have to be disjoint. So, how do you check for this ah this consistency of data, there is no easy way to do that.

So, the foreign key references and all those. So, handling of temporal data is another aspect which will have to be looked into very carefully, in the design and you will need to do some kind of design compromise and implementation has to take care of those issues.

(Refer Slide Time: 36:33)

**Module Summary**

- Understood multi-valued dependencies to handle attributes that can have multiple values
- Learned Fourth Normal Form and decomposition to 4NF
- Discussed aspects of the database design process
- Studied the issues with temporal data

SWAYAM-NPTEL MOOCs Instructor: Prof. P.P. Desai, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

16.32

©Silberschatz, Korth and Sudarshan

So, to summarize we have ah taken a full look into the multivalued dependencies and tried to understand what happens, when your attributes get multiple values. Learn the 4th normal form for that and the decomposition into that, and most importantly we have tried to summarize the core database design process. That we have been discussing for the last four modules, this is the fifth one including this and we have understood that and we have talked a little bit about the temporal data.

And with this we close our discussions on the relational database design, and from the next module we will move on to other aspects of the database systems.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 21**  
**Application Design and Development**

Welcome to module 21 of Database Management Systems in this module and the next too we will discuss about application design and development.

(Refer Slide Time: 00:31)

PPD

### Week 04 Recap

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>▪ <b>Module 16: Relational Database Design/1</b><ul style="list-style-type: none"><li>◦ Features of Good Relational Design</li><li>◦ Atomic Domains and First Normal Form</li><li>◦ Functional Dependencies</li></ul></li><li>▪ <b>Module 17: Relational Database Design/2</b><ul style="list-style-type: none"><li>◦ Decomposition Using Functional Dependencies</li><li>◦ Functional Dependency Theory</li></ul></li><li>▪ <b>Module 18: Relational Database Design/3</b><ul style="list-style-type: none"><li>◦ Algorithms for Functional Dependencies</li><li>◦ Lossless Join Decomposition</li><li>◦ Dependency Preservation</li></ul></li></ul> | <ul style="list-style-type: none"><li>▪ <b>Module 19: Relational Database Design/4</b><ul style="list-style-type: none"><li>◦ Normal Forms</li><li>◦ Decomposition to 3NF</li><li>◦ Decomposition to BCNF</li></ul></li><li>▪ <b>Module 20: Relational Database Design/5</b><ul style="list-style-type: none"><li>◦ Multivalued Dependencies</li><li>◦ Decomposition to 4NF</li><li>◦ Database-Design Process</li><li>◦ Modeling Temporal Data</li></ul></li></ul> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition      21.2      ©Silberschatz, Korth and Sudarshan

In the last week we have for the whole week in the five modules we have discussed about relational database design; in depth we have looked into what are the different aspects of that.

(Refer Slide Time: 00:43)

The slide is titled "Module Objectives" in red bold text at the top center. In the top right corner, there is a small "PPD" logo. On the left side, there is a small image of a sailboat on water. At the bottom left, it says "SWAYAM: NPTEL-MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur, Date: Apr. 2018". At the bottom right, it says "©Silberschatz, Korth and Sudarshan". The main content is a bulleted list of objectives:

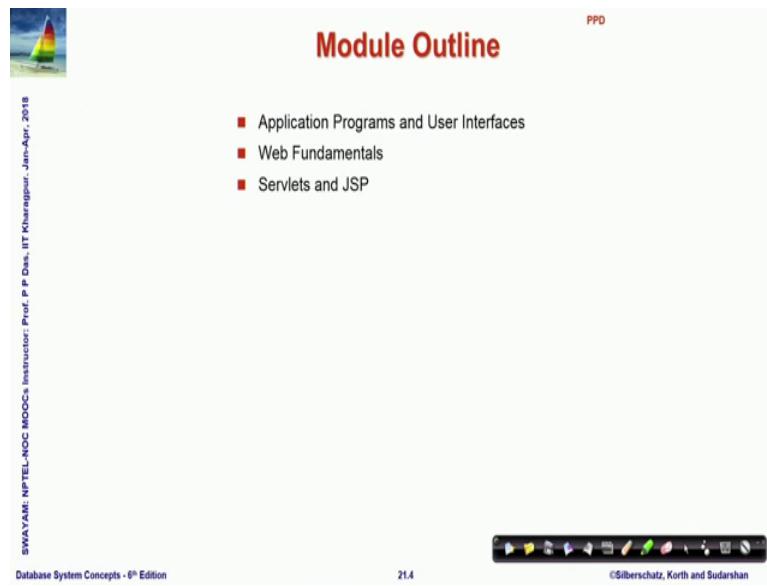
- To understand the requirements of Database Application Programs and User Interfaces
- To familiarize with the fundamental notions and technologies of Web
- To learn about Servlets and Java Server Pages

At the very bottom, there is a decorative footer bar with various icons.

And now we get into the core issue of if I have a relational database design existing and that is populated with the data then based on that how do we develop, how do we create an application where the user can interact and actually get answers to the questions that the user has or the user can actually update the data create new data, remove old data and so, on.

So, we would in that process like to familiarize with the fundamental notions of notions and technologies of web applications and specifically we would learned about servlets and Java server pages.

(Refer Slide Time: 01:30)

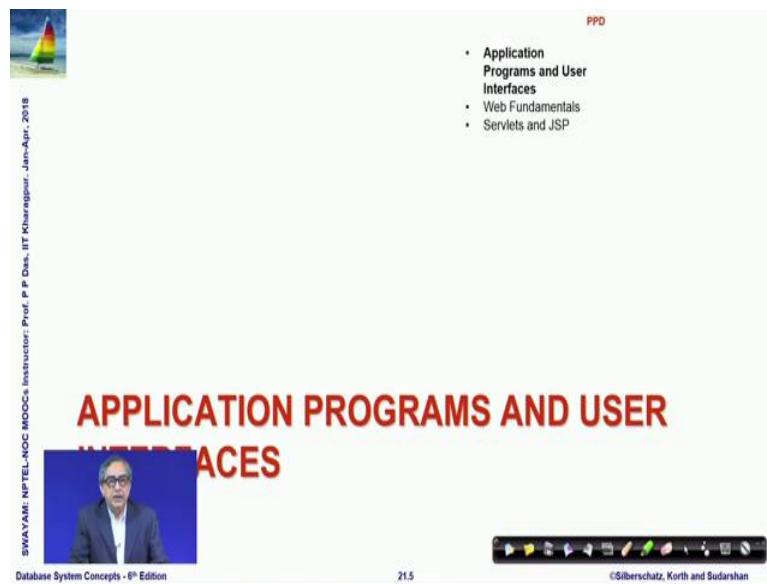


The slide is titled "Module Outline" in red at the top right. In the top left corner, there is a small image of a sailboat on water. The footer contains the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Desai, IIT Kharagpur - Jan-Apr. 2018", "PPD", "Database System Concepts - 8<sup>th</sup> Edition", "21.4", and "©Silberschatz, Korth and Sudarshan". A navigation bar with various icons is located at the bottom right.

- Application Programs and User Interfaces
- Web Fundamentals
- Servlets and JSP

So, these are the free topics to be covered.

(Refer Slide Time: 01:34)



The slide features a large title "APPLICATION PROGRAMS AND USER INTERFACES" in red at the top center. Below the title is a video frame showing a man speaking. The footer contains the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Desai, IIT Kharagpur - Jan-Apr. 2018", "PPD", "Database System Concepts - 8<sup>th</sup> Edition", "21.5", and "©Silberschatz, Korth and Sudarshan". A navigation bar with various icons is located at the bottom right.

- Application Programs and User Interfaces
- Web Fundamentals
- Servlets and JSP

So, we first start with application programs and user interfaces.

(Refer Slide Time: 01:41)

**Application Programs and User Interfaces**

■ Most database users do not use a query language like SQL  
■ An application program acts as the intermediary between users and the database  
■ Applications split into  
    ‣ frontend  
    ‣ middle layer  
    ‣ backend  
■ Frontend or Presentation Layer: user interface  
    ‣ Forms, Graphical user interfaces  
    ‣ Many interfaces are Web-based or Mobile App  
■ Middle Layer or Application / Business Logic Layer  
    ‣ Functionality of the Application – links front and backend  
■ Backend or Data Access Layer  
    ‣ Persistent data, large in volume, needs efficient access

**Overview of a 3-tier Architecture**

The diagram illustrates a 3-tier architecture with three layers: Presentation tier, Logic tier, and Data tier.

- Presentation tier:** The top-most level of the application is the user interface. The main function of this tier is to transfer data directly to something the user can understand.
- Logic tier:** This tier contains the application processing components, handle logical decisions and evaluations, and perform calculations. It receives and processes data between the two surrounding layers.
- Data tier:** Here information is stored and retrieved from a database or file system. The application in this tier passes data to the logic tier for processing and then eventually back to the user.

The diagram shows a flow from the user interface (Presentation tier) sending a "GET LIST OF ALL SALES LAST YEAR" query to the logic tier. The logic tier then sends a "SALES 1", "SALES 2", "SALES 3", and "SALES 4" response back to the user interface. The logic tier also interacts with a "Database" and "Storage" component.

So, the situation is the where we do have a relational database design it is populated with the required data, but how about the interaction with the user incidentally most of the you database users do not interact with the database or query the database using language like SQL because as you have seen it is not a very friendly language and it is not presentable in a way which I would we would always expect or we would like.

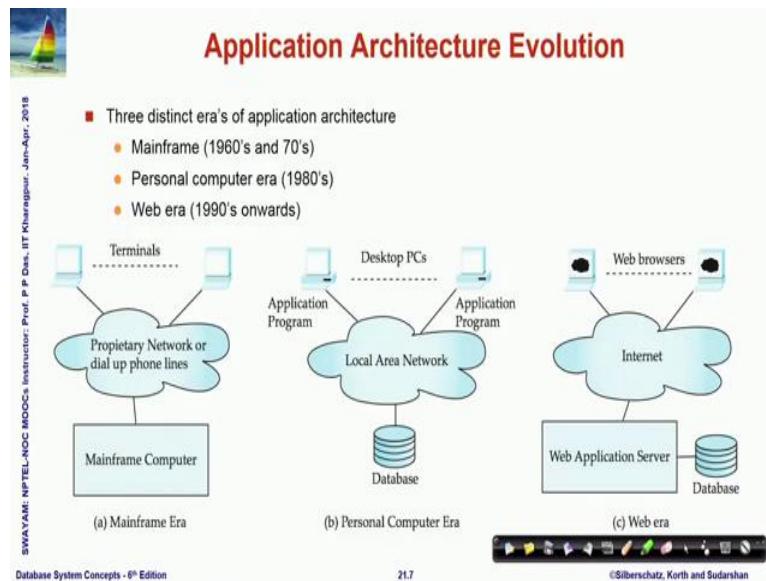
So, usually an application program acts as a as we say as an intermediately between the user and the database. And it is often split into three layers as we will say frontend middle layer and backend. So, on the right I have shown these three layers we will talk more about the this is just a representative diagram. So, the frontend is the user interface it is also called the presentation layer. So, it is the layer where it is the part of the application where there are forms GUIs and different ways to input as well as get output of the data.

Which is directly interacting with the user then we have a middle layer or its also called the business logic layer or the application layer where the functionality of the application the required operations of the or desired behavior of the application are coded and it is kind of acts as a link between the frontend and the backend and what is the backend?.

Backend is an actual data access layer or it is where the persistent data the database that we have created exist it is typically large in volume need sufficient access and so, on. So, in this module we will try to understand as to how such what are the different

requirements and what are the different technologies involved in creating such a layered application.

(Refer Slide Time: 03:55)



Now, if we historically look at; so, here we are just showing three phases initially 60s and 70s where the first database applications started then the interaction used to be takes based from the terminal.

And those to directly connect to the main frame computer where the data existed through either a direct connection dial up phone or proprietary network. And as we move to the 80s; then we saw the advent of local area networks to application programs or desktop would interact to with the database through these local area networks. And beyond that we have the what we call the web era which is 1990 onwards we in the that is that is about roughly the last 30 years where typically the applications are now based on web browsers.

So, the frontend where we actually interact are web browsers and that connect to the web application server the database everything through an internet. And I must tell you at this point that when you say this is the architecture it does not necessarily mean that the cloud shown as internet will have to be the web, it could be an internet which is created with a set of systems within your organization which we typically call as an intranet or couple of organizations across.

Which we say are extranet or it could even be a set of systems which are connected through the internet protocol within your lap or it could even be a single computer in which all these layers are integrated together, but by internet we mean it is a internet protocol and technologies will be used for doing this interaction.

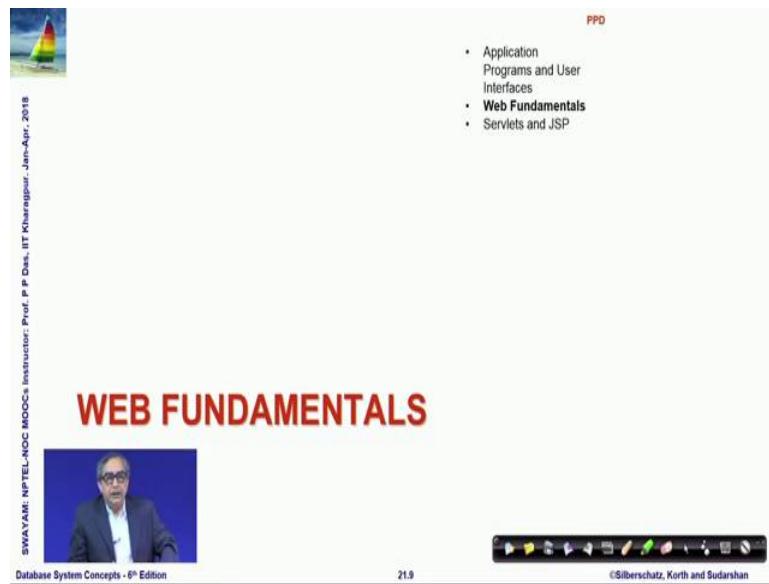
(Refer Slide Time: 05:52)

The slide has a title 'Web Interface' at the top right. On the left, there is a small image of a sailboat on water. The main content area contains two bulleted lists under red square bullet points. The first list discusses web browsers as the de-facto standard user interface to databases, mentioning their ability to enable large numbers of users to access databases from anywhere, avoid the need for specialized code, and run scripts like Javascript and Flash transparently. It also lists examples such as bank, airline, and rental car reservations, university course registration, and grading. The second list discusses mobile interfaces in mobile apps, noting they are getting popular, similar to web architecture but with significant differences, and will be discussed later. At the bottom left is a video thumbnail showing a person speaking. The footer includes the text 'SWAYAM: NPTEL-NOC MOOCs Initiator: Prof. P P Desai, IIT Kharagpur - Jan-Apr - 2018', 'Database System Concepts - 8th Edition', '21.8', and '©Silberschatz, Korth and Sudarshan'.

So, web interface has become the de-facto standard which gives a very distributed access to the database enables large number of users to access together. And it avoids the requirement of downloading or installing specialized code into that all that we need is just a web browser. And we have seen we are living through a variety of applications which are of this kind the banking application, the airline and railway reservations car rental hotel booking or web mail systems we will check mail in Gmail or Yahoo those are all different web interfaces through which we actually access a the required set of databases.

And every even every enterprise operations the ERP are now web based and that is become a de facto standard. So, in the web interface along with a web interface what has been imagined of let of the last about 10 years are mobile interfaces that we are getting use to using such applications from our mobile phone or tablet. And these are similar in architecture and workflow as of the web application, but there are significant differences to and at a later point in the next module, we will discuss about the specific requirements of mobile apps in this context as well.

(Refer Slide Time: 07:27)



PPD

- Application Programs and User Interfaces
- **Web Fundamentals**
- Servlets and JSP

## WEB FUNDAMENTALS



SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr., 2018

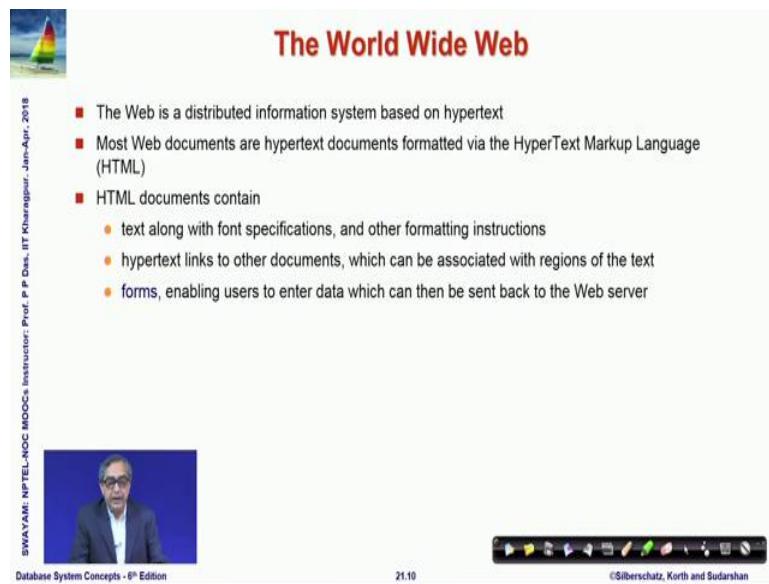
Database System Concepts - 8<sup>th</sup> Edition

21.9

©Silberschatz, Korth and Sudarshan

So, before we move forward in terms of the details of how to build these applications; we need to familiarize ourselves with the basic notions of the web as such. So, we call them as web fundamentals.

(Refer Slide Time: 07:43)



## The World Wide Web

- The Web is a distributed information system based on hypertext
- Most Web documents are hypertext documents formatted via the HyperText Markup Language (HTML)
- HTML documents contain
  - text along with font specifications, and other formatting instructions
  - hypertext links to other documents, which can be associated with regions of the text
  - forms, enabling users to enter data which can then be sent back to the Web server



SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

21.10

©Silberschatz, Korth and Sudarshan

So, web is a distributed information system which is based on hyper text a hyper text is one where you have a part of the text available to you and then you have links we say our hyper links which can connect to the different documents contents.

Which are located at other places at other servers and typically most web documents are hyper text documents which are formatted in terms of what we know as HTML Hyper Text Markup Language; you will be familiar with that I am sure. So, they contain text and along with that they can have other components like images, video, the text as specifications for font colors style all that. And in addition there are forms which can be used to enter data and send them back to the web server.

(Refer Slide Time: 08:35)

**Uniform Resources Locators**

In the Web, functionality of pointers is provided by Uniform Resource Locators (URLs).

URL example:

<http://www.acm.org/sigmod>

- The first part indicates how the document is to be accessed (protocol)
  - "http" indicates that the document is to be accessed using the Hyper Text Transfer Protocol.
- The second part gives the unique name of a machine on the Internet
- The rest of the URL identifies the document within the machine

The local identification can be:

- The path name of a file on the machine: A file at C:/WINDOWS/media/Alarm01.wav of local machine can be accessed as:
  - <file:///C:/WINDOWS/media/Alarm01.wav>
  - <file:///localhost/c:/WINDOWS/media/Alarm01.wav>
- An identifier (path name) of a program, plus arguments to be passed to the program: Searching google.com with 'silberschatz' has the url:
  - <http://www.google.com/search?q=silberschatz>

Database System Concepts - 8<sup>th</sup> Edition      21.11      ©Silberschatz, Korth and Sudarshan

Now, naturally when we operate on the web we need to have the functionality of pointing to different resources and this is done by what is known as URL or uniform resource locator. So, that is a URL is a procedure to which you can identify and point to a certain specific location of content. So, here I am showing an example of such a URL all of you be familiar with URLs.

But just to look into the different components the first component http : this http is actually a tells us the way the content would be accessed and this is typically called a protocol http its stands for hyper text transfer protocol which allows you different text to be accessed.

The second component in this URL which is between the two forward slash and the next slash www [dot] acm [dot] org identifies uniquely identifies a machine on the internet you will understand this is the symbolic name and the actual machine has what is known as an IP address.

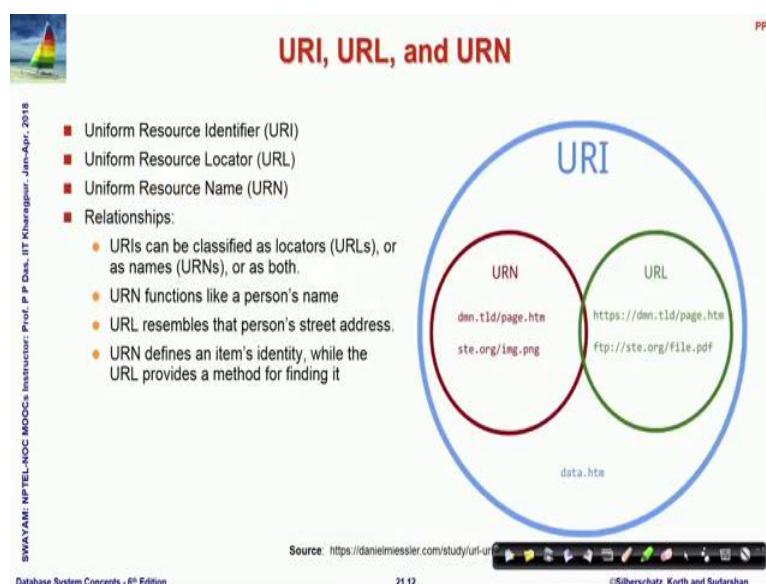
And we will not go into those details, but for us it is enough to understand that www [dot] acm [dot] org here is uniquely relatable to a particular machine on the internet. And the last part which is remaining is unique and if there are more and more parts, then it identifies the document inside within that machine. So, URL can be used in a multiple other ways also.

For example I can use URL locator to specify a file in my machine for example, I have shown an example of an AVI file in my C drive in windows and that is done through a similar URL where the protocol is not http the protocol is file telling me that it is actually residing in my local machines.

So, I have shown two ways to look at that and you will see that between the two forms; if you look at the second form you can easily understand that the machine to be identified is called the local host. And in the first form that part is missing because it is by default the machine where I am running this code and rest of it is same which is basically identifying the document to be located in that machine.

Similarly, this such URL can also in the last example you can see that www [dot] Google [dot] com is the basic machine where I am putting the URL and then the rest of it is search.

(Refer Slide Time: 11:56)



So, which actually takes it to a document where I tell the search to be performed and then there are parameters to this form the parameter is q is equal to silberschatz which is equivalent to same that I am asking Google [dot] com to search for contents which have silberschatz in it. So, this is the basic purpose of the uniform resource locator or URL incidentally you may have heard the names like URI and URN in addition to URL.

So, they are related, but they mean little bit different things as this Venn diagram shows. So, a URI can be either a URL or a URN or it could be both. So, URN functions like a person's name; so, you can conceive it that way universal resource name and URL resembles that of a person's street address. So, URN says what is the name of the content and URL says where that can be found. And URI in general could be either the name or the address or both of them.

In this context of our discussion we will continue to use the term URL only, but I just wanted you to be aware of the other two terms in case you come across them in the text.

(Refer Slide Time: 12:53)

**HTML and HTTP**

- HTML provides formatting, hypertext link, and image display features
  - including tables, stylesheets (to alter default formatting), etc.
- HTML also provides input features
  - Select from a set of options
    - ▶ Pop-up menus, radio buttons, check lists
  - Enter values
    - ▶ Text boxes
  - Filled in input sent back to the server, to be acted upon by an executable at the server
- HyperText Transfer Protocol (HTTP) used for communication with the Web server

SWAYAM-NIITEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr- 2018

Database System Concepts - 8<sup>th</sup> Edition

21.13

©Silberschatz, Korth and Sudarshan

So, we know by now that http HTML provides the formatting the hyper text links images and so, on and http provides the protocol through which the contents are exchanged between different machines in the internet. So, you can select from a set of options in terms of a HTML pop-up menus, radio buttons, check boxes and so, on; you can enter values to text box and once a form has been filled up that form will be sent back to the

server from where it came and would be acted upon by the server http helps in that transfer mechanism.

(Refer Slide Time: 13:39)

The slide title is "Sample HTML Source Text". It contains the following HTML code:

```
<html>
<body>
<table border>
<tr> <th>ID</th> <th>Name</th> <th>Department</th> </tr>
<tr> <td>00128</td> <td>Zhang</td> <td>Comp. Sci.</td> </tr>
...
</table>
<form action="PersonQuery" method="get">
Search for:
<select name="persontype">
<option value="student" selected>Student </option>
<option value="instructor"> Instructor </option>
</select> <br>
Name: <input type="text" size=20 name="name">
<input type="submit" value="submit">
</form>
</body> </html>
```

On the left, there is a vertical watermark: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018". At the bottom, it says "Database System Concepts - 6<sup>th</sup> Edition" and "21.14 ©Silberschatz, Korth and Sudarshan".

So, the here is a sample HTML code let me show you the effect of this in the next slide.

(Refer Slide Time: 13:46)

The slide title is "Display of Sample HTML Source". It shows the rendered output of the HTML code. On the left, there is a table:

ID	Name	Department
00128	Zhang	Comp. Sci.
12345	Shankar	Comp. Sci.
19991	Brandt	History

Below the table is a search interface with fields for "Search for:" (containing "Student"), "Name:" (containing "Zhang"), and a "submit" button. A video player window shows a person speaking. Red arrows point from the rendered table and search form back to the corresponding parts in the original HTML code on the right.

On the left, there is a vertical watermark: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018". At the bottom, it says "Database System Concepts - 6<sup>th</sup> Edition" and "21.14 ©Silberschatz, Korth and Sudarshan".

So, if you look in here then you can see that these are this is what is known as a tag `<>` and `</>` it kind of the tags are kind of given in the form of `( )` notation.

So, it has a opening and a closing and these have to have to actually match and you can see that the opening is written within corner brackets and the closing is write the same tag name, but you put a forward slash before the name. And then between the closing opening and the closing you can write more tabs in a nested manner.

So, here it says that I have HTML which has a body and the body expanse this much and then we are saying that there is a table. So, this is the table that you can get to see and then it also says that there is a form and this is the form that you get to see within the table you can see. So, it is saying that this is an ID there is a name; so, it is describing the first row the department. So, you can see each one of them the ID is here the name is here the department is here similarly this is a next row where it is saying it is 0 1; 00128 Zhang Computer Science.

So, this is how you can you actually in an HTML in a text form all these details will be given and when it is rendered by the web browser then you will see a table like this. Similarly here we are I am showing a an instance of a form which is use to input data. So, we are saying that here is a drop down and it is written out here in terms of options.

So, the first options student is visible here if you drop down you will actually see another option instructor here you will not see that because it is a frozen image. So, and then I have a qualifier name and there is a input text box where you can input any strain up to size 20. And once this has been done then you have an input which is submit, which is the submit button here which shows that you can now submit and then this form filled up form will be sent back to the web browser from where it originally came.

(Refer Slide Time: 16:23)

The slide has a header 'Web Servers' in red. On the left is a small sailboat icon. The main content is a bulleted list:

- A Web server can easily serve as a front end to a variety of information services
- The document name in a URL may identify an executable program, that, when run, generates a HTML document
  - When an HTTP server receives a request for such a document, it executes the program, and sends back the HTML document that is generated
  - The Web client can pass extra arguments with the name of the document
- To install a new service on the Web, one simply needs to create and install an executable that provides that service
  - The Web browser provides a graphical user interface to the information service
- Common Gateway Interface (CGI): a standard interface between web and application server

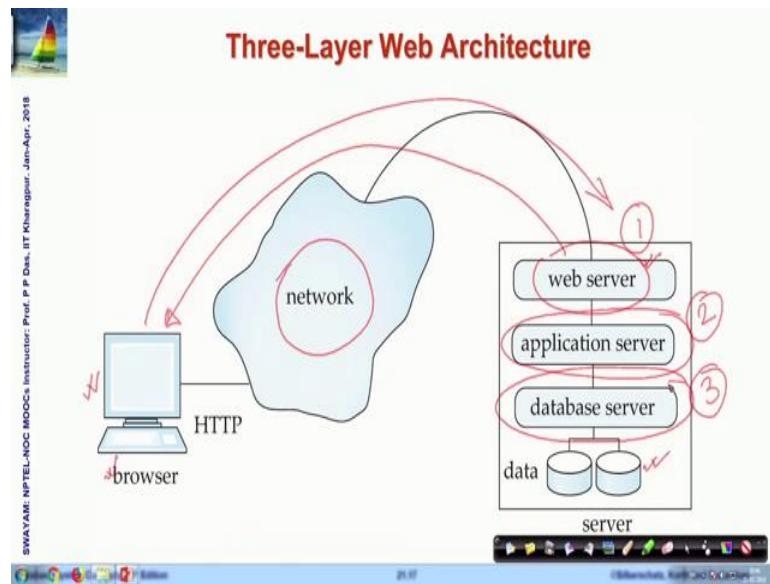
On the right side of the slide, there is a video frame showing a man speaking, a navigation bar with icons, and the text '©Silberschatz, Korth and Sudarshan'.

So, this is the basic mechanism of HTML you can learn little bit more about that and get familiar with it. So, a document name in a URL may identify a program that is written that generate. So, HTML could be either at the URL in the web server you can either have a HTML which we say is a static HTML or you could actually have a program which when you send the request it actually.

For example, when you are doing Google you said [www.google.com/search?q=silberschatz](http://www.google.com/search?q=silberschatz). Then actually at that location there is no HTML currently existing which contains the search result, but instead there is a program which will be executed based on the submission of this form and when run that will generate a HTML document. And once that is generated then this will be passed back to the to your web browser. So, that is a basic mechanism.

So, if you want a new service on the web then you all that you simply need to do is to create and install a new program that will provide that service and through this process we will see how easily this can be done and how web browser provides a graphical interface to this information service. There is there has been another other mechanisms of doing similar things also which was particularly popularly are called the common gateway interface or CGI, but now we have various other ways of doing the same thing.

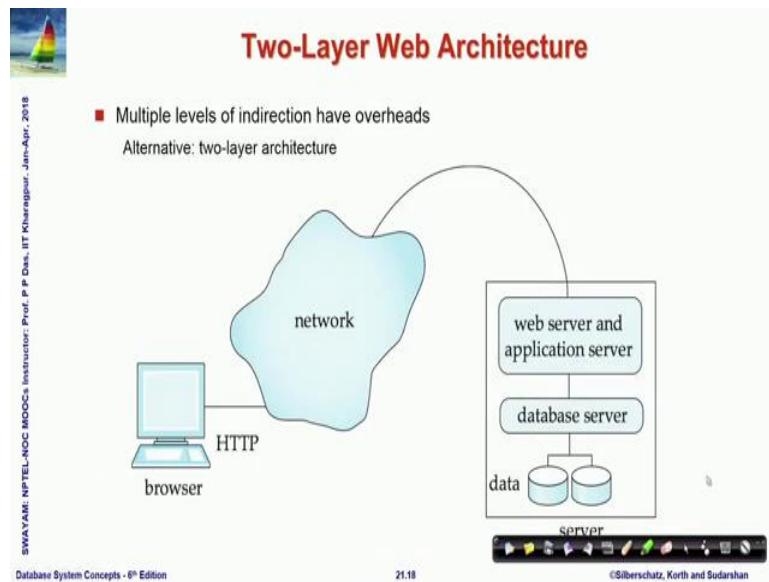
(Refer Slide Time: 18:03)



So, in this context then the basic three layers that we started discussing with are here. So, this is the most common way an application is. So, this is where your frontend is where you have the browser where you see that this is the network; we are just in general writing network it could be internet it could be intranet and a web server. So, when you send a request this is received by the server; web server somehow computes a result HTML and that send back to the browser and that is how the interaction keeps on happening.

So, the browser and the web server together often would be referred to as a frontend because that gives a presentation that presents the results the interaction to you. The next is the application layer which is the business logic where you write and then you have the data access layer or the database server and these are the actual disk where the data exist. So, this is a tier 1 this is tier 2 and this is tier 3 which is a very typical way a web application will be architected and these are the three layers or three tiers that we will usually find.

(Refer Slide Time: 19:22)



So, often actually three is not a very magical number in terms of tiers; it is possible that you could have more some applications have more tiers. And some applications may choose to have multiple functionality in the same layer for example, web server and application server functionality could be clubbed together and when this is done we say that we will then we have only two layers. So, the frontend and the middle layer are merge together and the backend or the database server becomes a second layer.

So, we would often might want to do that the reason we do that I will just take to back to the three layer view. So, if the question is naturally if we have the web server and we have these connections this is clear; now the question is what is this connection and what is this connection? Is it necessary that they will have to be on the same server physically or will they be on can be on different server and servers could be connected through a LAN or they themselves could be on different servers over the internet and may they may be connected through internet.

So, all of these are possibilities and the way we connect is the way we will write the application will be will not depend on the way these servers are connected between each other we will often assume that as if they are connected over a net and write it in a way so, that even when they may be connected over a LAN or even when they may actually be on the same machine things will work in the same way.

(Refer Slide Time: 20:57)

The slide has a title 'HTTP and Sessions' at the top right. On the left, there is a small logo of a sailboat on water. The main content area contains a bulleted list under several sections:

- The HTTP protocol is **connectionless**
  - That is, once the server replies to a request, the server closes the connection with the client, and forgets all about the request
  - In contrast, Unix logins, and JDBC/ODBC connections stay connected until the client disconnects
    - retaining user authentication and other information
  - Motivation: reduces load on server
    - operating systems have tight limits on number of open connections on a machine
- Information services need session information
  - E.g., user authentication should be done only once per session
- Solution: use a **cookie**

At the bottom left, it says 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr- 2018'. At the bottom center, it says 'Database System Concepts - 8<sup>th</sup> Edition' and '21.19'. At the bottom right, it says '©Silberschatz, Korth and Sudarshan'.

Now, one point that should be born in mind in terms of the http protocol is it is connectionless. So, this is a very very critical concept and it means that once I start the process I have an URL; I submit that and that goes to the server the server runs an application or it is a static page server picks up and returns me that HTML; the http loop is closed.

So, that is the all that happens and when I submit again something based on that. So, I have gone to Gmail **mail.google.com** I submit that and I get back a form which tells me to put my user ID and login I do that I submit and that when that goes then there is no memory about the earlier interaction that has happen.

So, when my login submission goes it is authenticated in the backend I am able to login and I am given back the first screen of my mail box which is the inbox screen. And as soon as I get that inbox the HTML containing the inbox on my browser that transaction has also been over. So, if I now want to look at a specific mail it has to be a new query and it is not remember anything from the previous query.

So, this connectionless property naturally makes it makes certain things more difficult; you will you will realize that many of the other connections that we do for example, if we login to UNIX system or to a window system if we use some database connections they are connected till the we disconnect them, but in http it is not. So, it is connectionless every time you do you have a separate session. So, naturally the question

this was I mean the there are there are reasons of why this is done this way this is to reduce load from the server and so, on.

But naturally the consequences therefore, we cannot remember information from one request response loop to the next. So, if I have logged in to my mail Gmail account and I have seen the I have got the inbox then when I want to check my first mail the system does not know any more that I am logged in because that session request response has is over and now I am making a new request that show me the first mail and I expect to see the whole body.

So, there is no information that is carried from one request to the other which makes http difficult to work with. So, the solution for that is something which is known as a cookie.

(Refer Slide Time: 23:46)

The slide has a header 'Sessions and Cookies' with a sailboat icon. The main content is a bulleted list about cookies:

- A **cookie** is a small piece of text containing identifying information
  - Sent by server to browser
    - ↳ Sent on first interaction, to identify session
    - Sent by browser to the server that created the cookie on further interactions
      - ↳ part of the HTTP protocol
    - Server saves information about cookies it issued, and can use it when serving a request
      - ↳ E.g., authentication information, and user preferences
  - Cookies can be stored permanently or for a limited time

At the bottom left is a video thumbnail of a professor, and at the bottom right is a navigation bar.

So, a cookie is a small piece of text which contain information which is identifying and which can go back and forth between the browser and the server. So, first it is sent by the sever to the browser and the what the browser does.

So, this happens the first time. So, when we logged in my browser has got a got some cookie from the mail Gmail server. Then the browser can send it back to the server when it is doing the next request so, that I can be identified as a logged in person and. So, the browser can keep it as a I mean locally in its memory or locally here and that is a part of the http protocol.

So, this keeps on this cookie keeps on going back and forth back and forth. So, every time I send a request the cookie actually has to go to tell the server that yes this is the Partha Pratim Das who is already logged in an authenticated himself for checking his mails. So, cookies are a big convenience and they are very important factor of the web applications. So, they can be stored permanently or for a limited period of time.

(Refer Slide Time: 25:04)

The slide has a title 'Servlets' in red at the top right. On the left is a small image of a sailboat. The main content area contains a bulleted list:

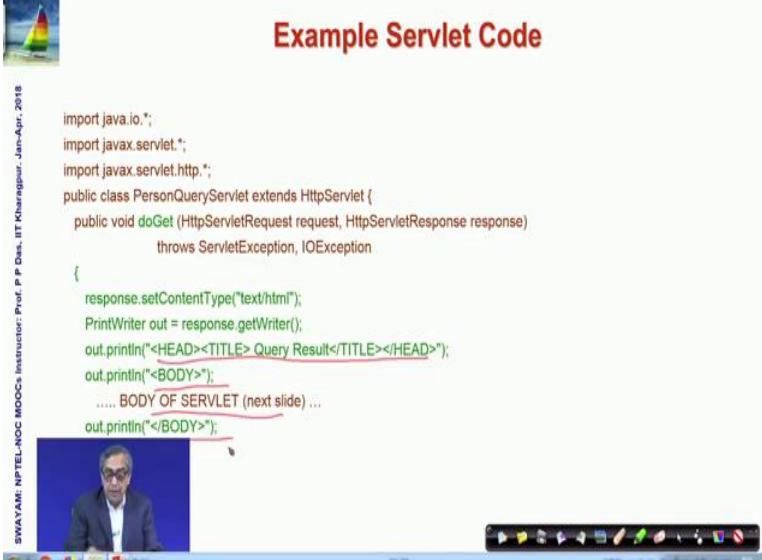
- Java Servlet specification defines an API for communication between the Web/application server and application program running in the server
  - E.g., methods to get parameter values from Web forms, and to send HTML text back to client
- Application program (also called a servlet) is loaded into the server
  - Each request spawns a new thread in the server
    - thread is closed once the request is serviced

At the bottom left is a video frame showing a man speaking, with the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr- 2018'. At the bottom center is the text 'Database System Concepts - 8<sup>th</sup> Edition' and '21.22'. At the bottom right is a navigation bar with icons for back, forward, search, etc., and the text '©Silberschatz, Korth and Sudarshan'.

Next let us look into some of the core technologies that are involved the first technology is called a servlet. Servlet is nothing but as you can understand from the name itself is as you have book booklet booklet is a small very small book servlet is a very small server.

So, it is a Java application which can do certain tasks; so, it is an kind of an application program and every time we request then the server actually spawns a new thread and in that thread this servlet would be running and once the request is serviced the thread will be closed.

(Refer Slide Time: 25:42)

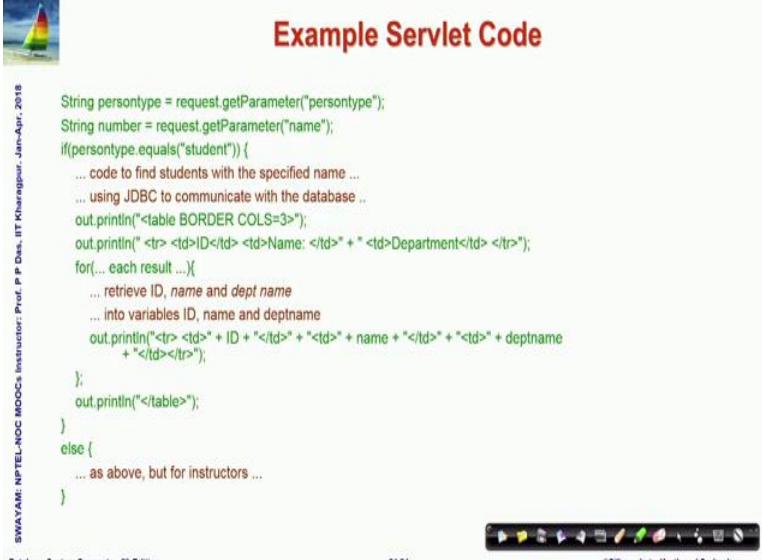


The screenshot shows a Java code editor with the title "Example Servlet Code". The code is a Java servlet named PersonQueryServlet. It imports java.io, javax.servlet, and javax.servlet.http. The doGet method prints an HTML response with a title and body sections. A watermark on the left reads "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018". On the right, there is a video player showing a man in a suit speaking, and a toolbar at the bottom.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class PersonQueryServlet extends HttpServlet {
    public void doGet (HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HEAD><TITLE>Query Result</TITLE></HEAD>");
        out.println("<BODY>");  
..... BODY OF SERVLET (next slide) ...
        out.println("</BODY>");
```

So, this is the typical server servlet view. So, which shows that in the servlet you are creating actually creating the requested I mean possible HTML response that you would like to have.

(Refer Slide Time: 25:58)



The screenshot shows a Java code editor with the title "Example Servlet Code". The code is a Java servlet with logic to handle student or instructor requests. It uses JDBC to query a database and prints the results as an HTML table. A watermark on the left reads "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018". On the right, there is a video player showing a man in a suit speaking, and a toolbar at the bottom.

```
String persontype = request.getParameter("persontype");
String number = request.getParameter("name");
if(persontype.equals("student")){
    ... code to find students with the specified name ...
    ... using JDBC to communicate with the database ...
    out.println("<table BORDER COLS=3>");
    out.println(" <tr> <td>ID</td> <td>Name: </td> <td>Department</td> </tr>");
    for(... each result ...){
        ... retrieve ID, name and dept name
        ... into variables ID, name and deptname
        out.println("<tr> <td>" + ID + "</td>" + <td>" + name + "</td>" + <td>" + deptname
        + "</td></tr>");
    }
    out.println("</table>");
}
else {
    ... as above, but for instructors ...
}
```

So, there are; so, there are different details you can read through that. So, this is the typical servlet code. So, it actually is a Java code which through print line will generate different lines of the HTML. Now naturally servlets maintain session the way we talked about.

(Refer Slide Time: 26:11)



## Servlet Sessions

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018

- Servlet API supports handling of sessions
  - Sets a cookie on first interaction with browser, and uses it to identify session on further interactions
- To check if session is already active:
  - if (`request.getSession(false) == true`)
    - .. then existing session
    - else .. redirect to authentication page
  - authentication page
    - check login/password
    - `request.getSession(true)`: creates new session
- Store/retrieve attribute value pairs for a particular session
  - `session.setAttribute("userid", userid)`
  - `session.getAttribute("userid")`

Database System Concepts - 8<sup>th</sup> Edition 21.25 ©Silberschatz, Korth and Sudarshan



So, that through an interaction I can continue to be identified and the servlet can check whether the session is on or the session is already over. So, these are the different ways of doing that in terms of shaking the user ID and several web servers application servers have support for servlet apache tomcat is one of the very popular one.

(Refer Slide Time: 26:33)



## Servlet Support

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018

- Servlets run inside application servers such as
  - Apache Tomcat, Glassfish, JBoss
  - BEA Weblogic, IBM WebSphere and Oracle Application Servers
- Application servers support
  - deployment and monitoring of servlets
  - Java 2 Enterprise Edition (J2EE) platform supporting objects, parallel processing across multiple application servers, etc



Database System Concepts - 8<sup>th</sup> Edition 21.26 ©Silberschatz, Korth and Sudarshan



So, which you must have heard the name of and there are, but there are several other servers as well.

(Refer Slide Time: 26:48)

The slide has a header 'Server-Side Scripting' in red. On the left is a small sailboat icon. The main content lists two bullet points:

- Server-side scripting simplifies the task of connecting a database to the Web
  - Define an HTML document with embedded executable code/SQL queries.
  - Input values from HTML forms can be used directly in the embedded code/SQL queries.
  - When the document is requested, the Web server executes the embedded code/SQL queries to generate the actual HTML document.
- Numerous server-side scripting languages
  - JSP, PHP
  - General purpose scripting languages: VBScript, Perl, Python

At the bottom, there's a footer with the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P.P. Desai, IIT Kanpur - Jan-Apr. 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '21.27', and '©Silberschatz, Korth and Sudarshan'.

Now, along with the servlet there is another concept which is called server side scripting. server side scripting is a mechanism where you define an HTML document and to within that HTML document can be used. So, you may have some inputs to that and they can be used to directly fire embedded SQL queries.

So, we talked about a madding of SQL query in while we discuss about the basic mechanism of host language and query language. So, here the HTML kind of a language is a host and you can embed the query right in as a part of that. And so, that query goes to the database query server and you get the answer and that answer is placed where your original query was there. So, that you continue to get very easily a my complete HTML as a response.

So, this kind of a mechanism is makes it very easy because a it is quite easy to conceive of the HTML and fill in. So, if I have asked for say logged in to the my mail Gmail service then I have given the input as my user name password and when that got gets authenticated. Then I get a response which is select mail from different respective tables where my authentication is there the user name is PPD and so, on. So, it becomes quite easy to actually create the HTML and there several such scripting language is JSP and PHP are the most popular ones.

(Refer Slide Time: 28:35)

The slide features a small sailboat icon in the top left corner. The title 'Java Server Pages (JSP)' is centered at the top in a red font. Below the title, there is a list of bullet points and some code snippets.

- A JSP page with embedded Java code

```
<html>
<head> <title> Hello </title> </head>
<body>
<% if (request.getParameter("name") == null)
{ out.println("Hello World"); }
else { out.println("Hello, " + request.getParameter("name")); }
%>
</body>
</html>
```

- JSP is compiled into Java + Servlets

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur Date: Jan-Apr- 2018

Database System Concepts - 8<sup>th</sup> Edition 21.28 ©Silberschatz, Korth and Sudarshan

So, this is how a typical JSP will look like. So, you can see that this actually looks like HTML, but inside that you have a, you have some part of a Java code. So, what will happen the body will get replaced when the when the response has come for example, here the response is doing hello world.

So, when this is executed then whatever is a result will replace the body in the HTML here and the result in the HTML will get generated. There is another mechanism of scripting which is also very popular called PHP. So, this is how it is done.

(Refer Slide Time: 29:19)

The slide features a small sailboat icon in the top left corner. The title 'Client Side Scripting' is centered at the top in a red font. Below the title, there is a list of bullet points and some descriptive text.

- Browsers can fetch certain scripts (client-side scripts) or programs along with documents, and execute them in "safe mode" at the client site
  - Javascript
  - Macromedia Flash and Shockwave for animation/games
  - VRML
  - Applets
- Client-side scripts/programs allow documents to be active
  - E.g., animation by executing programs at the local site
  - E.g., ensure that values entered by users satisfy some correctness checks
  - Permit flexible interaction with the user.
    - Executing programs at the client site speeds up interaction by avoiding many round trips to server

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur Date: Jan-Apr- 2018

Database System Concepts - 8<sup>th</sup> Edition 21.30 ©Silberschatz, Korth and Sudarshan

Similarly, you could have script an client side also for example, if you are entering data for say a month and in numeric and you happened to enter 14; then in most cases the page will immediately give a error saying that 14 cannot be a valid month; so, there is a validation involved.

So, in the client side in the browser there are some small script that can run a most typically it is a Java script which can too different authentication which is possible without actually accessing the data in the database. You cannot for example, validate a mail data based on the client side scripting sitting on the browser, but you can validate small things like valid data forms, range of data and so, on.

And it is it is very important because if you could not do that then all you required is you would have send that faulty month numbered 14 to the to the backend server and got an error and you would have come back and then have to correct it, but you can do this locally at the browser itself.

(Refer Slide Time: 30:26)

The slide has a header 'Client Side Scripting and Security' with a sailboat icon. The main content lists security mechanisms for client-side scripts:

- Security mechanisms needed to ensure that malicious scripts do not cause damage to the client machine
  - Easy for limited capability scripting languages, harder for general purpose programming languages like Java
- E.g., Java's security system ensures that the Java applet code does not make any system calls directly
  - Disallows dangerous actions such as file writes
  - Notifies the user about potentially dangerous actions, and allows the option to abort the program or to continue execution.

Navigation icons and footer text are visible at the bottom.

So, client side scripting has a lot of value, but you will have to have to remember that a if you are doing client side scripting then there are security issues.

Because it is quite possible that if you are doing things on the client side that is on the browser then we might also inadvertently or by a malicious intact actually make damages to the machine on which the browser is running. So, there are different kinds of care that

is to be taken for example, Java applet which is another way of doing client side computation disallows file writes and so, on.

(Refer Slide Time: 31:03)

The slide features a small sailboat icon in the top-left corner. The title 'Javascript' is in red at the top-right. The main content is a bulleted list under two main points:

- Javascript very widely used
  - forms basis of new generation of Web applications (called Web 2.0 applications) offering rich user interfaces
- Javascript functions can
  - check input for validity
  - modify the displayed Web page, by altering the underlying **document object model (DOM)** tree representation of the displayed HTML text
  - communicate with a Web server to fetch data and modify the current page using fetched data, without needing to reload/refresh the page
    - ▶ forms basis of AJAX technology used widely in Web 2.0 applications
    - ▶ E.g. on selecting a country in a drop-down menu, the list of states in that country is automatically populated in a linked drop-down menu

At the bottom left, it says 'Database System Concepts - 8<sup>th</sup> Edition'. In the bottom center, it shows '21.32'. At the bottom right, it says '©Silberschatz, Korth and Sudarshan'.

And Java script as I have said is widely used and it can function to check for input validity which I gave an example of it can modify the displayed web page, it can communicate with the web server to fetch data and so, on. And it is you should familiarize yourself with Java script more; it is a very powerful mechanism to do compute the sample things at the client side this is an example that I have given.

(Refer Slide Time: 31:20)

The slide features a small sailboat icon in the top-left corner. The title 'Javascript' is in red at the top-right. The main content is a bulleted list under one point:

- Example of Javascript used to validate form input

```
<html> <head>
<script type="text/javascript">
function validate() {
    var credits=document.getElementById("credits").value;
    if (isNaN(credits)|| credits<=0 || credits>=16) {
        alert("Credits must be a number greater than 0 and less than 16");
        return false
    }
}
</script>
</head> <body>
<form action="createCourse" onsubmit="return validate()">
    Title: <input type="text" id="title" size="20"><br />
    Credits: <input type="text" id="credits" size="2"><br />
    <input type="submit" value="Submit">
</form>
</body> </html>
```

At the bottom left, it says 'Database System Concepts - 8<sup>th</sup> Edition'. In the bottom center, it shows '21.33'. At the bottom right, it says '©Silberschatz, Korth and Sudarshan'.

So, you can read through and you will be able to if you know Java you will be able to understand Java script very easily, you could get through that.

(Refer Slide Time: 31:33)

The slide has a header 'USP of JSP' with a small sailboat icon to its left. The content is organized into four main sections: 'JSP vs. Active Server Pages (ASP)', 'JSP vs. Pure Servlets', 'JSP vs. JavaScript', and 'JSP vs. Static HTML'. Each section contains a bulleted list of pros or comparisons. The footer includes the text 'SWAYAM: NPTEL-NOCO MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr- 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '21.34', and '©Silberschatz, Korth and Sudarshan' along with a navigation bar.

- **JSP vs. Active Server Pages (ASP)**
  - ASP is a similar technology from Microsoft and is proprietary (uses VB).
  - JSP is platform independent and portable.
- **JSP vs. Pure Servlets**
  - JSP is a servlet but it is more convenient to write and to modify regular HTML than to have a million println statements that generate the HTML.
  - The Web page design experts can build the HTML, leaving places for the servlet programmers to insert the dynamic content.
- **JSP vs. JavaScript** JavaScript can generate HTML dynamically on the client.
  - "Client Side": Java Script code is executed by the browser after the web server sends the HTTP response. With the exception of cookies, HTTP and form submission data is not available to JavaScript.
  - "Server Side": Java Server Pages are executed by the web server before the web server sends the HTTP response. It can access server-side resources like databases, catalogs.
- **JSP vs. Static HTML** Regular HTML, of course, cannot contain dynamic information.

And so, there are multiple options of doing such things, but JSP has a unique position because it is very useful in many contexts. A JSP has certain USP like active server pages which are used in terms of the Microsoft platform is also another mechanism of doing server side scripting, but JSP is better because it is portable in comparison to pure servlets which we showed you in the beginning. JSP performs easier to use.

Because JSP has the structure of the HTML page whereas, in a pure servlet we will have to use print line to print every tag of the HTML which is cumbersome. JSP in contrast with Java script is certainly a different thing because Java script runs on the browser on the client side, JSP runs on the server side and certainly JSP is compared to static HTML is more powerful because it can handle dynamic information.

(Refer Slide Time: 32:39)

The slide is titled "Module Summary" in red font at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list of learning objectives:

- Understood the requirements of Database Application Programs and User Interfaces
- Familiarized with the Fundamentals notions and technologies of Web
- Learnt the notions of Servlets and Java Server Pages

At the bottom left, it says "Database System Concepts - 8<sup>th</sup> Edition". In the center bottom, it shows "21.35". At the bottom right, it says "©Silberschatz, Korth and Sudarshan". A decorative footer bar with various icons is also present.

So, in this that brings us to the end of this current modules; so, what we have done we have understood the basic requirements of database application programs and user interfaces understood the basic terminology of the web and took a look into the core notion, core technologies of application development which is in terms of the servlets and Java server pages.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 22**  
**Application Design and Development (Contd.)**

Welcome to module 22 of database management systems, we have been discussing application, design and development this is the second part of that discussion.

(Refer Slide Time: 00:27)

**Module Recap**

- Application Programs and User Interfaces
- Web Fundamentals
- Servlets and JSP

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018  
Database System Concepts - 8<sup>th</sup> Edition      22.2      ©Silberschatz, Korth and Sudarshan

In the last module, we have taken a quick look at the application programs and the user interfaces, looked at the fundamental notions of web and specifically the servlets and JSP.

(Refer Slide Time: 00:38)



**Module Objectives**

PPD

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr - 2018

- To understand Architectures Database Applications in detail
- To explore the Rapid Application Development Process
- To understand the issues in Application Performance
- To understand the issues in Application Security
- To appreciate how Mobile Apps are similar to and different web applications

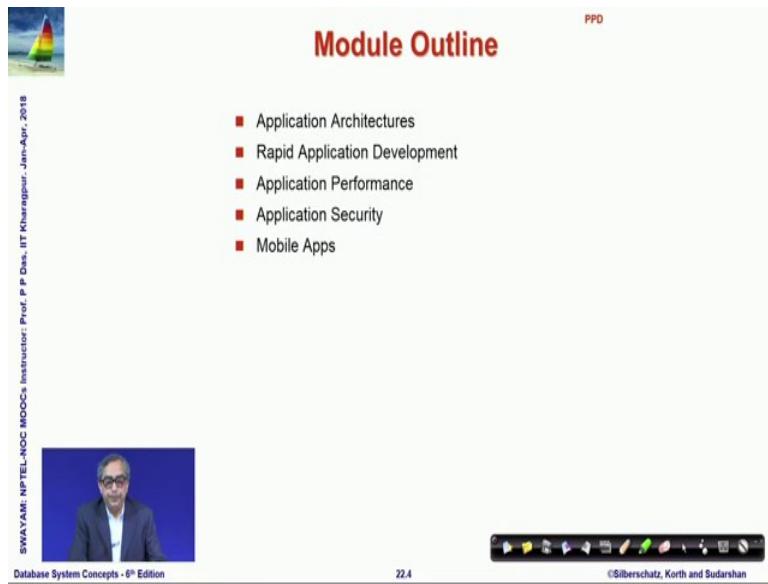
Database System Concepts - 8<sup>th</sup> Edition

22.3

©Silberschatz, Korth and Sudarshan

In the current module, we would like to understand the 3 tier architecture in little bit more detail, and explore quickly take a look into the rapid application development processes what kind of help is available, for quickly develop applications and then, we briefly we will look into the issues in terms of an applications performance and it is required security, and at the end we will discuss how a mobile app is similar to such web based database applications? And how they are different?

(Refer Slide Time: 01:14)



**Module Outline**

PPD

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr - 2018

- Application Architectures
- Rapid Application Development
- Application Performance
- Application Security
- Mobile Apps

Database System Concepts - 8<sup>th</sup> Edition

22.4

©Silberschatz, Korth and Sudarshan

So, this is the outline the 5 parts.

(Refer Slide Time: 01:19)

**Application Architectures**

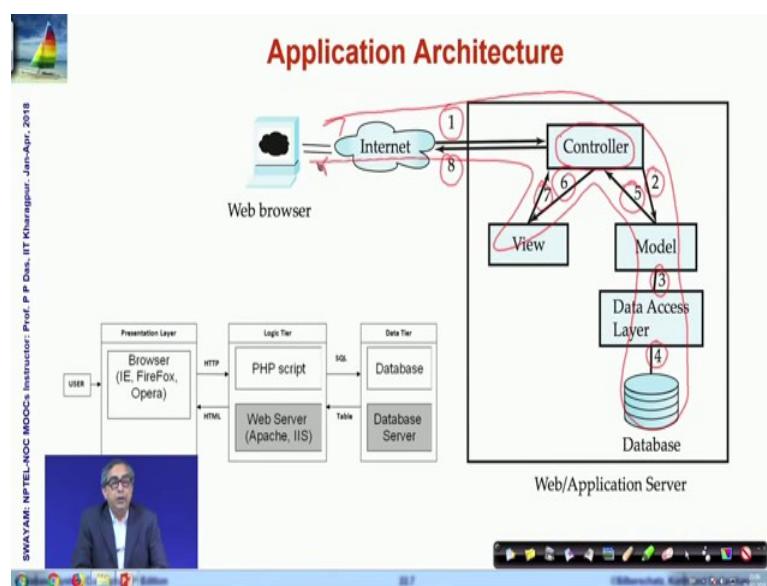
- Application layers
  - Presentation or user interface
    - **model-view-controller (MVC) architecture**
      - **model**: business logic
      - **view**: presentation of data, depends on display device
      - **controller**: receives events, executes actions, and returns a view to the user
    - **business-logic layer**
      - provides high level view of data and actions on data
      - often using an object data model
      - hides details of data storage schema
    - **data access layer**
      - interfaces between business logic layer and the underlying database
      - provides mapping from object model of business layer to relational model of database

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition      22.6      ©Silberschatz, Korth and Sudarshan

So, in terms of the application architecture again the presentation layer, or the user interface, business logic layer, and the data access layer, the frontend, the middle layer and the backend. Now, in the presentation layer or the user interface it is typical that applications follow, what is now known as MVC architecture, model view control architecture where, the model is the kind of the business logic that, that is implemented in terms of the frontend information. View is the actual presentation of the data, that HTML and controller is one who, receives different events execute actions and so on and then, we will go into the other layers.

(Refer Slide Time: 02:09)



So, here let us try to understand this flow. So, there is a request in the web browser say, to the service. So, this is the sequential now, let us say we are trying to log in to Gmail. So, in one we send a form HTML form which, has the username and the password and possibly encrypted, that comes to the controller ah. So, which basically controls the different events.

So, the controller knows that, it has to now decide whether, this what actions are required in terms of this input data. So, it is sends it to the model which, is the business logic here. So, the business logic model knows now well. So, at this there is an application which, deciphers the business logic which says that, business logic required here is we have a password and we have a user names now, we have to decide whether this user is a valid user and whether, he or she can be allowed to login. So, the model has to check on the user data, the user id and password data and therefore, it has to come from a database. So, it passes on this request to the data access layer, the data access layer in turn access the database.

So, you can think of data access layer is something like a SQL query layer where, you have formed a query select etc., from etc., where, user id is equal to PPD, password is equal to XXX, the database depending on what is found in the database is back to the model, and the model then sends it to the control it is says ok, this is what I have found. So, this is the result of the data that result of the request that has been prepared. So, it is says that well this is have been found and therefore, we have extracted the mails in the inbox that existed, or it is says that the authentication is not possible. So, it plugs in a error message and sends it to the controller, controller now knows that a response has to be framed. So, the controller sends it to the view of the MVC, the view we prepare the HTML that needs to go back. So, view prepares the HTML and sends it back to the controller. So, controller now has the response which it is sends back to the web browser through the internet, and we get to see that well now, my inbox and mails are all here.

So, this is a complete flow of starting from here, going through this, coming back, going here, going back here, is the is the whole route of the request, response that goes over the HTTP in a typical web or application scenario, that is the there is a way this application architecture is expected to work.

(Refer Slide Time: 05:13)

**Sample Applications in Multiple Tiers**

Application	Presentation	Logic	Data	Functionality
Web Mail	<ul style="list-style-type: none"> <li>• Login</li> <li>• Mail List View <ul style="list-style-type: none"> <li>• Inbox</li> <li>• Sent Items</li> <li>• Outbox</li> <li>• Trash</li> </ul> </li> <li>• Mail Composer</li> <li>• Filters</li> </ul>	<ul style="list-style-type: none"> <li>• User Authentication</li> <li>• Connection to Mail Server (SMTP, POP, IMAP)</li> <li>• Encryption / Decryption</li> </ul>	<ul style="list-style-type: none"> <li>• Mail Users</li> <li>• Address Book</li> <li>• Mail Items</li> </ul>	<ul style="list-style-type: none"> <li>• Send / Receive Mails</li> <li>• Manage Address Book</li> </ul>
Net Banking	<ul style="list-style-type: none"> <li>• Login</li> <li>• Account View</li> <li>• Add / Delete Account</li> <li>• Add / Delete Beneficiary</li> <li>• Fund Transfer</li> </ul>	<ul style="list-style-type: none"> <li>• User Authentication</li> <li>• Beneficiary Authentication</li> <li>• Transaction Validation</li> <li>• Connection to Banks / Gateways</li> <li>• Encryption / Decryption</li> </ul>	<ul style="list-style-type: none"> <li>• Account Holders</li> <li>• Beneficiaries</li> <li>• Accounts</li> <li>• Debit / Credit Transactions</li> </ul>	<ul style="list-style-type: none"> <li>• Check Balance and Transactions</li> <li>• Transfer Funds</li> </ul>
Timetable	<ul style="list-style-type: none"> <li>• Login</li> <li>• Add / Delete Courses, Teachers, Rooms, Slots</li> <li>• Assignments: <ul style="list-style-type: none"> <li>• Teachers → Course</li> </ul> </li> <li>• Allocations <ul style="list-style-type: none"> <li>• Course → Room, Slots</li> </ul> </li> <li>• Views</li> </ul>	<ul style="list-style-type: none"> <li>• User Authentication</li> <li>• Timetable Assignment Logic</li> <li>• Encryption / Decryption</li> </ul>	<ul style="list-style-type: none"> <li>• Courses</li> <li>• Teachers</li> <li>• Rooms</li> <li>• Slots</li> <li>• Assignments</li> <li>• Allocations</li> </ul>	<ul style="list-style-type: none"> <li>• Manage timetable for multiple courses taken by multiple teachers</li> </ul>

So, here I have created a small table, showing you the different activity is that happens at the presentation logic and data layer of different common applications like, web mail like, Google. So, at the presentation layer you will do things like, log in, mail list, view inbox, sent item, outbox so on, mail composer. So, we can write mails filters of checking at different mails. So, all the all these happens.

So, for example, if you talk about filters they might often happen in the java script itself, that trans within the browser, the logic the business logic will do user authentication, connection to mail server because, a mails have to come from a different server, they are not they may not be setting typed in terms of the database itself.

And then user encryption, decryption and the data side you will have different tables to represent the mail users, the users like you and me all who are users of the Gmail, the address book of each for each one of us the mail items and so on, and that will give us the functionality of send, receive mails, managing address book and so on. So, similarly I have listed an application for net banking which can where, we can check balance and do transactions transfer funds, or a timetable where you can manage time table for multiple courses taken by multiple teachers and so on.

So, you can if you think about an application then, you should be able to moderately map it is a functionality across the presentation logic and data layer. So, that you know what you want to do at the clients, and which is which is at the presentation layer, or what you

want at the absolute packet which is on the data, what tables and all the databases that you want to maintain, and the business logic of how actually this application will give you the result, how actually it will?

So, that is something where, you will have a whole lot of complex logic that might come in.

(Refer Slide Time: 07:18)

The slide has a header 'Business Logic Layer' with a sailboat icon. The content lists four main points about the Business Logic Layer:

- Provides abstractions of entities
  - e.g. students, instructors, courses, etc
- Enforces **business rules** for carrying out actions
  - E.g. student can enroll in a class only if she has completed prerequisites, and has paid her tuition fees
- Supports **workflows** which define how a task involving multiple participants is to be carried out
  - E.g. how to process application by a student applying to a university
  - Sequence of steps to carry out task
  - Error handling
    - e.g. what to do if recommendation letters not received on time

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jun-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition      22.9      ©Silberschatz, Korth and Sudarshan

So, coming to the business logic layer specifically, that provides abstraction of that will provide abstraction of various entities, students, instructors, courses, mails, your accounts, balance and so on, and that will enforce business tools for carrying out this. So, a student can enroll in a class only if she has completed prerequisites, you can transfer funds from one account to the other, provided, you have the authority to transfer, provided you have enough funds in the account that is going to get debited and so on. So, those are the different business tools, which will be employed by the business logic layer, and it will support a work flow which defines how a task will be carried out in terms of the multiple participants, and remember that all participants may not actually be human beings, they could be some could be human being some could be other machines or other applications as well.

So, it gives you the work flow. So, you can any of the 3 application that I just mentioned, everywhere you can find that there is a work flow, if you are want to check a mail then, there is a steps that you need to do go to the inbox, chose the particular mail item, get the

body and then if you want to reply to that, select that, the submit that, you get a new form when you write the reply, and within that form you get the original copy of the original mail and so on. So, all these kind of work flows will be supported by the business logic layer.

(Refer Slide Time: 08:47)

The slide has a header 'Object-Relational Mapping' with a sailboat icon. The main content is a bulleted list of pros and cons of O-R mapping:

- Allows application code to be written on top of object-oriented data model, while storing data in a traditional relational database
  - alternative: implement object-oriented or object-relational database to store object model
    - ▶ has not been commercially successful
- Schema designer has to provide a mapping between object data and relational schema
  - e.g. Java class *Student* mapped to relation *student*, with corresponding mapping of attributes
    - An object can map to multiple tuples in multiple relations
- Application opens a session, which connects to the database
- Objects can be created and saved to the database using `session.save(object)`
  - mapping used to create appropriate tuples in the database
- Query can be run to retrieve objects satisfying specified predicates

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kanpur - Jan-Apr - 2018

Database System Concepts - 8<sup>th</sup> Edition      22.10      ©Silberschatz, Korth and Sudarshan

Now, certainly you can understand that at the frontend, if we talk about what are different you know languages and models, that we are working within the frontend naturally our language is HTML tries for presentation, embedded with java script, at the backend in the data it is the database and the SQL query. So, it is a relational model, but what happens in between? What happens in the business layer which connects them? Now, naturally business layer you could write complex business tools.

For example, in the time table application we will have a complex algorithm, I know to find out what allocation of class rooms and slots, are feasible for assignments of teachers to courses availability of rooms and so on. So, often the business logic layer, the this tier would be convenient to write in some typical common high-level language like, C ++ or java, and naturally you would know from your experience of software engineering that, if you have a object based language then, that will be a very convenient to do that.

So, which means that, if you have say some entity as a student in your relational database then, most likely in your business logic, which is a java code you will have a class called student. So, the relation student is in the relational model, and your class student is in the

object based model, and you will need to define these in terms of certain mapping of the attributes, which we had shown when we talked about embedding of a languages, and this is what is commonly known as a object relational mapping.

So, that objects can map to multiple tuples, in multiple relations and can be viewed in a different way. So, so this mapping itself we could create a virtual view in the business logic layer, in the business logic language that you are looking at. Of course, they have been attempts to create a models, which are relational models which are also object oriented those are called object relational databases, some of them have been successful, but not really commercially successful.

So, we continue to work with SQL, and the relational database kind of things in the database level, and some kind of a high-level language like, C++, java and the object-based model in the middle tarred in the in the business logic, and continue to do the object relational mapping for solving the problems. So, you can here, I have given the points of what happens? How the objects get created? Application opens a session, which connects to the database because, we need to get the data from the database, they can be objects that can be created safe to the database.

They can be extracted from the database; new objects can be created and mapping use to create a appropriate tuples in the database. So, it is a two-way traffic that will keep on happening where, the business logic layer will continue to see entities as objects whereas, the database layer will continue to see them, as tuple in the relational database.

(Refer Slide Time: 12:07)



## Web Services

■ Allow data on Web to be accessed using remote procedure call mechanism  
■ Two approaches are widely used

- **Representation State Transfer (REST)**: allows use of standard HTTP request to a URL to execute a request and return data
  - ↳ returned data is encoded either in XML, or in **JavaScript Object Notation (JSON)**
- **Big Web Services**:
  - ↳ uses XML representation for sending request data, as well as for returning results
  - ↳ standard protocol layer built on top of HTTP

SWAYAM: NPTEL-NOOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition 22.11 ©Silberschatz, Korth and Sudarshan



So, there are also several web services, that can be used and you may be getting familiar with that, we will not go deep into this, but I will just mention that, web services a mechanism through which, you can access a data from remote server using what is known as a remote procedure call, and today very common approach for this is called rest representation state transfer, which allow standard HTTP request to a URL to execute a request and return data, and a several of the web services are based on that, and are the are big web services which you must have heard of, but I just mentioned it at this point because it is contextual, but we will not get into those in this course really. Now, coming to how do you actually develop applications?

(Refer Slide Time: 13:05)



## Rapid Application Development

■ A lot of effort is required to develop Web application interfaces

- more so, to support rich interaction functionality associated with Web 2.0 applications

■ Several approaches to speed up application development

- Function library to generate user-interface elements
- Drag-and-drop features in an IDE to create user-interface elements
- Automatically generate code for user interface from a declarative specification

■ Above features have been in used as part of **rapid application development (RAD)** tools even before advent of Web

■ Web application development frameworks

- Java Server Faces (JSF) includes JSP tag library
- Ruby on Rails
  - ↳ Allows easy creation of simple **CRUD** (create, read, update and delete) interfaces by code generation from database schema or object model

SWAYAM: NPTEL-NOOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition 22.11 ©Silberschatz, Korth and Sudarshan



Now, it is not an easy process that is a lot of effort required to develop web applications, you need to support the functionality that of current day web. So, you need several approaches to speed up applications. So, this is in parallel to if you think of how? What has been done to speed up development processes, development applications for different say C applications, or C++, or java applications.

So, one approach naturally is to variety of function library, which can help you easily get user interface elements like buttons, checkboxes, radio drag and drop features in the IDE, IDE stands for integrated development environment like, visual studio, front page these kind of which can use a, which can create user interference elements easily you can automatically generate code for user interface, and these are all parts of rapid application development tools, and some frameworks are very popular, this is primarily the java server faces or JSF is a framework where, you can rapidly develop fill in all the requirements of the different layers, in a web based database application in other very popular is ruby on rails, which allows easy creation of simple crud create, read, update, delete.

So, if you look at database applications and common applications will all applications will at least need to do this it lead to create data, read data, update data, delete data. So, you can do that quickly with ruby on rails. So, if you are looking into really the development of database applications, get yourself familiar with this rapid application development processes.

(Refer Slide Time: 14:53)

The slide has a header 'ASP.NET and Visual Studio' with a sailboat icon. The main content lists features of ASP.NET and Visual Studio, including drag-and-drop development, DataSet object association, Validator controls, JavaScript enforcement, user actions, and DataGrid usage. The footer includes copyright information for Prof. P. Das, IIT Kharagpur, Jan-Apr. 2018, and SWAYAM: NPTEL-NOOC MOOCs.

ASP.NET and Visual Studio

- ASP.NET provides a variety of controls that are interpreted at server, and generate HTML code
- Visual Studio provides drag-and-drop development using these controls
  - E.g. menus and list boxes can be associated with DataSet object
  - Validator controls (constraints) can be added to form input fields
    - ▶ JavaScript to enforce constraints at client, and separately enforced at server
  - User actions such as selecting a value from a menu can be associated with actions at server
  - DataGrid provides convenient way of displaying SQL query results in tabular format

SWAYAM: NPTEL-NOOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition 22.14 ©Silberschatz, Korth and Sudarshan

There are different other frameworks as well, another very popular and widely used framework is Microsoft specific, this is unfortunately not portable because, it is proprietary of Microsoft where, you can use the the [dot] net framework of Microsoft, which helps you with lot of an a resources and libraries which are already provided, and like we talked about JSP, we can use ASP[ dot] net here active server page in the[ dot] net framework, which provides a whole lot of controls and you have a very nice powerful id in terms of visual studio, high were being proprietary these need licenses and you need to pay for that. So, many a times, many developers may not be able to afford it or like it for that reason. Naturally, as you designed applications and create that, you will have to be careful about it is performance because certainly we all want very fast results.

(Refer Slide Time: 15:52)

The slide has a title 'Improving Web Server Performance' at the top right. On the left is a small image of a sailboat on water. The background is light green. The text is organized into bullet points under two main sections: 'Performance is an issue for popular Web sites' and 'Caching techniques used to reduce cost of serving pages by exploiting commonalities between requests'. The first section has one bullet point: 'May be accessed by millions of users every day, thousands of requests per second at peak time'. The second section has three bullet points: 'At the server site:' (with three sub-points: 'Caching of JDBC connections between servlet requests (a.k.a. connection pooling)', 'Caching results of database queries (Cached results must be updated if underlying database changes)', and 'Caching of generated HTML'), 'At the client's network' (with one sub-point: 'Caching of pages by Web proxy'). At the bottom left is the text 'SWAYAM: NPTEL-NOCO MOOCs Instructor: Prof. P. Doshi, IIT Kanpur - Jan-Apr. 2018'. At the bottom center is 'Database System Concepts - 6<sup>th</sup> Edition 22.16'. At the bottom right is '©Silberschatz, Korth and Sudarshan'.

So, if I log in to my Gmail application, and I would expect that as soon as I press the submit button with a couple of seconds, my inbox will be displayed on my browser, I am not ready to wait for 2 minutes, 3 minutes, 5 minutes for doing that.

So, while developing the application you will have to make an estimate of how often it will be used? How many users will use that every day and so on? how many, what is your expected heat rate? That is, when the maximum number of users are trying to use this then, what is the you know estimated number of request per second? That will come to your server. And to improve performance their different kinds of techniques can be used, the significant of them is called caching, caching is nothing but if you expect to request to be similar ah. So, that they are results should be similar then, after the first request besides sending the response back, you actually keep a local copy of that.

So, that if a similar request come in future, you can you may not re compute it, you can just send that cache copy. So, it can be done in terms of verity of JDBC connections called connection pooling, it can be done in terms of database queries, caching of generated, HTML and so on.

It can be done at the clients network side also by caching pages by web proxy; obviously, if you are using caching to improve performance, you will have to understand lot more of the web dynamics in depth because, certainly if you cache then there is a possibility, that somebody is asking is sending a request for which, the response would not be the same as what it was, when the last time the response was computed and cached. So, if

you send the cached information back then, you may be giving a dated information. So, possibly say for example, if you are checking at the net banking transaction, you have making a fund transfer and checking your account transactions after that one transfer, you would not expect a cached page whereas, if you are looking at the website of a say IIT Kharagpur then, it will be to cache that because, it is not expected to change very frequently. So, these are different factors. So, we do not have it in the scope to going to different issues of, how to improve performance? And what are the different challenges?

But I just want that you to be sensitive about these issues. Other very deep you know concerned, deep requirement about applications are the security of applications, there are this is a very involved topic, and there are several issues that are involved.

(Refer Slide Time: 18:48)

The slide has a title 'SQL Injection' in red. The content is a bulleted list of points:

- Suppose query is constructed using
  - "select \* from instructor where name = " + name + ""
- Suppose the user, instead of entering a name, enters:
  - X' or 'Y' = 'Y
- then the resulting statement becomes:
  - "select \* from instructor where name = " + "X" or 'Y' = 'Y" + ""
  - which is:
    - ✖ select \* from instructor where name = 'X' or 'Y' = 'Y'
- User could have even used
  - ✖ X'; update instructor set salary = salary + 10000; --
- Prepared statement internally uses:  
"select \* from instructor where name = 'X' or 'Y' = 'Y"
- Always use prepared statements, with user inputs as parameters
- Is the following prepared statement secure?
  - conn.prepareStatement("select \* from instructor where name = " + name + "")

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr-2018

Database System Concepts - 8<sup>th</sup> Edition      22.18      ©Silberschatz, Korth and Sudarshan

So, and I am talking about only a few of them because, the many of them require knowledge about several other fields particularly, in the field of security and an encryption and so on. So, one that is very common in terms of SQL query is known as a SQL injection where, based on I mean there is if you are expecting some inputs to come ah, and fill up certain parts of the SQL query then, you will have to be careful that, user should not be able to give such input say such strings.

So, that the query actually mean something different, this query may be which was just a query to read something, may update something, or give some different result. So, here I have just briefly highlighted some of those issues, there are other security issues like,

leakage of password if there are important things which are based on a single password then, use the password gets compromised because, it is shared or it is broken in a middle by some hacker and so on then, you will have a lot of risks involved.

(Refer Slide Time: 19:37)

The slide features a small sailboat icon in the top left corner. The title 'Password Leakage' is centered at the top in a red font. The main content consists of two bullet points:

- Never store passwords, such as database passwords, in clear text in scripts that may be accessible to users
  - E.g. in files in a directory accessible to a web server
    - ▶ Normally, web server will execute, but not provide source of script files such as file.jsp or file.php, but source of editor backup files such as file.jsp~, or .file.jsp.swp may be served
- Restrict access to database server from IPs of machines running application servers
  - Most databases allow restriction of access by source IP address

At the bottom, there is footer text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr- 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '22.19', and '©Silberschatz, Korth and Sudarshan'.

(Refer Slide Time: 19:59)

The slide features a small sailboat icon in the top left corner. The title 'Application Authentication' is centered at the top in a red font. The main content consists of two bullet points:

- Single factor authentication such as passwords too risky for critical applications
  - guessing of passwords, sniffing of packets if passwords are not encrypted
  - passwords reused by user across sites
  - spyware which captures password
- Two-factor authentication
  - e.g. password plus one-time password sent by SMS
  - e.g. password plus one-time password devices
    - ▶ device generates a new pseudo-random number every minute, and displays to user
    - ▶ user enters the current number as password
    - ▶ application server generates same sequence of pseudo-random numbers to check that the number is correct.

At the bottom, there is footer text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr- 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '22.20', and '©Silberschatz, Korth and Sudarshan'.

So, you should be you will need to make sure that, you do not store passwords you just store encrypted forms of that in which encrypted forms, you have must have observed for the last couple of years that, in many cases earlier you could just log in. So, giving just one password was enough, but now in many transactions for example, if you are logging

into a net banking application then, often you will be asked to provide some additional key information, or you will be asked to do a authentication by OTP one-time password and so on. So, these are common because, you know it is risky to work with just a single authentication, and you will find that some net banking applications for example, if you are doing a fund transfer, then they actually require two additional password authentication, one is a special password for fund transfer, and then possibly we will be asked to go through an OTP. So, depending on the criticality of your application and the potential vulnerability of the application, your authentication mechanism will have to be appropriately designed.

(Refer Slide Time: 21:09)

The slide has a title 'Application-Level Authorization' in red at the top right. On the left is a small logo of a sailboat on water. The main content area contains a bulleted list of issues with SQL authorization:

- Current SQL standard does not allow fine-grained authorization such as "students can see their own grades, but not other's grades"
  - Problem 1: Database has no idea who are application users
  - Problem 2: SQL authorization is at the level of tables, or columns of tables, but not to specific rows of a table
- One workaround: use views such as
 

```
create view studentTakes as
  select *
  from takes
  where takes.ID = syscontext.user_id()
```

  - where syscontext.user\_id() provides end user identity
    - ↳ end user identity must be provided to the database by the application
  - Having multiple such views is cumbersome

At the bottom, there is footer text: 'SWAYAM: NPTEL-NOCO MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018', 'Database System Concepts - 6<sup>th</sup> Edition', '22.21', and '©Silberschatz, Korth and Sudarshan'.

There are issues in terms of security, in terms of the application level also, for example, if you are looking at a student application where, you want to say that the student, every student can see his or her own grade, but they should not be able to see the grade of others. Now, how do you implement this? Now, two issues are one is the database cannot implement this as a part of access control because, database has no idea about who the application users are, and the second if you recall the way, we talked about authorization in SQL, that is in terms of tables or columns of the tables, but SQL has no mechanism to create authorization for rows of the table.

So, this will have to be handled, in terms of what is known as at the application layer. So, this is where you are. So, what this is saying that you are created a view where, you are created a view student text where, to do this you will have to read you are saying where, text [dot] id is equal to this function called is sys context [dot] user id. So, sys context is an object naturally, dot user I d this function called. So, what this function gives you call to the function gives you is an information about the end user identity, and that identity has to match, the identity that exist in the text ID for the result to be computed. So, this will ensure that depending on who is actually the current user, the same view will be evaluated for different results.

So, this is no not a not a very sound this is not a very comfortable situation because, here the authorization is not being implemented in the database level, but is being implemented at the application level, but that is way things a because, most of the fine grained authorization currently, is done entirely in the application level only a extension to SQL authorization, at for similar you know fine graining and so on has been proposed, but they have not been implemented because, there are several issues of implementing where there several issues of how do you represent? How do you module? And most importantly, these have potentials of slowing down the database, query process and significantly.

(Refer Slide Time: 23:08)

The slide has a header 'Application-Level Authorization (Cont.)' with a sailboat icon. The main content lists pros and cons of application-level authorization and introduces fine-grained authorization:

- Currently, authorization is done entirely in application
- Entire application code has access to entire database
  - large surface area, making protection harder
- Alternative: **fine-grained (row-level) authorization** schemes
  - extensions to SQL authorization proposed but not currently implemented
  - Oracle Virtual Private Database (VPD) allows predicates to be added transparently to all SQL queries, to enforce fine-grained authorization
    - ↳ e.g. add `ID= sys_context.user_id()` to all queries on student relation if user is a student

SWAYAM: NPTEL-NOC NOOC's Instructor: Prof. P. P. Dhas, IIT Kharagpur - Jan-Apr - 2018

So, often this is not a preferred mechanism either. Another way to ensure security is to keep audit trail, trail must log actions into it.

(Refer Slide Time: 23:46)

The slide has a title 'Audit Trails' in red at the top right. To its left is a small image of a sailboat on water. On the far left edge of the slide, there is vertical text that reads 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kanpur - Jan-Apr. 2018'. The main content is a bulleted list:

- Applications must log actions to an audit trail, to detect who carried out an update, or accessed some sensitive data
- Audit trails used after-the-fact to
  - detect security breaches
  - repair damage caused by security breach
  - trace who carried out the breach
- Audit trails needed at
  - Database level, and at
  - Application level

At the bottom of the slide, there is footer text 'Database System Concepts - 8<sup>th</sup> Edition' and '22.23' on the left, and '©Silberschatz, Korth and Sudarshan' on the right, along with a standard presentation navigation bar.

So, where you write down actions in terms of who carried out and update, or who access some sensitive data, or who did a delete and so on. So, audit trails can be used later on if the need arises to detect, if some security breach that happen, or if some damage has been caused by the security breach, that can be corrected and so on create a trace.

So, auditing is necessarily a very required, in a very required activity for any data-based application and audit trail had a good mechanism for that. So, when you develop applications you have to consider has to whether, you would like to support audit trail of course, if you support audit trail then, somewhat your application will get slowed down, it will require more disk to keep that trail because, every transaction every details you will get logged in to the audit trail, but it is very, very important for critical applications like, net banking where currently it is mandated every transaction that, you do every action that you do on your account, through the net banking is trailed in the banks end.

So, that if later there is a there is some dispute, there is some breach has found then, the same can be recovered and trace back to the actions, that you are actually taken ah. At the end of this module let me take a quick look into the mobile apps.

(Refer Slide Time: 25:32)

**What Is a Mobile App?**

- A type of application software designed to run on a mobile device, such as a smartphone or tablet computer
- Developed specifically for use on small, wireless computing devices, such as smartphones and tablets
- Designed with consideration for the demands and constraints of the devices and also to take advantage of any specialized capabilities
  - Form Factor – influences display and navigation
  - Limited Memory
  - Limited Computing Power
  - Limited Power
  - Limited Bandwidth
  - ...
  - + Availability of sensors like accelerometer
  - + Availability of touchscreen – Gesture-based Navigation
  - + ...

Source: <https://www.slideshare.net/hassandar18/architecture-of-mobile-software>

PPD

SWAYAM: NPTEL-NOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr - 2018

Database System Concepts - 8<sup>th</sup> Edition

22.25

©Silberschatz, Korth and Sudarshan

These are somewhat different, but it is very important to today to take a notion of the mobile app. So, it is a type of a application software, which is designed to run on a mobile devise. So, it this is not necessarily mean that, it has to be a smart phone, it could be a tablet or you know some small device, which typically is handled and carried around. So, it is typically specifically for use on small wireless computing devices, and it is designed with considerations for demands and coincident of the device. So, how are how is a different? If I want to use the application or want to have an application which does my task, but I want to do it as a mobile app, I want to do it for a small wireless handled device. Certainly, there are if we if we since the small device it has lot of constraints, there are lot of demands of the device, compare to if we were using a web browser in a desktop or a laptop where, you have lot of resources.

And also on the this is this is on the one kind of the restriction side and on the other side, these devices have certain capabilities, which your desktop or laptop may not have and therefore, it may be you may be able to do more interesting application using this mobile devices, and there could be really interesting mobile apps, and as you all must be using today use 10s of if not 100s of mobile apps for different applications, many of which actually support a database at the backend. So, these are in red are some of the negatives, which are restrictive in terms of a mobile application the first thing is form factor, which is the look the aspect ratio the size and the style of the device, which influences display and influences the way you navigate. In a desk of application you will typically use a mouse, or a roller and keyboard to navigate, but that is not possible or that is not easy in terms of a mobile device. So, you are navigation may happen in in a different way.

The presentation itself may happen in a different way, you have a very limited display area. So, you might want to stack multiple responses one after the other whereas, the in a in a web application running on a desktop, you would probably have shown them on different tabs side by side, or would have just shown them side by side on the display. Then, mobile devices typically have limited memory, they have limited computing power, very importantly most of them run on battery and therefore, they have one limited power available. So, you are applications will lead to be power optimized, which is not the case with the normal web based application, you have a wireless connection, so which may be limited in bandwidth based on your connectivity and that time, and so on. So, you while developing a mobile app corresponding to a possible web application, your considerations would be very different. So, if I say that a bank say, HDFC bank has a net banking application, which runs on the browser. And it is also having a mobile app, which runs on say the android phone then, the their requirements and their style of solutions will have to be very, very different. And at the same time if I talk about a mobile app then, the mobile devices often have features which desktop do not have for example, you have a number of senses available like, accelerometer and so on which you can make to advantage for example, we I mean in in many smart phones, if we just rotate the screen, rotate that device, then the display self-rotate automatically, which we use we use some some kind of an accelerometer inputs for that, you have a touch screen which can allow a wide range of gesture based navigation, which is typically not possible in desktop and most of the laptops.

(Refer Slide Time: 29:48)

**Mobile Website vis-à-vis Mobile App**

**■ Mobile Website**

- Similar to any other website in that it consists of browser-based HTML pages
- Can display text content, data, images and video
- Typically accessed over WiFi or 3G or 4G networks
- Designed for the smaller handheld display and touch-screen interface
- Can also access mobile-specific features such as click-to-call (to dial a phone number) or location-based mapping

**■ Mobile Apps**

- Actual applications that are downloaded and installed on the mobile device
- Users download apps from device-specific portals such as App Store, Google Play Store
- The app may
  - pull content and data from the Internet, in similar fashion to a website, or
  - download the content so that it can be accessed without an Internet connection

Source: <https://www.slideshare.net/hassandar18/architecture-of-mobile-software>

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr- 2018

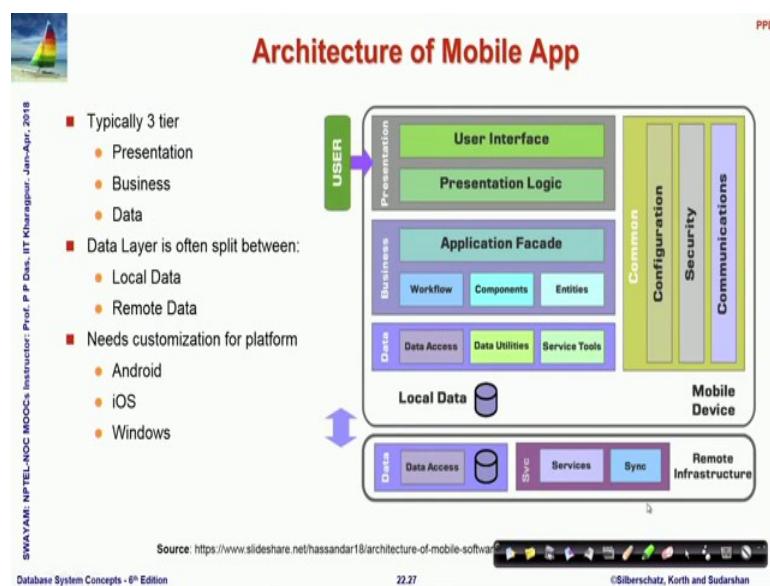
PPD

Database System Concepts - 8<sup>th</sup> Edition      22.26      ©Silberschatz, Korth and Sudarshan

So, having said that, naturally there are two aspects of mobile apps as well. So, when we talking about mobile apps, we have talking about stand alone mobile apps, there are other ways of developing applications also for example, we can do a typical web-based application, but we can use a website which is specifically designed catering to the mobile devices. So, these send back pages, which are smaller in size are stack differently and so on. So, in contrast to that a mobile app are actually, applications that are downloaded and runs on the system.

So, these are all different possibilities and even though mobile website is also becoming popular, but certainly mobile apps are very, very common in terms of a majority of critical applications of data bases that we have.

(Refer Slide Time: 30:40)



So, this is the typical architecture of a mobile app as you can see again here, that you have the three layers presentation, business and data the only difference being now, since you have a device, which can go anywhere and you have a connectivity.

So, this is what shows the connectivity, this is the device side and this is the pure backend or your service provider side. So, since your connectivity is not may not be very strong, you try to create all the layers of a presentation, business, as well as data on the mobile phone itself, but naturally all the data you will not have on your device ah, you are checking mails on the Gmail naturally, you will not have all the data on your Gmail.

So, what you do is you use the connection to connect to the remote database provided by the service, but primarily most of this layer of this presentation, business and a small part of the data layer are all realized within the phone itself, or within the mobile device itself. So, the data layer is split in this case, which is very different from how you do the web-based applications? And specifically, it might need customizations based on the kind of platform, you are using whether you have using android, iOS and windows, and these are all custom solutions for that and you could also note that, there are different types of mobile apps one large class is known as native application where, which is completely written in the native language of the platform.

So, for if you are doing an iOS mobile app then you will write it in object C, objective C if you are doing an android one you will write it in C or C++ or java is platform specific whereas, there is another class of mobile apps, which are known as web apps which run completely inside the web browser, so much like the way java scripts work.

(Refer Slide Time: 32:12)

**Types of Mobile Apps**

- **Native Apps:** Completely written in the native language of a platform
  - iOS → Objective-C, Android → Java or C/C++
  - Platform specific (heavily dependent on OS)
- **Web Apps:** Run completely inside of a Web browser.
  - Features interfaces built with HTML or CSS
  - Powered via Web programming languages →Ruby on Rails, JavaScript, PHP, or Python
  - Portable across any phone, tablet, or computer
- **Hybrid Apps:** Combines attributes of both native and Web apps.
  - Attempts to use redundant, common code that can be used across platforms, and
  - Tailors required attributes to the native system

SWAYAM: NPTEL-NOCO Instructor: Prof. P. Das, IIT Kharagpur - Jam-Apr- 2018

https://www.slideshare.net/hassandar18/architecture-of-mobile-software

Database System Concepts - 6<sup>th</sup> Edition

22.28

©Silberschatz, Korth and Sudarshan

So, they feature interfaces built with HTML and the style shades, and they have powered by different web programming languages like, ruby on rails java script to PHP and so on. And certainly there is a third kind, when you combine the attributes of both native and wave application and you try to you know. So, you can very easily understand, that if your application is a native one then it is not portable across devices, this is not an iOS application is not run on android and so on.

If it is a web app app, then it is portable because it is portable across different phone tablet or computer because, you are using generic technologies. So, you could do a hybrid app also where, you could use redundant or common code, which is usable across platform and add some tailor functionality in terms of the native system. So, these are the typical kinds of mobile apps that.

(Refer Slide Time: 33:50)

The slide has a header 'Design Issues' in red. Below it is a bulleted list of nine items:

- Determine Device
- Note Device Resources – memory, power, speed
- Consider Bandwidth
- Decide on Architecture Layers
- Select Technology
- Define User Interface
- Select Navigation
- Maintain Flow

On the left edge of the slide, there is vertical text that reads: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr - 2018'. In the top right corner, there is a small 'PPD' logo. At the bottom, there is a source link: 'Source: <https://www.slideshare.net/hassandar18/architecture-of-mobile-software>' and a set of navigation icons.

You might be developing and there are several factors to consider in the design issue, you have to first decide on the device, you have to take specific note of the typical resources, that your device will support memory, power, speed. You have to consider what should be the bandwidth should it be 2 G, 3 G, 4 G or wireless you know LAN, what kind of connections you would expect? Decide on the layers in the architecture, select the technology based on the device choice and the other factors define the user interface, navigation and maintain the work flow.

So, these are very variety I will not go into details of this, but just wanted to give a glimpse of the fact, that in today's time while you are talking about database applications then it is a very reality, that you will create that as a mobile app.

(Refer Slide Time: 34:32)



## Module Summary

- Studied the aspects of Database Applications Architectures
- Understood the steps in the Rapid Application Development Process
- Exposed to the issues in Application Performance
- Exposed to the issues in Application Security
- Learnt the distinctive features of Mobile Apps

SWAYAM: NPTEL-NOOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018



Database System Concepts - 6<sup>th</sup> Edition

22.30

©Silberschatz, Korth and Sudarshan

So, in this module to summarize your study aspects of database application architecture, understood the steps of rapid development, and took a quick look into the issues of application performance security, and what it takes to? What are the distinctive features of a mobile app for database applications?

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 23**  
**Application Design and Development (Contd.)**

Welcome to module 23 of Database Management Systems. We have been discussing about application design and development and this is the third and concluding module in that regard.

(Refer Slide Time: 00:27)

PPD

**Module Recap**

- Application Architectures
- Rapid Application Development
- Application Performance
- Application Security
- Mobile Apps

GUANAJAMANNUDEP-HOC Instructor: Prof. P. P. Das, IIT Kharagpur Date: 2018  
Database System Concepts - 8<sup>th</sup> Edition

23.2

©Silberschatz, Korth and Sudarshan

In the last module we have discussed about different aspects of application architecture rapid development process issues and performance and security and took a glimpse in terms of, what is required for doing a mobile app.

(Refer Slide Time: 00:44)

The slide is titled "Module Objectives" in red at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bullet point: "■ To design the schema for a Library Information System". At the bottom left, there is a video thumbnail showing a person speaking. The bottom right corner features a navigation bar with icons for back, forward, search, and other presentation controls. The footer includes text: "Database System Concepts - 8<sup>th</sup> Edition", "23.3", and "©Silberschatz, Korth and Sudarshan". A vertical column on the far left contains the text: "CHAKRABORTY NOC MOOCs", "Instructor: Prof. P. P. Das, IIT Kharagpur", and "2018".

In this module, we will take care case study in terms of a library information system. We will try to design the schema for that as you have seen that for a database application design there are frontend designs there are middle tier business logic design and there will be issues in terms of the database design.

Since we are in the DBMS course, I will not focus on the whole aspects of application design, but we will focus specifically on the database aspect. So, we will start with a basic requirement specification for a library information system and then from the starting from that we will try to extract different entities and attributes and their relationships and we will make a relational schema and use different notions of dependency and how to write queries we will look into those aspects and refine that and finalize a schema for this problem. So, let us work through that. So, the outline the module issue is library information system.

(Refer Slide Time: 01:55)

The slide has a header 'Library Information System (LIS)' with a small sailboat icon to the left. In the top right corner, there is a red 'PPD' logo. The main content discusses an institute library's book management system. It mentions that the library has over 200,000 books and 10,000 members. Books are issued on loan and returned after a period. The system manages books, members, and the issue-return process. Every book has a title, author (if multiple authors, only the first is maintained), publisher, year of publication, ISBN number (unique for the publication), and accession number (unique for the copy in the library). There may be multiple copies of the same book in the library. Members are categorized into undergraduate students, postgraduate students, research scholars, and faculty members. Each student has a name, roll number, department, gender, mobile number, date of birth, and degree (undergrad, grad, doctoral). Each faculty member has a name, employee ID, department, gender, mobile number, and date of joining. The library also issues a unique membership number to every member. Every member has a maximum quota for the number of books they can issue for the maximum duration allowed to them. Currently, these are set as:

- Each undergraduate student can issue up to 2 books for 1 month duration
- Each postgraduate student can issue up to 4 books for 1 month duration
- Each research scholar can issue up to 6 books for 3 months duration
- Each faculty member can issue up to 10 books for 6 months duration

The library has the following rules for issue:

A book may be issued to a member if it is not already issued to someone else (trivial). Also, a book may not be issued to a member if another copy of the same book is already issued to the same member. No issue will be done to a member if at the time of issue one or more of the books issued by the member has already exceeded its duration of issue. No issue will be allowed also if the quota is exceeded for the member.

Finally, it is assumed that the name of every author or member has two parts – first name and last name.

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

23.5

©Silberschatz, Korth and Sudarshan

So, let us start with a basic this is a very small description. So,, but yet it will give us lot of food for thought in this work and while you actually go through the rest of the video. I would suggest that you take a print out of this page or keep this page separately because you will frequently need to refer to it.

So, let me quickly go through this we are talking about an institute library that has over 2 lakh books and over 10,000 members who can use regularly issue the books on loan and return them on the expiry of the period or before that and the library needs a library information system to manage the books the members and as well as the issue return process. So, we are just looking into these aspects not the procurement of books and organization of the books and so on.

Now what is given every book has it is title author publisher etcetera a number of different attributes ISBN number accession number in terms of a book I must explain to you at this stage that any book that you that is published has an ISBN number which is an international number given so, that you can uniquely identify that book.

So, if we are talking about the database management book we are following here then that has a ISBN number, but that does not mean that number actually is unique for the book would not for a specific copy of the book. So, if you buy three copies of the book and put it in the library, then these three copies need to be identified separately by another number which is typically called the accession number.

So, naturally the LIS need that every book has ISBN number which says which book it is and the accession number which says which specific copy it is, because they are may be multiple copies of the same book in the library on the member side. There are four categories of members broadly: students and teachers, but amongst students if they could be undergraduate postgraduate or research students and the faculty members. The student are specified by their name, roll number, department, gender, mobile number, date of birth, and the degree that they are doing and every faculty member also has a name, employee id, department, gender, mobile number.

And the date of joining the library to manage these members library also issues a unique membership number to every member and every member has a maximum quota for the number of books that she or he can issue and the maximum duration for which those books can be retain once issued. So, there are different specification for at different categories of member can have different quota and different duration the general rule as specified by this libraries the a book may be issued to a member naturally if it is not issued to someone else the book has to be available also a book may not be issued to a member if another copy of the same book is already issued to the same member.

So, you cannot issue two copies of the same book at the same time no issue will be done to a member if he is kind of a default that is the time of at the time of issue one or more of the other books already issued by the member has already exceeded the duration of the issue.

So, they are overdue for return in that case no issue will be allowed no issue will; obviously, be allowed if the quota exceeds and it is also specified that whenever we talk about name every name will have two parts the first name and the last name. So, this is the this is the given specification based on this we will need to make a good relational schema to represent the tables and manage the queries.

(Refer Slide Time: 05:45)

LIS Queries

LIS should support the following operations / query:

- Add / Remove members, categories of members, books.
- Add / Remove / Edit quota for a category of member, duration for a category of member.
- Check if the library has a book given its title (part of title should match). If yes: title, author, publisher, year and ISBN should be listed.
- Check if the library has a book given its author. If yes: title, author, publisher, year and ISBN should be listed.
- Check if a copy of a book (given its ISBN) is available with the library for issue. All accession numbers should be listed with issued or available information.
- Check the available (free) quota of a member.
- Issue a book to a member. This should check for the rules of the library.
- Return a book from a member.
- and so on

CHAKRABORTY, NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

23.6

©Silberschatz, Korth and Sudarshan

So, let us take a quick look into some of the sample queries this is just a indicative sample. Now, naturally we need a whole lot of you know insert delete update kind of queries for adding or removing members categories of members shapes the books and so, on adding or removing or changing the quota of a category of member the duration for that also we will have queries like to check.

If the library has a given book with a given title and if it is found then the details of those should be listed or we need to check if library has a book given it is author. So, if I say the author it should be possible to locate a book we should able to check, if a copy of a book if I specify the ISBN number, then whether it is available with the library for issue. So, there may be as I said multiple copies of the same book.

So, given the ISBN number it will return all the copies the accession number of all the copies and their issue status; whether they are issued or they are available it should be available it should be possible to check the quota available free quota of a member and certainly it should have features of issuing a book to a member and they should check for the rules of the library as stated it should be possible to return a book and so, on. So, these are the typical queries against which in the backdrop of which we will try to design the database system. So, what we will do?

(Refer Slide Time: 07:18)

**Entity Sets: books**

PPD

CHAKRABORTY, NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr., 2018

- Every book has title, author (in case of multiple authors, only the first author is maintained), publisher, year of publication, ISBN number (which is unique for the publication), and accession number (which is the unique number of the copy of the book in the library). There may be multiple copies of the same book in the library.
- Entity Set:
  - books
- Attributes:
  - title
  - author\_name (composite)
  - publisher
  - year
  - ISBN\_no

Database System Concepts - 8<sup>th</sup> Edition

23.7

©Silberschatz, Korth and Sudarshan

We will initially start with a specification as given. So, I hope you already have kept a copy to refer to and we will try to extract the different entity sets and the attributes. Naturally, the first entity said that we extract we let us call it books which is about books where in the beginning I have given the basic statement that is given in the so, in the specification.

So, from that we can see that books will be an entity set and it will have attributes like title author name which is a composite one, because it will have two parts into that the first name and the last name the publisher year ISBN number and accession number. So, these are the attributes available for books. So, this is my first entity set.

(Refer Slide Time: 08:11)

The slide is titled "Entity Sets: students" in red text at the top right. On the left, there is a small logo of a sailboat on water. The main content is a bulleted list:

- Every student has name, roll number, department, gender, mobile number, date of birth, and degree (undergrad, grad, doctoral).
- Entity Set:
  - **students**
- Attributes:
  - member\_no – is unique
  - name (composite)
  - roll\_no – is unique
  - department
  - gender
  - mobile\_no – may be null
  - dob
  - degree

At the bottom left, it says "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018". At the bottom center, it says "Database System Concepts - 8<sup>th</sup> Edition" and "23.8". At the bottom right, it says "©Silberschatz, Korth and Sudarshan".

The second entity set are students. So, this is the statement about the student and from that statement, we can easily extract that entity set here is student and the attributes are member number, because certainly in the context of the library system as we said the; student has to be a member. So, it should be there will be a member number which is has to be unique the composite name the student has a roll number. So, we assuming that the role number is a unique field.

So, knows to students will have the same roll number, then there is department gender mobile number which could be null the; date of birth degree that the student has done is doing. So, those are the different attributes for this entity set.

(Refer Slide Time: 09:03)

**Entity Sets: faculty**

- Every faculty has name, employee id, department, gender, mobile number, and date of joining.
- Entity Set:
  - **faculty**
- Attributes:
  - member\_no – is unique
  - name (composite)
  - id – is unique
  - department
  - gender
  - mobile\_no – may be null
  - doj

Database System Concepts - 8<sup>th</sup> Edition

23.9

©Silberschatz, Korth and Sudarshan

So, we have books we have students similarly we will have faculty again the there is this process will continue by extracting different parts from the specification. So, these are statement about the faculty and we know that there will be a faculty entity set with attributes like member number name id and so, on.

(Refer Slide Time: 09:25)

**Entity Sets: members**

- Library also issues a unique membership number to every member. There are four categories of members of the library: undergraduate students, post graduate students, research scholars, and faculty members.
- Entity Set:
  - **members**
- Attributes:
  - member\_no
  - member\_type (takes a value in ug, pg, rs or fc)

Database System Concepts - 8<sup>th</sup> Edition

23.10

©Silberschatz, Korth and Sudarshan

So, it should be quite obvious library has talked of that it can each it will issue unique membership number to every member and there are 4 categories of member. So, let us

we are just making attentive you know suggestion that there could be. So, it looks like members are in entity which the library has to interact with in terms of issue.

So, let us create an entity set members which has the member number and the member type. So, which can take different types of undergraduate post graduate these different one of these four specific values let us say.

(Refer Slide Time: 10:00)

**Entity Sets: quota**

- Every member has a maximum quota for the number of books she / he can issue for the maximum duration allowed to her / him. Currently these are set as:
  - Each undergraduate student can issue up to 2 books for 1 month duration
  - Each postgraduate student can issue up to 4 books for 1 month duration
  - Each research scholar can issue up to 6 books for 3 months duration
  - Each faculty member can issue up to 10 books for six months duration
- Entity Set:
  - quota
- Attributes:
  - member\_type
  - max\_books
  - max\_duration

Database System Concepts - 8<sup>th</sup> Edition

23.11

©Silberschatz, Korth and Sudarshan

The rules have talked about having a quota. So, every member will according to it is member category we will have different quota. So, to represent so, quota becomes an entity set. So, we would like to represent that for different member type which is a category; how many number of maximum books can be taken and what could be the maximum duration? So, let us say we will put the maximum duration say in terms of months and with these three attributes we will have an entity set which is quota.

(Refer Slide Time: 10:33)

**Entity Sets: staff**

- Though not explicitly stated, library would have staffs to manage the LIS.
- Entity Set:
  - **staff**
- Attributes: (speculated – to ratify from customer)
  - name (composite)
  - id – is unique
  - gender
  - mobile\_no
  - doj

CHANDRAKANTAPURI NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

PPD

23.12

©Silberschatz, Korth and Sudarshan

Here, I am talking about another entity sets staff if you again carefully read the specification you will find that there is no mention of this staff in the in the total of the specification, but if we logically think and this is what we often need to do. When we deal with a practical specification like somewhat like, the one that I have given here is that we would need to look little beyond the specification.

So, think about it if I have the library with books students faculty issue process quota and all that then certainly they will have to be some staff of the library; that will actually do all this operation. So, they will be able to log in to the database and actually issue a book return a book check for validity and so, on.

So, let us assume that we have a entity set staff which certainly a staff should have name and id which id should be unique, gender, mobile number and date of joining. So, this is kind of a speculative addition to the design of entity sets and this must be ratified from the customer; when the opportunity arise, but something like that must be there for to make the design complete.

(Refer Slide Time: 11:48)

The slide has a header 'Relationships' in red. On the left, there's a small logo of a sailboat on water. On the right, it says 'PPD'. The main content is a bulleted list:

- Books are regularly issued by members on loan and returned after a period. The library needs an LIS to manage the books, the members and the issue-return process.
- Relationship
  - book\_issue
- Involved Entity Sets
  - students / faculty
    - ▶ member\_no
  - books
    - ▶ accession\_no
- Relationship Attribute
  - doi – date of issue
- Type of relationship
  - Many-to-many

At the bottom, there's vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P.P. Desai, IIT Kanpur Date: Jan-Apr- 2018'. Below the slide are standard presentation navigation icons and the text 'Database System Concepts - 8<sup>th</sup> Edition' and '23.13 ©Silberschatz, Korth and Sudarshan'.

So, these are ah; obviously, the immediately visible entity sets that we see. So, the other part that we will now have to check on is a relationship. So, again we pick up the relevant statement in this regard books are regularly issued by members on loan and returned after a period the library needs an LIS to manage the books members and the issue return process.

So, certainly there will have to be a relationship of issue between the student or faculty in the books. So, we loosely define this relationship on one side there has to be the books and the other side would be the involved entity set would be either student or faculty so, actually though I am just noting it here as a as a single relationship, but actually there is a relationship of issue between students and books and faculty and books.

So, we will have to see how to handle these kind of situation now certainly the students or faculty are identified by the unique member number of the library that has been given books as we have said are identified by the accession number. Again, please note that we are not talking about the ISBN number, because ISBN number could be same for multiple copies of the book, but when you issue you issue a specific copy. So, that accession number which is a unique for a specific copy needs to be tracked. So, these are the two attributes, which will certainly be involved in the relationship and then you would recall that often relationships of their own attributes.

So, here we have one the date of issue needs to be recorded, because we want to check conditions like; if the borrower is has issued the book and how many for how many months he or she is keeping that book. So, if you have to check that we need to know when the book was issued. So, this is a attribute which does not exist in any of the entity sets that are involved in this relationship neither in students or faculty nor in books, but this is a attribute of the relationship which needs to be specified.

And of course, it is this relationship is a many to many relation; because every student or faculty can issue multiple books and every book may be issued by different, but we can we can in general we can be specific to say that this is many to one also if we want to maintain that at a any given instant a book can be issued only by one person. So, if you look at it from that perspective this will not be many to many this will be treated as many to one.

(Refer Slide Time: 14:24)

**Relational Schema**

- books(title, author\_fname, author\_lname, publisher, year, ISBN\_no, accession\_no)
- book\_issue(members, accession\_no, doi)
- members(member\_no, member\_type)
- quota(member\_type, max\_books, max\_duration)
- students(member\_no, student\_fname, student\_lname, roll\_no, department, gender, mobile\_no, dob, degree)
- faculty(member\_no, faculty\_fname, faculty\_lname, id, department, gender, mobile\_no, doj)
- staff(staff\_fname, staff\_lname, id, gender, mobile\_no, doj)

NPTEL MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr 2018

Database System Concepts - 8<sup>th</sup> Edition

23.14

©Silberschatz, Korth and Sudarshan

So, having done this finding having found out this entity an attributes and the relationships let us now start with a relational schema.

So, what I have done here; in terms of the relational schema that for each of the entity set. I have created a relational schema by the same name and have put the attributes as identified as attributes of that relational schema. So, this is a very straight forward once we have identification, made this identification process from the original specification. This is a more straight forward process where you just convert every entity set into a

relational schema and also we have converted the relationship there was a there is a relationship called book issue here as you can see the book issue this also we have converted to a relational schema involving the two attributes of member number and the accession number.

So, let us see how this will span out later parts so, having done this.

(Refer Slide Time: 15:38)

The screenshot shows a presentation slide titled "Schema Refinement". The slide content is as follows:

- books(title, author\_fname, author\_lname, publisher, year, ISBN\_no, accession\_no)
  - ISBN\_no → title, author\_fname, author\_lname, publisher, year
  - accession\_no → ISBN\_no
  - Key: accession\_no
- Redundancy of book information across copies
- Good to normalize:
  - book\_catalogue(title, author\_fname, author\_lname, publisher, year, ISBN\_no)
    - ▶ ISBN\_no → title, author\_fname, author\_lname, publisher, year
    - ▶ Key: ISBN\_no
  - book\_copies(ISBN\_no, accession\_no)
    - ▶ accession\_no → ISBN\_no
    - ▶ Key: accession\_no
- Both in BCNF. Decomposition is lossless join and dependency preserving

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jun-Apr. 2018

The next task would be to work on the refinement of the schema. So, now, it is time that the relational schema is available the first schema is available. So, now, we have to apply all different notions of the relational design to refine this schema and finalize. So, for what will do; we will take every relational schema and we will try to identify the functional dependencies and use that to identify the key.

So, if I look at the books relational schema, then we easily know that **ISBN number → title, author first name; authors last name, publisher, year**. So, here also you may just note that since name is stated to be a composite attribute I have designed here to use two different attributes the first name and the last name. So, with that we have this functional dependency which tells me that given ISBN number, I know these details, but of course, that does not tell me what is the accession number because there could be multiple copies, but another functional dependency must hold because, every copy of the same book must have the same ISBN number. So, given the accession number, ISBN number should be determinable.

So, **accession number → ISBN number**. So, if you do the sample computation on this you will easily figure out that a key of this is accession number. So, this is what we currently have this this is what we have done now. So, you can see that if there are say five copies of a book having different accession numbers which are the key they are has been number would may all be same.

So, if I have the same ISBN number I can multiple accession numbers and therefore, there are different records, but if that accession number of two books are I am sorry the ISBN number of two books are same then all of that title, author and first name last name all this attributes will be repeated and if there are multiple copies of the book then each one of them will have a separate entry because they have a separate accession number.

But, their ISBN number and everything else will be redundant. So, we can easily see that the given relational schema actually has redundancy across copies of the book and if we want to look at this from formal theory we can easily check that the keys accession number. So, **ISBN number → title, author, .... etcetera** that functional dependency violates the boyce code normal form actually this is the also not in the third normal form right now.

So, we would do well to reduce this redundancy and it would be good to normalize. So, here I have not gone through the actual steps of normalization, but I am showing you more intuitively as to, how you can normalize; because it is a quite obvious that the accession number is involved only with the ISBN number and which keeps track of the copies.

So, we propose to have under normalization we propose to have one which is let me just highlight and show you. So, book copies where the ISBN number and accession number will be maintained. So, for every accession number we will be able to see the ISBN number which will tell you which book it is and rest of the book details will be kept in this say another new relational schema called book catalogue which will have all of the earlier attributes expect the accession number.

So, now, if you project the; this dependency on book catalogue the dependency will be fully projected. So, the whole dependency is preserved if you project this dependency on the book copies it will also be fully preserved. So, this decomposition is a dependency

preserving and you can easily check that these two can be joint by a natural join using ISBN number as the common attribute.

So, if we take the intersection of the two sets of attributes then ISBN number would be the attribute and **ISBN number → all attributes** in the book catalogues. So, our lossless join condition  $r_1 \cap r_2 \rightarrow r_1$  here. So, this will also give me a lossless join and if you check on each one of these relational schema then this functional dependency the left hand side is a key and therefore, book catalogue is in Boyce Codd normal form and book copies is also in Boyce Codd normal form.

So, you can you could come to the same result by doing the formal process of functional Boyce Codd normal form decomposition, but I have just shown you here as to intuitively how you get this. So, intuitively becomes very clear that you have details of the book that you keep separate in a separate table, because that is not change across the copies and we have a separate table to maintain the specific information about the copies. So, that takes care of the books relationship moving on let us look at the other.

(Refer Slide Time: 21:16)

The slide is titled "Schema Refinement". It contains the following text:

- **book\_issue(member\_no, accession\_no, doi)**
  - member\_no, accession\_no → doi
  - Key: members, accession\_no

On the left margin, there is vertical text that reads: "SWAYAM: NPTEL-NOC Instructor: Prof. P. P. Deshpande, IIT Kanpur - Jan-Apr. 2018". At the bottom, it says "Database System Concepts - 8<sup>th</sup> Edition" and "23.16". The footer also includes "©Silberschatz, Korth and Sudarshan".

So, book issue is a relational schema which comes from the relationship between the now between what now book issue is to happen between student faculty and books. So, it is I mean we cannot have two of them of these entity sets occurring in the same. So, initially let us just put that well the library has a concept of a member. So, what if the book issue is a relation simply between the members and the books of course, we

do not know the relationship between members and students or members and faculty, but we can at least refine the book issue to be between member and the book and therefore, member number and accession number together becomes the key which → the date of issue this is already in normal form as you can see.

(Refer Slide Time: 22:11)

The slide has a title 'Schema Refinement' in red at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list:

- **quota(member\_type, max\_books, max\_duration)**
  - $\text{member\_type} \rightarrow \text{max\_books, max\_duration}$
  - Key: `member_type`

At the bottom left, it says 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Desai, IIT Kanpur - Jan-Apr- 2018'. At the bottom center, it says 'Database System Concepts - 8<sup>th</sup> Edition' and '23.17'. At the bottom right, it says '©Silberschatz, Korth and Sudarshan'.

Quota is a simple relational schema where member type → the max books and duration and that is the key in normal form

(Refer Slide Time: 22:19)

The slide has a title 'Schema Refinement' in red at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list:

- **members(member\_no, member\_type)**
  - $\text{member\_no} \rightarrow \text{member\_type}$
  - Key: `member_no`
  - Value constraint on `member_type`
    - ▶ ug, pg, or rs: if the member is a student
    - ▶ fc: if the member is a faculty
  - How to determine the `member_type`?

At the bottom left, it says 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Desai, IIT Kanpur - Jan-Apr- 2018'. At the bottom center, it says 'Database System Concepts - 8<sup>th</sup> Edition' and '23.18'. At the bottom right, it says '©Silberschatz, Korth and Sudarshan'.

The members I mean we have identified something like a member which should list all the members of the library.

So, existence of a member number in that list will mean that someone holding that member number is actually a member and it is should also tell me what type of member or what category of member he or she is. So, member number must → the member type and the member type will be constrained in terms of possible 4, 1 of the four possible values are stated here now of course, we will still have to figure out is to where do we get this member type from and so, on and that information is not present here. So, further refinements will be required in this regard.

(Refer Slide Time: 23:02)

The slide has a header 'Schema Refinement' and a small logo of a sailboat in the top left. In the top right corner, there is a small red 'PPD' icon. On the left side, there is vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kanpur - Jan-Apr-2018'. On the right side, there is a video frame showing a man with glasses and a blue background. Below the video frame is a toolbar with various icons. At the bottom, there is footer text: 'Database System Concepts - 8<sup>th</sup> Edition', '23.19', and '©Silberschatz, Korth and Sudarshan'.

**■ students(member\_no, student\_fname, student\_lname, roll\_no, department, gender, mobile\_no, dob, degree)**

- roll\_no → student\_fname, student\_lname, department, gender, mobile\_no, dob, degree
- member\_no → roll\_no
- roll\_no → member\_no
- 2 Keys: roll\_no | member\_no

**■ Issues:**

- member\_no is needed for issue / return queries. It is unnecessary to have student's details with that.
- member\_no may also come from **faculty** relation.
- member\_type is needed for issue / return queries. This is implicit in degree – not explicitly given.

Let us go to the students this is the whole schema that we had done. So, **roll number → all the attributes** specifically **roll number → member number**, **member number → roll number**, because every student has a unique roll number and; obviously, has a unique member number also **number → all attrib.** So, **member number → roll number** **roll number → member number** and so **roll number → rest of attributes.**

Which mean that this design this relational schema has two keys the roll number and the member number now are we happy with this design that is a question we need to ask. So, we can figure out that member number is needed for issue return or queries book issue has member number, that is; what we were thinking of; now when we want to deal with

this book issue issuing a book to a student and returning a book from the student and soon.

Is it necessary to have all the students details with that one that make every record very heavy and if we want to extra connect with that do some join or if you want to do some query unnecessarily we will have to deal with very large units of data and as we will see in the next couple of modules that if a record becomes larger dealing with it become naturally becomes more cumbersome. So, there is some kind of discomfort at this design similarly a member number is not here it is the student relation. So, it is coming from the student relation, but in general in terms of issue it may also come from faculty relations.

So, how is that handled we do not know then issue return also needs the member type which is implicit here in terms of the field in terms of the attribute degree which tells you that student is a undergraduate postgraduate or research, but it is not explicitly maintained anywhere. So, these are the issues in this form of the design that we that we came.

(Refer Slide Time: 25:06)

The slide has a header 'Schema Refinement' in red. On the left, there is a small logo of a sailboat on water. On the right, there is a 'PPD' watermark. The main content is organized into two sections:

- faculty**(member\_no, faculty\_fname, faculty\_lname, id, department, gender, mobile\_no, doj)
  - id → faculty\_fname, faculty\_lname, department, gender, mobile\_no, doj
  - id → member\_no
  - member\_no → id
  - 2 Keys: id | member\_no
- Issues:**
  - member\_no is needed for issue / return queries. It is unnecessary to have faculty details with that.
  - member\_no may also come from **students** relation.
  - member\_type is needed for issue / return queries. This is implicit by the fact that we are in faculty relation.

At the bottom, there is a footer with the text 'SWAYAM: NPTEL-NOC's Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '23.20', and '©Silberschatz, Korth and Sudarshan'. There is also a set of navigation icons at the bottom right.

Moving on look at the faculty we have a very similar situation as of the **student id → all attributes** of the faculty , **id → member\_number**, **member number → id** every faculty has two unique numbers two keys. And we have similar type of issues as we have seen in terms of student. So, we will need to do something in terms of this.

(Refer Slide Time: 25:34)

The slide has a header 'Schema Refinement' and a small logo of a sailboat in the top left corner. On the right, there is a small video window showing a person speaking. The main content is a bulleted list under the heading 'Consider a query:'.

- Consider a query:
  - Get the name of the member who has issued the book having accession number = 162715
    - \* If the member is a student,
      - SELECT student\_fname as First\_Name, student\_lname as Last\_Name
      - FROM students, book\_issue
      - WHERE accession\_no = 162715 AND book\_issue.member\_no = students.member\_no;
    - \* If the member is a faculty,
      - SELECT faculty\_fname as First\_Name, faculty\_lname as Last\_Name
      - FROM faculty, book\_issue
      - WHERE accession\_no = 162715 AND book\_issue.member\_no = faculty.member\_no;
    - Which query to fire!

At the bottom, there is a note: 'These are sample indicative code not checked for syntax.' and some navigation icons. The footer includes 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kanpur', 'Database System Concepts - 8<sup>th</sup> Edition', '23.21', and '©Silberschatz, Korth and Sudarshan'.

So, what can us; I mean what kind of issues we will get into. So, let us consider a query that the query is to get the name of a member of the member who has issued a book say having accession number some accession number 162715.

Now, if we have to write a select query for this we will need to know whether the select query should be written on the student or with the faculty. So, if this member is a student then we need the first query where we do a join between student and book issue to find out the student member number who is issued that book where the accession number gives me the particular book issue record from this result will come. Similarly, if the member is a faculty I need a different query. Now, it is a problem because if I have two possible queries and based on the member number I have to decide which query to fire we have not done any mechanism like that. So, this design has problems.

(Refer Slide Time: 26:36)

The slide has a header 'Schema Refinement' and a small logo of a sailboat in the top left. On the right, there is a small video window showing a man speaking. The main content discusses the refinement of a 'members' entity set based on member categories: undergraduate students, post graduate students, research scholars, and faculty members. It lists attributes like member\_no, member\_class (student or faculty), member\_type (ug, pg, rs, fc), roll\_no, and id. It also mentions hidden relationships between student and faculty members and the type of relationship (one-to-one).

There are four categories of members of the library: undergraduate students, post graduate students, research scholars, and faculty members. This leads to the following specialization relationships.

- Consider the entity set **members** that represent the behavior of the member of a library and refine:
  - Attributes:
    - ✖ member\_no
    - ✖ member\_class – ‘student’ or ‘faculty’, used to choose table
    - ✖ member\_type – ug, pg, rs, fc, ...
    - ✖ roll\_no (if member\_class – ‘student’. Else null)
    - ✖ id (if member\_class – ‘faculty’. Else null)
  - We can then exploit some hidden relationship:
    - students IS\_A members
    - faculty IS\_A members
  - Type of relationship
    - One-to-one

So, let us see what we can do? So, we again go back to the specification and see what the specification says; it said that there are four categories of members undergraduate, postgraduate, research scholar, and faculty members and least to the least to the specialization of this relationship and we have already done a members concept members entity we did.

Which has a member number and the member type, but now we have just seen that it actually matters as to whether the member is a student or is a faculty is a more unique deciding factor in terms of, how we design our query? So, we introduce a new attribute which was not specified explicitly say; let us call it member class which can take only two values either student or faculty and then retain the member type. So, what it will mean that the; if the student class is student then the member type could be ug, pg or rs and if the member class is faculty.

Then the person is a faculty than the member type should be only fc and then also maintain the roll number and id in this members table. So, the roll number ah; obviously, if it is if the member class is student then the person is a student and the roll number will exist, but the id which is an employee id will be null and at the same time if member class is a faculty for a record then the roll number will be null because, a faculty cannot have a roll number, but the id will be present.

So, we extend the members entity set in relational schema with these attributes and once we do that then we kind of exploit a hidden relationship which was not very explicitly stated anywhere that; now we can say that students is a members faculty is a members, that is; from the perspective of issuing books and using the library both students and faculty can be considered to be members it is kind of a you know virtual entity that we can see here and certainly there will be a one to one relationship between the students and members and faculty and members.

(Refer Slide Time: 28:39)

**Schema Refinement**

- Consider the old query again:
  - Get the name of the member who has issued the book having accession number = 162715

```

SELECT
  ((SELECT faculty_fname as First_Name, faculty_lname as Last_Name
  FROM faculty
  WHERE member_class = 'faculty' AND members.id = faculty.id)
UNION
  (SELECT student_fname as First_Name, student_lname as Last_Name
  FROM students
  WHERE member_class = 'student' AND members.roll_no = students.roll_no))
FROM members, book_issue
WHERE accession_no = 162715 AND book_issue.member_no = members.me
  
```

These are sample indicative code not checked for syntax errors.

Database System Concepts - 8<sup>th</sup> Edition      23.23      ©Silberschatz, Korth and Sudarshan

So, let us see what is a consequence of that; in terms of the earlier query. So, we look at go back and look at the query again we will see that problems have got significantly solved, because we now still have to find that member name and this is the basic condition which say.

But, I am sorry this is a basic condition which says the member number who has issued that book, but the actual problem of finding the name can be solved here, because I can write two queries and take a union of the first query runs on faculty the other query runs on student. Now, here since I have a member class which tell me whether the member is a faculty or the member is a student. So, one of these queries will actually return a null result will not return any record because it will not match this condition, but the other one we will match and will give me the record give me the corresponding names first name and last name of the member and certainly to ensure that

when we are looking at the faculty. We need to check the equality of member id between the members relation and the faculty relation and while dealing with the student we need to check for the equality of the roll number.

So, in this way by using intelligently using the union feature and the you know nested query feature we can easily write a query and where implicitly. Now, based on the data the switching of which table I am I am actually looking the data from will get solved.

(Refer Slide Time: 30:17)

The screenshot shows a presentation slide with the title "Schema Refinement" in red at the top center. To the left of the title is a small icon of a sailboat on water. On the right side, there is a small video window showing a man with glasses and a blue shirt, likely the lecturer. The main content area contains a bulleted list under the heading "members(member\_no, member\_class, member\_type, roll\_no, id)". The list includes:

- members(member\_no, member\_class, member\_type, roll\_no, id)
  - member\_no → member\_type, member\_class, roll\_no, id
  - member\_type → member\_class
  - Key: member\_no

At the bottom of the slide, there is some footer text: "SWAYAM: NPTEL-NOC's Instructional Platform", "Prof. P. P. Das, IIT Kharagpur - Jan-Apr, 2018", "Database System Concepts - 8th Edition", "23.24", and "©Silberschatz, Korth and Sudarshan".

So, with this refinement my members schema now turns out to be **member number → member class, member type, roll number, and id member, number** and **member type → member class**, because if I know some member type is undergraduate I know member class is student and so, on key; obviously, is one number.

(Refer Slide Time: 30:38)

The slide has a header 'Schema Refinement' and a footer with the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018'.

**■ students**(student\_fname, student\_lname, roll\_no, department, gender, mobile\_no, dob, degree)

- roll\_no → student\_fname, student\_lname, department, gender, mobile\_no, dob, degree
- Keys: roll\_no
- Note:
  - ▶ member\_no is no longer used
  - ▶ member\_type and member\_class are set in **members** from degree at the time of creation of a new record.

**■ faculty**(faculty\_fname, faculty\_lname, id, department, gender, mobile\_no, doj)

- id → faculty\_fname, faculty\_lname, department, gender, mobile\_no, doj
- Keys: id
- Note:
  - ▶ member\_no is no longer used
  - ▶ member\_type and member\_class are set in **members** at the time of creation of a new record

PPD

Database System Concepts - 8<sup>th</sup> Edition      23.25      ©Silberschatz, Korth and Sudarshan

So having done that, we again go back to the student and faculty, because now we do not need member number in that anymore; because these will not directly be involved in the issue return, because all that you need is just the member number which is already there in the members. So, now, it is a roll number which determines everything member number is no longer used.

And member type and member class which we have assumed is exist in the in the members will have to be derived from the degree value at the time when a new record is created. So, when a new student record is created an entry will happen in this relation as well as a corresponding entry we will need happen in the members relation where the membership number is given and the membership type will be derived from the degree similar change will happen in terms of the faculty as well.

(Refer Slide Time: 31:31)

The slide title is "Schema Refinement – Final". The content lists eight relational schemas:

- book\_catalogue(title, author\_fname, author\_lname, publisher, year, ISBN\_no)
- book\_copies(ISBN\_no, accession\_no)
- book\_issue(member\_no, accession\_no, doi)
- quota(member\_type, max\_books, max\_duration)
- members(member\_no, member\_class, member\_type, roll\_no, id)
- students(student\_fname, student\_lname, roll\_no, department, gender, mobile\_no, dob, degree)
- faculty(faculty\_fname, faculty\_lname, id, department, gender, mobile\_no, doi)
- staff(staff\_fname, staff\_lname, id, gender, mobile\_no, doi)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr- 2018

Database System Concepts - 8<sup>th</sup> Edition 23.26 ©Silberschatz, Korth and Sudarshan

So, after refinement now we have these eight relational schema from book catalogue to give the details of every book copies which keeps the information about, how many; what are the different copies book issue; is the basic issuing information. We have quota members has the virtual kind of relation that we have created to support the notion of members of the library and students and faculty are related to this members either through roll number or through id in a selective manner and staff we have not touched.

(Refer Slide Time: 32:12)

The slide title is "Module Summary". The content lists two key points:

- Using the specification for a Library Information System, we have illustrated how a schema can be designed and then refined for finalization
- Coding of various queries based on these schema are left as exercises

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr- 2018

Database System Concepts - 8<sup>th</sup> Edition 23.27 ©Silberschatz, Korth and Sudarshan

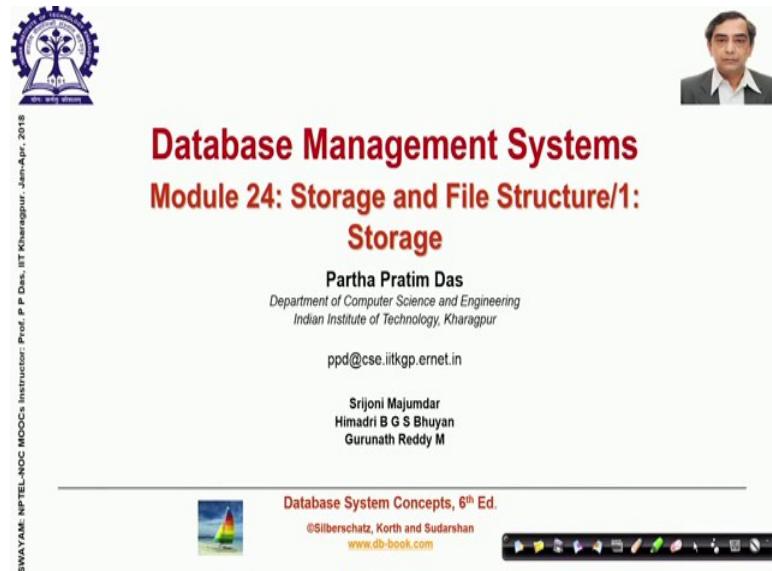
So, this is the final relational schema. In this module I have shown you illustrated you that how starting from a very simple specification you can reason and work through in terms of creating the entity sets attributes and relationships and then get into the relational schema look at the possible functionality dependency and refine that and also look into what will be the requirements of doing your query and come to a final refined schema.

So, various queries that we had talked of and others can be can now be coded on this, but I leave that as an exercise to you please try out coding those queries and you will be able to learn in terms of how to do that well and I will try to also supply some supplementary information on this offline.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 24**  
**Storage and File Structure: Storage**

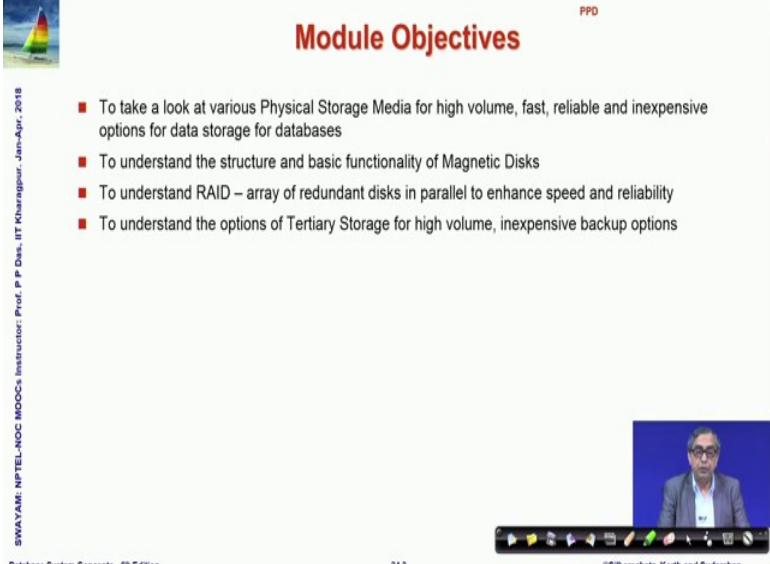
(Refer Slide Time: 00:16)



The slide is titled "Database Management Systems" in red, followed by "Module 24: Storage and File Structure/1: Storage". It features a portrait of Prof. Partha Pratim Das on the right. The footer includes the book title "Database System Concepts, 6<sup>th</sup> Ed.", authors "Silberschatz, Korth and Sudarshan", and the website "www.db-book.com". A decorative footer bar with various icons is also present.

Welcome to module 24 of database management systems in this module and the next we will take a look at the storage and file structure of database systems. So, we will start with the storage.

(Refer Slide Time: 00:30)



This slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner and a video player window showing a man speaking on the right. The text on the slide lists four objectives:

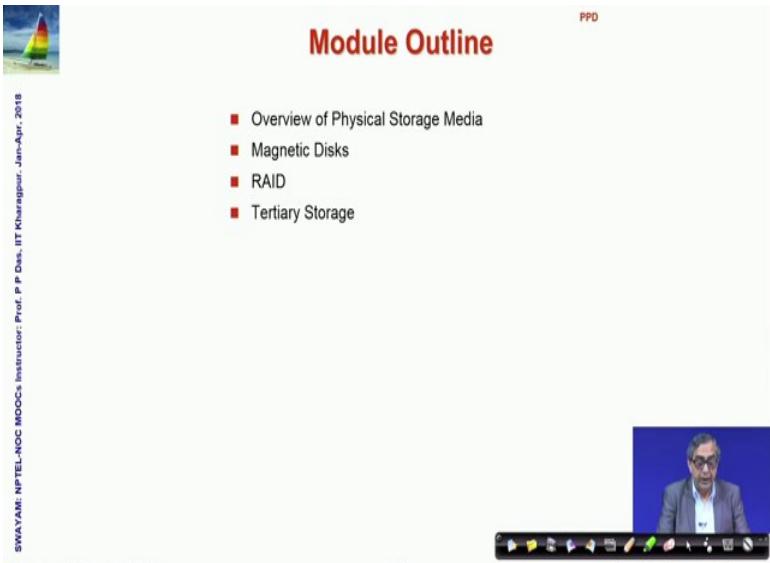
- To take a look at various Physical Storage Media for high volume, fast, reliable and inexpensive options for data storage for databases
- To understand the structure and basic functionality of Magnetic Disks
- To understand RAID – array of redundant disks in parallel to enhance speed and reliability
- To understand the options of Tertiary Storage for high volume, inexpensive backup options

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr- 2018  
PPD

Database System Concepts - 6<sup>th</sup> Edition 24.3 ©Silberschatz, Korth and Sudarshan

So, specifically we want to look at various physical storage medium because. So, far we have been talking only about the logical layer of the database design and now we want to actually look at the in physical terms how the data will be stored what could be the physical storage medium for high volume fast reliable inexpensive options for databases. We would like to understand the structure and basic functionality of magnetic disks, because they are they are most widely used we will try to take the glimpse about RAID which is a kind of a good option in terms of reliable databases and also look at options for the tertiary storage.

(Refer Slide Time: 01:12)



This slide is titled "Module Outline" in red at the top right. It features a small sailboat icon in the top left corner and a video player window showing a man speaking on the right. The text on the slide lists five topics:

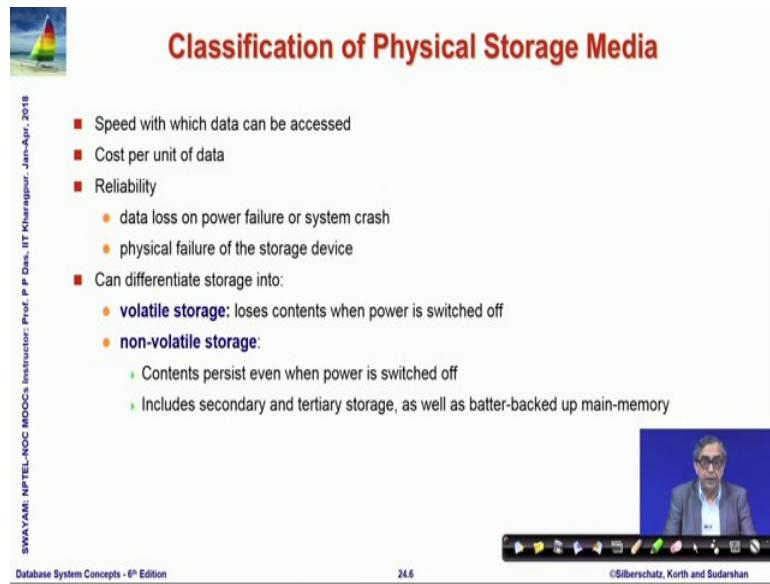
- Overview of Physical Storage Media
- Magnetic Disks
- RAID
- Tertiary Storage

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr- 2018  
PPD

Database System Concepts - 6<sup>th</sup> Edition 24.4 ©Silberschatz, Korth and Sudarshan

So, these are the topics that we will quickly cover in this.

(Refer Slide Time: 01:17)



The slide title is "Classification of Physical Storage Media". It features a small sailboat icon in the top left corner. On the right side, there is a video player window showing a man speaking. The slide content includes a list of factors for classification:

- Speed with which data can be accessed
- Cost per unit of data
- Reliability
  - data loss on power failure or system crash
  - physical failure of the storage device
- Can differentiate storage into:
  - **volatile storage**: loses contents when power is switched off
  - **non-volatile storage**:
    - ↳ Contents persist even when power is switched off
    - ↳ Includes secondary and tertiary storage, as well as battery-backed up main-memory

Small text on the left edge reads: SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition      24.6      ©Silberschatz, Korth and Sudarshan

So, first let us take a overview of the physical storage medium I am sure all of you have known all or parts of this. So, this is, but this is more for completeness to look at from the perspective of a database application. So, some of the the classification of storage media done on different factors the factors includes speed which is the first thing the how fast the data can be accessed the cost per unit of data you can say the rupees per bit or rupees per byte or rupees per kilobyte something like that.

So, which is a cost per unit of data the reliability that is the if we will the data get lost if power fails or if the system crashes and or if this physical you know failure of the storage device and so, on. So, what is the reliability on that; and broadly as you all know we can differentiate storage into volatile storage which loses contents. When the power is switched off and the non volatile storage which are secondary and tertiary storage where the data will continue to stay even when power is off even you have some parts of the memory which may be battery backup which also will be non volatile.

(Refer Slide Time: 02:29)

The slide has a header 'Physical Storage Media' with a sailboat icon. It contains three main sections: 'Cache', 'Main memory', and 'Volatile'. The 'Cache' section lists: fastest and most costly form of storage, volatile, managed by the computer system hardware. The 'Main memory' section lists: fast access (10s to 100s of nanoseconds; 1 nanosecond =  $10^{-9}$  seconds), generally too small (or too expensive) to store the entire database, capacities of up to a few Gigabytes widely used currently, Capacities have gone up and per-byte costs have decreased steadily and rapidly (roughly factor of 2 every 2 to 3 years). The 'Volatile' section lists: contents of main memory are usually lost if a power failure or system crash occurs. The footer includes 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '24.7', and '©Silberschatz, Korth and Sudarshan'.

So, in terms of the physical storage certainly the absolute starting point of the storage is registered in the CPU; we are not talking about that because they are primarily meant for temporary computations. So, in terms of data the first possible level is the cache which is the fastest and most costly form of storage it is volatile in nature and is managed by the computer system hardware.

So, cache typically is a fast semiconductor memory that exist between your main memory and the disk system and it is very fast to work with then comes the main memory which is which has fast access, but compare to cache it may be may be much bigger, but overall it is too small to store an entire database, but I mean every regularly the size of this main memory is increasing. So, the capacity of couple of gigabytes are common these days, but still it is small compare to the requirement of the databases and main memory typically is volatile. So, if the power goes up the system crashes all the data is lost.

(Refer Slide Time: 03:48)

The slide has a title 'Physical Storage Media (Cont.)' at the top right. On the left, there is a small logo of a sailboat on water. The main content is a bulleted list under the heading 'Flash memory'. The list includes:

- Data survives power failure
- Data can be written at a location only once, but location can be erased and written to again
  - ↳ Can support only a limited number (10K – 1M) of write/erase cycles
  - ↳ Erasing of memory has to be done to an entire bank of memory
- Reads are roughly as fast as main memory
- But writes are slow (few microseconds), erase is slower
- Widely used in embedded devices such as digital cameras, phones, and USB keys

At the bottom left, it says 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018'. At the bottom center, it says 'Database System Concepts - 8<sup>th</sup> Edition' and '24.8'. At the bottom right, it says '©Silberschatz, Korth and Sudarshan'.

We have flash memory where the data can survive across power failure; it can be they had data can be written at a location only once, but you can erase and write it again.

So, it is not like in the main memory where you can read write read write like that here you can write and then if you want to write again then you will have to erase and write it. So, the read is very fast in case of flash memory which is almost as fast as a main memory, but writes are slow particularly when you have erase and write it will be a slow process all the kinds of USB keys pen drives digital phone memory that we are often using are actually flash memory.

(Refer Slide Time: 04:35)

The slide is titled "Physical Storage Media (Cont.)" in red at the top right. On the left, there is a small image of a sailboat on water. The main content area is divided into sections. The first section is titled "■ Magnetic-disk" and contains the following bulleted points:

- Data is stored on spinning disk, and read/written magnetically
- Primary medium for the long-term storage of data
  - ↳ typically stores entire database
- Data must be moved from disk to main memory for access, and written back for storage
  - ↳ Much slower access than main memory
- direct-access
  - ↳ possible to read data on disk in any order, unlike magnetic tape
- Capacities range up to roughly 16~32 TB
  - ↳ Much larger capacity and cost/byte than main memory/flash memory
  - ↳ Growing constantly and rapidly with technology improvements (factor of 2 to 3 every 2 years)
- Survives power failures and system crashes
  - ↳ disk failure can destroy data, but is rare

On the right side of the slide, there is a video player window showing a man speaking, likely the instructor. Below the video player is a navigation bar with icons for back, forward, search, and other controls. At the bottom of the slide, there is footer text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018", "Database System Concepts - 8<sup>th</sup> Edition", "24.9", and "©Silberschatz, Korth and Sudarshan".

Then you have the magnetic disk where the data is stored on spinning disk and it is typically written and read magnetically. So, this is the primary medium for long term storage of large volume of data. So, data needs to be moved from disk to the main memory and written back for permanent storage. So, it is ways slower compare to the main memory and, but it is has a kind of direct access which means that it is possible to read data on this disk in any arbitrary order in compare to magnetic disk.

Which is the serial device here it is a, it is kind of a I can do things in parallel at random in any order capacity is go up to tens of terabytes easily and it can survive for failure and system crashes, because it will the magnetic recording will still be there if the disk itself fails then it will the data will get distract, but such a situation is usually rear.

(Refer Slide Time: 05:40)

The slide is titled "Physical Storage Media (Cont.)" and features a small sailboat icon in the top left corner. On the right side, there is a video player window showing a man speaking. The video player has a blue background and includes standard controls like play, pause, and volume. The main content area contains a bulleted list under the heading "Optical storage".

**Optical storage**

- non-volatile, data is read optically from a spinning disk using a laser
- CD-ROM (640 MB) and DVD (4.7 to 17 GB) most popular forms
- Blu-ray disks: 27 GB to 54 GB
- Write-one, read-many (WORM) optical disks used for archival storage (CD-R, DVD-R, DVD+R)
- Multiple write versions also available (CD-RW, DVD-RW, DVD+RW, and DVD-RAM)
- Reads and writes are slower than with magnetic disk
- Juke-box systems, with large numbers of removable disks, a few drives, and a mechanism for automatic loading/unloading of disks available for storing large volumes of data

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018

Database System Concepts - 8<sup>th</sup> Edition

24.10

©Silberschatz, Korth and Sudarshan

We have different optical storage devices CD-ROM, DVD and so, on the juke box systems and which are also non volatile and data is written optically here, that is by using a laser light on the spinning disk use a typically optical storages are removable media.

(Refer Slide Time: 06:03)

The slide is titled "Physical Storage Media (Cont.)" and features a small sailboat icon in the top left corner. On the right side, there is a video player window showing a man speaking. The video player has a blue background and includes standard controls like play, pause, and volume. The main content area contains a bulleted list under the heading "Tape storage".

**Tape storage**

- non-volatile, used primarily for backup (to recover from disk failure), and for archival data
- sequential-access**
  - much slower than disk
- very high capacity (40 to 300 TB tapes available)
- tape can be removed from drive  $\Rightarrow$  storage costs much cheaper than disk, but drives are expensive
- Tape jukeboxes available for storing massive amounts of data
  - hundreds of terabytes (1 terabyte =  $10^{12}$  bytes) to even multiple **petabytes** (1 petabyte =  $10^{15}$  bytes)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018

Database System Concepts - 8<sup>th</sup> Edition

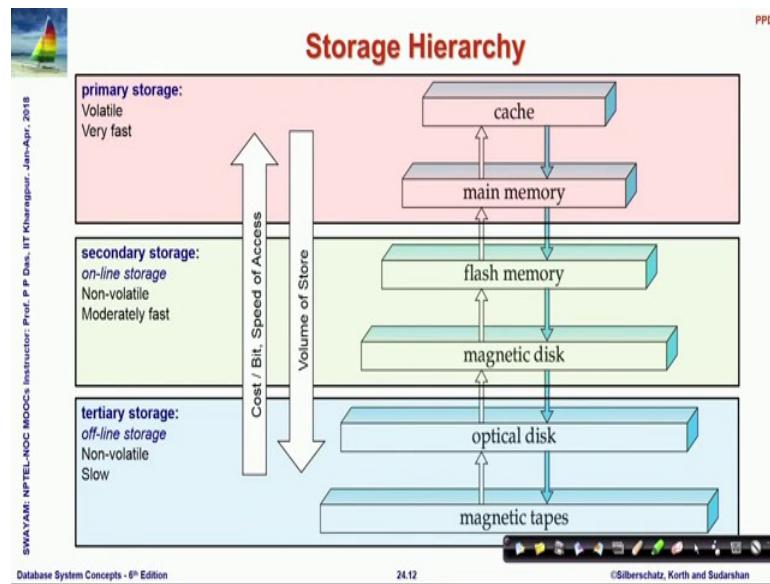
24.11

©Silberschatz, Korth and Sudarshan

Then you have the tape storage which usually is a largest volume of storage, but it is as a name suggests it is a tape it is a linear device. So, access can only be sequential. So, if you want to read the 6th record you have to skip of a record 1 to 5, but it can be a very high capacity usually it is slow.

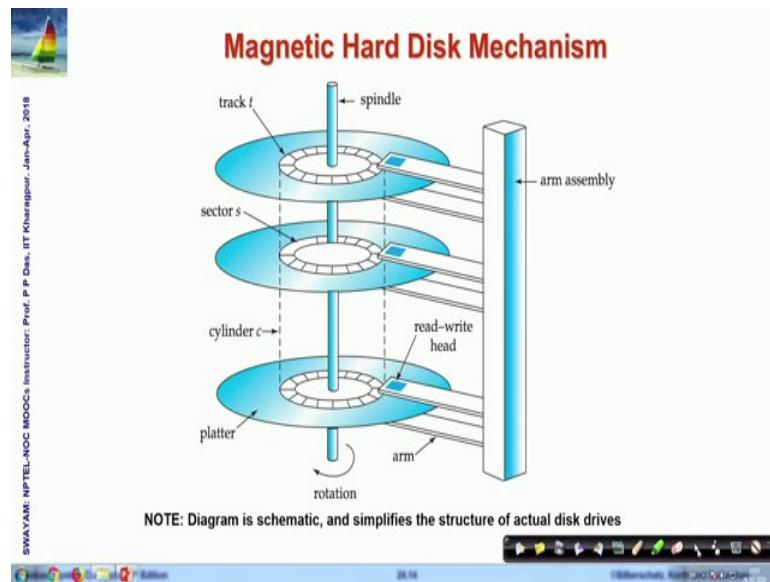
But very large in volume and tape juke boxes can support hundreds of terabytes or even multiple of petabyte. So, that is for offline storage.

(Refer Slide Time: 06:35)



This is a big medium. So, this is the basic storage hierarchy. So, we broadly classify them into three groups primary storage which is volatile and very fast cache and main memory is within that or secondary storage which is an online store which is non volatile and moderately fast and tertiary storage is called the offline store which is non volatile and slow. So, flash memory and magnetic disk are secondary storage they are non volatile and moderately fast and they are online. So, they exist with the system whereas, optical disk and magnetic disk can be removed and taken elsewhere.

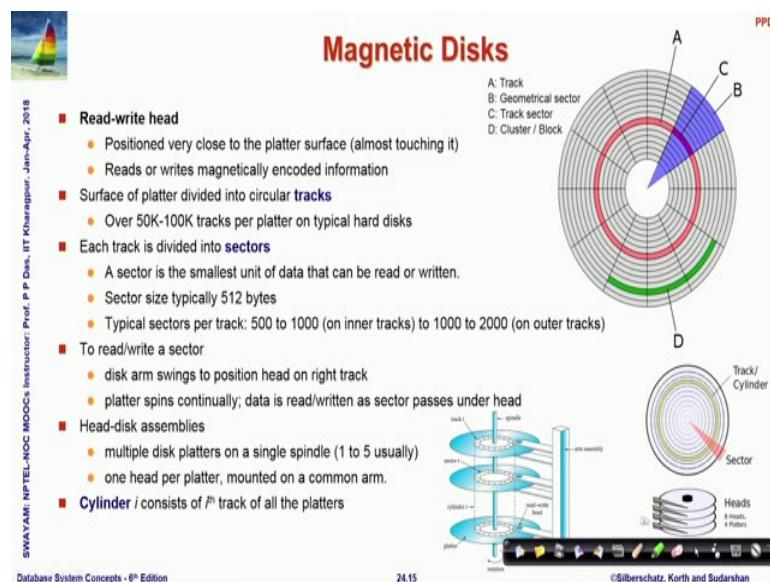
(Refer Slide Time: 07:12)



So, let us quickly take a look into the magnetic disk this is how a typical magnetic disk looks like; these are the different cylinders these are the different disks that we have and these are all different read write head.

So, as you can see all of them can work along a path backward forward like this and they can come and parallelly all of them parallelly can read from the different disks and this disks keep on spinning to help you look at the data anywhere on the disk. So, that is a typical structure of a magnetic.

(Refer Slide Time: 07:50)



Disk if you look specifically into. So, this is this is the structure we saw and if you specifically look into one particular disk then the disk is radially divided into different sectors portions. So, these are these are all separate portions and you can at a time the head can read one such sector.

So, these are called the geometric sectors and the whole of the ring that you can see the cylindrical ring that you can see is called a track. So, this is a track sector the orange one is a track sector and often we take multiple sectors from a particular track and combine them into one unit this is called the block of data and we will often talk about the block of data.

So, here at the typical numbers of how these sizes of this different units turn out to be.

(Refer Slide Time: 08:44)

The slide has a title 'Magnetic Disks (Cont.)' in red. To the left is a small sailboat icon. The main content is a bulleted list under two sections:

- Earlier generation disks were susceptible to head-crashes
  - Surface of earlier generation disks had metal-oxide coatings which would disintegrate on head crash and damage all data on disk
  - Current generation disks are less susceptible to such disastrous failures, although individual sectors may get corrupted
- Disk controller – interfaces between the computer system and the disk drive hardware
  - accepts high-level commands to read or write a sector
  - initiates actions such as moving the disk arm to the right track and actually reading or writing the data
  - Computes and attaches **checksums** to each sector to verify that data is read back correctly
    - If data is corrupted, with very high probability stored checksum won't match recomputed checksum
  - Ensures successful writing by reading back sector after writing it
  - Performs remapping of bad sectors

At the bottom, there is a small video window showing a person speaking, the text 'Database System Concepts - 8<sup>th</sup> Edition', the number '24.16', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, the early generation where of disk were susceptible to head crashes, but now a days it is moved it quite stable there are disk controllers which regularly check and manage the; read write into in terms of the sectors naturally the. So, many heads being in parallel the data can be written on to multiple sectors at the same time and so, for doing that.

(Refer Slide Time: 09:12)

The diagram illustrates a Disk Subsystem architecture. At the top, there is a small sailboat icon. Below it, the title "Disk Subsystem" is centered. A horizontal line labeled "system bus" runs across the middle. On the left side of the bus, there is a rectangular box labeled "disk controller". Four vertical lines descend from the controller to four circular shapes labeled "disks". A vertical line of text on the left edge reads: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018". On the right side, there is a video frame showing a man speaking, with a progress bar and other video controls below it. At the bottom of the slide, the text "Database System Concepts - 8<sup>th</sup> Edition" and "24.17" are on the left, and "©Silberschatz, Korth and Sudarshan" is on the right.

We will see what is the mechanism for that; and we also have disk subsystems where if you need a large volume of data to be managed which is larger than a single disk. Then you can connect multiple disk and through a disk controller you can actually read write data on to them and there are different such disk interface standards that you may have heard of.

(Refer Slide Time: 09:38)

The diagram illustrates a Disk Subsystem architecture. At the top, there is a small sailboat icon. Below it, the title "Disk Subsystem" is centered. A horizontal line labeled "system bus" runs across the middle. On the left side of the bus, there is a rectangular box labeled "disk controller". Four vertical lines descend from the controller to four circular shapes labeled "disks". A vertical line of text on the left edge reads: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018". On the right side, there is a video frame showing a man speaking, with a progress bar and other video controls below it. At the bottom of the slide, the text "Database System Concepts - 8<sup>th</sup> Edition" and "24.18" are on the left, and "©Silberschatz, Korth and Sudarshan" is on the right.

We will just mention that these are the typical storages the SAN and NAN are the typical storages.

So, if you come across these terms we will not go into details of that that takes us into a different course of hardware discussion, but these are the very common storages for database disk systems.

(Refer Slide Time: 09:56)

The slide has a decorative header with a sailboat icon and the title 'Performance Measures of Disks' in red. The content is organized into sections with bullet points:

- **Access time** – the time it takes from when a read or write request is issued to when data transfer begins. It consists of:
  - **Seek time** – time it takes to reposition the arm over the correct track
    - Average seek time is 1/2 the worst case seek time
      - Would be 1/3 if all tracks had the same number of sectors, and we ignore the time to start and stop arm movement
    - 4 to 10 milliseconds on typical disks
  - **Rotational latency** – time it takes for the sector to be accessed to appear under the head
    - Average latency is 1/2 of the worst case latency.
    - 4 to 11 milliseconds on typical disks (5400 to 15000 rpm)
- **Data-transfer rate** – the rate at which data can be retrieved from or stored to the disk.
  - 25 to 100 MB per second max rate, lower for inner tracks
  - Multiple disks may share a controller, so rate that controller can handle is also important
    - E.g. SATA: 150 MB/sec, SATA-II 3Gb (300 MB/sec)
    - Ultra 320 SCSI: 320 MB/s, SAS (3 to 6 Gb/sec)
    - Fiber Channel (FC2Gb or 4Gb): 256 to 512 MB/s

Small video player window showing a person speaking is visible in the bottom right corner. The footer contains the text 'SWAYAM: NPTEL-NOCOCS Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '24.19', and '©Silberschatz, Korth and Sudarshan'.

Now basic question is for doing this read write on the disk what kind of performance measures we should look at. So, one main measure is access time if I want to access a record from the disk, then how much time shall I need. So, this is based on two major components one is a seek time now naturally as we have seen that the disk platter as a whole range of read heads which are positioned.

So, to be able to get the data the platter has to the head has to move forward or backward to the right track on which the data is there. So, this time it takes to go from current position to the correct track where, I find the data is called the seek time. Now, even when it is come to the; correct position in terms of the correct track it may not actually be on the correct sector.

Because, it is at the whole track is a is kind of a circle. So, it may be at one part of the circle and the actual data may be in a sector which is in a different part of the circle. So, the disk will have to rotate. So, that the correct sector comes under the head which is already positioned through the seek. So, the time to seek plus the rotational latency the time it takes the for the sake sector to be access is gives the total access time and then comes the other measure which is the data transfer rate as to you using this then what is

the rate how many megabytes and so, on can be transferred at from this. So, it that depends on a besides the access time you will have to see what is the rate at which the disk can be copied from the magnetic medium to the semiconductor medium and transferred.

(Refer Slide Time: 11:52)

The screenshot shows a presentation slide with a title 'Performance Measures (Cont.)'. On the right side, there is a video player window displaying a man speaking. The video player has standard controls like play, pause, and volume. The slide content includes a section on 'Mean time to failure (MTTF)' with bullet points explaining its definition, typical values (3 to 5 years), and the probability of failure for new disks (500,000 to 1,200,000 hours). A note also states that MTTF decreases as disk ages. The footer of the slide includes the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Date, IIT Kanpur - Jan-Apr- 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '24.20', and '©Silberschatz, Korth and Sudarshan'.

Ah the other performance measure that we are concerned with in the database system is known as MTTF which is mean time to failure.

Because, database systems as you know are dealing with persistent data. So, data has to exist and therefore, the disk on which we keep the data must be very very reliable. So, meantime to failure is conceptually that if you consider two failures of the database of the disk to consecutive failures then what is the time the time elapsed between them the average of the time that has elapse between them now. So, if we say the, it is typically now mean time to failure for this magnetic disk that we use today at typically 3 to 5 years.

So, the probability of the failure of a new disk is very very low. So, it is kind of a theoretical MTTF we which will be said like this which; obviously, in terms of hours it it does not really make a make a physical science. So, what it in technical in in more practical terms, what it means that if you are given thousand relatively new disks then on average one of them will fail every twelve hundred hours.

So, this is what is a. So, MTTF certainly decrease it will decrease as a disk becomes old. So, it decreases with age. So, they will start failing sooner than it is to do.

(Refer Slide Time: 13:22)

The slide has a title 'Optimization of Disk-Block Access' at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list under the heading 'Block – a contiguous sequence of sectors from a single track'. The list includes:

- data is transferred between disk and main memory in blocks
- sizes range from 512 bytes to several kilobytes
  - Smaller blocks: more transfers from disk
  - Larger blocks: more space wasted due to partially filled blocks
  - Typical block sizes today range from 4 to 16 kilobytes

SWAYAM: NPTEL-NOC: Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

24.21

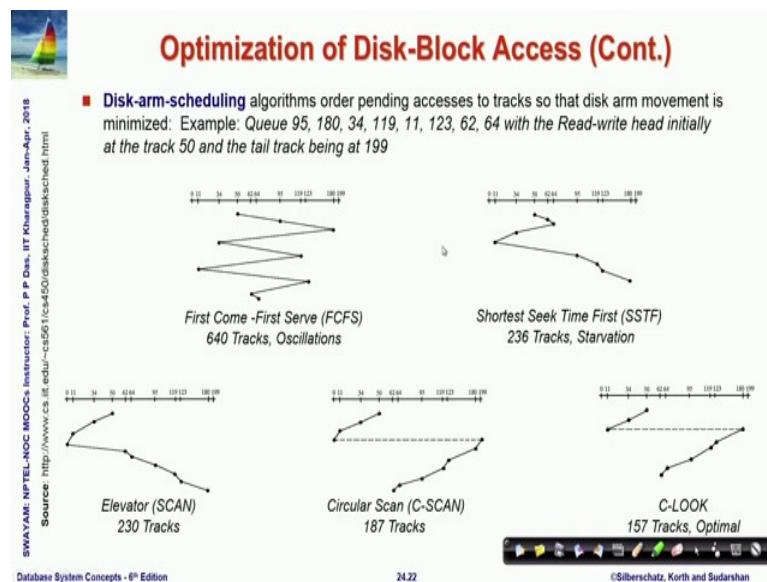
©Silberschatz, Korth and Sudarshan

Now we have to basic objective is to transfer the data at a fast rate. So, what we try to optimize is a; what is called a block the contiguous sequence of sectors from a single track which I mentioned earlier. So, this is what we will be read at one go. So, once you access the data one block will be read block size can range from 512 bytes to several kilobytes. If the blocks are smaller than naturally will need more transfers from the disk if the blocks are larger then you might waste lot of space because your part of the block will not can be used with the data.

So, with all that consideration the typical blocks size that use today is 4 to 16 kilobytes. So, you will see that in all our subsequent discussions particularly with the access and the file organization and the indexing we will consider that a block is one unit.

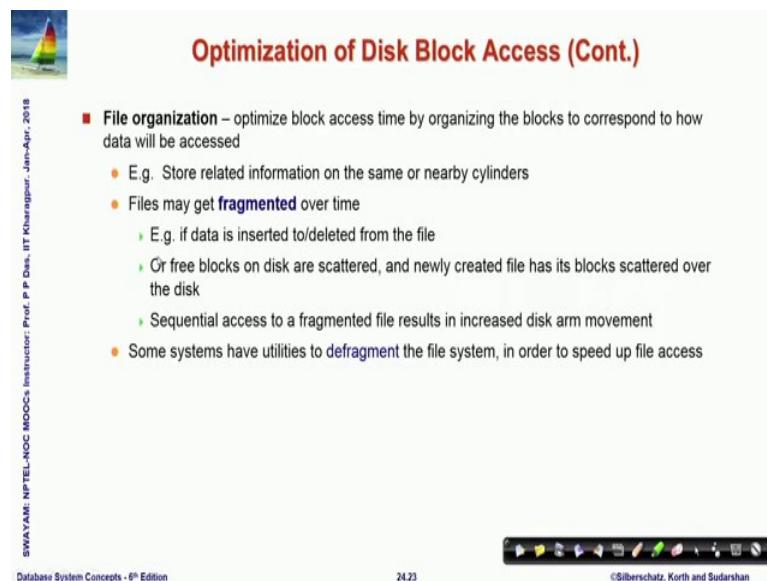
Which can be fetched in one go and that determines the size of the basic information node where the information about the records will be maintained.

(Refer Slide Time: 14:36)



This is the more details in terms of how you move your disk head. So, I think will skip this is more advance material.

(Refer Slide Time: 14:46)



So, to optimize the block access we need to organize the blocks corresponding to how the data will be accessed. So, certainly if the related data are kept in nearby blocks then naturally you are disk head and the rotation will have to be mean will get minimized. So, the basic idea is store the related information in nearby cylinders in the nearby cylindrical practice, but as you keep on using you may start with that, but as you keep on

using for various types of insert delete and overflows that keeps on happening. So, the data gets very fragmented which means that the data gets spread over a whole lot of widely separated cylinders and so, on.

So, the time to actually seek the data increases. So, we can correct that by doing what is called a defragmentation process. So, by defragmentation what you do is your data which is got distributed all over the disk you try to bring them together again to logically continuous physically contiguous blocks so, that their access time can improve. So, databases often will periodically defragment the file system.

(Refer Slide Time: 16:06)

The slide has a title 'Optimization of Disk Block Access (Cont.)' in red at the top right. On the left is a small logo of a sailboat on water. The main content area contains three bullet points under different sections:

- **Nonvolatile write buffers** speed up disk writes by writing blocks to a non-volatile RAM buffer immediately
  - Non-volatile RAM: battery backed up RAM or flash memory
    - Even if power fails, the data is safe and will be written to disk when power returns
  - Controller then writes to disk whenever the disk has no other requests or request has been pending for some time
  - Database operations that require data to be safely stored before continuing can continue without waiting for data to be written to disk
  - Writes can be reordered to minimize disk arm movement
- **Log disk** – a disk devoted to writing a sequential log of block updates
  - Used exactly like nonvolatile RAM
    - Write to log disk is very fast since no seeks are required
    - No need for special hardware (NV-RAM)
- File systems typically reorder writes to disk to improve performance
  - **Journaling file systems** write data in safe order to NV-RAM or log disk
  - Reordering without journaling: risk of corruption of file system data

At the bottom right is a video frame showing a man speaking. The footer includes the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr-2018', 'Database System Concepts - 8<sup>th</sup> Edition', '24.24', and '©Silberschatz, Korth and Sudarshan'.

The other way look at the optimize block access is by using buffers. So, idea of the buffer is suppose you want to write some data to the disk. So, naturally for writing the data also you will need a seek time you will need the rotational latency then we will have to data do the data transfer.

So, if you are trying to do some write you have you can take another option that you actually write that to a buffer a buffer which is in the memory in the it is a in the a semiconductor buffer where you can very quickly write and then when you have enough data in the buffer then you can take them in a single go to the disk.

And write that or when the disk is not actually doing something some access you can use those cycles to write that now naturally if you write things onto the buffer then it is

possible that while your buffer has some data which has actually not been written to the disk if the power fails then you will lose that data. So, parts of the data which you think have been written actually have not gone to the disk. So, to take care of that often non volatile memory ram are used which are battery backed up or flash memory.

So, that even if power fails the right buffers will not be lost. So, we say that use non volatile buffers and other option that is used often is you maintain a log disk a log disk is nothing, but when you are writing to the disk you make a sequential log of the updates that you are doing. So, this kind of is a report. So, you are saying that I have written this data to this block written this data to this block.

So, if there is a failure, then if you can go through the log and you can actually retrieve the information of what was lost how far it actually worked correctly and you can use exactly like the non volatile ram and using the log you can find out. So, you are keeping a kind of a general link system you are keeping information of this is what I did this is what I did. So, all this write information are maintained in terms of the log.

(Refer Slide Time: 18:30)

The slide is titled "Flash Storage" in red at the top right. It features a small sailboat icon in the top left corner. The content is organized into two main sections: "NOR flash vs NAND flash" and "NAND flash".

- NOR flash vs NAND flash
- NAND flash
  - used widely for storage, since it is much cheaper than NOR flash
  - requires page-at-a-time read (page: 512 bytes to 4 KB)
  - transfer rate around 20 MB/sec
  - **solid state disks**: use multiple flash storage devices to provide higher transfer rate of 100 to 200 MB/sec
  - erase is very slow (1 to 2 millisecs)
    - erase block contains multiple pages
    - remapping of logical page addresses to physical page addresses avoids waiting for erase
      - **translation table** tracks mapping
        - also stored in a label field of flash page
      - remapping carried out by **flash translation layer**
    - after 100,000 to 1,000,000 erases, erase block becomes unreliable and cannot be used
      - **wear leveling**

You can use flash storage nowadays the NAND based flash storage is very common.

(Refer Slide Time: 18:40)

The slide has a header 'PPD' and a sidebar with the text: 'Overview of Physical Storage Media', 'Magnetic Disks', 'RAID', and 'Tertiary Storage'. It features a large red title 'RAID: REDUNDANT ARRAY OF INDEPENDENT DISKS'. The footer includes 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '24.26', '©Silberschatz, Korth and Sudarshan', and a navigation bar.

So, here are some details on that let us move on to understanding. So, we just took a look into the basic storage hierarchy and the use of magnetic disks and what are the parameters that control the performance in terms of the read write in terms of the disk RAID is a the full form is redundant array of independent disks.

(Refer Slide Time: 19:05)

The slide has a header 'RAID' and a sidebar with the text: 'RAID: Redundant Arrays of Independent Disks'. It lists several points about RAID, including its benefits (high capacity and speed), reliability (redundant storage), and failure probability. It also notes its cost-effectiveness and current use. The footer includes 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '24.27', '©Silberschatz, Korth and Sudarshan', and a video thumbnail of a professor speaking.

So, the disk organization that manage a large number of disk, but the view that you get you do not get to see the multiple disk you get to see a single disk.

So, by this you can create very high capacity and very high speed and. So, because you have multiple disks so, you organize a data in such a way that when you are writing or you are reading actually you can parallelly read or write from multiple different disks. So, that not only increases capacity, but actually increases a throughput and you can get high reliability also by storing the data redundantly; that is keeping multiple copies and that is what RAID is often quite known for.

So, originally when RAID was originally designed actually the, I in red stood for inexpensive. So, it was kind of a disk array which was inexpensive to afford, but now it is not particularly the expenses is not the primary factor for which we do go for RAID, but we go for RAID for the high capacity high speed and high reliability. So, the I is now interpreted as independent array.

(Refer Slide Time: 20:21)

**Improvement of Reliability via Redundancy**

SWAYAM: NPTEL-NOCO MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018

- **Redundancy** – store extra information that can be used to rebuild information lost in a disk failure
- **Mean time to data loss** depends on mean time to failure, and **mean time to repair**
  - E.g. MTTF of 100,000 hours, mean time to repair of 10 hours gives mean time to data loss of  $500 \times 10^6$  hours (or 57,000 years) for a mirrored pair of disks (ignoring dependent failure modes)
- **Mirroring (or shadowing)**
  - Duplicate every disk. Logical disk consists of two physical disks.
  - Every write is carried out on both disks
    - » Reads can take place from either disk
  - If one disk in a pair fails, data still available in the other
    - » Data loss would occur only if a disk fails, and its mirror disk also fails before the system is repaired
      - Probability of combined event is very small
      - » Except for dependent failure modes such as fire or building collapse or electrical power surges

Database System Concepts - 8<sup>th</sup> Edition      24.28      ©Silberschatz, Korth and Sudarshan

So, we can improve. So, now, the main issue in red is to improve reliability. So, the main the key idea is have redundancy to improve the reliability that is stored the data in multiple copies and can rebuild from the copies once a disk has failed. So, we earlier looked at the MTTF mean time to failure. Now, we are look at another parameter which we say is a mean time to data loss. So, which depends on mean time to failure, because if you are if you are failed then you have lost the data, but when you have failed you have a possibility now, to recover the data to repair it; because you have redundant copies.

So, mean time to data loss it depends on the MTTF plus the mean time to repair how quickly can we use your redundant copies and get back the original data. So, mean time to data loss is the actual key factor which needs to be minimized and for this red does a mirroring or shadowing that is for every disk there is a duplicate there is a clone.

So, it logically you have one disk, but physically you have actually two disk. So, every write that you do is carried out to both the disks and when you read the read happens on either of the disk usually it switches between the disks. So, if one of the disk in the pair would fail, then the data can still be recovered from the other and the data loss would occur if a disk fail, but the mirror disk also has to fail before the system has been repair.

So, if one of the disk fail you get the data from the middle disk you continue to do that from the mirror disk you restore the other disk you may be replace and put a new one mirror it again and. So, on, but you can restore that. So, in between this time if the mirror disk also fails; then you have actually lost data, but the probability of that is very very small. So, mirroring gives a at the expense of naturally having lot more of redundant storage the mirroring can actually give you a much higher reliability.

(Refer Slide Time: 22:39)

The slide is titled "Improvement of Reliability via Redundancy". It features a small sailboat icon in the top left corner. The main content is a bulleted list of redundancy techniques:

- **Bit-level striping** – split the bits of each byte across multiple disks
  - In an array of eight disks, write bit  $i$  of each byte to disk  $i$
  - Each access can read data at eight times the rate of a single disk
  - But seek/access time worse than for a single disk
    - ▶ Bit level striping is not used much any more
- **Block-level striping** – with  $n$  disks, block  $i$  of a file goes to disk  $(i \bmod n) + 1$ 
  - Requests for different blocks can run in parallel if the blocks reside on different disks
  - A request for a long sequence of blocks can utilize all disks in parallel

At the bottom of the slide, there is footer text: "SWAYAM: NPTEL-NOCO MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jam-Apr-2018", "Database System Concepts - 6<sup>th</sup> Edition", "24.29", and "©Silberschatz, Korth and Sudarshan". There is also a set of navigation icons at the bottom right.

That we expect today another some of the other techniques which improve reliability is bit level and byte level block level striping techniques. So, what you in the basic bit level striping what you do is a say every byte has 8 bits. So, when you are writing a byte you

do not write all the bytes to the same disk you write them to multiple disks. So, you take an array of 8 disks. So, write bit  $I$  of each byte to disk  $I$  it is. So, did interesting concept you have fragmenting it in a very peculiar way.

So, you have 8 disks and every byte first byte first bit is written to one disk second byte is second bit is written to the second disk, third bit is written to the third disk and so, on. And when you access you can access from all these 8; now naturally which means that this will decrease your throughput to some extent, because you have to collect from all of that reconstruct. So, bits levels striping is not much in use any more instead you have block level striping where with end disk a block  $I$  of a file goes to the disk  $i \bmod n + 1$ .

So, circularly so, the first goes if you have say five disks in the first block goes to disk one second to disk two-fifth to disk 5 and the 6th again back to disk 1. So, the request to different blocks can run in parallel and reside on different disk and if they can easily utilize this parallelism to improve the throughput.

(Refer Slide Time: 24:30)

**Improvement of Reliability via Redundancy**

■ **Bit-Interleaved Parity** – a single parity bit is enough for error correction, not just detection, since we know which disk has failed

- When writing data, corresponding parity bits must also be computed and written to a parity bit disk
- To recover data in a damaged disk, compute XOR of bits from other disks (including parity bit disk)

■ **Block-Interleaved Parity**: Uses block-level striping, and keeps a parity block on a separate disk for corresponding blocks from  $N$  other disks

- When writing data block, corresponding block of parity bits must also be computed and written to parity disk
- To find value of a damaged block, compute XOR of bits from corresponding blocks (including parity block) from other disks.

SWAYAM: NPTEL-NOC's Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018

Database System Concepts - 8<sup>th</sup> Edition 24.30 ©Silberschatz, Korth and Sudarshan

At the same time improve the reliability now how do you improve the reliability. So, for improving the reliability you use the basic you know error correcting coding concept you just use a bit level that again two options one is you can do a bit inter lift parity which means a single parity bit is used which is good for error correction. Usually, we know that if we use a single parity bit we can know a single error we can detect a single error,

but here with a single bit you can correct the single error also because you know in case of a failure you know which particular disk has failed. So, then you can exalt with a data from the other disk and reconstruct the error bit.

So, the other is naturally block inter leaving of the parity which uses block level striping and keeps a parity block on a separate disk for corresponding blocks from n other disks and you can reconstruct in a very similar manner. So, by using block interleaved parity with block striping you can really have a higher throughput with a better reliability.

(Refer Slide Time: 25:44)

**Choice of RAID Level**

- Factors in choosing RAID level
  - Monetary cost
  - Performance: Number of I/O operations per second, and bandwidth during normal operation
  - Performance during failure
  - Performance during rebuild of failed disk
    - ↳ Including time taken to rebuild failed disk
- RAID 0 is used only when data safety is not important
  - E.g. data can be recovered quickly from other sources
- Level 2 and 4 never used since they are subsumed by 3 and 5
- Level 3 is not used anymore since bit-striping forces single block reads to access all disks, wasting disk arm movement, which block striping (level 5) avoids
- Level 6 is rarely used since levels 1 and 5 offer adequate safety for most applications

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

24.31

©Silberschatz, Korth and Sudarshan

So, it is a win-win situation now naturally when you go for RAID you will find that there are different levels of read that are available today goes up to level 6 right. Now, and the factors that certainly has to be considered is I mean different RAID at different kind of cost the what is the performance what is the performance during failure; that is performance during failure is typically the MTTF and performance during rebuilt is a mean time to data loss so, including the rebuilding and so, on. So, based on these factors different rate levels can be looked at RAID 0 is used only when data safety is not important. So, that is not very common the RAID level 2 and 4 are never used.

Because, they are they got subsumed in level 3 and 5. So, you can ignore that level three is also not used anymore, because it used bits striping and naturally we talked of that that is not that is worse compare to the block striping which level 5 uses and level 6 is also really used. Since level 1 and 5 adequately support all applications.

(Refer Slide Time: 27:08)



## Choice of RAID Level (Cont.)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur, Jan-Apr, 2018

- Level 1 provides much better write performance than level 5
  - Level 5 requires at least 2 block reads and 2 block writes to write a single block, whereas Level 1 only requires 2 block writes
  - Level 1 preferred for high update environments such as log disks
- Level 1 had higher storage cost than level 5
  - disk drive capacities increasing rapidly (50%/year) whereas disk access times have decreased much less ( $\times 3$  in 10 years)
  - I/O requirements have increased greatly, e.g. for Web servers
  - When enough disks have been bought to satisfy required rate of I/O, they often have spare storage capacity
    - so there is often no extra monetary cost for Level 1!
- Level 5 is preferred for applications with low update rate, and large amounts of data
- Level 1 is preferred for all other applications



Database System Concepts - 8<sup>th</sup> Edition 24.32 ©Silberschatz, Korth and Sudarshan

So, the conclusions simply, is that you either use RAID level 1 or you use RAID level 5. So, RAID level 1 gives a better right performance, than RAID 5 and it is certainly level. So, therefore, level 1 is preferred for high update environments such as log disks and so, on whereas, level one has higher storage cost than 5 also and level 5 is preferred for applications that has low update rate and large volume of data. So, if you have very high update you go for level one rate. So, which will give you which will cost you more, but if you have a level 5, then you will be able to get a low update, but have large amount of data stored reliably with less amount of money invested into that.

(Refer Slide Time: 28:15)



## Optical Disks

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur, Jan-Apr, 2018

- Compact disk-read only memory (CD-ROM)
  - Removable disks, 640 MB per disk
  - Seek time about 100 msec (optical read head is heavier and slower)
  - Higher latency (3000 RPM) and lower data-transfer rates (3-6 MB/s) compared to magnetic disks
- Digital Video Disk (DVD)
  - DVD-5 holds 4.7 GB, and DVD-9 holds 8.5 GB
  - DVD-10 and DVD-18 are double sided formats with capacities of 9.4 GB and 17 GB
  - Blu-ray DVD: 27 GB (54 GB for double sided disk)
  - Slow seek time, for same reasons as CD-ROM
- Record once versions (CD-R and DVD-R) are popular
  - data can only be written once, and cannot be erased.
  - high capacity and long lifetime; used for archival storage
  - Multi-write versions (CD-RW, DVD-RW, DVD+RW and DVD-RAM) also available



Database System Concepts - 8<sup>th</sup> Edition 24.34 ©Silberschatz, Korth and Sudarshan

And so, these are the RAID levels then of course, finally there are different tertiary storages compact disk CD-ROM we all are familiar that DVD the record once versions where you just record and use that particularly for different kind of distribution these.

(Refer Slide Time: 28:34)

**Magnetic Tapes**

- Hold large volumes of data and provide high transfer rates
  - Few GB for DAT (Digital Audio Tape) format, 10-40 GB with DLT (Digital Linear Tape) format, 100 GB+ with Ultrium format, and 330 GB with Ampex helical scan format
  - Transfer rates from few to 10s of MB/s
- Tapes are cheap, but cost of drives is very high
- Very slow access time in comparison to magnetic and optical disks
  - limited to sequential access.
  - Some formats (Accelis) provide faster seek (10s of seconds) at cost of lower capacity
- Used mainly for backup, for storage of infrequently used information, and as an off-line medium for transferring information from one system to another.
- Tape jukeboxes used for very large capacity storage
  - Multiple petabytes ( $10^{15}$  bytes)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

24.35

©Silberschatz, Korth and Sudarshan

storages are used there are magnetic tapes which are very large volume and provide a high transfer rate and they are go currently they go into different couple of orders of terabytes even petabytes in size, but the tapes are not really very expensive. So, we can use them to hold really really large databases they are good for Backups and so, on, but the tape drives are quite expensive. So, that will have to keep in mind.

(Refer Slide Time: 29:08)

The slide is titled "Module Summary" in red at the top right. On the left, there is a small sailboat icon. The main content area contains a bulleted list of learning objectives:

- Looked at the options of Physical Storage Media for high volume, fast, reliable and inexpensive options for data storage for databases
- Understood the structure and basic functionality of Magnetic Disks
- Understood RAID – array of redundant disks in parallel to enhance speed and reliability
- Understood the options of Tertiary Storage for high volume, inexpensive backup options

At the bottom left, it says "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P.P. Desai, IIT Kanpur Date: Jan-Apr., 2018". At the bottom center, it says "Database System Concepts - 8<sup>th</sup> Edition" and "24.36". At the bottom right, it says "©Silberschatz, Korth and Sudarshan". There is also a small video player window showing a man speaking.

So, to summarize we have looked at different physical storage media. So, this was I mean besides all the details in the discussion the key take way point for us here is to understand that primarily.

We have a memory which is expensive and which is small in size very high speed. So, all operations that we need that need to be done we will finally, have to happen when the once the data is in memory and on the other side we have all the persistent data in a in some kind of a magnetic disk in certain structure and that is that can support large storage it is persistent it is reliable, but it is relatively slow to access and so, it needs to be used in a intelligent manner and one point that you have specifically noted that there is some unit for every disk system.

There is some unit called a block or disk block, which is a basic unit of data that will be transferred every time you access the disk. So, you are if your design is aligned with a size of the disk block which is couple of kilobytes then it will be easier to be able to design more optimal physical storage for your files and you can speed up the whole process of search and update that you want to do in the database.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 25**  
**Storage and File Structure : File Structure**

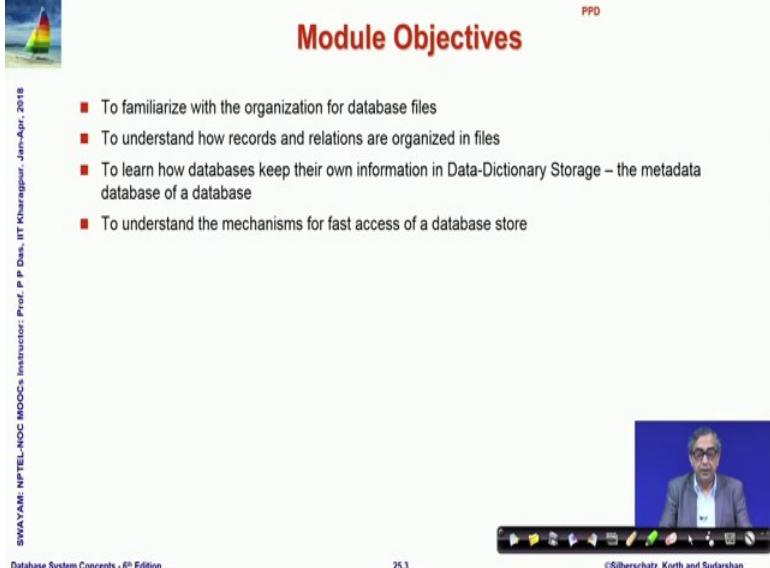
Welcome to module 25 of Database Management Systems. We have been discussing about storage and file structure.

(Refer Slide Time: 00:22)

The slide is titled "Module Recap" in red at the top right. It features a small sailboat icon in the top left corner. A vertical column of text on the left side reads: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018". At the bottom left is the text "Database System Concepts - 8<sup>th</sup> Edition". The main content area contains a bulleted list of storage topics: "■ Overview of Physical Storage Media", "■ Magnetic Disks", "■ RAID", and "■ Tertiary Storage". On the right side, there is a video frame showing a man with glasses and a blue background, likely the professor. Below the video frame is a toolbar with various icons. The slide is labeled "PPD" in the top right corner.

In the last module we have talked about different storage options.

(Refer Slide Time: 00:27)



The slide is titled "Module Objectives" in red. It features a small sailboat icon in the top left corner and a video player window in the bottom right showing a man speaking. The text on the slide lists four objectives:

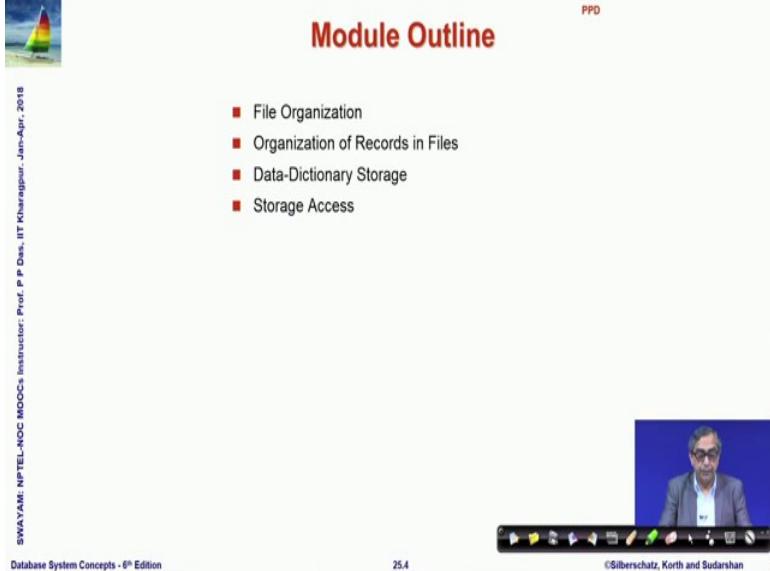
- To familiarize with the organization for database files
- To understand how records and relations are organized in files
- To learn how databases keep their own information in Data-Dictionary Storage – the metadata database of a database
- To understand the mechanisms for fast access of a database store

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018  
PPD

Database System Concepts - 8<sup>th</sup> Edition 25.3 ©Silberschatz, Korth and Sudarshan

And in this one we will talk about the; organization of database files, what should be the typical structure to store the records in the files. And how the overall database which manage itself we will talk about those issues.

(Refer Slide Time: 00:41)



The slide is titled "Module Outline" in red. It features a small sailboat icon in the top left corner and a video player window in the bottom right showing a man speaking. The text on the slide lists five topics:

- File Organization
- Organization of Records in Files
- Data-Dictionary Storage
- Storage Access

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018  
PPD

Database System Concepts - 8<sup>th</sup> Edition 25.4 ©Silberschatz, Korth and Sudarshan

So, the file organization; so if you look at a database; what is the database? It is a collection of relations.

(Refer Slide Time: 00:43)

The slide has a title 'File Organization' at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list under two main points:

- A database is
  - A collection of *files*. A file is
    - A sequence of *records*. A record is
      - A sequence of *fields*
- One approach:
  - assume record size is fixed
  - each file has records of one particular type only
  - different files are used for different relations

This case is easiest to implement; will consider variable length records later

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

25.6

©Silberschatz, Korth and Sudarshan

A video player interface is visible on the right side of the slide, showing a video of a man speaking. Below the video player are several small icons.

So, it is a collection of files every relation is a file. Now, what is a file? A file is a sequence of records, and what is a record? It is a sequence of fields. So, this is the hierarchy that exists and this will have to be kept in mind, when we design the organization of how we keep this data.

Now, one starting approach could be we can assume that all records are of fixed size which makes a life easier and each file has records of only one type again a simplifying assumption and different files are used for different relations. So, this is a easiest case to implement.

(Refer Slide Time: 01:30)

The slide has a header 'Fixed-Length Records' with a sailboat icon. It includes a sidebar with course information: SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018.

**Simple approach:**

- Store record  $i$  starting from byte  $n * (i - 1)$ , where  $n$  is the size of each record.
- Record access is simple but records may cross blocks
  - Modification: do not allow records to cross block boundaries

**Deletion of record  $i$ :**

alternatives:

- move records  $i + 1, \dots, n$  to  $i, \dots, n - 1$
- move record  $n$  to  $i$
- do not move records, but link all free records on a free list

record #	name	subject	score	
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Navigation icons: back, forward, search, etc.

Page footer: Database System Concepts - 8<sup>th</sup> Edition, 25.7, ©Silberschatz, Korth and Sudarshan

So, we will start with that; so this is what we have we store these are fixed size records. So, we store them one after the other and based on the fixed size, we can easily know what is the starting address of any record and we can access it accordingly. Now, if a record is deleted, then there are several things that I can do see that this is a different alternatives, that is if I record delete record I then. So, if we delete any record then we can actually move the records. So, that we consume that space or we can take the last record and move it there or we can simply do not do any move, but use an some additional pointers to denote that these records have become free rather give it to a free list.

(Refer Slide Time: 02:22)

The slide title is "Deleting record 3 and compacting". It features a small sailboat icon in the top left corner and a copyright notice for Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018 on the right. The main content is a table of student records:

	ID	Name	Major	GPA
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Below the table is a navigation bar with icons for back, forward, search, etc. At the bottom, it says "Database System Concepts - 8<sup>th</sup> Edition", "25.8", and "©Silberschatz, Korth and Sudarshan".

So, these are the three main strategies. So, here we showing the first one the record three has been removed. So, all records have moved up in this it is we have move the last record 11 in the place of record 3. So, record 3 is gone, but still the whole thing remains compact only the point that must be noted that in the earlier one, where well we moved everything then the ordering that existed here of this key of this key field is maintained, but if we move the last record the naturally that ordering has got destroyed. So, it will have implications in terms of indexed organization that will cover in the next modules.

(Refer Slide Time: 03:08)

The slide title is "Free Lists". It features a small sailboat icon in the top left corner and a copyright notice for Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018 on the right. The main content is a table of student records with arrows pointing from specific cells to a free list header:

	ID	Name	Major	GPA
header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	15151	Mozart	Music	40000
record 2	22222	Einstein	Physics	95000
record 3	33456	Gold	Physics	87000
record 4	58583	Califieri	History	62000
record 5	76543	Singh	Finance	80000
record 6	76766	Crick	Biology	72000
record 7	83821	Brandt	Comp. Sci.	92000
record 8	98345	Kim	Elec. Eng.	80000
record 11				

Arrows point from the value "22222" in record 2, the value "33456" in record 3, and the value "76766" in record 6 to a "header" row above the table. Below the table is a video frame of a professor speaking, a navigation bar with icons, and the footer information "Database System Concepts - 8<sup>th</sup> Edition", "25.10", and "©Silberschatz, Korth and Sudarshan".

The third option could be use a free list, which is a nice one because you would you do not neither here neither you destroy the order that existed and no one have to really move records which is expensive, but you just start with a pointer and keep on pointing to the empty records.

And once you delete it you use that space itself to point to the next deleted record. So, whenever you have to you know delete a record all that you need to do is adjust this pointer. So, which is pretty fast and quite efficient way of getting this linked together in terms of; so there is as such no space over it and it is a fastest possible that you can do now in contest to fix length record.

(Refer Slide Time: 03:54)

**Variable-Length Records**

- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file
  - Record types that allow variable lengths for one or more fields such as strings (**varchar**)
  - Record types that allow repeating fields (used in some older data models)
- Attributes are stored in order
- Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes
- Null values represented by null-value bitmap

Null bitmap (stored in 1 byte)  
0000

21, 5	26, 10	36, 10	65000	10101	Srinivasan	Comp. Sci.		
Bytes 0	4	8	12	20	21	26	36	45

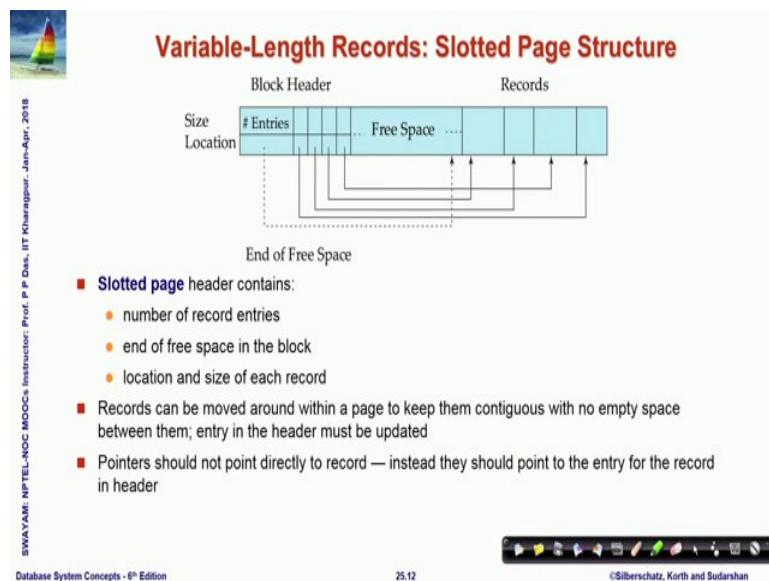
SWAYAM-NPTEL-MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
25.11  
©Silberschatz, Korth and Sudarshan

If the record becomes variable length, then certainly every record may of very different size and it is very common for example, we have types like varchar a lot of strings are varchar. So, we just we do not know how much it will take. So, the typical way you represent that is a you represent as to; what is a starting pointer of a particular of the actual value and the size of the value the number of bytes it will take. So, when we say **21,5** which we mean that this field will actually start from location 21 and we will have 5 locations 5 bytes, then the next one is **26, 10**.

So, this will start to 26 and go for 10 such; so what happens is; if you look into this part of the data, then that part is actually for all practical purposes the fix length 1, because here you are just keeping double x for the variable length data or you have some field

which is a fixed length data anyway or you have a null which is stored in one byte, and then you have all the variable stuff at one end. So, you can actually make part of this fixed length by using this kind of encoding. So, this is what is explained here.

(Refer Slide Time: 05:22)



So, for variable length records a one main issue is if you if you keep it like this, then since you are using actually you are using pointers here we saying that this data actually is on 21. So, what will happen is if you change the position of the record if you relocate the record, then all these references will have to be updated. So, that becomes a slotted thing. So, what the slotted page structure does is it does a [li/little] little bit of adjustment it ports a puts a records here at the at the end.

And it has a header it has a. So, it has a block header as in here and the block header has actually pointers to the records and then you have a an entry which points to the end of the free space where more records can still be stored. So, when you refer to a particular record you do not actually refer here. So, you do not refer here, but you refer here. So, what you maintain is the header is actually not changed, but if there are relocations required adjustments required, then that will be done with respect to this. And so, this value will change, but any references made to this location will remain invariant. So, that is the basic idea of the slotted page structure, which can allow you to have the variable length record with easy re locatability in the design.

Now, let us see the given this what is the organization of the records in the file.

(Refer Slide Time: 07:05)

The slide has a title 'Organization of Records in Files' at the top right. On the left is a small sailboat icon. The main content is a bulleted list of four organization types: Heap, Sequential, Hashing, and Multitable clustering file organization. A small video thumbnail of a professor is on the left, and a navigation bar is at the bottom.

**Organization of Records in Files**

- **Heap** – a record can be placed anywhere in the file where there is space
- **Sequential** – store records in sequential order, based on the value of the search key of each record
- **Hashing** – a hash function computed on some attribute of each record; the result specifies in which block of the file the record should be placed
- Records of each relation may be stored in a separate file. In a **multitable clustering file organization** records of several different relations can be stored in the same file
  - Motivation: store related records on the same block to minimize I/O

SWAYAM-NPTEL-MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

25.14

©Silberschatz, Korth and Sudarshan

So, there are different organizations that have been tried out the simplest is the heap is a record can be placed anywhere in the file where there is space. And you can link to that that is that is one way certainly there is nothing very smart in terms of doing that, but you can you would possibly like to do better than that. So, one is you can store the; records in a sequential manner let us store records in a sequential order in terms of certain search key.

So, based on the value of the search key you put them in the sequential orders. So, what it will mean that it will become easier to search the records in that way, but it has consequences or you can hash you can use a hash function on a some of the attributes of the record and the results specified on which block which disk block the record will be placed. So, these are the different option and a records of which relation may be stored in a separate file that is a basic convention, but in some cases there could be multi table clustering as well.

So, let us quickly take a look at these options.

(Refer Slide Time: 08:16)

The slide features a title 'Sequential File Organization' at the top right. On the left, there is a small logo of a sailboat and some text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018'. The main content is a table of student records with sequential links. The table has columns for ID, Name, Subject, and Marks. Arrows show the sequential links between records. The table is as follows:

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

So, these sequential file organizations. So, these things are kept sequentially here as you can see there all consequentially here and this is the link key of those.

(Refer Slide Time: 08:34)

The slide features a title 'Sequential File Organization (Cont.)' at the top right. On the left, there is a small logo of a sailboat and some text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018'. The main content is a continuation of the previous slide's table, showing the same records with sequential links. The table is as follows:

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

So, this is the issues of deletion and if you delete you use pointer chains. As you have we have discussed earlier, and if you have to insert then you look for a free space, if you find a free space you can put it there you insert it there if there is no free space then you have to use a overflow block, where you can go and place that separately as the dilemma

shows here; in a multi table clustering what you would do is more than one relation could be kept in the same file.

(Refer Slide Time: 09:00)

**Multitable Clustering File Organization**

Store several relations in one file using a **multitable clustering** file organization

<i>department</i>	<i>dept_name</i>	<i>building</i>	<i>budget</i>
	Comp. Sci.	Taylor	100000
	Physics	Watson	70000

<i>instructor</i>	<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
	10101	Srinivasan	Comp. Sci.	65000
	33456	Gold	Physics	87000
	45565	Katz	Comp. Sci.	75000
	83821	Brandt	Comp. Sci.	92000

<i>multitable clustering of department and instructor</i>	<i>Comp. Sci.</i>	<i>Taylor</i>	<i>100000</i>
	45564	Katz	75000
	10101	Srinivasan	65000
	83821	Brandt	92000
	Physics	Watson	70000
	33456	Gold	87000

For example, here we are showing two relations department name building and budget these attributes doing department and instructor, id name, department name, and salary is other instructor in a way keeping them together here naturally, where we keep them together.

For example here in we have one which is here in we have one, which is and entry of record from the department relation. Similarly here is another which is from the department relation whereas, these are entries from the instructor relation; please note that since we are doing it multi table with a department we do not need to keep these information in as a part of the record, but what you mean is if there is a computer science entry here. Then all those records which follow this computer science entry are actually instructors in the computer science till I actually come across another departments entry where which will be followed by instructors for that department. So, that is a basic multi table convention that is to be followed here.

(Refer Slide Time: 10:26)

The slide title is "Multitable Clustering File Organization (cont.)". It features a small sailboat icon in the top left and a portrait of a man in the bottom left. On the right, there is a table with data and a pointer chain diagram.

**Multitable Clustering File Organization (cont.)**

SWAYAM-NPTEL-MOOCs Instructor: Prof. P.P. Desai, IIT Kharagpur - Jan-Apr. 2018

■ good for queries involving *department*  $\bowtie$  *instructor*, and for queries involving one single department and its instructors  
■ bad for queries involving only *department*  
■ results in variable size records  
■ Can add pointer chains to link records of a particular relation

	Comp. Sci.	Taylor	100000	
45564	Katz	75000		
10101	Srinivasan	65000		
83821	Brandt	92000		
Physics	Watson	70000		
33456	Gold	87000		

A pointer chain diagram shows arrows connecting the last record of one page to the first record of the next page, illustrating how records of a particular relation are linked.

Database System Concepts - 8<sup>th</sup> Edition      25.18      ©Silberschatz, Korth and Sudarshan

Now, it is actually good for queries that involved joining department with instructor, because based on the value of the department you have the instructors club together and they could be very easily quickly taken together and it is also good for single queries with departments and it is instructors, because as you can see you can if you want to know for example, who are the faculty for at computer science department; then it be very easy to answer that, because you need to search for computer science and then you know all the list of the faculty will be in consecutive block.

So, you can easily lift that, but certainly this is not true, if you want to involve queries which have department only; because that department information are all now sparsely distributed. So, if your query has the department based information to be to be accumulated, and then this may not be a good option. So, that will result in then you can have supporting pointer chains to actually link the department information. So, this is a one kind of a design that you have ok.

Now think about; so the whole so, we have; so, far talked about the relations and relations going to either single files or multi table relations; multi table file where multiple relations are on the same file. Now, if you look at the database as a whole. So, what is a data base?

(Refer Slide Time: 12:08)

The slide has a header 'Data Dictionary Storage' with a sailboat icon. The main content discusses what the Data dictionary stores, listing various types of metadata. A footer includes course details and copyright information.

**Data Dictionary Storage**

The **Data dictionary** (also called **system catalog**) stores **metadata**; that is, data about data, such as:

- Information about relations
  - names of relations
  - names, types and lengths of attributes of each relation
  - names and definitions of views
  - integrity constraints
- User and accounting information, including passwords
- Statistical and descriptive data
  - number of tuples in each relation
- Physical file organization information
  - How relation is stored (sequential/hash/...)
  - Physical location of relation
- Information about indices

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kanpur Date: Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition      25.20      ©Silberschatz, Korth and Sudarshan

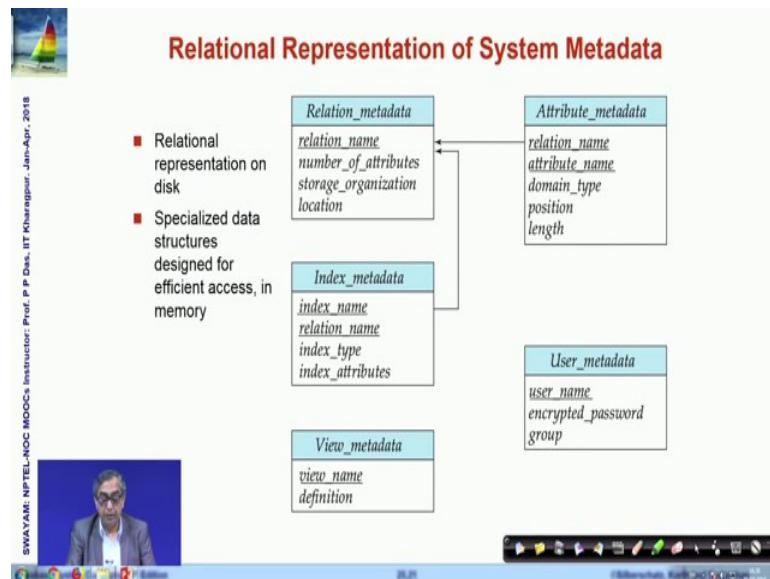
The database as a whole has a whole lot of tables; and so far we have just been focus on the fact that tables we know the tables we know their attributes and the data resides in inside, but if you think in terms of the database, then somebody; somewhere we will need to remember that what are my tables? What are my relations? For a given relation I need to know what are the attributes that the relation has, what is the; you know length type of this attributes I did remember what are the views that I have created on the database the constraints that exist.

So, all of these information which you can say is databases own metadata information needs to be also maintained; and what is done is that; also is maintained as a database within the database system. And such a metadata system is usually known as the name of data dictionary or system catalogues.

So, it has informations like this. So, you put them again, you create the schema design based on the all this metadata information that you need, also you can have you will need to maintain information for users, accounts, passwords and so, on. Then you may have statistical information, where you would like to; we will see the use of statistical information when we talk about indexing in the following week you will see that you need to know, what is the you know how frequently the different queries are fired, what are the number of peoples in each relation and so, on; you may also need to have

information in terms of what has been your choices in terms of the physical locations of file the storage options and so on the index files.

(Refer Slide Time: 14:05)



So, here is a sample one. So, what if you look into; so you can again see a number of schema. So, this is saying that the; this is the relational metadata schema which is talking about, what is the different relations? So, every record here is not keeping the data of your application, but its keeping the information that here you have these different relations. For example, couple of modules back we are talking about the library information system.

So, in the library information system we had different we designed different relations the book issue, the book catalogue, the book copies and so, on. So, those relation names and the how many attributes they have? How will you organize the storage, where is that storage will go to this particular table? Then depending on the kind of index that you are creating we have still not discussed about index we will do subsequently; but those index information can be preserved the view information we can have attribute metadata.

So, it is for the relation name what is attribute name and what is the type of that attributes. So, if the relation name is say book catalogue, then the attribute name is title, then what is the domain type. So, we will say this is a varchar; then the position of that attribute, the length if it is given the user metadata all of this are typical things that will go into this system catalogue or the data dictionary that we will require.

(Refer Slide Time: 15:47)

The slide has a header 'Storage Access' with a sailboat icon. The main content is a bulleted list:

- A database file is partitioned into fixed-length storage units called **blocks**
  - Blocks are units of both storage allocation and data transfer
- Database system seeks to minimize the number of block transfers between the disk and memory
  - We can reduce the number of disk accesses by keeping as many blocks as possible in main memory
- **Buffer** – portion of main memory available to store copies of disk blocks
- **Buffer manager** – subsystem responsible for allocating buffer space in main memory

On the left, vertical text reads: SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018. At the bottom left is a video thumbnail of the professor, and at the bottom right are navigation icons.

So, finally, the access to the storage the database file as I have been repeatedly saying is partition into fix length units called blocks, because blocks are defined; so that they can be easily allocated and transfer and they are the fastest unit of data that can be transferred between the disk and the memory.

So, unlike many of our typical algorithmic considerations; so when we talk about different algorithms, what we try to minimize? We try to minimize certain expensive operations in the CPU; say the comparison operation or the assignment or may be the memory read operation. But in terms of database systems block is a basic unit of data transfer and the data transfer to and from the disk is the most time taking factor much takes much larger time compare to any in memory operation that we do. So, this kind of becomes the primary unit of cost that we want to minimize.

So, normally we will see that as we talk about index saying and other different kinds of mechanisms, our primary target is to minimize the number of block transfers. So, certainly we can do that by; can reduce the number of disk access by keeping as many blocks as possible in the main memory. So, we can how can you minimize that transfer, if we can keep more of the blocks in our main memory and naturally of course, there is a limitation because main memory is much smaller. So, often we make use of different buffers.

So, a portion of the main memory will be kept to store copies of the disk block. So, every time you need a block you may not want to need to bring it from the; disk storage. So, you keep it in the buffer in main memory, and then you have a management strategy to manage this buffer. So, whenever you want to actually access a record which should be in a particular block; you check whether that block is already available in the buffer; if it is available in the buffer it use that if it is not available in the buffer, then you take it from the disk you will need quite a bit of cycles for that and as you get that from the disk then you keep a copy in the buffer so that it can be used in future.

(Refer Slide Time: 18:33)

The slide has a decorative header with a sailboat icon and the title 'Buffer Manager' in red. The content is organized into two main sections: a bulleted list and a numbered list. On the left, there is a vertical watermark-like text: 'SWAYAM-NPTEL-NOOC MOOCs Instructor: Prof. P. Doss, IIT Kharagpur - Jan-Apr. 2018'. At the bottom, there is a video player showing a person speaking, with a progress bar indicating 25.28 seconds.

- Programs call on the buffer manager when they need a block from disk.
  1. If the block is already in the buffer, buffer manager returns the address of the block in main memory
  2. If the block is not in the buffer, the buffer manager
    1. Allocates space in the buffer for the block
      - 1. Replacing (throwing out) some other block, if required, to make space for the new block
      - 2. Replaced block written back to disk only if it was modified since the most recent time that it was written to/fetched from the disk
    2. Reads the block from the disk to the buffer, and returns the address of the block in main memory to requester

Now as you keep on doing that, naturally soon you will run out of the buffer memory. So, you will come to a situation, where we need to bring a block from the disk to the memory, but the buffer does not have enough space. So, then we will have to create replace some of the blocks and create space for that. So, here is a basic buffer management strategy.

So, as I said if we if we start if the block is already there in the buffer, then that is given out if the block is not there in the buffer the buffer manager will need to allocate some space how do you allocate space by throwing out some other block which is not required or replace the; then replace the block return back to disk and if it was modified and make space free and then read from the disk and keep a copy in the buffer. So, that is a simple strategy as you can see.

(Refer Slide Time: 19:36)

The slide has a title 'Buffer-Replacement Policies' at the top right. On the left is a small sailboat icon. The main content is a bulleted list:

- Most operating systems replace the block **least recently used** (LRU strategy)
- Idea behind LRU – use past pattern of block references as a predictor of future references
- Queries have well-defined access patterns (such as sequential scans), and a database system can use the information in a user's query to predict future references
  - LRU can be a bad strategy for certain access patterns involving repeated scans of data
    - For example: when computing the join of 2 relations r and s by a nested loops
      - for each tuple  $tr$  of  $r$  do
      - for each tuple  $ts$  of  $s$  do
      - if the tuples  $tr$  and  $ts$  match ...
  - Mixed strategy with hints on replacement strategy provided by the query optimizer is preferable

At the bottom left, vertical text reads: SWAYAM-NPTEL-NOC-MOOCs Instructor: Prof. P.P. Desai, IIT Kanpur-2018. At the bottom right, there is a toolbar and a status bar showing '29.39'.

Now, certainly when you have to replace the block in the buffer, then the question is which block would you replace. Now if you recall from your from similar situations in the in the operating system in terms of memory management, you have read about different strategies for doing replacement and one of the very common strategy more often used is the LRU strategy of the least recently used strategy. So, the idea of behind LRU is use the past pattern of block references as to predict the future. So, least recently used is if this is not been used in the recent past. So, it has less likely hood of being used in the future.

Now, to queries; now here we are trying to do the similar thing in terms of queries. So, they have a well defined access pattern and database system can make use of that and as it turns out LRU can be a bad strategy for example, often you are doing computations in terms of such what you say such nested loops.

So, you have for each tuple you do this; so you have basically trying to do a join. So, you have two relations and you are trying to do a join. So, when you do this when you are going through the inner loop, there will be lot of transfers that will happen; and the original block where you have been holding this is here and you are not accessing that for quite some time.

So, while you are doing this if you if this block, which was which was holding  $r$  at that time if that turns out to be a LRU; then you will throw it away, but with that when you

complete this loop and come back here, you will again have to read it from the disk. So, this is not LRU for such nested computations may not be a good strategy, so maybe some kind of mixed strategy would work better.

(Refer Slide Time: 21:50)

The slide has a title 'Buffer-Replacement Policies (Cont.)' in red. On the left is a small sailboat icon. The right side contains a bulleted list of buffer replacement strategies:

- **Pinned block** – memory block that is not allowed to be written back to disk
- **Toss-immediate** strategy – frees the space occupied by a block as soon as the final tuple of that block has been processed
- **Most recently used (MRU) strategy** – system must pin the block currently being processed. After the final tuple of that block has been processed, the block is unpinned, and it becomes the most recently used block.
- Buffer manager can use statistical information regarding the probability that a request will reference a particular relation
  - E.g., the data dictionary is frequently accessed. Heuristic: keep data-dictionary blocks in main memory buffer
- Buffer managers also support **forced output** of blocks for the purpose of recovery

At the bottom, there is footer text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr- 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '25.26', and '©Silberschatz, Korth and Sudarshan'.

So, there are several that are used in terms of buffer replacement one is called pin block, where you mark a certain memory block which is not allow to be return back to the disk it has to stay in the buffer or a toss immediate strategy is quite often used. So, it frees the space occupied by a block; as soon as the final tuple has been removed.

So, it is a toss immediate. So, as soon as you are done you just throw it out you are you are done with it so you write it back. Another which is commonly uses most recently used the; if whenever the block is currently being processed, then the system will kind of keep a marker a pin. So, that it is not removed, but after the final tuple has been processed, the block will be unpinned and then it becomes a most recently used block and you can go with defining the most recently used block and having the strategy.

(Refer Slide Time: 23:04).

The slide is titled "Module Summary" in red at the top right. On the left, there is a small sailboat icon. The main content area contains a bulleted list of learning objectives:

- Familiarized with the organization for database files
- Understood how records and relations are organized in files
- Learnt how databases keep their own information in Data-Dictionary Storage – the metadata database of a database
- Understood the mechanisms for fast access of a database store

On the left side of the slide, there is vertical text that reads: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr-2018". Below the main content area, there is a video thumbnail showing a man speaking, with the text "Database System Concepts - 8th Edition" underneath it. At the bottom right, there is a navigation bar with icons for back, forward, search, and other presentation controls, along with the text "©Silberschatz, Korth and Sudarshan".

You can certainly use different kinds of statistical information and in summary; so on this we have talked about the basic organization of database files starting from the fixed record to variable record handling of the different file organization and try to take a look in terms of how records and relations are organized in terms of the in terms of the files and what are the options that we have and we took a look at the data dictionary storage the basic system catalogue, where database keeps its own information.

And then noted that block happens to be the major unit of data transfer between the disk and the main memory and therefore, that is the unit of defining unit of cost that we have to incur, and to minimize that we have a buffering strategy in the main memory, where the disk blocks will be kept a few disk blocks will be kept for quick use whenever required. And there needs to be various different smart replacements strategies for good management of this buffer.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 26**  
**Indexing and Hashing/1: Indexing/1**

Welcome to module 26 of Database Management Systems. In this module and the next 4; that is for the whole of this week we will talk about indexing and hashing.

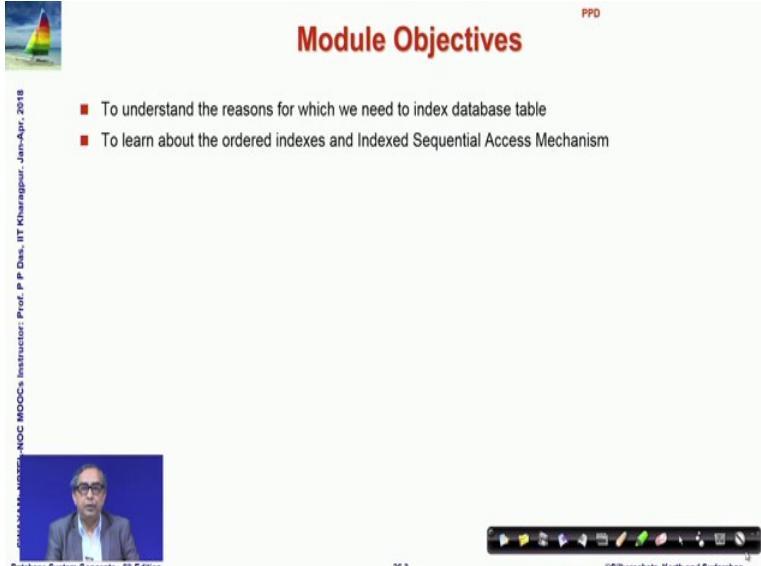
(Refer Slide Time: 00:33)

The slide is titled "Week 05 Recap" in red at the top right. It features a small sailboat icon on the left. The content is organized into four main sections, each with a bullet point and a list of topics. The sections are: "Module 21: Application Design and Development/1", "Module 22: Application Design and Development/2", "Module 23: Application Design and Development/3", and "Module 24: Storage and File Structure/1 (Storage)". A vertical sidebar on the left contains the text "SWAYAM: NPTEL MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018". At the bottom, there is a navigation bar with icons and the text "Database System Concepts - 8<sup>th</sup> Edition", "26.2", and "©Silberschatz, Korth and Sudarshan".

- **Module 21: Application Design and Development/1**
  - Application Programs and User Interfaces
  - Web Fundamentals
  - Servlets and JSP
- **Module 22: Application Design and Development/2**
  - Application Architectures
  - Rapid Application Development
  - Application Performance
  - Application Security
  - Mobile Apps
- **Module 23: Application Design and Development/3**
  - Case Studies of Database Applications
- **Module 24: Storage and File Structure/1 (Storage)**
  - Overview of Physical Storage Media
  - Magnetic Disks
  - RAID
  - Tertiary Storage
- **Module 25: Storage and File Structure/2 (File Structure)**
  - File Organization
  - Organization of Records in Files
  - Data-Dictionary Storage
  - Storage Access

So, this is a quick recap of what we did in the last week primarily talking about application development and then specifically; we focused on storage and file structure that is how a database file can be stored how it can be organized and in a manner so that, we have efficient representation for that now.

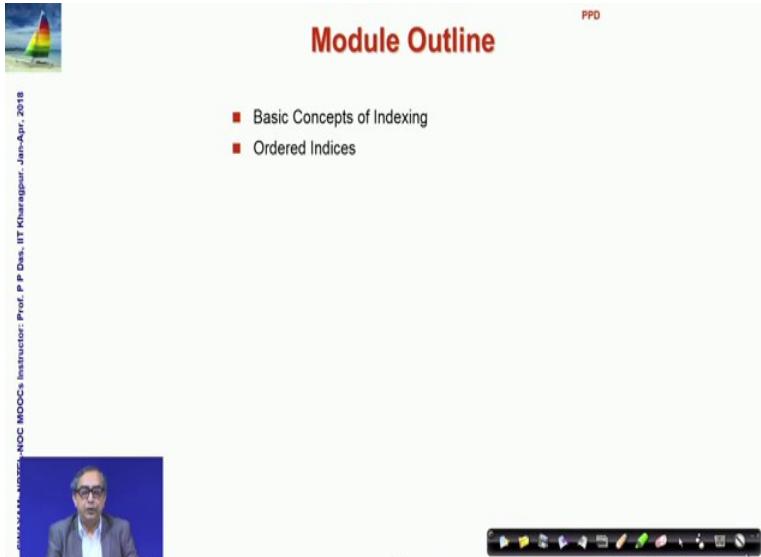
(Refer Slide Time: 01:01)



This slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. On the left edge, there is vertical text: "CHAKRABARTI-NODES-NOC MOOCs", "Instructor: Prof. P. P. Das, IIT Kharagpur", and "Date: Apr. 2018". The main content area contains two bullet points: "To understand the reasons for which we need to index database table" and "To learn about the ordered indexes and Indexed Sequential Access Mechanism". At the bottom left is a video thumbnail showing a man speaking. The bottom right corner includes the text "Database System Concepts - 8<sup>th</sup> Edition", "26.3", and "©Silberschatz, Korth and Sudarshan". A navigation bar with various icons is at the bottom.

We will move on from there to and focus on the basic feature of a database system, which is the ability to find information in a very fast manner the ability to update in a very fast manner. And we will see the reasons for which we do something on the database tables called indexing and we will talk about ordered index in this module and about the index sequential access mechanism.

(Refer Slide Time: 01:31)



This slide is titled "Module Outline" in red at the top right. It features a small sailboat icon in the top left corner. On the left edge, there is vertical text: "CHAKRABARTI-NODES-NOC MOOCs", "Instructor: Prof. P. P. Das, IIT Kharagpur", and "Date: Apr. 2018". The main content area contains two bullet points: "Basic Concepts of Indexing" and "Ordered Indices". At the bottom left is a video thumbnail showing a man speaking. The bottom right corner includes the text "Database System Concepts - 8<sup>th</sup> Edition", "26.4", and "©Silberschatz, Korth and Sudarshan". A navigation bar with various icons is at the bottom.

So, introduction of the basic indexing concepts and the order indices and we start with the basic concepts.

(Refer Slide Time: 01:40)

The slide is titled "Search Records" and features a table illustrating a search process. The table is divided into three main sections: "Index on 'Name'", "Table 'Faculty'", and "Index on 'Phone'".

Index on "Name"		Table "Faculty"			Index on "Phone"	
Name	Pointer	Rec #	Name	Phone	Pointer	Phone
Anupam Basu	2	1	Partha Pratim Das	81998	6	81664
Pabitra Mitra	6	2	Anupam Basu	82404	1	81998
Partha Pratim Das	1	3	Ranjan Sen	84624	2	82404
Prabir Kumar Biswas	7	4	Sudeshna Sarkar	82432	4	82432
Rajib Mall	5	5	Rajib Mall	83668	5	83668
Ranjan Sen	3	6	Pabitra Mitra	81664	3	84624
Sudeshna Sarkar	4	7	Prabir Kumar Biswas	84772	7	84772

Below the table, there are two sections of bullet points:

- Consider a table: Faculty(Name, Phone)
- How to search on Name?
  - Get the phone number for 'Pabitra Mitra'
  - Use "Name" Index – sorted on 'Name', search 'Pabitra Mitra' and navigate on pointer (rec #)
- How to search on Phone?
  - Get the name of the faculty having phone number = 84772
  - Use "Phone" Index – sorted on 'Phone', search '84772' and navigate on pointer (rec #)
- We can keep the records sorted on 'Name' or on 'Phone' (called the primary index), but not on both.

On the left side of the slide, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018". On the right side, there are icons for navigation and "PPD". At the bottom, it says "Database System Concepts - 8<sup>th</sup> Edition", "26.6", and "©Silberschatz, Korth and Sudarshan".

So, consider a very simple example, let us consider an example where there is a relation faculty which has name and phone number and additionally. So, just focus on the middle part the pinkish part of the table which is table faculty besides the two attributes I have put a serial number for the records which kind of can be considered as internal numbers of the database to identify each record.

Now, let us say let us assume that we have to search on names suppose we have a query that get me the phone number of Pabitra Mitra. So, you can see that Pabitra Mitra the record name occurs at position 6. So, if we have to get the phone number, then naturally we have to look at all these entries from one end to the other till we come across Pabitra Mitra match the name Pabitra Mitra and we can access the phone number.

Now, similarly if we have to search for a phone number we will have similar situation. So, if we want to get the phone numbers of the faculty having phone number 84772. Again, we will have to search on the phone number from one end to the other and find the name of the faculty. Now, certainly this will mean that if there are n records in the database then we will need or the order of about  $n/2$  comparisons to be done or accesses record accesses to be done before.

We can find the desired record which is certainly not a very efficient way of doing things and you know from your knowledge of basic algorithms that; if we have a set of data and we want to find a particular one then we can make it efficient by actually keeping the

data in a sorted manner and doing some kind of a binary search that is one one possible mechanism through which you can do that.

So, we can use that same context same concept now and just look at the left side the blue part of the blue part tableware, what I have done have collected all the names of the faculty and I have sorted them in lexicographically increasing order and with that I have kept the record number as a pointer. Now, since this is a sorted array. So, here I can search for Pabitra Mitra for the first query in a binary search. So, at least in the  $\log n$  order of comparisons I should be able to find professor Pabitra Mitra and get the fact; that it is record number is 6 navigate to the record number 6 in the table in the middle and take out the phone number.

Similarly, without disturbing the actual faculty table I can build a similar kind of supportive table on the right the yellow one where I collect all the phone numbers and I have sorted on the phone numbers I can again do binary search on the phone number 84772 and find the phone number find the record number where this phone number occurs and go and find the name of the faculty.

So, you can see that naturally this gives us a alternate way of finding out and certainly you would say that we could have kept the record sorted on name or on phone number, but certainly if we keep it sorted on name the first query we will get benefited the second query will still take a linear time if we get keep it sorted by phone number the first query will take a linear time. So, if we cannot actually keep the data sorted on both and therefore, this is just an alternate mechanism called the indexing mechanism through which even though the original data is not sorted by maintaining some auxiliary data we can actually make our search mechanism more efficient and that is the core idea of indexing.

(Refer Slide Time: 05:39)

The slide has a header 'Basic Concepts' with a sailboat icon. The content includes:

- Indexing mechanisms used to speed up access to desired data.
  - For example:
    - Name in a faculty table
    - author catalog in library- Search Key** - attribute to set of attributes used to look up records in a file
- An **index file** consists of records (called **index entries**) of the form

search-key	pointer
------------	---------
- Index files are typically much smaller than the original file
- Two basic kinds of indices:
  - Ordered indices**: search keys are stored in sorted order
  - Hash indices**: search keys are distributed uniformly across "buckets" using a "hash function"

At the bottom left is a video thumbnail of a professor, at the bottom center is the text 'Database System Concepts - 8<sup>th</sup> Edition', and at the bottom right is the text '©Silberschatz, Korth and Sudarshan'.

So, indexing mechanism is used to speed up accesses to desired data. So, as you have just seen. So, what we search on is called the search key and an index file consists of the index entries as you have just seen it has a search key and the pointer, pointer is the record number or the internal address of the record where it occurs and certainly I have shown an example where there just two attributes in general there will be a large number of attributes. So, the entry of every index would be much smaller than the corresponding record.

So, the overall index file will be a much smaller one than the original and we can index it and there are two basic ways to index and; that is what we will discuss through these modules one is called ordered index where this search keys are stored in sorted order as you have just seen and the other is hash indices where the search keys are distributed randomly across different buckets. Using, concepts of hash function which you must have studied as a part of your data structure course.

(Refer Slide Time: 06:44)

The slide is titled "Index Evaluation Metrics". It features a small sailboat icon in the top left corner. In the bottom left, there is a video frame showing a man with glasses and a grey jacket. The slide is part of a MOOC on Database System Concepts, 8th Edition, taught by Prof. P. P. Deshpande at IIT Kharagpur, dated Jan-Apr. 2018. The main content is a bulleted list of metrics:

- Access types supported efficiently. For example,
  - records with a specified value in the attribute, or
  - records with an attribute value falling in a specified range of values
- Access time
- Insertion time
- Deletion time
- Space overhead

At the bottom right, there is a navigation bar with icons for back, forward, search, and other presentation controls.

Now, if we mean why should we index on the basic question is should we index on all attributes should we index on one attribute should we index on combination of attributes. So, access. So, there are certain measures to decide as to what is a good way to index. So, that will be that will be measured against the basic requirements of the database that is I should be able to index in a way.

So, that the access time the insertion time the deletion time all these should get as minimized as possible and as you have just seen indexing might mean maintaining additional structures. So, that overhead of space should also be minimal. So, with that with indexing we should be able to do at least certain kind of accesses where a particular value matches the attribute of a record or falls within a range of values in a record those kind of searches those kind of queries should become very efficient and these metrics will have to always keep in mind.

(Refer Slide Time: 07:51)

## Ordered Indices

- In an **ordered index**, index entries are stored sorted on the search key value. For example, author catalog in library
- **Primary index:** in a sequentially ordered file, the index whose search key specifies the sequential order of the file
  - Also called **clustering index**
  - The search key of a primary index is usually but not necessarily the primary key
- **Secondary index:** an index whose search key specifies an order different from the sequential order of the file
  - Also called **non-clustering index**
- **Index-sequential file:** ordered sequential file with a primary index

CHAKRABORTY, NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

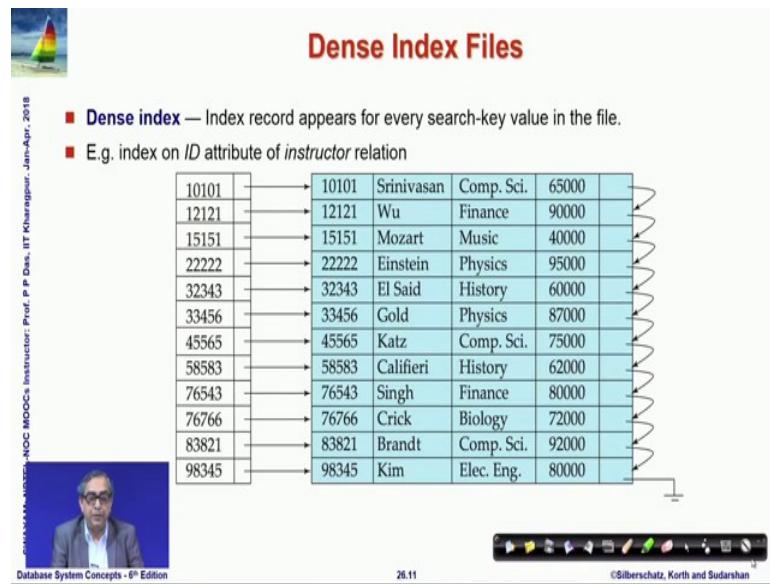
Database System Concepts - 8<sup>th</sup> Edition

26.10

©Silberschatz, Korth and Sudarshan

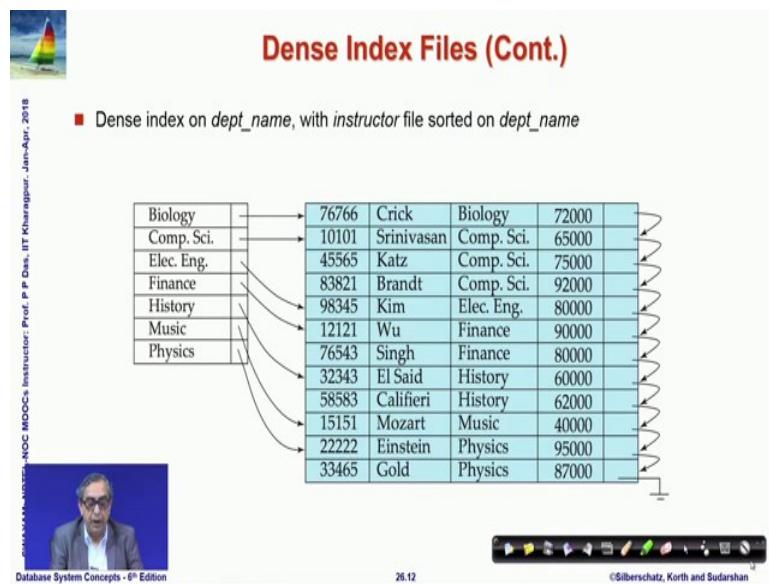
So, let us look at the first concept of ordered index which is pretty much like what we have just sent. So, in a sequentially ordered file the index whose search key specifies the sequential order is known as the primary index. So, the sequential ordering is done based on that primary index it is called also called the clustering index and please keep in mind that the search key of a primary index is usually the primary key, but not necessarily the primary key it could be different from the primary key also and if I create an index who search key is different from the sequential order of the file then we say it is a secondary index and it is also called the non clustering index. So, index sequential file is ordered sequential file which is ordered on the primary index.

(Refer Slide Time: 08:44)



So, here I show an example and this is example of dense index file. So, you can see the blue part is actual table which has different fields the id the name of the faculty, the department, and the salary and this is as you can see that it is completely sorted on the id in the increasing value of id and on the left the white is basically just the ids and with every idea we have a pointer they record the number possibly of the record that it corresponds to here all the indices that feature in the table are also put on the index file and therefore, it is called a dense index you should also note on the right that the records are interlinked through a chain I mean kind of they are being formed in terms of a linked list whose purpose would be seen later on.

(Refer Slide Time: 09:40)

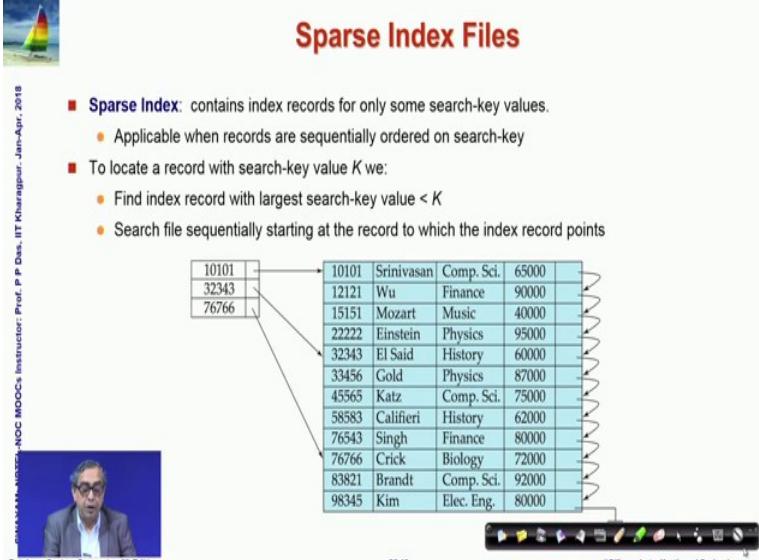


Now, instead of doing id if I want to do a dense index on department name, then naturally the sequential file has to be indexed primarily on the department name and as you can note that unlike the id which is also the primary key and therefore, every id value was unique the department name is not a primary key it is a different attribute which allows for multiple value multiple records having the same attribute value and therefore, when we sort we have one instance of biology, but three of computer science two of finance and so, on.

So, when we make the index we made the index collect all the distinct values of the department names and for every department name we put a pointer we put the index marking the first record in the sequential file with that department name. So, you can see that your computer science points to the record starting with 1 0 1 0 1 and then the; it goes on to the next two records.

So, now you can understand why we need the link list; because if we are looking for the all those teachers all those faculty who are associated with department computer science, then I can find it on the index file with the department name, computer science traveled to the record first record in that which is of Srinivasan and then keep going on this list through the linked list that we have on write and find the record for Katz and record for Brandt and till we moved to the next record for Kim which does not match the department name anymore.

(Refer Slide Time: 11:26)



The slide title is "Sparse Index Files". It features a small sailboat icon in the top left and a video camera icon in the bottom left showing a person speaking. The text discusses sparse index characteristics and search methods. A diagram shows a sparse index structure with three indexed entries (10101, 32343, 76766) pointing to a sequential file of 15 records. The file contains records like Srinivasan (id 10101), Wu (id 12121), Mozart (id 15151), etc.

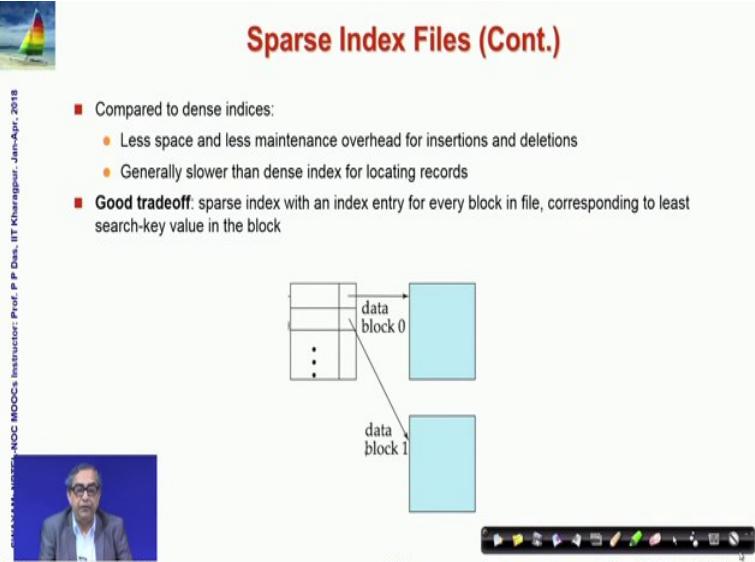
**Sparse Index:** contains index records for only some search-key values.  
Applicable when records are sequentially ordered on search-key

To locate a record with search-key value  $K$  we:  
Find index record with largest search-key value  $< K$   
Search file sequentially starting at the record to which the index record points

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Now instead of dense index we could also do sparse index. With parse index it means that instead of maintaining all the search key values that exist in the file we just take a subset of that and we maintain those in the index file. So, here again we are showing a index sequential structure with id being the primary index and we have chosen three of the ids and kept them in the index file. So, in the sorted order so, this if you want to say look at a record with search K value K we can find the index record with the largest key value largest search key **value id < K** and then search sequentially or onwards because these are all linked together in the sequential manner.

(Refer Slide Time: 12:18)



The slide title is "Sparse Index Files (Cont.)". It features a small sailboat icon in the top left and a video camera icon in the bottom left showing a person speaking. The text compares sparse indices to dense indices and discusses good tradeoffs. A diagram shows a sparse index entry pointing to two data blocks.

Compared to dense indices:  
Less space and less maintenance overhead for insertions and deletions  
Generally slower than dense index for locating records

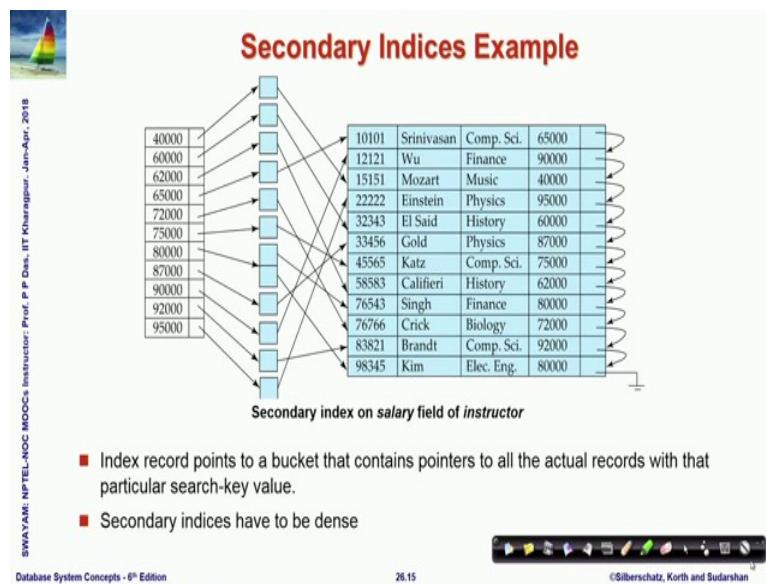
**Good tradeoff:** sparse index with an index entry for every block in file, corresponding to least search-key value in the block

Diagram illustrating a sparse index entry pointing to two data blocks:

```
graph LR; A[ ] --> B["data block 0"]; A --> C["data block 1"]
```

So, naturally compared to the dense index of sparse index takes less space and therefore, less overhead for maintenance when we do insertion deletion you must have already noted that if I were dense index then every time I insert a value it will have to be the dense index file will also have to change changes will be required there for insertion as well as for deletion for sparse index it will not be required, because it will just be required when I am actually changing the record or creating a record which is existing on the sparse index.

(Refer Slide Time: 13:11).



So that way it is better than the dense index, but certainly in the dense index I can go to any record directly from indexing in the sparse index I need to first index and then go sequentially. So, it will be in general slower in terms of performance and will need to look at this tradeoff now it is also. So, these were the ways to do the primary indexing where we decide the order in which we actually keep the record sorted or the fields on which we do that and the index associated with it, but once since the data can be primarily indexed on one or couple of attributes and that gets fixed.

But we may want to search for other values also to make that efficient we create a secondary index. So, here we show an example where the primary index is id. So, the whole recall records are sorted in terms of that and we are creating an index on the salary. So, that we can quickly find the salary of and given the salary of an employee we can quickly find the employee or the employees who get that salary.

So, you can see that for secondary index now it is quite possible that there are multiple entries for the same value of salary for example, if you look at the salary eighty thousand that is received by Professor Singh as well as Professor Kim. So, if you look at the index file of the index sequence of the salary values you will find that against 80,000 we have two entries which mark to two different records corresponding to the faculty who are drawing that salary. So, secondary index naturally has to be dense and is created when we want we feel that there is a need to quickly search on that field or that set of fields and we will discuss more about those issues later on.

(Refer Slide Time: 14:46)

The slide features a sailboat icon in the top left corner. The title 'Primary and Secondary Indices' is centered in red. To the left of the title is a vertical column of text: 'MOOCs-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018'. Below the title is a bulleted list of points:

- Indices offer substantial benefits when searching for records
- BUT: Updating indices imposes overhead on database modification --when a file is modified, every index on the file must be updated
- Sequential scan using primary index is efficient, but a sequential scan using a secondary index is expensive
  - Each record access may fetch a new block from disk
  - Block fetch requires about 5 to 10 milliseconds, versus about 100 nanoseconds for memory access

At the bottom left is a small video thumbnail showing a man speaking. The bottom right contains navigation icons and the text 'Silberschatz, Korth and Sudarshan'.

So, indices offer substantial benefits when searching for records, but updating index impose over it; because if you create an index whenever you insert a record or a delete a record or you change the value of a field which is indexed in a record then certainly all these indices will have to be also updated and therefore, while your access time significantly reduces, because of indexing your actual update insert delete update time will increase and therefore, the indexing will have to be done carefully.

So, sequential scan using primary index is efficient, but sequential scan using secondary index is expensive. So, you will have to bring them in blocks and that will require couple of millisecond versus the amount of time that you need in the memory access. So, these are the factors that we will have to take into consideration and we will talk more about those.

(Refer Slide Time: 15:44)

**Multilevel Index**

- If primary index does not fit in memory, access becomes expensive
- Solution: treat primary index kept on disk as a sequential file and construct a sparse index on it
  - outer index – a sparse index of primary index
  - inner index – the primary index file
- If even outer index is too large to fit in main memory, yet another level of index can be created, and so on
- Indices at all levels must be updated on insertion or deletion from the file

SWAYAM: NPTEL-MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr, 2018  
Database System Concepts - 8<sup>th</sup> Edition  
26.17  
©Silberschatz, Korth and Sudarshan

Now, if I have a primary index then naturally to be able to access the records I will have to first access the primary index and then do a search in that and then traverse the point at to go to the actual record in the file. So, to access the primary record we would prefer that if the primary index can actually fit into the memory. So, that I can do a in memory search like we do the binary search in a in an array.

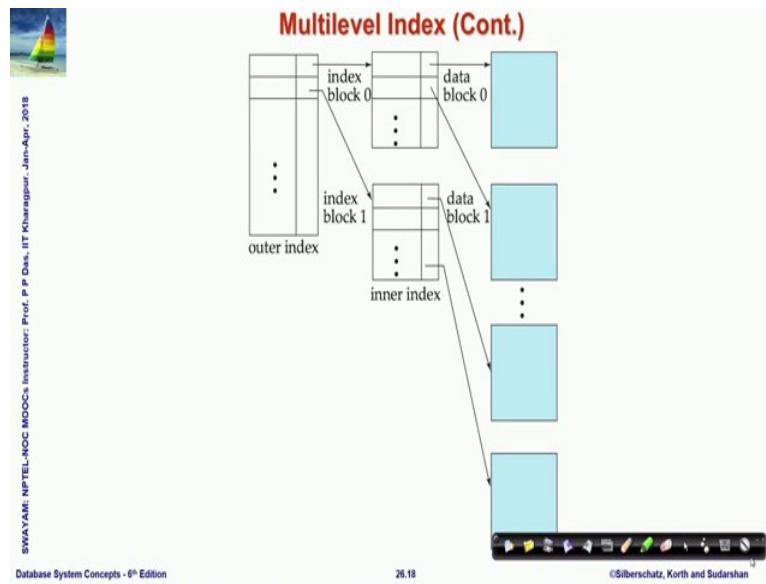
So, because if the primary index is large then that also has to exist in the disk and therefore, bringing that primary index into the memory and then searching will add to additional access cost of the disk and you have already seen in the earlier modules as. So, how these costs are expensive these accesses are expensive. So, primary index if it is not in the memory then we usually have a lot of disadvantage.

So, to take care of that if you have primary index actually is large enough. So, that it does not fit in memory then we simply apply the notion of indexing once again on the primary index file itself. So, we construct a say on the primary index we construct a past sparse index. So, which is called the outer index which is a sparse index of the primary index and in that index is the actual primary index file.

So, if now in turn the outer index the sparse index of the primary index also is too large to fit in memory then you need to do yet another level of indexing on it and. So, on till you come to a index of list which fits into a memory can be directly accessed. So, the cost for that so, this is what is called a multi level indexing.

Because, you are following multiple levels for indexing and the cost naturally of update insert delete increases; because now all of these multiple levels of indices will have to be updated.

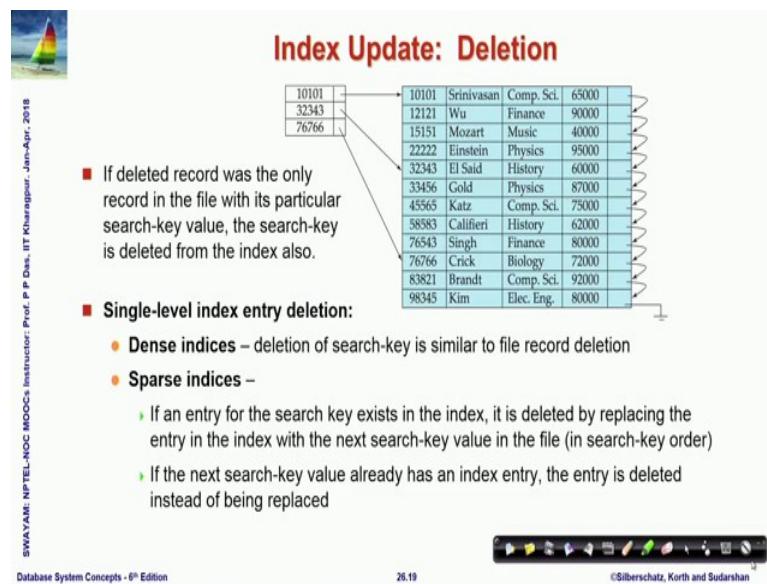
(Refer Slide Time: 17:48)



When you do an update so, this is what is a view of the multi level index you can see the outer index which is sparsely index and index those lead to different blocks of primary index of the of the inner index which is the primary index and then you traverse to the specific block where your record is expected. So, with this you since your outer index are in memory. So, you need to do one disk fetch for finding out the inner index block which should be one disk page or disk block one access and then based on that to find another access to for the block in which the record exists.

So, with this you would be able to manage with to block accesses in this case and that is how the multiple this would not have been possible if in this case you would not have done the sparse outer index, because you would have required the different parts of the inner index of the primary index to be fetched repeatedly from the disk till you act could actually find the final result in that.

(Refer Slide Time: 18:59)



The slide is titled "Index Update: Deletion". It features a small sailboat icon in the top left and a decorative footer bar at the bottom. The main content includes a table of data and a detailed list of deletion rules.

**Table Data:**

Index	Name	Subject	Score
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

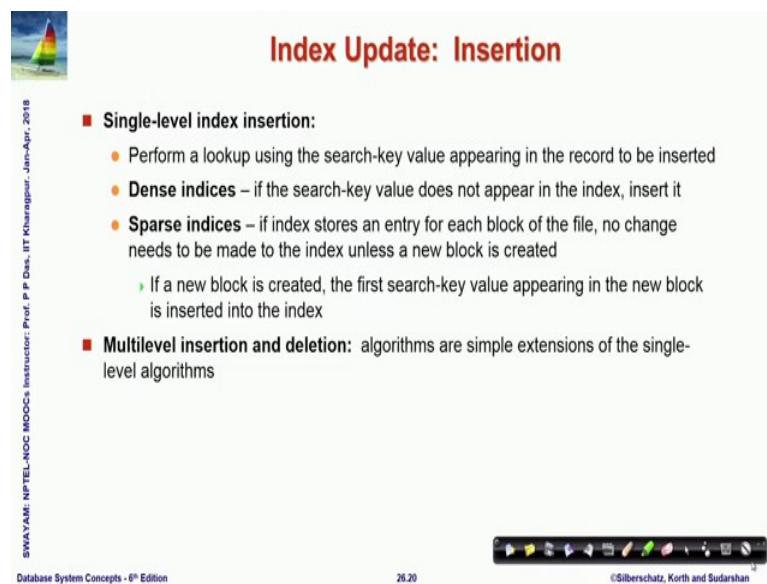
**List of Rules:**

- If deleted record was the only record in the file with its particular search-key value, the search-key is deleted from the index also.
- Single-level index entry deletion:
  - Dense indices – deletion of search-key is similar to file record deletion
  - Sparse indices –
    - › If an entry for the search key exists in the index, it is deleted by replacing the entry in the index with the next search-key value in the file (in search-key order)
    - › If the next search-key value already has an index entry, the entry is deleted instead of being replaced

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr- 2018  
Database System Concepts - 6<sup>th</sup> Edition  
26.19  
©Silberschatz, Korth and Sudarshan

So, updating the index particularly if you do deletion then the if it is a dense index then the deletion of the search key is similar to deletion of the actual record in the file because it is dense if these parts, then naturally you will have to take care of some of the cases, because if it falls within a range then just deleting it would not matter, but if it falls on the boundary where it is actually sparsely indexed then that will have to be appropriately updated.

(Refer Slide Time: 19:32)



The slide is titled "Index Update: Insertion". It features a small sailboat icon in the top left and a decorative footer bar at the bottom. The main content includes a table of data and a detailed list of insertion rules.

**Table Data:**

Index	Name	Subject	Score
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

**List of Rules:**

- Single-level index insertion:
  - Perform a lookup using the search-key value appearing in the record to be inserted
  - Dense indices – if the search-key value does not appear in the index, insert it
  - Sparse indices – if index stores an entry for each block of the file, no change needs to be made to the index unless a new block is created
    - › If a new block is created, the first search-key value appearing in the new block is inserted into the index
- Multilevel insertion and deletion: algorithms are simple extensions of the single-level algorithms

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr- 2018  
Database System Concepts - 6<sup>th</sup> Edition  
26.20  
©Silberschatz, Korth and Sudarshan

Similar thing will have to be taken care of in terms of insertion. So, first you will have to look up to find out where the record needs to be inserted and then if it is a dense index if the search key does not appear in the index then you will have to insert it in case of sparse index you will have to do the additional care that whether it already belongs to the range and or whether if it will have to be a new block has to be created then it has to be also entered in terms of the sparse index and if you have multi level indexing then insertion deletion will be extensions of these basic algorithms.

(Refer Slide Time: 20:14)

The slide has a title 'Secondary Indices' in red at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list:

- Frequently, one wants to find all the records whose values in a certain field (which is not the search-key of the primary index) satisfy some condition
  - Example 1: In the *instructor* relation stored sequentially by ID, we may want to find all instructors in a particular department
  - Example 2: as above, but where we want to find all instructors with a specified salary or with salary in a specified range of values
- We can have a secondary index with an index record for each search-key value

At the bottom left, there is a video player showing a person speaking. The video player interface includes controls for volume, brightness, and other media functions. The bottom right corner shows the text '©Silberschatz, Korth and Sudarshan'.

For secondary indices frequently we want to find all records where certain value in a certain field which is not the search key or the; of the primary index satisfy some condition. And, we can have a secondary index with an index record for each of this search key values depending on what we expect what we would think or likely fields on which more search will be done.

(Refer Slide Time: 20:45)

**Module Summary**

- Appreciated the reasons for indexing database tables
- Understood Indexed Sequential Access Mechanism (ISAM) and associated notions of the ordered indexes

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr- 2018

Database System Concepts - 8<sup>th</sup> Edition

26.22

©Silberschatz, Korth and Sudarshan

Or more range queries will be done. So, to summarize we have taken a brief look at the basic reasons for indexing database tables which is to speed up the process of access insert and delete and we have seen that primarily indexing primarily focuses on speeding up the access process.

So, in that maintenance of indexing whereas, insertion and deletion might have some additional overhead, but since insertion and deletion both inherently needs access to be done because you can insert only when you have tried to access and have not found exactly where the record should occur or for deletion; obviously, you need to first find the record to be able to delete it.

So, even though it is focused indexing is focused on improving the access time it actually improves the time for access insert delete all of that, but we will have to keep in mind that in the process there are certain overheads of index update for insert and delete that will have to be kept as a minimal and the additional storage requirement will also have to be kept as a least overhead. So, with this we will close this module we have taken the basic look at the index sequential access mechanism this is called index sequential access mechanism associated with different database index files.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 28**  
**Indexing and Hashing/2: Indexing/2**

Welcome to module 27 of Database Management Systems. We are discussing indexing and hashing mechanisms in a database and this is the second in that series.

(Refer Slide Time: 00:28)

**Module Recap**

- Basic Concepts of Indexing
- Ordered Indices

SWAYAM: NPTEL-NOC's MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr. 2018

PPD

Database System Concepts - 8<sup>th</sup> Edition      27.2      ©Silberschatz, Korth and Sudarshan

In the last module, we have discussed the basic requirement of indexing and we have learnt about ordered indexes using primary index.

(Refer Slide Time: 00:42)

The slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. On the left edge, there is vertical text: "CHANDRASHEKAR NOC MOOCs", "Instructor: Prof. P. P. Deshpande", and "Date: Jan-Apr. 2018". The main content area contains two bullet points:

- To recap Balanced Binary Search Trees as options for optimal in-memory search data structures and understand the issues relating to external search data structures for persistent data
- To study 2-3-4 Tree as a precursor to B/B+-Tree for an efficient external data structure for database and index tables

At the bottom left is a small video thumbnail showing a man speaking. The bottom right corner includes the text "Database System Concepts - 8th Edition", the page number "27.3", and the copyright notice "©Silberschatz, Korth and Sudarshan". A decorative footer bar with various icons is also present.

Which can be dense or parts and the multi level indexes. Now, in this module we would try to look for the basis of how indexing the index file structure can be very efficiently represent it. So, we will do a quick recap of our notions in algorithms goes.

Where we have talked about earlier balanced binary search trees I mean not in this particular course delivery, but I expect that you have gone through algorithms course where you have learnt about balanced binary search trees as options for optimal in memory search data structure and from that will try to understand the issues relating to external search data structures for persistent data and very specifically we will study 2-3-4 tree as a precursor to B/B+ tree which is an very efficient external data structure for databases and index tables. So, these are the two topics to cover.

(Refer Slide Time: 01:47)

The slide is titled "Search Data Structures". It features a sidebar with a sailboat icon and the text "PPD". On the left, there's a video thumbnail of Prof. P. P. Deshpande. The main content area has a blue header bar with the title. Below it, there's a list of search methods:

- How to search a key in a list of  $n$  data items?
  - Linear Search:  $O(n)$ : Find 28 → 16 comparisons
    - Unordered items in an array – search sequentially
    - Unordered / Ordered items in a list – search sequentially
  - Binary Search:  $O(\log_2 n)$ : Find 28 → 4 comparisons – 25, 36, 30, 28
    - Ordered items in an array – search by divide-and-conquer
    - 01 05 08 10 12 15 20 22 25 28 30 36 38 40 45 48 50 END
    - Binary Search Tree – recursively on left / right

At the bottom right is a binary search tree diagram with root 25, levels 20, 16, 10, 22, 30, 40, and leaf nodes 1, 5, 8, 12, 15, 28, 38, 45, 48, 50.

So, first let me quickly take you around with search data structure now I should warn you that here I am looking at we are looking at a little different kind of a problem here we are looking at search as it is performed in the algorithms course which mean that when you talk about data structures in algorithms course.

You typically talk of data structures which have two basic properties one they are volatile data structures transient that is they are created when the program starts and you operate on the data structure find queries and then as soon as your program ends they disappeared. So, they are not persistent data in contrast when you are dealing with database we are dealing with persistent data which stays even when no queries being performed.

And the second which is the consequence of the first is the data structure that you are study in algorithms work in memory. So, they work in a small limited space and they could be volatile whereas, the database the data structure required for databases has to reside in the disk. So, we have seen the tradeoff between the; on the storage hierarchy between memory and disk and other layers.

So, they will be brought to memory and then certain operations done and then possibly written back and so, on. So, there is similarity in terms of the strategies, but there is a very significant difference in that that because of the persistency the data structures that are used in the database application in the persistent data application has to work with a

very different kind of cost. So, when we talk about cost of an algorithm say a search algorithm we will say that a search algorithm or a sort algorithm has a certain complexity you saw, you know the merge sort has the complexity order  **$n \log n$**  and what it actually means.

Is a number of comparisons you can estimate to do it in sorting  $n$  numbers is approximately  $n \log n$ , but when we talk about external data structure or disk base data structure then your cost may often not be the operations in the CPU like comparison or addition or assignment your cost will shift to actually the number of disk accesses the page access that you have to do, because as you have already noted that the cost of a disk access is much larger couple of orders larger compared to the basic cost of different processor operations.

So, I will start with this in this module I will start talking about data structures which we are found to be efficient in memory and then we will migrate to seeing how they can be used in a with simple extension in the as external data structures as well. So, what we have if you have given a to search a key in a list of  $n$  data items what are the different choices I could make use of a linear search either items could be in an array ordered or unordered in an array or they could be on a list either ordered or unordered I can search that list sequentially and find the data item.

So, I have just shown an example here there is a couple of data items given in that array and trying to find 28, there are 16 comparisons that I need to do and we all know that we can do much better if we keep the this item sorted in terms of in an increasing order or say decreasing order here it is increasing order and then we can do a binary search divide and conquer and I can find the same value 28.

Now, in just four comparisons and from that we have come to the also know that this whole binary search order can be easily represented in terms of a binary tree structure called the binary search tree.

(Refer Slide Time: 05:44)

The slide has a header 'Search Data Structures' with a sailboat icon. On the left, there's vertical text: 'CHAKRABORTY NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018'. The main content includes a table comparing data structures based on worst-case time complexity:

Data Structure	Search	Insert	Delete	Remarks
Unordered Array	O(n)	O(1)	O(1)	The time to Insert / Delete an item is the time after the location of the item has been ascertained by Search.
Ordered Array	O(log n)	O(n)	O(n)	
Unordered List	O(n)	O(1)	O(1)	
Ordered List	O(n)	O(1)	O(1)	
Binary Search Tree	O(h)	O(1)	O(1)	

Below the table are three bullet points:

- Between an array and a list, there is a trade-off between search and insert/delete complexity
- For a BST of  $n$  nodes,  $\log n \leq h \leq n$ , where  $h$  is the height of the tree
- A BST is balanced if  $h \sim O(\log n)$  – this is what we desire

At the bottom left is a video thumbnail of a professor, and at the bottom right are navigation icons.

So, if we compare worst case time here again here please keep in mind that we are talking about in memory data structure. So, here the time is primarily that of comparison. So, if you look at the different data structure like unordered array ordered array unordered list and ordered list and binary search tree and if you check the complexity of the three basic operations which we will need in that in a database application the search insert and delete you can find that the search. Usually, is order  $n$  unless you have an ordered array I mean between array and list the search is order  $n$  unless you have an ordered array and you can do a binary search.

The insert the time that I show for insert or for delete is usually order one, because when I say insert is order one what it means its that after I have been able to search an item which I need to insert after I have been able to find its position, what is the additional time that you need to actually insert that item? So, that insert cost is often for unorder array and any kind of list is order one because you can just manipulate a couple of pointers and insert that, but if you are using an ordered array to make your search efficient then to insert you need a order  $n$  insertion because at the right place you need to move the elements to the right to make the space for the new element, because you need to maintain the ordering that the elements have.

So, kind of we find that between the array and the list there is kind of a trade off in terms of if you want to make search better you have ordered array and that degrades the insert

delete complexity and vice versa. So, to take the benefit of both we the binary search tree is device to a you expect that the search will take the worst case order  $h$  cost which  $h$  is the height of the tree, because that is the maximum number of comparisons that we will need to reach that leaf node and a BST binary search tree if it is balanced then  $h$  would be of the order of  $\log n$  and this is what we desire.

(Refer Slide Time: 07:58)

**Balanced Binary Search Trees**

- A BST is balanced if  $h \sim O(\log n)$
- Balancing guarantees may be of various types:
  - Worst-case
    - AVL Tree
  - Randomized
    - Randomized BST, Skip List
  - Amortized
    - Splay
- These data structures have optimal complexity for all of search, insert and delete –  $O(\log n)$ . However:
  - Good for in memory operations
  - Work well for small volume of data
  - Has complex rotation and / or similar operations
  - Do not scale for external data structures

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kanpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

27.8

©Silberschatz, Korth and Sudarshan

So, if you look at BST is balanced  $h$  is of the order of  $\log n$  I am not going into the theory of proving why balancing means  $h$  is of the order of  $\log n$  or how this order of  $\log n$  comes if you have doubts please refer to your algorithms book you will find plenty of that now that naturally now if a in the data structure if I am regularly inserting and deleting data.

Then it is not guaranteed that it will remain balanced it might for example, if I am inserting the data in a in a say in increasing order in a in a binary search tree then every time the insertion will happen on the rightmost node and if. So, that will mean that I will have along the rightmost node I will have a long chain and therefore, it become like a linear list and therefore, any search in that will not remain optimally it will take order in time.

So, there are different strategies that you have studied in terms of balancing just to remind you there are strategies which give you the best possible worst case time of  $\log n$  which is the famous AVL tree there are randomized strategies in terms of randomized BST skip

list there are amortized strategies which say that I do not really care about what happens with a particular insertion search or deletion, but what I care is if I have done a large number of insert delete search operations on the array then on the average what it should be balanced on the average it should be of ordered  $\log n$ .

So, we have seen all of these different strategies and. So, in an order in an attempt to generalize them for external data structure we note that these are typically good for in memory operations these are good worked well; when you deal with a small volume of data I mean you may be that may still be large, but is small in the sense that the whole data fits into the memory and you can manipulate muscles the whole data in memory and it of course, all these many many of these algorithms.

Particularly, the AVL tree and quite a bit of the randomized BST have complex rotation operations that need to be performed the order different random generators need to be involved in the randomized BST or skip list. So, there are other complexities in this whole factor comparison cost is not the only cost that you have and in the simple thing is they do not scale, what external data structures they are not optimized in terms of minimizing or optimizing the disc accesses or they do not scale to millions and millions of entries and so, on.

(Refer Slide Time: 10:38)

The slide has a header '2-3-4 Trees' in red. On the left, there is a small sailboat icon and some text: 'MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. The main content is a bulleted list of properties:

- All leaves are at the same depth (the bottom level).
  - Height,  $h$ , of all leaf nodes are same
    - ▶  $h \sim O(\log n)$
    - ▶ Complexity of search, insert and delete:  $O(h) \sim O(\log n)$
- All data is kept in sorted order
- Every node (leaf or internal) is a 2-node, 3-node or a 4-node, and holds one, two, or three data elements, respectively
- Generalizes easily to larger nodes
- Extends to external data structures

At the bottom, there is a video player showing a man speaking, the text 'Database System Concepts - 6<sup>th</sup> Edition', the number '27.10', and the copyright '©Silberschatz, Korth and Sudarshan'.

So, you need to look at a different approach and this different approach in the day in terms of database application database structures is called B + tree and what I am going

to discuss is an in memory version of that which is a 2-3,-4 trees. So, that we can understand the basic principle of such search data structure which can work with external data with this data, but first understand them in as an in memory version.

So, what is a 2-3-4 tree a 2-3-4 tree in contrast to other BSTs or in contrast to the typical BSTs where you know every operation needs the height of the tree to be balanced because some leaves could happen at a much deeper level some could be at a much shallow level in a 2-3-4 tree all leaves always are at the same level the same depth the bottom level.

So, height  $h$  is the height of all the leaf nodes and that is guaranteed to be of order of  $\log n$ , if the tree has  $n$  number of nodes the complexity of search, delete and insert all are order  $h$  that is a consequence of a constant height  $h$  or or a fixed height  $h$  that given  $n$  that is maintained all data are kept in a sorted order. So, if you do a in order traversal of the tree you will get the data in the sorted order, but the (Refer Time: 12:13) difference is in contrast to the BST where every node is a binary node is a is a (Refer Time: 12:19) one key and has two children here every node either a leaf node or an internal node every node is one of the three types it can either be a 2-node or a 3-node or a 4-node.

A 2-node holds one, data 3-node once holds 2 data and 4-node holds 3 data and there they give the name get the name 2-node 3-node and 4-node based on the number of children that they can have the 2-node can have 2 children, 3-node 3 children and 4-node 4 children. They can generalized easily to larger nodes which can have a large number of different types of nodes and it extends very naturally to external data structure. So, that is with the basic about the 2- 3- 4 tree.

(Refer Slide Time: 13:05)

The slide is titled "2-3-4 Trees". It features a small sailboat icon at the top left and a "PPD" logo at the top right. A vertical sidebar on the left contains the text "CHALMANS NOC NOC INSTRUCTOR: Prof. P. P. Das, IIT Kharagpur - Jan-Apr- 2018". The main content area has a red header "2-3-4 Trees". Below it, a bulleted list states: "■ Uses 3 kinds of nodes satisfying key relationships as shown below:" followed by four points: "● A 2-node must contain a single data item (S) and two links", "● A 3-node must contain two data items (S, L) and three links", "● A 4-node must contain three data items (S, M, L) and four links", and "● A leaf may contain either one, two, or three data items". Three diagrams illustrate these nodes: a 2-node with a single data item 'S' and two links labeled "Search keys < S" and "Search keys > S"; a 3-node with two data items 'S' and 'L' and three links labeled "Search keys < S", "Search keys > S and < L", and "Search keys > L"; and a 4-node with three data items 'S', 'M', and 'L' and four links labeled "Search keys < S", "Search keys > S and < M", "Search keys > M and < L", and "Search keys > L". At the bottom left is a video player showing a man speaking, and at the bottom right are various icons.

So, let us just go through it in little bit more detail it uses three kinds of node as I have said and now let me just show you. So, if you are talking about a 2-node. So, this is a 2-node. So, there is one data item S and all search keys which are less than S around this side all search keys which are on greater than S around this side.

We are just for the simplicity of discussion where I am assuming that all keys in this data structure are unique. So, there is no repeated keys the repeated keys can be handled in a very easy manner. So, this is one type of node a second type of node is a 3-node which must contain two data items S we are calling it S and L and three links the first is less than S(**Search key<S**) the last is greater than L(**Search key>L**) and the middle is greater than S, but less than L(**S<Search Key<L**) which means actually what we enforce in terms of the two keys that exist at the node  $S < L$ .

So, values less than L go on one link values between S and L go on the middle link and values  $> n$  go on the third link. Similarly, if I have a 4-node here I have three values where  $S < M < L$ . So, now, you can understand why the acronyms S, M and L small, medium and large. So, on and we have four links the first link gives you values **Search Key < S** second is **S<Search Key<M** third between **M< Search Key< L** and fourth **Search Key >L**. So, these are the or three different types of nodes that a 2-3-4 tree can support and if it is a leaf node then it can be it can contain either 1, 2 or 3 get items. So, let us go forward.

(Refer Slide Time: 15:01)

2-3-4 Trees: Search

- Search
  - Simple and natural extension of search in BST

CHANDRAKANTAPUDI-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

27.12

©Silberschatz, Korth and Sudarshan

Now, to search to search is a simple extension of BST you know how to search in a BST you start with the route see whether the given key to search is greater than the key at the route if it is greater you go to right if it is less you go to left and you do the same thing here for a 2-node for a 3-node all that you will need to find out is between S and L whether it **Search Key < L** then **Search Key < S** then you take the leftmost whether it is **Search Key > L** then you take the rightmost if it is **S < Search Key < L** you take the middle similar strategy you do for 4-node and search is a simple extension of the BST algorithm. So, you can suddenly work it out.

(Refer Slide Time: 15:38)

2-3-4 Trees: Insert

- Insert
  - Search to find expected location
    - ▶ If it is a 2 node, change to 3 node and insert
    - ▶ If it is a 3 node, change to 4 node and insert
    - ▶ If it is a 4 node, split the node by moving the middle item to parent node, then insert
  - Node Splitting
    - ▶ A 4-node is split as soon as it is encountered during a search from the root to a leaf
    - ▶ The 4-node that is split will
      - Be the root, or
      - Have a 2-node parent, or
      - Have a 3-node parent

CHANDRAKANTAPUDI-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

27.13

©Silberschatz, Korth and Sudarshan

Now, what we will need to do in terms of an insert, insert is very interesting. So, for insert first you search and find the expected location where you are expecting it. So, that is not there. So, you will expect. So, now, what are the possibility? Possibilities where you have found the expected location that node could be a 2-node if it is a 2-node all that you simply need to do is change that where 3-node inserts the second item if it is a 2-node it has only one item. So, just insert this new item there and making it into a 3-node good.

In the second case if you find the location if you find that it is a 3-node. So, it has two items you just change it to a 4-node and insert this is as a third item; obviously, when you insert you will have to decide has to whether where you should insert that depends on whether you are given key is greater than the key that already existed or smaller than that and so, on. Now, the question is what happens; when if you locate the place to be insert itself is already a 4-node. Now, naturally you do not have anything  $>$  4-node. So, if it is a 4-node then all that you need is to split that node you have to split the node. So, you have a 4-node.

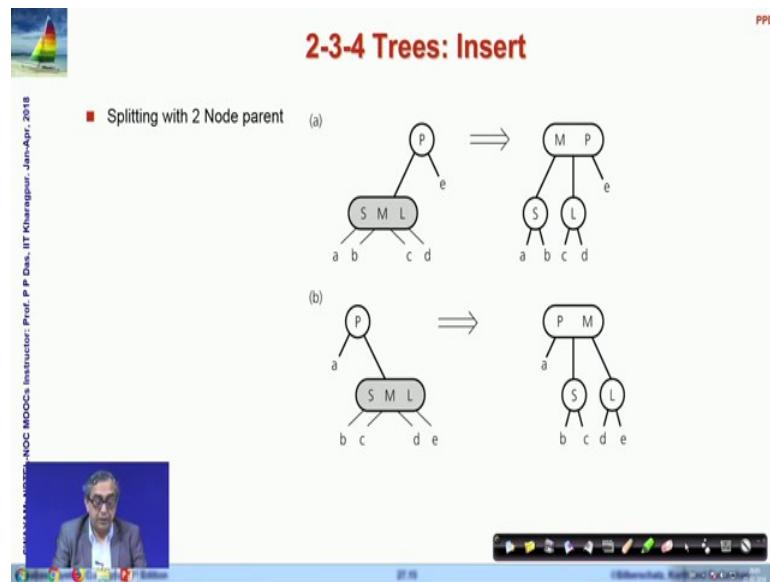
So, you have a 4-node. So, you have you have a data 1 here you have data 2 here, you have data 3 here and you are you know that the your d has to get inserted in this. So, you cannot insert it by changing the node type because this is a maximum allowed. So, all that you want to do is to change that into at different structure and that structure is called a node splitting structure. So, we will we will see in the next slide as to how this is done and we will see how different splitting sequence can happen and what will happen when a 4-node will split when it was a root or the different kinds of parents that it has let us just go there.

(Refer Slide Time: 17:42)

The slide is titled "2-3-4 Trees: Insert". It features a diagram illustrating the splitting of a root node. On the left, a rounded rectangle contains three nodes labeled "S", "M", and "L". Below it, two pairs of nodes are labeled "a b" and "c d". An arrow points to the right, where a tree structure is shown. The root node is labeled "M". It has two children: a left child labeled "S" which has two children "a" and "b", and a right child labeled "L" which has two children "c" and "d". The slide also includes a small video window showing a professor speaking, and a footer with text and icons.

So, what we are saying is the suppose you have a 4-node which is a root now splitting that is actually doing this. So, you have to convince yourself that enough 2-3-4 tree what we have already assumed; whether I represent this or I represent this are algorithmically their equivalent. So, a single root 4-node and a such a structure of your 3- node, two nodes are equivalent why is it? So, for example, if you are looking say if you are looking for a here then it is it say it is less than s. So, you come here now if you are looking for the same a here what happens  $a < S$ . So, it is it **Search Key < M** remember  $S < M < L$  So,  $a < M$ . So, you take this part it is  $< S$  to you come here let us take a case of c lets say c. So, if it is c you should come here, now if you check here c if it is falling on this link; that means, it is  $c > M$  and  $c < L$ . So,  $c > M$  you come here  $c < L$ . So, you come here. So, you reach the same link. So, you can see that actually a 2-3-4 tree is not a unique representation depending on the requirement I can replace 4-nodes in terms of other 2-nodes and actually create a new tree configuration. So, if it is at the root I can get rid of a 4-node and replace it by this equivalent tree.

(Refer Slide Time: 19:27)



Now, suppose there are the 4-node is not at the root it is somewhere else. So, what are the possibilities the; if it is not at the root it must have a parent now the parent could be a 2-node. So, if it is a 2-node then these are the two possibilities if it is a 2-node, then. So, this is a parent node which is a 2-node. So, then the 4-node could be a left child of that or it could be a right child of that if it is a left child, then we use split take the middle item and insert it in the parent and by that process a parent becomes a 3-node from a 2-node and make these become two different 2-nodes.

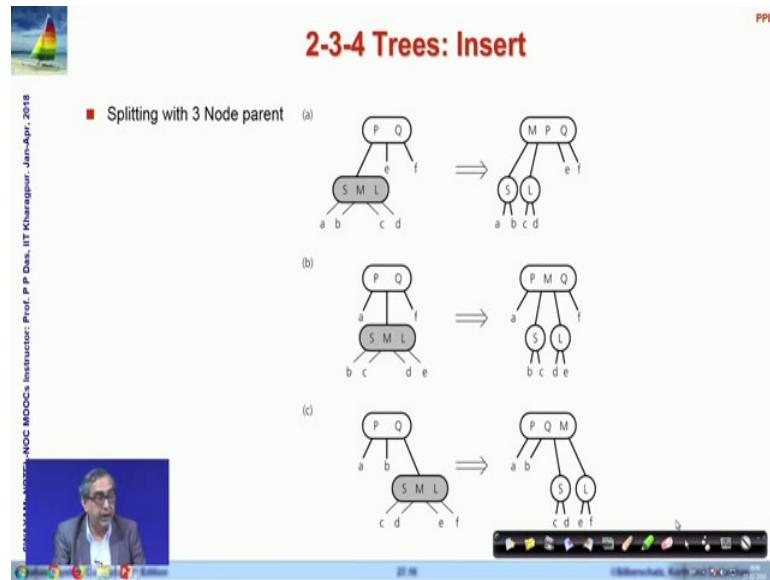
Again the way I was just explaining in the in the previous slide you can convince yourself that this structures are equivalent for example, if I am looking for d let us say if I am looking for d in this tree.

So, if I am looking for  $d > L$ . So, how do I arrive here now since this is a left child? So, it  $d < P$ , otherwise it could not have occurred on this side. So,  $d < P$ . So,  $d < P, d > L$ . So, given that if I search for d here so,  $d < P$  and since it is we I also have from this the  $d > M$ , otherwise it would not have come to the; this third link this fourth link it would have been elsewhere. So, I know that  $d > M$ .

So, and  $d < P$ . So, if I combined this two, then d has to go on this middle ink, because it is  $d > M$  and  $d < P$  and then I have it is  $d > L$ . So, it has to be on the right. So, it comes to the right position.

So, in this way you can convince yourself in every search case that if the parent is a 2-node, then you can equivalently split the 4-node and make an insertion in the parent 2-node converted into 3-node and get rid of the 4-node altogether that lives us with.

(Refer Slide Time: 21:45)

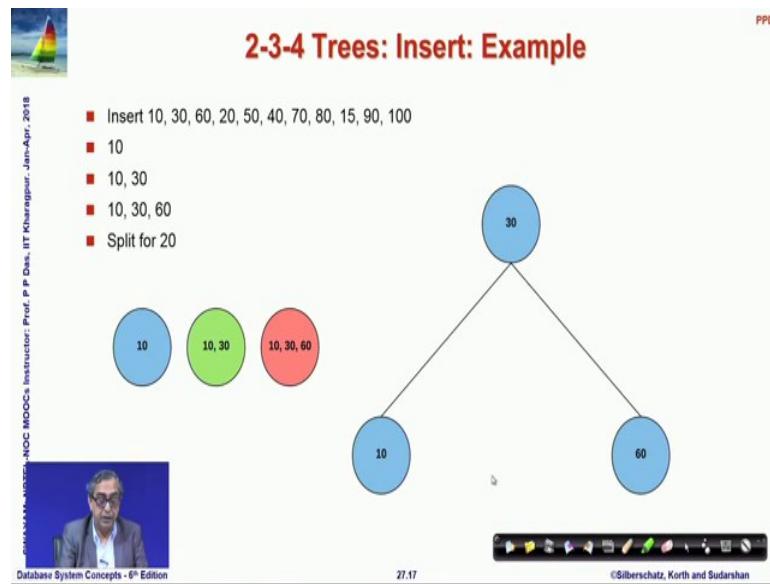


One other case which is if the parent is a 3-node naturally if it is the parent is a 3-node then there are three possibilities. Now, because your 4-node could be a left child a middle child or a right child and in every case you do the same thing you split the 4-node take the middle item put it to the parents. So, parent converts from 3-node to 4-node. Now and your rest of the split and pointed adjustments had done. So, that you get rid of this.

So, what happens in this process in and like the earlier ones here you do not get rid of the 4-nodes altogether, because in replacing one 4-node your creating another 4-node, but in the process what is happening the 4-node is moving up it is going one level up.

So, again what you will do is in recursively the new parent 4-node parent will again be split and I just split if it is parent that is parent of the parent if that parent is also a 3-node, then that will become a 4-node this will continue till your root becomes a 4-node and we know that when root becomes a 4-node I can always change it to a configuration of 3, 2-nodes. So, in this process as we have shown that we can actually get rid of the 4-nodes in the whole of the 2-3-4 tree when it is required.

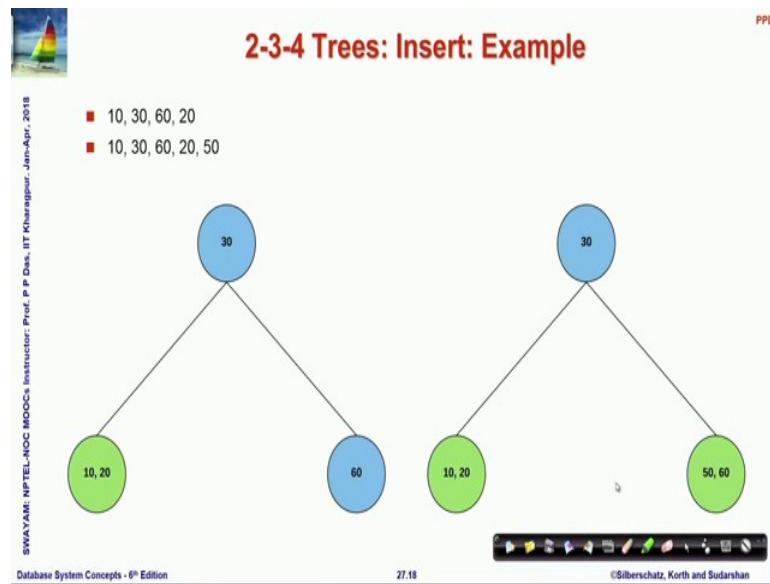
(Refer Slide Time: 23:03)



So, the basic strategy is very simple that will keep on constructing the 2-3-4 tree and whenever we come across a 4-node for the first time we will split that and rearrange. So, that I can get rid of it so, here what I show you is a basically an insert sequence over the next couple of slides where which is trying to insert the data in this following order starting with an empty 2-3-4 tree. So, we first insert 10. So, you get this then we insert 30 that is here. So, 2-node becomes. So, the here the convention that I am I am following is blue is a 2-node green is a 3-node and red is a 4-node. So, then you insert 60 it becomes a 4-node and as soon as it becomes a 4-node the next element to be entered is 20.

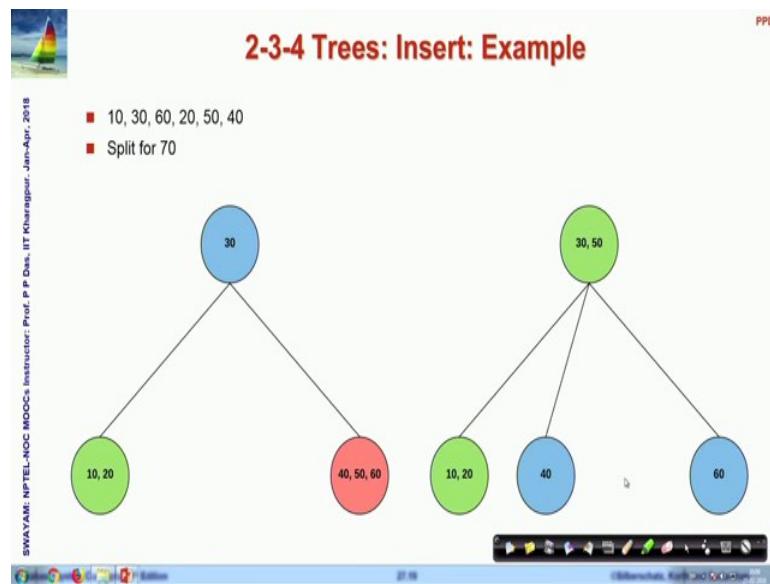
But, before that this 4-node will have to be split and this is the case of splitting at the root, because this is this has no child yet. So, you split and you get the middle moves to the top here as 30, then you have two children 10 and 60 and after the splitting you move on to actually inserting the intended 20 into it.

(Refer Slide Time: 24:19)



So, 20 gets inserted on this side 20 is inserted on this side, because a smaller than this and comes here. So, this 2-node becomes a 3-node then you insert 50 which goes on this side which goes on this side and gets inserted here.

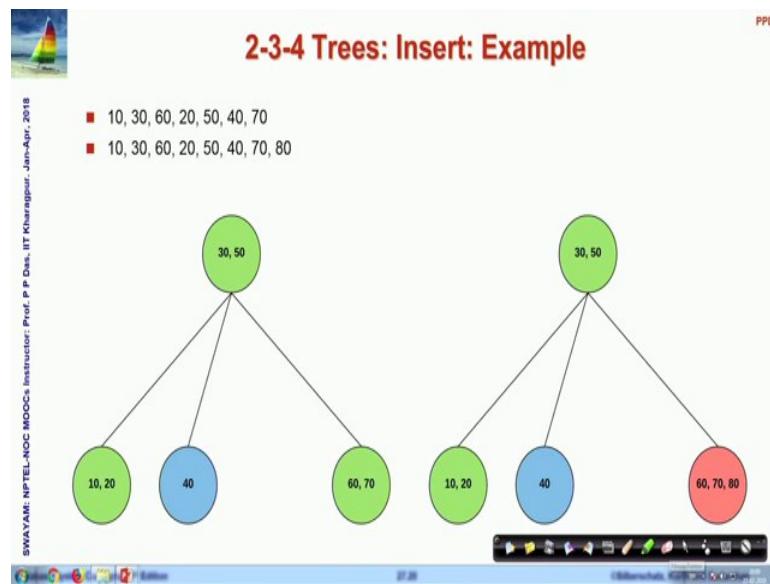
(Refer Slide Time: 24:41)



So, your insertion continues your next to insert is 40 which gets the inserted here it becomes a 4-node and as soon as it becomes a 4-node and before the next insertion of 70 can happen you need to do a split. So, you do a split.

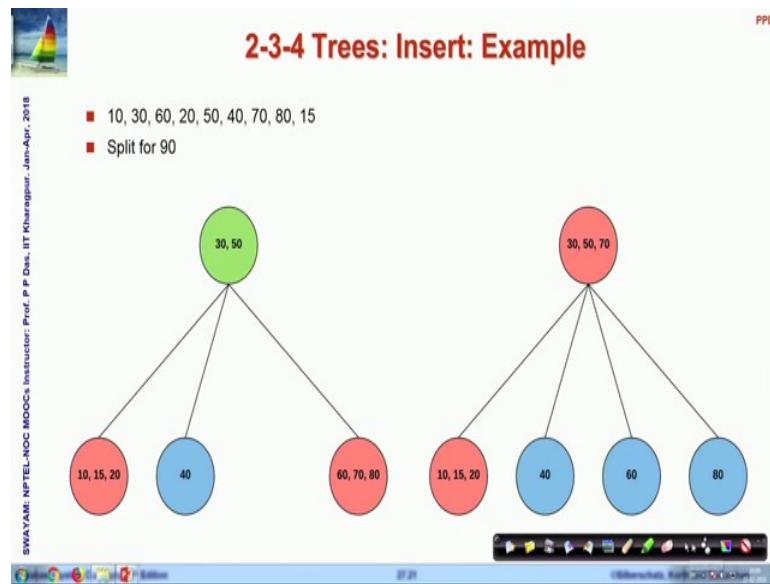
If you do a split then your 50 moves to the parent this becomes a 3-node. Now and your 40 and 60 becomes two 2-node children as this. So, it is a same information is represented, but now in this new 2-3-4 tree you do not have any 4-node. So, in you can go ahead and insert 70.

(Refer Slide Time: 25:19)



So, insert 70, 70 gets inserted here the 2-node becomes 3-node the, this is done. So, then you insert 80, 80 gets inserted here is greater than it comes here and this becomes a 4-node and. So, naturally the next would be two insert 15. So, 15 goes in on to the left. So, 15 has come in here which becomes a 4-node.

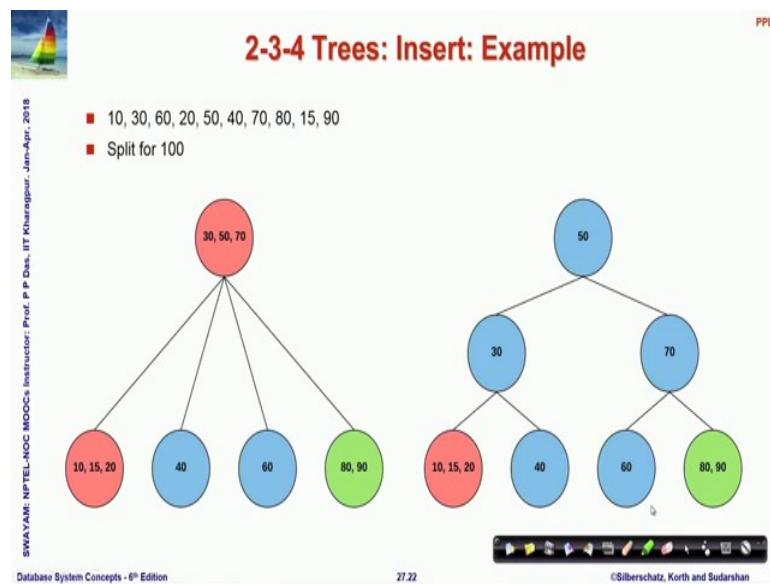
(Refer Slide Time: 25:43)



And the next is to insert 90. So, which has to get inserted here it should have got inserted here. So, this is already a 4-node. So, you need to split.

So, in split and as you split you get 40, 40 was already there you get 60 and 80 and the 70, that existed in the middle goes to the root and your root becomes a 4-node.

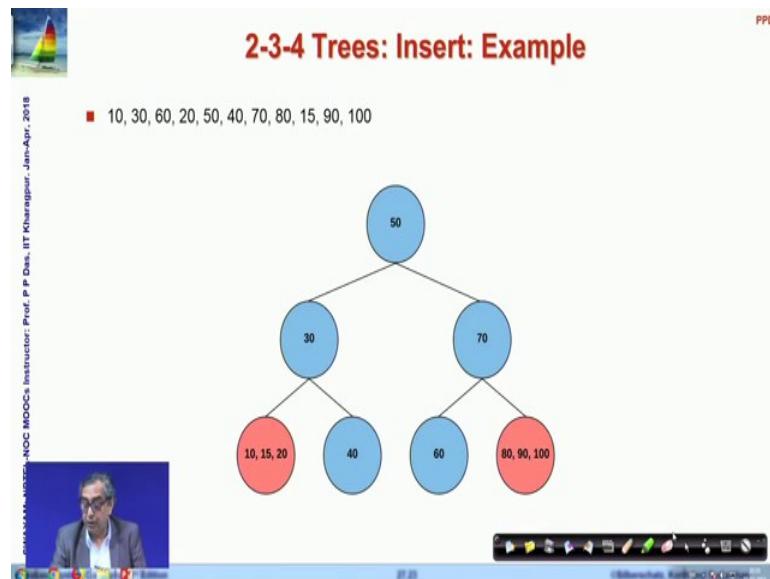
(Refer Slide Time: 26:27)



So, this is the configuration you move on to your 90 gets inserted. Now, you have to insert 100. So, you before that this your root has become a 4-node. So, you do a split. So,

as you do the split this is a new configuration that has happened and this is what has resulted from the split of this 4-node at the root.

(Refer Slide Time: 26:59)



And then naturally you can go ahead and insert 100 and 100 is now inserted. Here, you could do adjustments of changing this this 4-node into by splitting it and moving it onwards I have not shown that, but if you now go back if you just now go back on on these this whole process and you will see that specifically that we claim that all leaf nodes would be in the same level. So, you can see that initial three trees all are at the same level, then leaf and their level 0 and then when you have had this split then the leaf nodes 10 and 60 are both at level once.

So, we can see that when actually you split at the root you add one level to all the leaf nodes and that is the only time your height changes. So, beyond that you look at this the level has not changed I am going to the next the level has not changed instruments to be height 1 level does not change height 1 does not change does not change till the left here and then we have a case again of splitting a 4-node at the root.

When the one level has been added to all the; so, all the leaf nodes where at level 1 now all of them are at level 2. So, in a 2-3-4 tree you achieve this invariance of all leaf nodes being at the same level by maintaining that the only time the height changes is when you split a 4-node at the root and add one level uniformly to all of them.

And then rest of the logic is quite simple that these are here you can do actually follow the same logic as of the binary search tree analysis that if there are  $n$  items here; then the maximum height could actually be  $\log n$ ; because we though all nodes are not binary, but the nodes that are not binary actually have more data.

So, they will be they will the height will always be  $\log n$  or less than that of course, there is an issue of concluding the complexity, because now you will argue that there are 3-nodes or 4-nodes where more than one comparison is required to decide about the node, but the counter argument to that is even if that be the case even if you have along the path of the tree from the root to any leaf node.

Even if all of the nodes are of are 4-node that cannot happen as you have seen because you will keep on splitting and distributing that, but if all of them are or 4-node also then what will add is simply a factor of three two rather additional with the  $\log n$  and the overall complexity remains to be  $\log n$  will go.

(Refer Slide Time: 29:59)

The slide has a title '2-3-4 Trees: Delete' at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list under the heading 'Delete':

- Delete
  - Locate the node  $n$  that contains the item  $theItem$
  - Find  $theItem$ 's inorder successor and swap it with  $theItem$  (deletion will always be at a leaf)
  - If that leaf is a 3-node or a 4-node, remove  $theItem$
  - To ensure that  $theItem$  does not occur in a 2-node
    - ▶ Transform each 2-node encountered into a 3-node or a 4-node
    - ▶ Reverse different cases illustrated for splitting

At the bottom left, there is a video player showing a man speaking, with the text 'Database System Concepts - 8<sup>th</sup> Edition'. At the bottom right, there is a navigation bar with icons and the text '27.24' and '©Silberschatz, Korth and Sudarshan'.

Now, if you have to delete you have to do very similar operations I have not shown the details, but you look at the node and then actually find the in order successor for that and swap it with the item and then you can do the other arrangements of collapsing the nodes as we have done the splitting of nodes. Now I have to do the collapsing of nodes following the same river structure and leave that as an exercise to you just work it out at home.

(Refer Slide Time: 30:28)

The slide is titled "2-3-4 Trees" in red at the top right. At the top left is a small sailboat icon. On the right side, there is some small text that appears to be "PPD".  
**Advantages**

- All leaves are at the same depth (the bottom level): Height,  $h \sim O(\log n)$
- Complexity of search, insert and delete:  $O(h) \sim O(\log n)$
- All data is kept in sorted order
- Generalizes easily to larger nodes
- Extends to external data structures

**Disadvantages**

- Uses variety of node types – need to destruct and construct multiple nodes for converting a 2 Node to 3 Node, a 3 Node to 4 Node, for splitting etc.

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
27.25  
©Silberschatz, Korth and Sudarshan

So, if you look at if the 2-3-4 tree and there is a time to justify why we are doing this all leaves are the same depth which is a great advantage the height is order  $\log n$  always complexity of search insert delete all are order  $\log n$  all data kept in sorted order it generalizes easily to larger nodes are and extends to external data structure. So, what you mean by larger nodes, let us let us look at that a little bit before that of course, 2-3-4 tree has a major disadvantage compared to binary research trees of the other kinds because it uses a variety of node types. So, you I mean when you change from a say 2-node to a 3-node you actually if you think in programming terms you have lot of additional cost, because you need to distract the 2-node and create a 3-node.

If you split a 4-node and create 3, 2-nodes as required, then you have to distract the 4-node and create 3, 2-nodes. So, there are lot of over rights in terms of that.

(Refer Slide Time: 31:32)

The slide is titled "2-3-4 Trees" in red at the top right. On the left, there is a small sailboat icon. The main content is a bulleted list of properties:

- Consider only one node type with space for 3 items and 4 links
  - Internal node (non-root) has 2 to 4 children (links)
  - Leaf node has 1 to 3 items
  - Wastes some space, but has several advantages for external data structure
- Generalizes easily to larger nodes
  - All paths from root to leaf are of the same length
  - Each node that is not a root or a leaf has between  $\lceil n/2 \rceil$  and  $n$  children.
  - A leaf node has between  $\lceil (n-1)/2 \rceil$  and  $n-1$  values
  - Special cases:
    - » If the root is not a leaf, it has at least 2 children.
    - » If the root is a leaf, it can have between 0 and  $(n-1)$  values.
- Extends to external data structures
  - B-Tree
  - 2-3-4 Tree is a B-Tree where  $n = 4$

At the bottom left, it says "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kanpur Date: Jan-Apr., 2018". At the bottom center, it says "Database System Concepts - 8th Edition" and "27.26". At the bottom right, it says "©Silberschatz, Korth and Sudarshan".

So, what leaf if we if we just simplify that process and if you assume that, there is only one node type which has enough space for three items and four links we do not have two I mean we functionally there will be 2-node 3-node 4-node, but physically let them with a same type of node. So, any internal node can have what is specification? So, there the same type any internal node has 2 to 4 children and a leaf node has 1 to 3 items that is what all that we are saying.

So, by doing this we will waste some space, but we have several advantages particularly when you look at this for external data structure. So, what will happen is if I can think about 2-3-4 tree in that manner then I can; obviously, generalize that it is not necessary that I will have to restrict myself at 4 links, 4 children, I can do more than that.

So, in general I can say that there are a node has  $n$  children and it is each node is not a leaf not a root or a leaf will have between  $n / 2$  and  $n$  children. So, put  $n$  as 4 you will find that it becomes 2-3-4 tree and a leaf node will have  $(n-1) / 2$  which is 1 and for enough  $n$  being 4 and  $n-1$  or 3 values which is what did you have.

So, if you just generalize from 2-3-4 to  $n/2$  to  $n$  and have a container have a node container which can have either  $n/2$  or  $(n / 2) + 1$  or  $(n / 2) + 2$  or maximum up to  $n - 1$  data items and corresponding number of children then we will be very easily be able to arrange for a similar data structure which will have all the nodes all the leaf nodes at the same level and. So, this is the structure which is which extends very easily and this is a

fundamental structure of what he says a B-tree or a B + tree will see the differences shortly.

But this is a basic notion and the strategies of node splitting and node merging in case of deletion and the algorithm of insertion deletion that we have discussed here will simply get generalize in case of B-tree.

(Refer Slide Time: 33:57)

**Module Summary**

- Recapitulated the notions of Balanced Binary Search Trees as options for optimal in-memory search data structures
- Understood the issues relating to external data structures for persistent data
- Explored 2-3-4 Tree in depth as a precursor to B/B+-Tree for an efficient external data structure for database and index tables

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jun-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition      27.27      ©Silberschatz, Korth and Sudarshan

When we go to the next module so, we have recapitulate in on the balanced binary search tree and we introduced a the notion of a 2- 3- 4 tree which is a precursor to B + tree B-tree which is which are efficient external data structures and will be covered in the next module.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 28**  
**Indexing and Hashing/3 : Indexing/3**

Welcome to module 28 of Database Management Systems. We have been discussing about indexing and hashing.

(Refer Slide Time: 00:28)

PPD

**Module Recap**

- Balanced Binary Search Trees
- 2-3-4 Tree

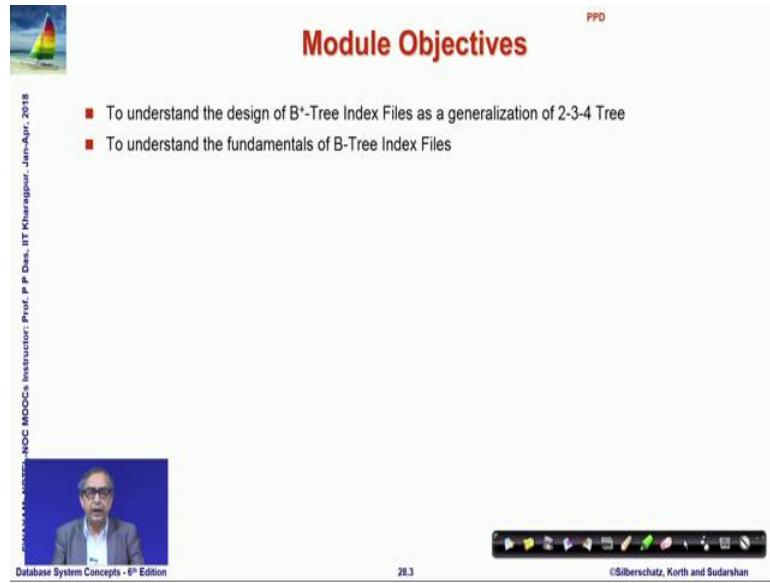
Database System Concepts - 8<sup>th</sup> Edition

28.2

©Silberschatz, Korth and Sudarshan

This is the third module; in that continuation.

(Refer Slide Time: 00:38)



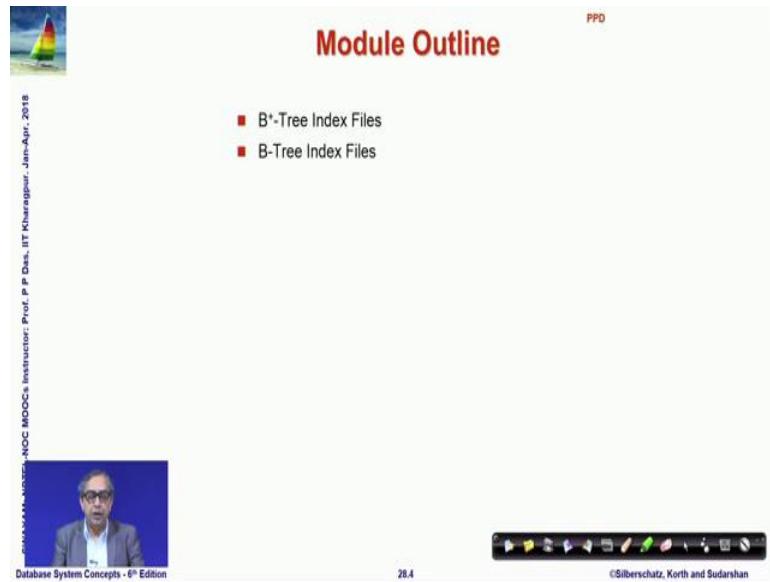
This slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. On the left side, there is vertical text: "CHANDRAKANTAPURE-NOC-MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur, Jam-Apr-2018". In the center, there is a list of objectives:

- To understand the design of B+-Tree Index Files as a generalization of 2-3-4 Tree
- To understand the fundamentals of B-Tree Index Files

At the bottom left is a video frame showing a man speaking. The bottom right contains the text "Database System Concepts - 8<sup>th</sup> Edition", the page number "28.3", and the copyright notice "©Silberschatz, Korth and Sudarshan". A navigation bar is at the very bottom.

In the last module we have taken a quick look at the balanced BST and specifically a and different kind of inline data structure called 2-3-4 tree, which can be of very good use in terms of understanding B+ tree, which we want to study in this module and we will also take a quick look at the B tree.

(Refer Slide Time: 00:50)



This slide is titled "Module Outline" in red at the top right. It features a small sailboat icon in the top left corner. On the left side, there is vertical text: "CHANDRAKANTAPURE-NOC-MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur, Jam-Apr-2018". In the center, there is a list of topics:

- B+-Tree Index Files
- B-Tree Index Files

At the bottom left is a video frame showing a man speaking. The bottom right contains the text "Database System Concepts - 8<sup>th</sup> Edition", the page number "28.4", and the copyright notice "©Silberschatz, Korth and Sudarshan". A navigation bar is at the very bottom.

So, now, B + tree is the main data structure is or one of the main data structures to be used for index files.

(Refer Slide Time: 01:00)

**B<sup>+</sup>-Tree Index Files**

B<sup>+</sup>-tree indices are an alternative to indexed-sequential files

- Disadvantage of indexed-sequential files
  - performance degrades as file grows, since many overflow blocks get created
  - Periodic reorganization of entire file is required
- Advantage of B<sup>+</sup>-tree index files:
  - automatically reorganizes itself with small, local, changes, in the face of insertions and deletions
  - Reorganization of entire file is not required to maintain performance
- (Minor) disadvantage of B<sup>+</sup>-trees:
  - extra insertion and deletion overhead, space overhead
- Advantages of B<sup>+</sup>-trees outweigh disadvantages
  - B<sup>+</sup>-trees are used extensively

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

28.6

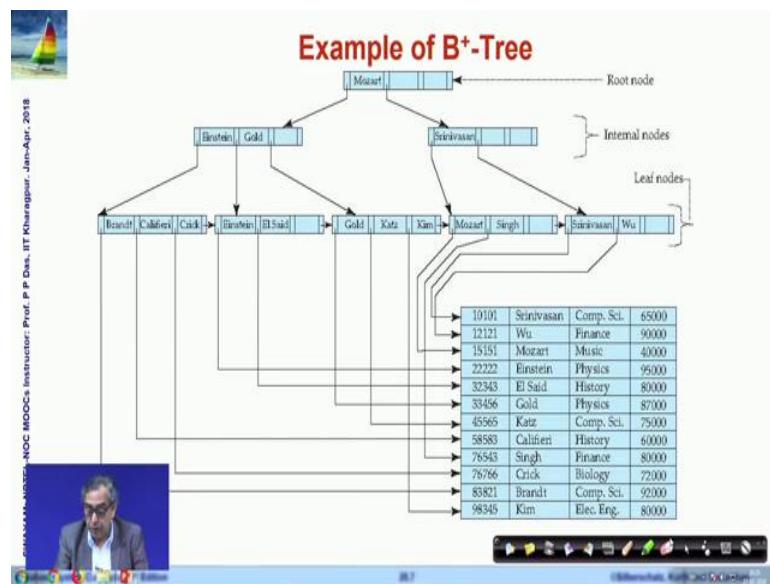
©Silberschatz, Korth and Sudarshan

So, B + tree has now; what we have seen we have seen the ordered indexes. We have seen the index sequential files, where you could keep the index file in a sorted manner in the primary index you could build secondary index on that and so, on, but that is not an efficient way of doing things, because the performance keeps on degrading as the file grows.

Since many overflow blocks will get created, because certainly if you are growing, then naturally you have created say sparse index on uncertain values and if there are more records in that bucket. Then naturally you need to have linked buckets. So, periodic reorganization of the entire file becomes required which is a very costly affair.

In contrast advantage of B + tree is it automatically reorganizes itself in small bits and pieces with local changes and so, on; whenever insertions and deletions happen and the reorganization of the entire file is not required for the purpose of maintenance. Of course, there are a little bit of disadvantage the extra insertion and deletion overhead exist for the small you know micro reorganization there is little bit of space over it, but in the face of the advantage that we get it outweighs the advantage is outweighs the disadvantages and B + trees are used quite extensively.

(Refer Slide Time: 02:28)



So, just recall the notion of 2-3-4 tree that we had discussed and look at this diagram. So, 2-3-4 tree have different types of node : 2 node 3 node and 4 node. So, we said that there could be a node which can be only partially filled and it has a different number of children pointing to the; conditions of how different keys are ordered in that particular node.

So, here we I show an instance of a B + tree, which is basically trying to represent this file in terms of the creating indexes. So, if the index is actually based on the name. So, this is the root node that you have and for an instance; we are taking a structure where every node can have 3 data items and 4 links and it could be it could be more it could be less, but this is just for an example. So, as you can see; so if we have this link, then on the left of Mozart, then it means all keys which are less than Mozart will be available on this link below; the link that exists here is for all keys which are greater than Mozart and less than right. Now there is nothing.

So, those will occur here. So, as you can see that Einstein, Gold, Brandt all these will come on this length Srinivasan, Singh, Wu all this come on this side the Mozart itself comes on this side. Now, if I look at this node the next level loads. Now this link has values which are less than Einstein as you can see this has values which are between Einstein and Gold. So, Einstein and I set these are values which are more than gold.

So, this is this is a and as you can see that though all nodes are shown to be of the same type as we had mentioned at the end of the 2-3-4 tree discussion, but it has variable number of entries. So, the number of links are between  $n/2$  and  $n$ . So,  $n$  here is 4. So, you have at least either at least two entries or maximum up to 4 entries that can go on here.

(Refer Slide Time: 05:08)

The slide has a title 'B+-Tree Index Files (Cont.)' in red. Below the title is a small image of a sailboat. The main content is a bulleted list of properties for a B+-tree:

- All paths from root to leaf are of the same length
- Each node that is not a root or a leaf has between  $\lceil n/2 \rceil$  and  $n$  children.
- A leaf node has between  $\lceil (n-1)/2 \rceil$  and  $n-1$  values
- Special cases:
  - If the root is not a leaf, it has at least 2 children.
  - If the root is a leaf (that is, there are no other nodes in the tree), it can have between 0 and  $(n-1)$  values.

On the left side of the slide, there is vertical text: 'MANAKARAN NOORI, NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018'. At the bottom left is a video player showing a man speaking, with the text 'Database System Concepts - 8th Edition'. At the bottom right is a navigation bar with icons for back, forward, search, etc., and the text '©Silberschatz, Korth and Sudarshan'.

So, this is the basic observe definition of a B + tree. All paths from root to leaf are of the same length. This is again something you should observe here, because if you see all of these paths all of them have the same length here then the length is 2. So, that is a basic property of 2-3-4 tree generalized into B + tree.

So, each node that is not a root is a leaf level has between  $n / 2$  to  $n$  children. Leaf node has  $(n - 1) / 2$  to  $n - 1$  value. And the if the root is not a leaf, then it has at least 2 children and if the root is a leaf there is no other nodes in the tree then it can have between 0 to  $n - 1$  values which are quite obvious.

(Refer Slide Time: 05:54)

## B<sup>+</sup>-Tree Node Structure

■ Typical node

$P_1$	$K_1$	$P_2$	$\dots$	$P_{n-1}$	$K_{n-1}$	$P_n$
-------	-------	-------	---------	-----------	-----------	-------

- $K_i$  are the search-key values
- $P_i$  are pointers to children (for non-leaf nodes) or pointers to records or buckets of records (for leaf nodes).
- The search-keys in a node are ordered  
$$K_1 < K_2 < K_3 < \dots < K_{n-1}$$
(Initially assume no duplicate keys, address duplicates later)

Database System Concepts - 8<sup>th</sup> Edition

28.9

©Silberschatz, Korth and Sudarshan

So, naturally a typical node will look like this, where the pointers and key values alternate starting with a pointer P<sub>1</sub>, then key K<sub>1</sub> and so, on and ending with a point at P<sub>n</sub>. And the search keys are strictly ordered  $K_1 < K_2 < K_{n-1}$  these are facts that we have seen for 2-3-4 tree.

(Refer Slide Time: 06:14)

## Leaf Nodes in B<sup>+</sup>-Trees

Properties of a leaf node:

- For  $i = 1, 2, \dots, n-1$ , pointer  $P_i$  points to a file record with search-key value  $K_i$
- If  $L_i, L_j$  are leaf nodes and  $i < j$ ,  $L_i$ 's search-key values are less than or equal to  $L_j$ 's search-key values
- $P_n$  points to next leaf node in search-key order

leaf node

Brandt	Califieri	Crick	→ Pointer to next leaf node
--------	-----------	-------	-----------------------------

Database System Concepts - 8<sup>th</sup> Edition

28.10

©Silberschatz, Korth and Sudarshan

So, for a leaf node the pointer P<sub>i</sub> points to the file record with the search key K<sub>i</sub> and if there are two leaf nodes L<sub>i</sub> and L<sub>j</sub> and  $i < j$ , then  $L_i \leq L_j$  search key values. So, this is

the basic ordering that we had seen in 2-3-4 tree, that is what is getting generalized for a non leaf node.

(Refer Slide Time: 06:40)



## Non-Leaf Nodes in B<sup>+</sup>-Trees

■ Non leaf nodes form a multi-level sparse index on the leaf nodes. For a non-leaf node with  $m$  pointers:

- All the search-keys in the subtree to which  $P_1$  points are less than  $K_1$ ,
- For  $2 \leq i \leq n - 1$ , all the search-keys in the subtree to which  $P_i$  points have values greater than or equal to  $K_{i-1}$  and less than  $K_i$
- All the search-keys in the subtree to which  $P_n$  points have values greater than or equal to  $K_{n-1}$

$P_1$	$K_1$	$P_2$	$\dots$	$P_{n-1}$	$K_{n-1}$	$P_n$
-------	-------	-------	---------	-----------	-----------	-------

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr- 2018

Database System Concepts - 6<sup>th</sup> Edition

28.11 ©Silberschatz, Korth and Sudarshan

Similarly all search-keys in the subtree which  $P_1$  points to are  $< K_1$ , then for all that  $P_n$  points to are  $> K_{n-1}$ . And in the other cases they are between the two consecutive key values that exist between the pointers.

(Refer Slide Time: 07:01)



## Example of B<sup>+</sup>-tree

■ Leaf nodes must have between 3 and 5 values ( $\lceil (n-1)/2 \rceil$  and  $n-1$ , with  $n = 6$ )

■ Non-leaf nodes other than root must have between 3 and 6 children ( $\lceil n/2 \rceil$  and  $n$  with  $n = 6$ )

■ Root must have at least 2 children

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr- 2018

Database System Concepts - 6<sup>th</sup> Edition

28.12 ©Silberschatz, Korth and Sudarshan

So, this is an example of a simple case which is  $n$  where  $n = 6$ .

(Refer Slide Time: 07:10)

**Observations about B<sup>+</sup>-trees**

- Since the inter-node connections are done by pointers, "logically" close blocks need not be "physically" close
- The non-leaf levels of the B<sup>+</sup>-tree form a hierarchy of sparse indices
- The B<sup>+</sup>-tree contains a relatively small number of levels
  - ▶ Level below root has at least  $2 \cdot \lceil n/2 \rceil$  values
  - ▶ Next level has at least  $2 \cdot \lceil n/2 \rceil \cdot \lceil n/2 \rceil$  values
  - ▶ ... etc.
    - If there are K search-key values in the file, the tree height is no more than  $\lceil \log_{n/2}(K) \rceil$
    - thus searches can be conducted efficiently
- Insertions and deletions to the main file can be handled efficiently, as the index can be restructured in logarithmic time

CHAKRABORTY NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

28.13

©Silberschatz, Korth and Sudarshan

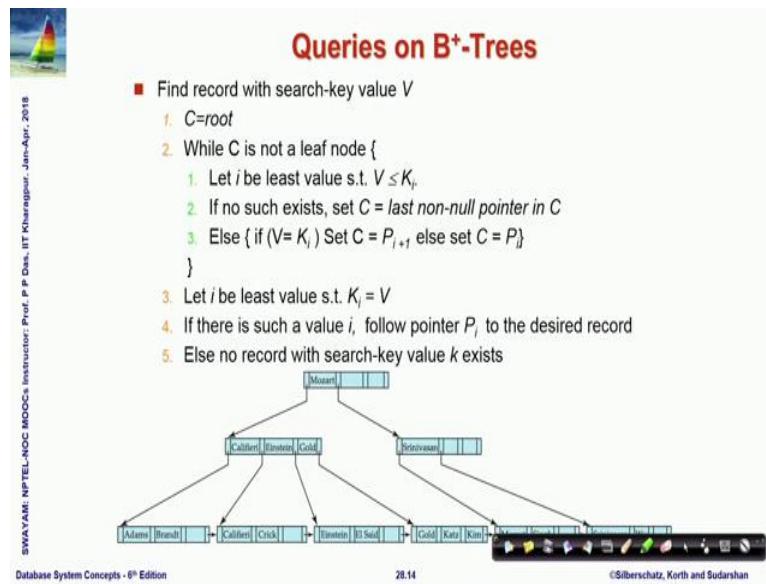
So, since the inter-node connections are done by pointers, “logically” closed blocks are not “physically” close. So, that is a key idea there is a key observation about the B + tree. So, 2 nodes the records which are logically closed are may not actually be physically close, because the pointers actually define the closeness in terms of the ordering of the values.

So, B + tree contains relatively small number of levels, we will see what that level would be? So, what will happen; if the level below root has two values at the most at least and the below that will have  $n/2$  values, because every node has to be at least half full. We have said every node we will have to have  $n/2$  lengths to  $n$  links it cannot be less than that less than  $n / 2$  link.

So, the next level as  $n / 2$ , then the next level has  $2 * n/2 * n/2$  and so, on. So, every time you go down you can basically increasing by a factor of  $n/2$ , which as you all know simply means that the number of levels or the height is  $\log K$  to the base  $n/2$ , where  $K$  is a number of search key values that exist on the tree. So, larger the end smaller is this value. So, larger the node size is smaller is a height and therefore, the number of insertion number of you know access operations that need to be performed.

So, insertion, deletions to the main file can be handled efficiently as the index can be restructured in logarithmic time as you have just seen.

(Refer Slide Time: 09:01)



The slide title is "Queries on B<sup>+</sup>-Trees". It features a logo of a sailboat on the left and a decorative footer bar at the bottom right. The main content is a flowchart of a search algorithm:

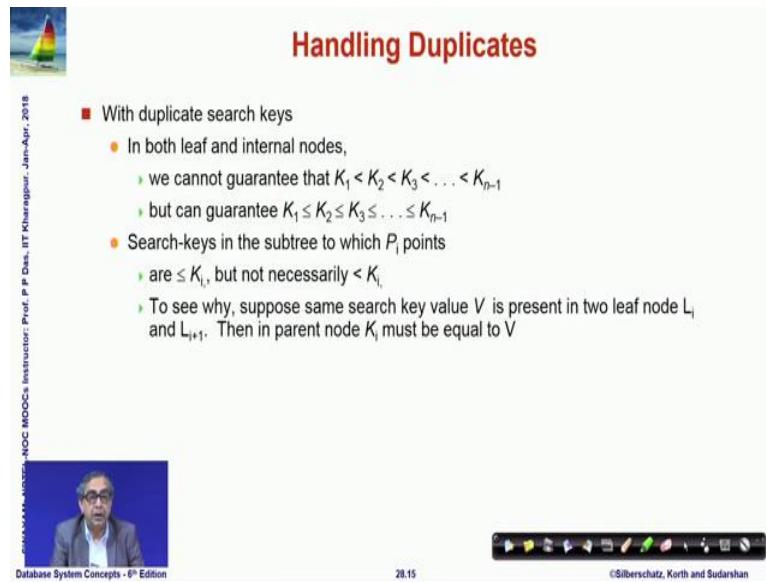
```
graph TD; C["Mozart"] --> C1["Calder"]; C --> C2["Einstein"]; C --> C3["Gold"]; C3 --> L1["Adams | Braud"]; C3 --> L2["Crick | El Said"]; C3 --> L3["Einstein | Gold | Katz | Kau | ..."]
```

**Algorithm Steps:**

- Find record with search-key value  $V$ 
  - $C = \text{root}$
  - While  $C$  is not a leaf node {
    - Let  $i$  be least value s.t.  $V \leq K_i$
    - If no such exists, set  $C = \text{last non-null pointer in } C$
    - Else { if ( $V = K_i$ ) Set  $C = P_{i+1}$  else set  $C = P_i$ }
  - Let  $i$  be least value s.t.  $K_i = V$
  - If there is such a value  $i$ , follow pointer  $P_i$  to the desired record
  - Else no record with search-key value  $k$  exists

So, search should be very simple, because its just an extension of what you did in 2-3-4 trees. So, algorithm is given here I will skip it, because we have already done this in detail.

(Refer Slide Time: 09:13)



The slide title is "Handling Duplicates". It features a logo of a sailboat on the left and a video player window showing a professor on the right. The main content is a list of points:

- With duplicate search keys
  - In both leaf and internal nodes,
    - we cannot guarantee that  $K_1 < K_2 < K_3 < \dots < K_{n-1}$
    - but can guarantee  $K_1 \leq K_2 \leq K_3 \leq \dots \leq K_{n-1}$
  - Search-keys in the subtree to which  $P_i$  points
    - are  $\leq K_i$ , but not necessarily  $< K_i$
  - To see why, suppose same search key value  $V$  is present in two leaf node  $L_i$  and  $L_{i+1}$ . Then in parent node  $K_i$  must be equal to  $V$

Now, what we introduced I started saying that there are no duplicates. So, the keys follow strict ordering, but the whole assumption will also hold good, if you allow the equality between the consecutive keys, but only difference is there could be multiple keys which are all equal; and if that happens then you have to use the same key value

present at the two leaf nodes and the parent will also have the same leaf node same value.

(Refer Slide Time: 09:43)



## Handling Duplicates

■ We modify find procedure as follows

- traverse  $P_i$  even if  $V = K_i$
- As soon as we reach a leaf node  $C$  check if  $C$  has only search key values less than  $V$ 
  - if so set  $C = \text{right sibling of } C$  before checking whether  $C$  contains  $V$

■ Procedure printAll

- uses modified find procedure to find first occurrence of  $V$
- Traverse through consecutive leaves to find all occurrences of  $V$

\*\* Errata note: modified find procedure missing in first printing of 8<sup>th</sup> edition

Database System Concepts - 8<sup>th</sup> Edition      28.16      ©Silberschatz, Korth and Sudarshan

So, for doing in the case of such duplicates will have to a little bit modify the procedure for doing the search and say printing all values and so, on. So, you could go through that.

(Refer Slide Time: 09:58)



## Queries on B+-Trees (Cont.)

■ If there are  $K$  search-key values in the file, the height of the tree is no more than  $\lceil \log_{n/2}(K) \rceil$

■ A node is generally the same size as a disk block, typically 4 kilobytes

- and  $n$  is typically around 100 (40 bytes per index entry)

■ With 1 million search key values and  $n = 100$ 

- at most  $\log_{50}(1,000,000) = 4$  nodes are accessed in a lookup

■ Contrast this with a balanced binary tree with 1 million search key values — around 20 nodes are accessed in a lookup

- above difference is significant since every node access may need a disk I/O, costing around 20 milliseconds



Database System Concepts - 8<sup>th</sup> Edition      28.17      ©Silberschatz, Korth and Sudarshan

So, if there is a key search-key values in the file, then let us see what the cost is coming to actually, then the height of the tree is not more than  $\log K_n / 2$ . So, if we say that the every node. So, how large would be the node. Now again I would remind you that we are

moving from 2-3-4 tree, which was a in memory data structure to a external data structure. So, our main cost is a disk access. So, what would you like to make this node size, if we make the node size too small, then there will be too many nodes and every node will have to be accessed? So, as you can see this is  $\log n/2$ .

So, we benefit by making  $n$  larger, larger the  $n$  this log value or the height will be less, but can I make  $n$  arbitrary large then  $n$  will not fit into one disk block. So, it would it cannot be accessed in one fetch from the disk to the memory. So, we would typically like to make it is customary to make the node as the same size as the disk block, which is typically say 4 kilobyte or 8 kilobyte like that and therefore, the if that is a size then it the  $n$  will be typically around 100, because if 4 kilobytes is a is a total space and if I assume that 40 bytes per index entry, which is very typical, then  $n$  would be about 100.

So, if I assume that my index file has actually 1 million search key values to look for, then I will need 1 million to the base 100 by 250. So, 1 million  $\log 1$  million to the base 50 which is approximately 4 node accesses in a lookup table. So, that is amazingly fast if you contrast this with binary balanced binary tree which will be  $\log 1$  million to the base 2; which would be about 20 nodes accesses 20 disk accesses for this lookup. So, this is the core reason that B + trees are preferred and with this if even, when you have couple of million records in a in a table you can actually manage with a very small number of node accesses for the lookup, which makes the realization of algorithms possible in the next couple of slides.

(Refer Slide Time: 12:23)

**Updates on B<sup>+</sup>-Trees: Insertion**

1. Find the leaf node in which the search-key value would appear
2. If the search-key value is already present in the leaf node
  - 1. Add record to the file
  - 2. If necessary add a pointer to the bucket
3. If the search-key value is not present, then
  - 1. Add the record to the main file (and create a bucket if necessary)
  - 2. If there is room in the leaf node, insert (key-value, pointer) pair in the leaf node
  - 3. Otherwise, split the node (along with the new (key-value, pointer) entry) as discussed in the next slide

CHAKRABORTY, NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur  
Database System Concepts - 8<sup>th</sup> Edition

28.18 ©Silberschatz, Korth and Sudarshan

I have discussed about how to update B + trees talked about the insertion and the deletion process. I will skip them in the presentation, now because as we have discussed the process of insertion in depth in terms of the 2-3-4 tree the only difference here is that this is in a generalized framework, but follows exactly the same idea of node splitting and keeping in mind that in case of 2-3-4 tree you move from 2 to 3 and 3 to 4 node here. All that you will have to remember is you always make sure that you have every node half filled, because  $n / 2$  is a minimum requirement.

So, you keep on inserting in a node till it becomes full, when it becomes full you cannot insert any more you divide it and split it into two nodes. So, that each one of them become at least half filled and that is the simple logic and rest of it you can figured out by following on the 2-3-4 tree insertion. So, this is the first algorithm.

(Refer Slide Time: 13:33)

The slide title is "Updates on B+-Trees: Insertion (Cont.)". It features a small sailboat icon in the top left and a photo of the instructor in the bottom left. The content is organized into sections:

- Splitting a leaf node:
  - take the  $n$  (search-key value, pointer) pairs (including the one being inserted) in sorted order. Place the first  $\lceil n/2 \rceil$  in the original node, and the rest in a new node
  - let the new node be  $p$ , and let  $k$  be the least key value in  $p$ . Insert  $(k, p)$  in the parent of the node being split
  - If the parent is full, split it and propagate the split further up
- Splitting of nodes proceeds upwards till a node that is not full is found
  - In the worst case the root node may be split increasing the height of the tree by 1

Diagram illustrating the splitting of a leaf node:

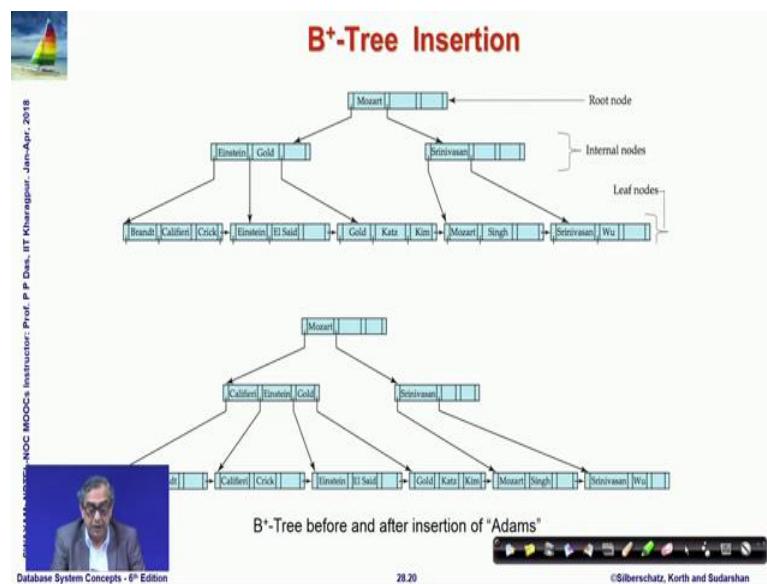
```
graph LR; A[Adams, Brandt] --> B[Califieri, Crick];
```

Result of splitting node containing Brandt, Califieri and Crick on inserting Adams  
Next step: insert entry with (Califieri.pointer-to-new-node) into parent

Database System Concepts - 8<sup>th</sup> Edition      28.19      ©Silberschatz, Korth and Sudarshan

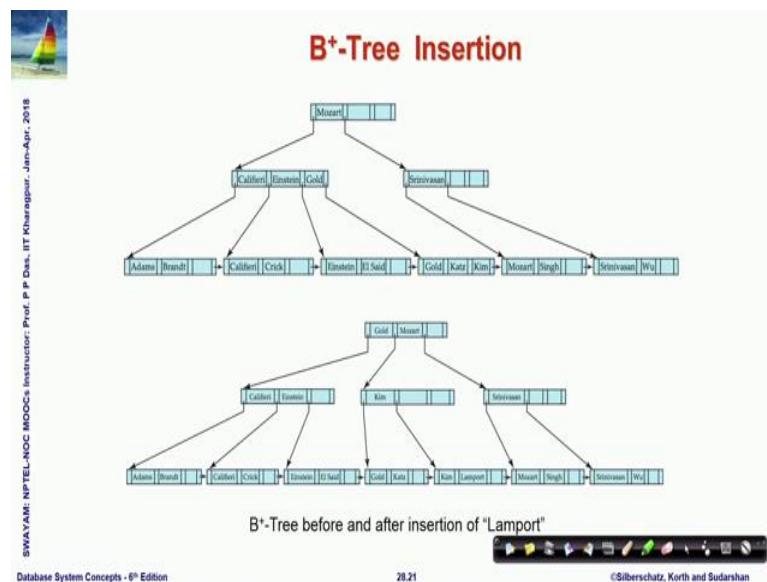
Then we have shown here the strategy to splitting the node, which I have just you know discussed and the same notion of propagating the middle element of the split continues here go to next and here the examples shown in terms of the B + tree.

(Refer Slide Time: 13:47)



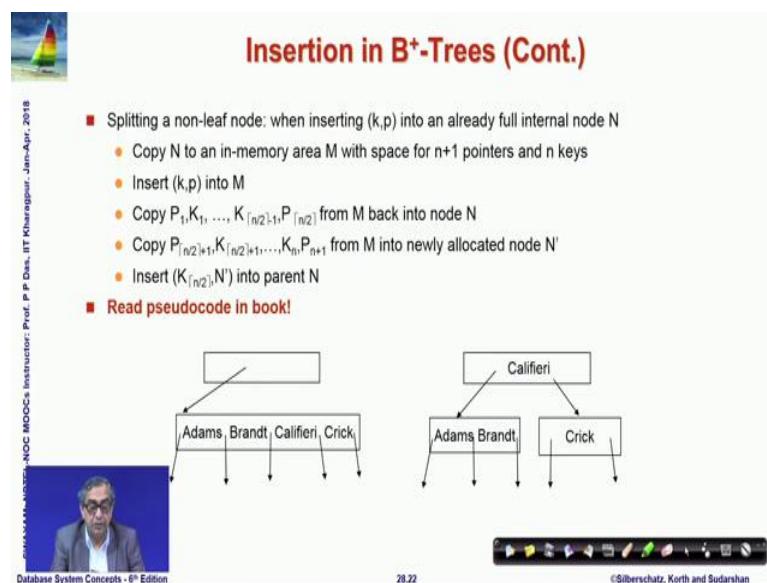
Before and after insertion of a certain key you can go through that and convince yourself.

(Refer Slide Time: 13:57)



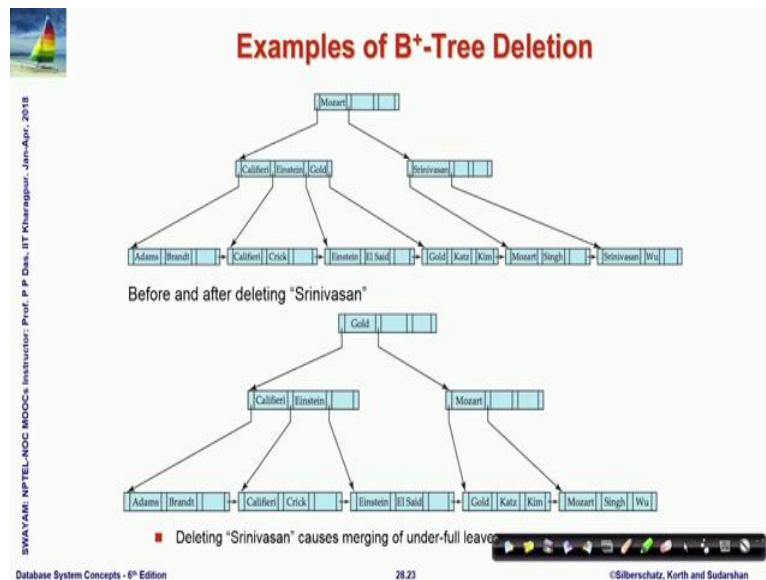
There are some more steps in the algorithm please go through them carefully and try to understand.

(Refer Slide Time: 14:04)



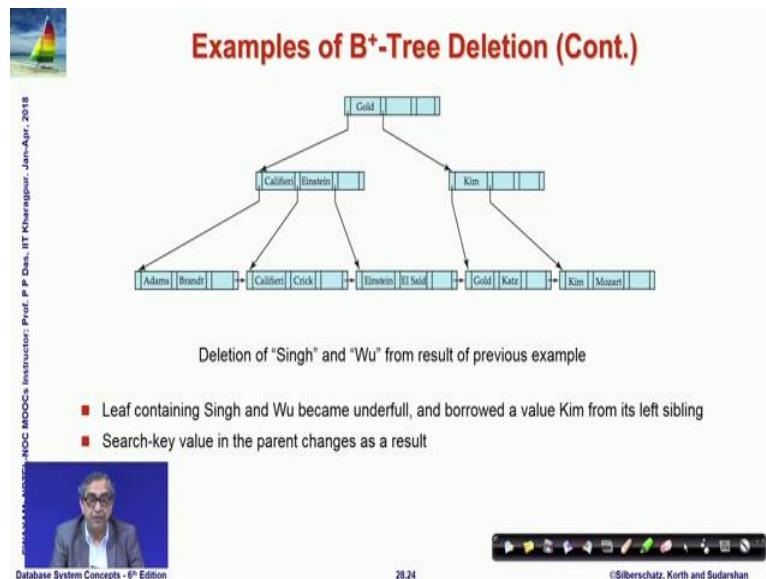
The whole process and then this is the basic algorithm written in a very cryptic pseudocode, I should say you should refer to the book actually to, for and study the whole pseudocode to understand the algorithm better and work through examples as well.

(Refer Slide Time: 14:19)



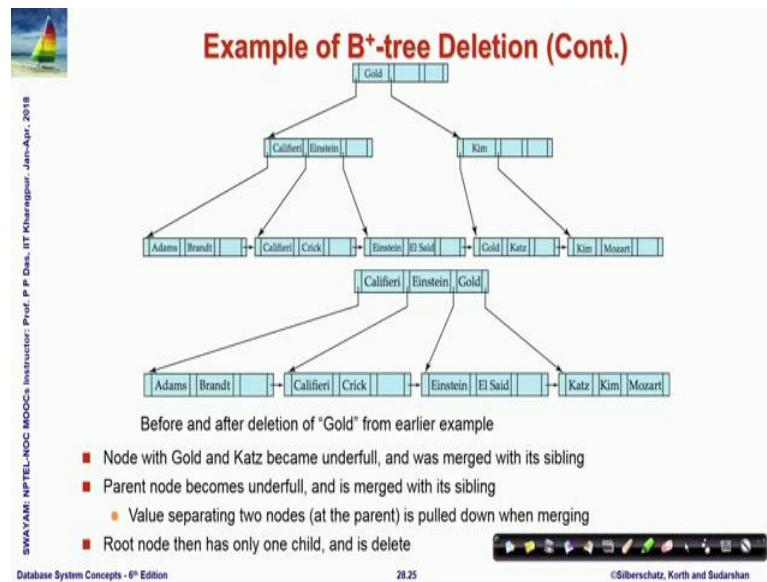
Similarly, examples of deletion in B + tree; so the trees are shown before and after deletion of Srinivasan, then if we delete like that; now in case of in contrast to splitting.

(Refer Slide Time: 14:43)



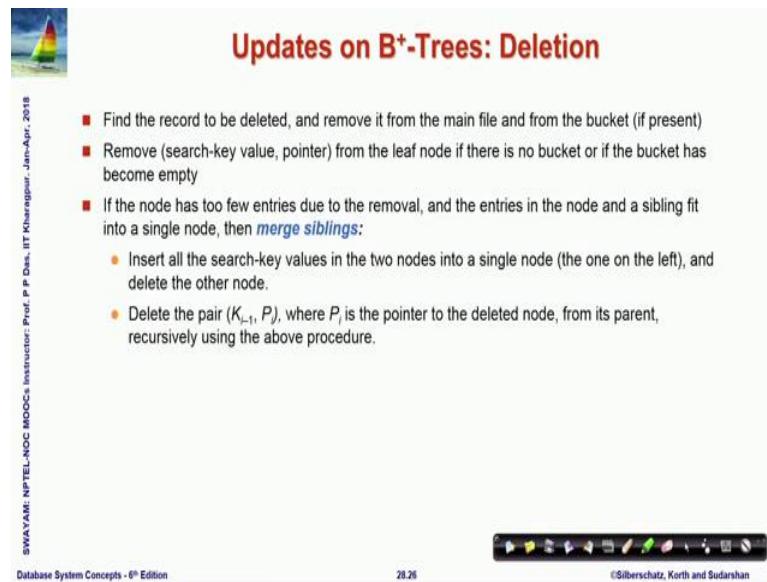
Now I will have merging of nodes which will start happening there are some more steps in the deletion shown here, please go through them and work this out they should not be you should not have any difficulty in understanding them given your background in the 2-3-4 tree.

(Refer Slide Time: 14:56)



So, more steps in the deletion. So, this is the deletion process in terms of algorithmic steps and what you need to do for deletion.

(Refer Slide Time: 15:05)



So, this is all detailed here just. So, B + tree file organization is takes care of the degradation problem.

(Refer Slide Time: 15:12)

The slide has a title 'Updates on B<sup>+</sup>-Trees: Deletion' at the top right. On the left, there is a small video window showing a man speaking. The main content area contains a bulleted list of points:

- Otherwise, if the node has too few entries due to the removal, but the entries in the node and a sibling do not fit into a single node, then **redistribute pointers**:
  - Redistribute the pointers between the node and a sibling such that both have more than the minimum number of entries
  - Update the corresponding search-key value in the parent of the node
- The node deletions may cascade upwards till a node which has  $\lceil n/2 \rceil$  or more pointers is found
- If the root node has only one pointer after deletion, it is deleted and the sole child becomes the root

At the bottom left, it says 'Database System Concepts - 8<sup>th</sup> Edition'. At the bottom right, it shows a progress bar at 28.27 and the copyright notice '©Silberschatz, Korth and Sudarshan'.

In terms of the index files which would have happened, if we were used pure ordered indices like, the index sequential access method for storing the index files. So, that is now taken care of and even the data File degradation problem can also be solved by using B + Tree organization.

(Refer Slide Time: 15:20)

The slide has a title 'B<sup>+</sup>-Tree File Organization' at the top right. On the left, there is a small video window showing a man speaking. The main content area contains a bulleted list of points:

- Index file degradation problem is solved by using B<sup>+</sup>-Tree indices
- Data file degradation problem is solved by using B<sup>+</sup>-Tree File Organization
- The leaf nodes in a B<sup>+</sup>-tree file organization store records, instead of pointers
- Leaf nodes are still required to be half full
  - Since records are larger than pointers, the maximum number of records that can be stored in a leaf node is less than the number of pointers in a non-leaf node
- Insertion and deletion are handled in the same way as insertion and deletion of entries in a B<sup>+</sup>-tree index

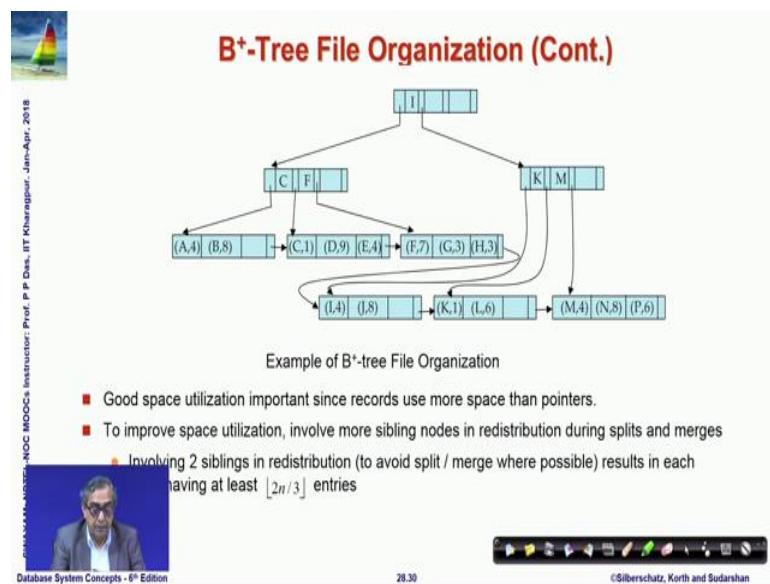
At the bottom left, it says 'Database System Concepts - 8<sup>th</sup> Edition'. At the bottom right, it shows a progress bar at 28.29 and the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, it can be used for both maintaining the the index as well as the actual data file and the leaf nodes in the B + tree file organization stored the records instead of pointers. So, you finally, have the records there and the leaf nodes are still required to be half full

since they are records, but since records are larger than the maximum number of records that can be stored would be less than the number of pointers in a non leaf node insertion and deletions are handled in the same way as in the B + tree index file.

So, here all that we are explaining that. So, far we have not explained the whole B + tree in terms of index file organization and we are saying that you can do the same thing with the data file and only at the leaf level you will have to actually keep the data records for maintenance.

(Refer Slide Time: 16:47)



So, this is showing some instances of the B + tree organization.

(Refer Slide Time: 16:54)

The slide has a header 'Other Issues in Indexing' with a sailboat icon. The main content is a bulleted list under 'Record relocation and secondary indices':

- If a record moves, all secondary indices that store record pointers have to be updated
- Node splits in B+-tree file organizations become very expensive
- Solution: use primary-index search key instead of record pointer in secondary index
  - Extra traversal of primary index to locate record
  - Higher cost for queries, but node splits are cheap
  - Add record-id if primary-index search key is non-unique

On the left, there is a small video window showing a professor. On the right, there is a navigation bar with icons and the text 'Database System Concepts - 8<sup>th</sup> Edition', '28.31', and '©Silberschatz, Korth and Sudarshan'.

So, there is a couple of other issues the record relocation and secondary index, if a record moves all secondary indices that store record pointers will also have to be updated node splits in B + tree file organization is very expensive. So, what we do is? We use primary index search key instead of record pointer in the secondary index. So, in the secondary index we do not actually keep the direct record pointer instead, we keep the search-key of the primary index and we know that the primary index can be very efficiently searched. So, what happens is when in the secondary index when you have been able to actually find that you do not get a pointer directly to the record, but you get the search key through which you can use the primary index and actually go to that.

But with that you get yourself get rid of the requirement of maintaining different secondary index structures and getting into several record relocation problems.

(Refer Slide Time: 18:05)

The slide has a header 'Indexing Strings' with a sailboat icon. The main content lists points under two sections: 'Variable length strings as keys' and 'Prefix compression'. A small video player window shows a man speaking, and the footer includes course information and navigation icons.

- Variable length strings as keys
  - Variable fanout
  - Use space utilization as criterion for splitting, not number of pointers
- Prefix compression
  - Key values at internal nodes can be prefixes of full key
    - ▶ Keep enough characters to distinguish entries in the subtrees separated by the key value
      - E.g. "Silas" and "Silberschatz" can be separated by "Silb"
    - Keys in leaf node can be compressed by sharing common prefixes

CHAKRABORTY, NOC MOOCs Instructor: Prof. P. P. De, IIT Kharagpur - Jan-Apr., 2018  
Database System Concepts - 8<sup>th</sup> Edition

28.32 ©Silberschatz, Korth and Sudarshan

There are your indexing also may need to take care of other issues of string your variable length string could be keys which are variable fan out and so, the general strategy in handling indexing with string is to do a kind of what is known as prefix compression. So, you kind of find out what is the shortest prefix which can distinguish between the strings. So, if you have Silas and Silberschatz then you can easily make out that Silb would be a separating string between these two. So, Silb will match with Silberschatz, but or not will match with the first one. So, you do not need to look beyond that so, we can just keep enough characters to distinguish entries in the subtree separated I by the key values and keys in the leaf node can be compressed by sharing common prefixes.

(Refer Slide Time: 19:12)

**B-Tree Index Files**

- Similar to B+tree, but B-tree allows search-key values to appear only once; eliminates redundant storage of search keys
- Search keys in non-leaf nodes appear nowhere else in the B-tree; an additional pointer field for each search key in a non-leaf node must be included
- Generalized B-tree leaf node

Diagram (a) shows a generalized B-tree leaf node structure:

$P_1$	$K_1$	$P_2$	$\dots$	$P_{n-1}$	$K_{n-1}$	$P_n$
-------	-------	-------	---------	-----------	-----------	-------

(a)

Diagram (b) shows a non-leaf node structure:

$P_1$	$B_1$	$K_1$	$P_2$	$B_2$	$K_2$	$\dots$	$P_{m-1}$	$B_{m-1}$	$K_{m-1}$	$P_m$
-------	-------	-------	-------	-------	-------	---------	-----------	-----------	-----------	-------

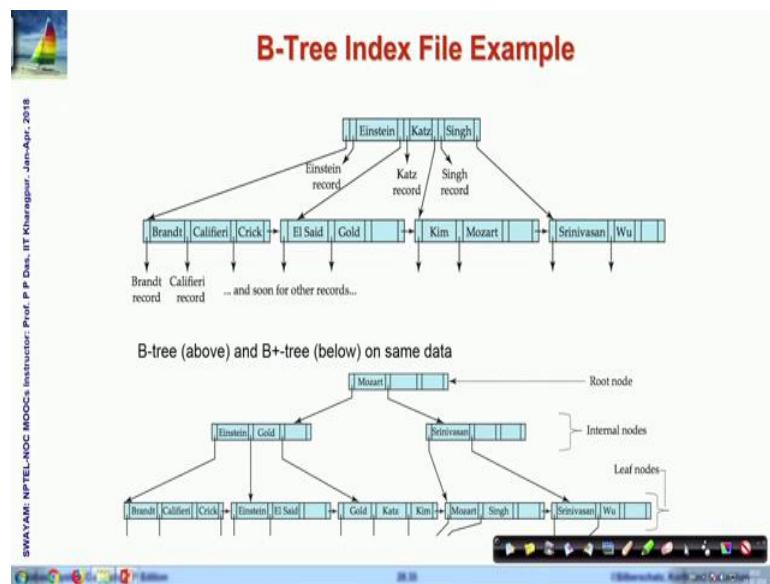
(b)

- Non-leaf node – pointers  $B_i$  are the bucket or file record pointers

So, that is a very common strategy next let us just take a quick look into the B tree index file which is another alternate possibility the basic difference between a B + tree. And B tree allows search key values to appear only once, if you if you remember in the B + tree your search key values where which occurs in an internal load keeps on occurring at multiple node levels also B+, B tree does not allow that the search key non leaf nodes appear nowhere else in the B tree.

So, if it does not then naturally the question is when where will the actual record value we found out for this key. So, what you do is in the node itself you introduce another field after along with the key which is the; pointer to the actual record. So, as you can as you can see here let us get back. So, as you can see this is this was a general structure of the B + tree node. And, now what we are doing is we are putting in separate pointers along with the key which will actually maintain the data for that key which will be pointers to the actual record, because this earlier in B + tree all records. Finally, appear in terms of the leaf level nodes only they are their pointers come in the leaf level whereas, here the there is no repetition of the search key along the structure. So, they come wherever there.

(Refer Slide Time: 20:43)



So, let me just show you an example. So, if you look into this carefully. So, this is what you have seen is a B + tree. So, you can see that Mozart happened here, it also happened here and this is the leaf level. So, from here actually you get pointers to the; to the record for Mozart.

Similarly, Einstein happens here and it happens here Srinivasan happens here in. So, there are multiple times there happening this in contrast is a B tree representation where Einstein, if it happens then alongside with it the pointer to the Einstein's record exists, if brands happen here along with it the brands record exists and Einstein would not happen anywhere else in the tree. So, you do not have the second instance of the Einstein or this instance of the Mozart in the B tree. So, naturally that is the basic optimization that B tree does?

(Refer Slide Time: 21:48)

The slide has a title 'B-Tree Index Files (Cont.)' at the top right. On the left, there is a small sailboat icon and some vertical text: 'CHAKRABORTY NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. The main content is organized into three sections with bullet points:

- Advantages of B-Tree indices:
  - May use less tree nodes than a corresponding B<sup>+</sup>-Tree
  - Sometimes possible to find search-key value before reaching leaf node
- Disadvantages of B-Tree indices:
  - Only small fraction of all search-key values are found early
  - Non-leaf nodes are larger, so fan-out is reduced. Thus, B-Trees typically have greater depth than corresponding B<sup>+</sup>-Tree
  - Insertion and deletion more complicated than in B<sup>+</sup>-Trees
  - Implementation is harder than B<sup>+</sup>-Trees
- Typically, advantages of B-Trees do not outweigh disadvantages

At the bottom left is a video thumbnail of a professor, and at the bottom right are navigation icons and the text 'Database System Concepts - 8<sup>th</sup> Edition', '28.36', and '©Silberschatz, Korth and Sudarshan'.

. So, it is advantages it may use less notes than the corresponding B + tree sometimes it is possible to find the search key value even before reaching the leaf node. So, search could be efficient, but it does have a lot of disadvantages, because what happens is only small fraction of all search key values are actually found early non leaf nodes are larger.

Now, because you have pointers to the data as well, so the fan out gets reduced which means that the number of children you can have is gets reduced. So, though you are expecting to get a benefit, because you are not having to go to the leaf every time, but you pay off because your fan out gets less. So, if your fan out get less naturally the tree has a greater depth. Now, because you can you are fanning out less number of children at every node. So, it has a greater depth. So, eventually your cost increases the naturally the deletions insertions are more complicated than in B + tree and implementation is more difficult.

So, typically the advantages of B tree do not outweigh the disadvantages.

(Refer Slide Time: 23:01)

**Module Summary**

- Understood the design of B+-Tree Index Files in depth for database persistent store
- Familiarized with B-Tree Index Files

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

28.37

©Silberschatz, Korth and Sudarshan

So, it is not very frequent that you will use B trees, but they are used at times, but that is not a very common thing and we stick to B + tree for both of the data file as well as index file storage. So, in this module you have understood the design of B + index B + tree index files in depth for the purpose of data base persistent store and I would again remind you that whole discussion of how B + tree is organized and how operations of access insert delete are done in B + tree. I have introduced them in keeping in parallel with the simpler in memory data structure for this which is a 2-3-4 tree discussed in the last module.

So, while going through the insertion deletion processes of B+ tree, if you have difficulty following I would request that you go back to the 2-3-4 tree that is kind the simplest situation that can have that can occur and understand that and then you come back to the specific points in the B + tree algorithm and also always keep in mind. When you refer to 2-3-4 tree for understanding also always keep in mind that in case of B + tree all node types are same and the basic requirement is every node must be at least half full all the time except of course, for the root and in addition we have also familiarized with B tree and reason that B tree possibly is not a very powerful is not powerful enough it does not give enough advantages so, that to we would like to use it in place of B + tree.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 29**  
**Indexing and Hashing/4 : Hashing**

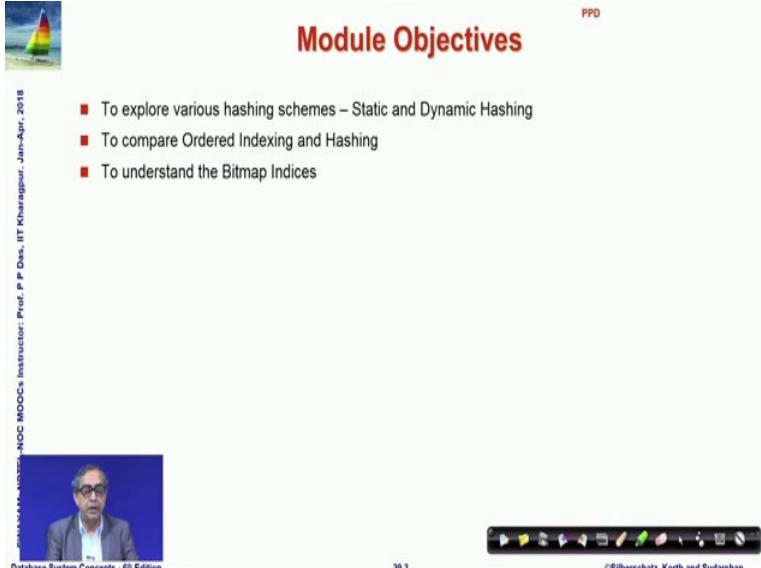
Welcome to module 29 of Database Management Systems; we have been talking about indexing and hashing and this is a fourth in the series.

(Refer Slide Time: 00:26)

The slide has a header 'Module Recap' in red. On the left, there is a small image of a sailboat on water. The footer contains copyright information: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr., 2018'. The bottom right corner features the 'Silberschatz, Korth and Sudarshan' logo.

In the previous 3 we have talked about different aspects of indexing and specifically in the last module, we have introduced the most powerful data structure B+ tree for index files.

(Refer Slide Time: 00:39)

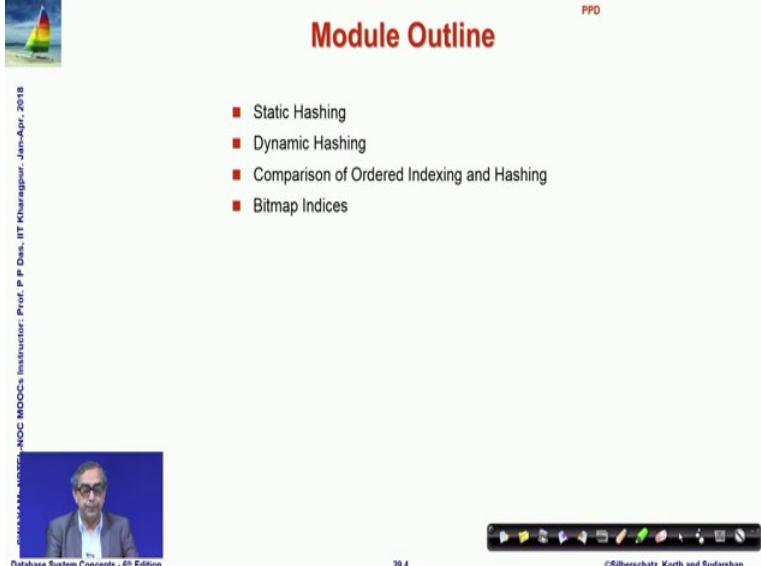


This slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. On the left edge, there is vertical text: "CHAKRABARTI, NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018". At the bottom left is a video frame showing a man with glasses. The bottom right contains the text "Database System Concepts - 8<sup>th</sup> Edition", "29.3", and "©Silberschatz, Korth and Sudarshan". A decorative toolbar is at the bottom.

- To explore various hashing schemes – Static and Dynamic Hashing
- To compare Ordered Indexing and Hashing
- To understand the Bitmap Indices

In this module, we will take a look into a explore into various hashing schemes for achieving the similar targets we will look at static and dynamic hashing. And we will then compare it between the ordered indexing that we have discussed already and hashing and we will also understand about what are called bitmap indices.

(Refer Slide Time: 01:03)



This slide is titled "Module Outline" in red at the top right. It features a small sailboat icon in the top left corner. On the left edge, there is vertical text: "CHAKRABARTI, NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018". At the bottom left is a video frame showing a man with glasses. The bottom right contains the text "Database System Concepts - 8<sup>th</sup> Edition", "29.4", and "©Silberschatz, Korth and Sudarshan". A decorative toolbar is at the bottom.

- Static Hashing
- Dynamic Hashing
- Comparison of Ordered Indexing and Hashing
- Bitmap Indices

So, these are the module outline.

(Refer Slide Time: 01:07)

**Static Hashing**

- A **bucket** is a unit of storage containing one or more records (a bucket is typically a disk block)
- In a **hash file organization** we obtain the bucket of a record directly from its search-key value using a **hash function**
- Hash function  $h$  is a function from the set of all search-key values  $K$  to the set of all bucket addresses  $B$
- Hash function is used to locate records for access, insertion as well as deletion
- Records with different search-key values may be mapped to the same bucket; thus entire bucket has to be searched sequentially to locate a record

CHAKRABORTY, NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

29.6

©Silberschatz, Korth and Sudarshan

So, static hashing I am assuming that all of you know about basic concept of hashing. So, it what it does a there is a bucket is a unit of storage containing one or more records. So, that is the basic logical concept typically in physical terms a bucket can be a disk block. So, a hash file organization obtains we in a hash file organization; we attempt to obtain a bucket for a record directly from its search key value using a hash function.

So, this is where it becomes very different compared to the ordered indexing for which we saw all this LSM method and the B+ tree where we went through different index structure, but here we want to make use of a mathematical hash function. So, that given the key ideally I should be able to get the bucket in which that particular record containing the key exists that is the requirement.

So, hash function  $h$  is a function from the set of all search key values  $K$  to the set of all bucket addresses  $B$ . So, it is a mathematical function and it is used to locate the records for access insert as well as delete records with different search key values may be mapped to the same bucket right this is not what ideally we wanted, but it is possible there is a entire bucket has to be searched sequentially once you reach there to look at a record, we can make use of other techniques there we will come to that.

(Refer Slide Time: 02:33)

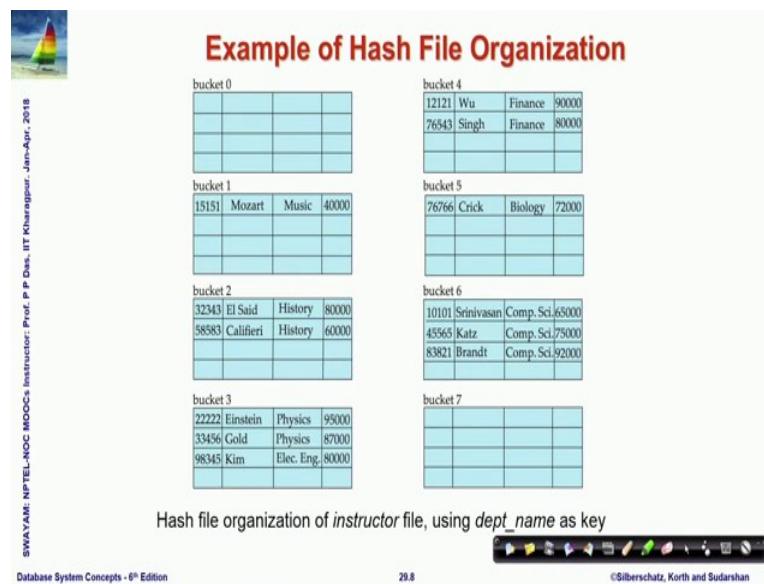
The slide has a header 'Example of Hash File Organization' with a sailboat icon. The main content discusses hash file organization for an 'instructor' file using 'dept\_name' as a key. It lists four points: 1. There are 10 buckets. 2. The binary representation of the  $i^{\text{th}}$  character is assumed to be the integer  $i$ . 3. The hash function returns the sum of the binary representations of the characters modulo 10. 4. Examples include  $h(\text{Music}) = 1$ ,  $h(\text{History}) = 2$ ,  $h(\text{Physics}) = 3$ , and  $h(\text{Elec. Eng.}) = 3$ . The footer includes a photo of a professor, the title 'Database System Concepts - 8<sup>th</sup> Edition', page number '29.7', and copyright '©Silberschatz, Korth and Sudarshan'.

So, let us take a quick example hash file organization of an instructor file using say department named as key. So, we need to design a hash function. So, let us assume that on the address space B we have 10 buckets. So, every bucket is designated by a serial number bucket 0 to bucket 9 and we take department name is a key. So, it is a character string.

So, we take the binary representation of the  $i^{\text{th}}$  character and assume it to be the integer I simply every character you take its binary representation and think as if it is an integer. And then as a hash function we add these integer values of binary representations modulo 10. So, M hash value of music we take the binary representation of m which is the ascii code of M capital M; then add the ascii code of u the lower case u and so, on and do that modulo 10 and we get a value which is 1.

So, naturally since we are doing modulo 10 which is the number of buckets here. So, we will get a result for the hash function which is between 0 to 9 which, is a bucket address where it is expected.

(Refer Slide Time: 03:51)



So, here we are showing an example. So, you can see in the earlier slide we are showing music is 1 history is 2 physics and electrical engineering both are hash value 3.

So, you can see here in bucket 2 since history has value 2. So, those records El Said and Califieri records go to bucket 2 whereas, physics and electrical engineering both have hash value 3. So, that Einstein gold and Kim came all go to the bucket 3 similarly it happens with the other buckets as well not all buckets are shown here shown only 8 buckets are shown, but in this way we can actually directly map them to the buckets.

(Refer Slide Time: 04:33)

Hash Functions

- Worst hash function maps all search-key values to the same bucket; this makes access time proportional to the number of search-key values in the file
- An ideal hash function is **uniform**, i.e., each bucket is assigned the same number of search-key values from the set of *all* possible values
- Ideal hash function is **random**, so each bucket will have the same number of records assigned to it irrespective of the *actual distribution* of search-key values in the file
- Typical hash functions perform computation on the internal binary representation of the search-key
  - For example, for a string search-key, the binary representations of all the characters in the string could be added and the sum modulo the number of buckets could be returned

Database System Concepts - 8<sup>th</sup> Edition

And; so, such a hash function would be really useful now the question is a it is a mathematical function; so, how good or how bad it is. So, we will say that the worst possible hash function is one which maps all key values to the same bucket. So, that everything will have to within them serially; so, that is of no use.

So, the ideal one would be which will distribute the different search keys values in different buckets in an uniform manner to the from the set of all possible values. So, that would be that will be nice to have and ideal would be that if the hash function is random which means that. So, that each bucket will I mean it will generate from the key value it will generate the bucket number, it will generate the bucket address in kind of a random manner.

So, that in a random phenomena; so, that irrespective of what kind of actual distribution the search keys may have the buckets over which the distribute will be more or less the same. So, every bucket will have same number of records things will be balanced.

A typical hash function performs computation on the internal binary representation of the search key that is the basic that that is the one that you have just seen. So, if it is a string then you treat the characters as they are binary representations as integer do some modulo a number exactly what we did in the last case.

(Refer Slide Time: 05:11)

**Handling of Bucket Overflows**

- Bucket overflow can occur because of
  - Insufficient buckets
  - Skew in distribution of records. This can occur due to two reasons:
    - ▶ multiple records have same search-key value
    - ▶ chosen hash function produces non-uniform distribution of key values
- Although the probability of bucket overflow can be reduced, it cannot be eliminated
  - it is handled by using *overflow buckets*

Course Name: NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr-2018

Database System Concepts - 8<sup>th</sup> Edition

29.10

©Silberschatz, Korth and Sudarshan

Now the question is the buckets have a certain size we said that a bucket is a disk block. So, a bucket can overflow because there may not be enough sufficient buckets to keep all the records. So, it will not fit in or your distribution could be skewed. So, there may be many buckets where there are lot of space left, but some buckets may have a too many records coming on to it because of the behavior of the hash function. So, that multiple records have the same key value or chosen hash function produces non uniform distribution and so, on.

So, if that happens then the probability of bucket flow; bucket overflow will happen and we can try to reduce that, but it cannot be eliminated. So, all that you do is to have overflow bucket which is nothing, but having other buckets connected to this target bucket in a chain.

(Refer Slide Time: 07:04)

## Handling of Bucket Overflows (Cont.)

**■ Overflow chaining** – the overflow buckets of a given bucket are chained together in a linked list

**■ Above scheme is called **closed hashing****

- An alternative, called **open hashing**, which does not use overflow buckets, is not suitable for database applications

```

graph TD
    bucket0[bucket 0] --- bucket1[bucket 1]
    bucket1 --- overflow1[overflow buckets for bucket 1]
    bucket1 --- overflow2[overflow buckets for bucket 1]
    bucket2[bucket 2]
    bucket3[bucket 3]
  
```

DR. MANOHAR - NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

29.11 ©Silberschatz, Korth and Sudarshan

So, this is called a overflow chaining as you can see there are 4 buckets shown here and bucket 1 we are saying showing are connected with other two buckets which are the overflow buckets for bucket 1. So, that this kind of a scheme is called closed hashing there is an alternate scheme called open hashing, which does not use a bucket overflow and, but it is not therefore, suitable for database applications and we will not discuss it here.

(Refer Slide Time: 07:31)

The slide has a header 'Hash Indices' in red. On the left is a small sailboat icon. The main content is a bulleted list:

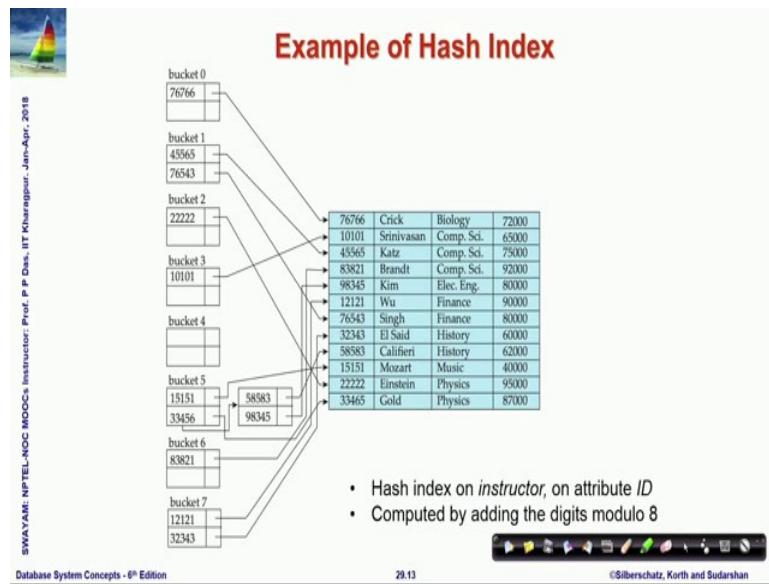
- Hashing can be used not only for file organization, but also for index-structure creation
- A **hash index** organizes the search keys, with their associated record pointers, into a hash file structure
- Strictly speaking, hash indices are always secondary indices
  - if the file itself is organized using hashing, a separate primary hash index on it using the same search-key is unnecessary
  - However, we use the term hash index to refer to both secondary index structures and hash organized files

At the bottom left is vertical text: SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018. At the bottom center: Database System Concepts - 8<sup>th</sup> Edition, 29.12. At the bottom right: ©Silberschatz, Korth and Sudarshan.

So, hash indices can be used only for file organization I mean not only for file organization, but they can also be used for indexed structure creation like we did for B plus tree we can use the hash indices for index structure also. So, hash index organizes the search keys with their associated record pointers into a hash file structure exactly in the same way its hashing otherwise.

So, but the you can note that the hash indices are always kind of secondary indices because if a file itself is organized using hashing; then a separate primary hash index on it using the same search keys are necessary. Because if if you are talking about primary hash indexing then it will mean that you are taking the primary search key and creating a hash index on that, but if the file is hash created by hash indexing then that already exists. So, anything that you create in terms of indexing is basically a secondary indexing structure in a hash organized file.

(Refer Slide Time: 08:38)



So, this is kind of hash indexing example. So, here I am showing the hash indexing with the ID of this table and the index is computed by adding the digits modulo 8 assuming that there are 8 buckets. So, if you take; so, if you look at bucket 0 then the key that has gone there is 76766 which is  $7 + 6; 13, 20 + 6; 26 + 6 \equiv 32 \pmod{8}$  is 0.

So, it goes into bucket 0 it happens that way if, but if we look into bucket 4 you will find that the 4 IDs actually all have this value 5 under the hash function. So, they all need to go to this bucket and therefore, but the bucket size assumed here is just 2. So, after the 2 indices have gone in there a overflow chain is created and another overflow bucket is used to keep the next two IDs there. So, this is how a hash index can be created.

(Refer Slide Time: 09:45)

**Deficiencies of Static Hashing**

- In static hashing, function  $h$  maps search-key values to a fixed set of  $B$  of bucket addresses.  
Databases grow or shrink with time
  - If initial number of buckets is too small, and file grows, performance will degrade due to too much overflows
  - If space is allocated for anticipated growth, a significant amount of space will be wasted initially (and buckets will be underfull).
  - If database shrinks, again space will be wasted
- One solution: periodic re-organization of the file with a new hash function
  - Expensive, disrupts normal operations
- *Better solution:* allow the number of buckets to be modified dynamically

CHAKRABORTY, NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
29.14 ©Silberschatz, Korth and Sudarshan

Now, this is this kind of a scheme where you start with a fixed number of buckets and then you design a hashing function which maps the search key values to this fixed set of buckets is known as a static hashing it is static because you start with a fixed number of buckets.

So, yeah naturally the question is what should be this value of  $B$  the number of buckets. Now if it is initially too small then the file keeps on growing the performance will degrade because you will have too many overflow chains and if the all advantages of having done the hashing will get lost. On the other hand if you take a too large a  $B$  then you will unnecessarily allocate a lot of space anticipating growth, but it may take a very significant amount of time to utilize that that space or also it is possible that it the database at certain point of time grew to a large size and then it started shrinking and then again space will get wasted.

So, static hashing has these limitations. So, naturally what you will have to do is to periodically reorganize the file with a new hash function which is certainly very expensive because it changes the positions of all records. So, it disrupts the normal operation; so, it would be better if we could allow to change the number of buckets to be changed dynamically at the as the database grows. So, if the database grows it can use more and more buckets and if we could adjust this in the hashing scheme inherently; then

it will certainly be better as a solution. So, that gives rise to what is known as dynamic hashing.

(Refer Slide Time: 11:30)

The slide has a title 'Dynamic Hashing' in red at the top right. On the left is a small sailboat icon. The main content is a bulleted list under three main points:

- Good for database that grows and shrinks in size
- Allows the hash function to be modified dynamically
- **Extendable hashing** – one form of dynamic hashing
  - Hash function generates values over a large range — typically  $b$ -bit integers, with  $b = 32$
  - At any time use only a prefix of the hash function to index into a table of bucket addresses
  - Let the length of the prefix be  $i$  bits,  $0 \leq i \leq 32$ 
    - ▶ Bucket address table size =  $2^i$ . Initially  $i = 0$
    - ▶ Value of  $i$  grows and shrinks as the size of the database grows and shrinks
    - Multiple entries in the bucket address table may point to a bucket (why?)
      - ▶ Thus, actual number of buckets is  $< 2^i$
      - ▶ The number of buckets also changes dynamically due to coalescing and splitting of buckets

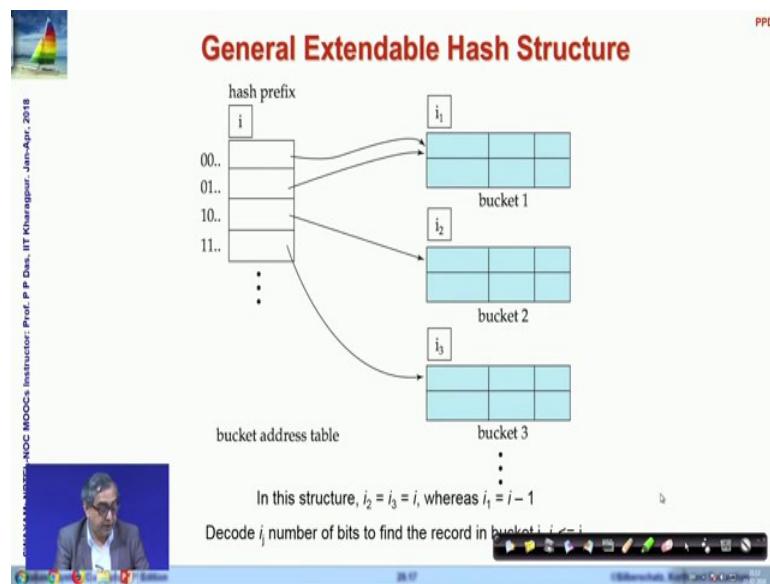
So, it is certainly good for databases that regularly grows and shrinks in size, allows the hash function to be modified dynamically. Of the different dynamic hashing schemes I will discuss the extendable hashing which is a very popular scheme. So, let us see what how it works. So, it at the hash function will generate the value over a large range say typically a B bit integer say 32 bit integer now.

So, what you have is you have generated a value which is hash value which is say over 32 bits, but what you do at any time you use only a prefix of that; you only use a part frontal part of that to index the table to the bucket address and the length of that prefix is  $i$  bits; then naturally it could be at least theoretically it could be 0 that is you do not use any prefix and it could be up to that you use all the prefixes.

And so, therefore, if you are using  $i$  bits then the bucket address table the possible you know bucket addresses that you could have is  $2^i$  initially you keep that as 0.

So, then the address table will actually point to different buckets let us start moving to an example and see what is happening.

(Refer Slide Time: 12:57)



So, this is the general scheme. So, you have a hash prefix which is using  $i$  number of bits and therefore, different values of  $i$  number of bits. So, there will be  $2^i$  entries naturally you have different buckets here, but you may not actually have all  $2^i$  buckets you may have less than that as it is shown here that bucket 2 and bucket 3 exist, but bucket 1 is a holder for both this prefix 0 0 as well as prefix 0 1.

So, and on top of every bucket you have a kind of bucket depth given. So, it is a number of bits that you need to explore in the representation in the; so, that you can distinguish the different records of that bucket.

Naturally the maximum value of any of these  $i$  is the  $i$  here, but it could be less than that. So, I am sure this probably is not making much sense immediately. So, let me move to my detailed discussion.

(Refer Slide Time: 14:19)

**Use of Extendable Hash Structure**

- Each bucket  $j$  stores a value  $i_j$ 
  - All the entries that point to the same bucket have the same values on the first  $i_j$  bits
- To locate the bucket containing search-key  $K_j$ 
  - Compute  $h(K_j) = X$
  - Use the first  $i$  high order bits of  $X$  as a displacement into bucket address table, and follow the pointer to appropriate bucket
- To insert a record with search-key value  $K_j$ 
  - Follow same procedure as look-up and locate the bucket, say  $j$
  - If there is room in the bucket  $j$  insert record in the bucket
  - Else the bucket must be split and insertion re-attempted (next slide)
    - ▶ Overflow buckets used instead in some cases (will see shortly)

CHAMAKAM NOETHER NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

29.18

©Silberschatz, Korth and Sudarshan

So, what I was saying that each bucket  $j$  stores a value  $i_j$ . So, this is the all the entries that point to this bucket has the same value on the first  $i_j$  bits. So, this  $i_j$  bits are identical. So, all of them have come to this bucket. So, how do you look at the bucket that contains the search key  $K_j$ (subscript)? So, it compute the hash function which is  $X$  user prefix  $i$  bits of  $X$  as a displacement into the buckets address table and follow the pointer to the appropriate bucket.  $h_i(K_j)=X$

Now if I have to insert a record with a search key  $K_j$ (subscript); you will follow that same procedure as a lookup and look at the bucket  $j$  and then you will have to look for making some space. So, let me do something.

(Refer Slide Time: 15:15)



## Deletion in Extendable Hash Structure

- To delete a key value,
  - locate it in its bucket and remove it
  - The bucket itself can be removed if it becomes empty (with appropriate updates to the bucket address table)
  - Coalescing of buckets can be done (can coalesce only with a "*buddy*" bucket having same value of  $i_j$  and same  $i_j - 1$  prefix, if it is present)
  - Decreasing bucket address table size is also possible
    - Note: decreasing bucket address table size is an expensive operation and should be done only if number of buckets becomes much smaller than the size of the table



Navigation icons: back, forward, search, etc.

Let me before going through this statement of the algorithm.

(Refer Slide Time: 15:20)



## Use of Extendable Hash Structure: Example

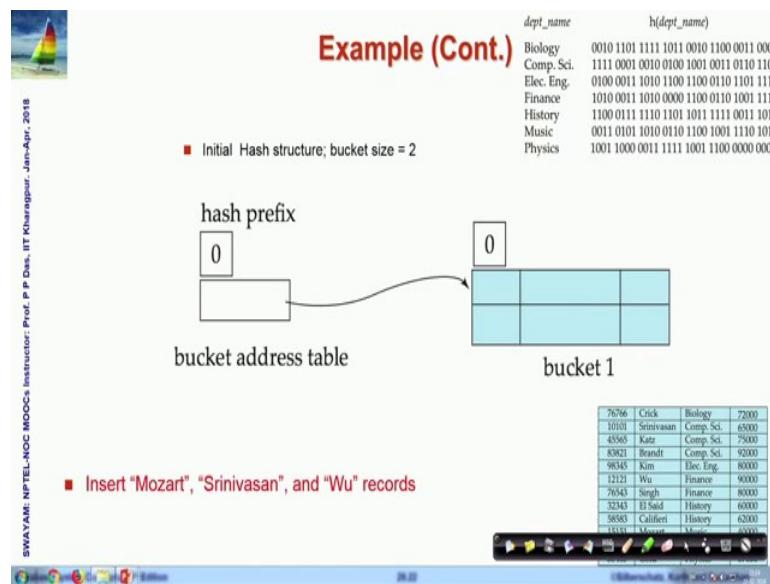
<i>dept_name</i>	$h(dept\_name)$
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001



Let me just go through an example first and we can come back to this formal statement.

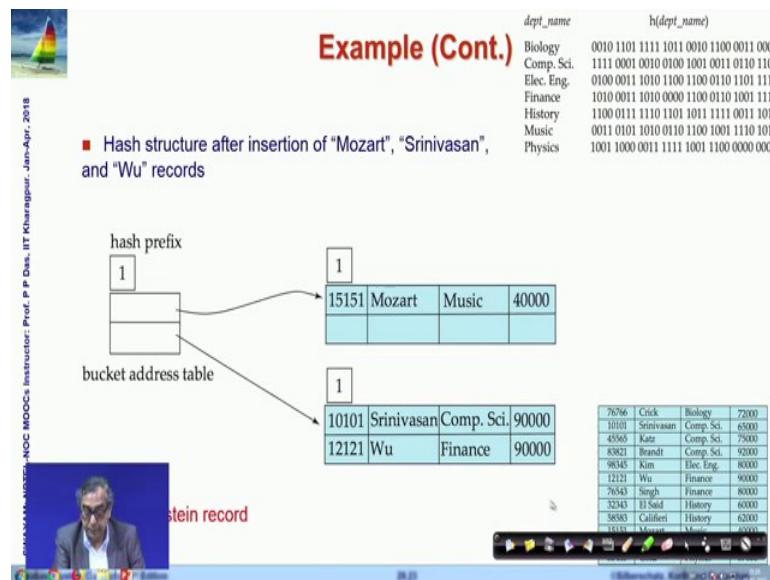
So, what we are trying to do is there is the department names which we are using as a key to do this hashing index and they are represented in terms of the binary representation. So, this is this is the hash of that department name and hashed into you can you can easily see this is 1, 2, 3, 4, 5, 6, 7, 8. So, this is hashed into 32 bit number.

(Refer Slide Time: 16:01)



Now, what do we do? So, initially we start with. So, this is all the different hash values that you can see I am sorry this is all the different hash values and this is the table that I need to actually represent. So, initially there is nothing. So, I try to I will try to insert Mozart Srinivasan and these 3 records here. So, let me try that.

(Refer Slide Time: 16:33)



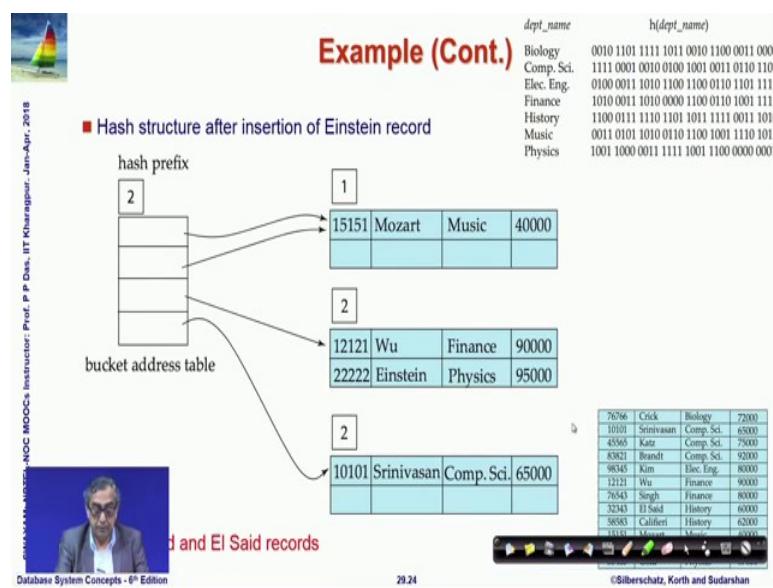
So, if I look at Mozart then Mozart is from the department of music. So, and Srinivasan is from computer science Wu is from finance. So, let us look at this. So, Mozart is from music Srinivasan is from computer science and Wu is from finance. So, these are the 3

now if we look into the prefixes of their hash values; you can see that their hash values are 1 1 and for music it is 0 right.

So, if I use a hash prefix which has just one bit and naturally therefore, I have two entries 0 and 1 then music with the value 0 maps to this bucket where I entered the record for music. And computer science and finance the records corresponding to them has a hash value prefix 1. So, they both map to discipline this is how it can get started. So, you find out while inserting you find out where is Mozart and based on that you create this.

Now, let us try to insert Einstein.

(Refer Slide Time: 18:09)



So, to insert Einstein what do we find? So, what all we already have? We have music, we have computer science, we have finance and now Einstein comes in Einstein is from physics. So, what would happen when you try to insert Einstein? You already had computer science with 1 as 1 prefix and finance with 1 prefix.

So, you had in bucket two corresponding to 1 you already have 2 records that bucket is full assuming that it can take only 2 records. So, now, you get another which is value 1; so, its value is 1. So, what I need to do? I need to actually expand this now how do I do that? I cannot expand this because there is no more space left. So, all that I need to do is to actually expand the bucket address table.

So, earlier if I just go back. So, if I just go back earlier we had only two entries because we are using only 1 bit in the prefix and with that I could not have inserted Einstein oh is from department of physics which also has a 1 bit prefix which is 1 it was. So, I need more space; so, I have increased the prefix level to 2 going here and now naturally I have increases to 2; now I have let me erase these entries and now I look at for music I look at 2 for physics 1 0, for finance 1 0, for computer science 1 1.

So, now, I find that after I have moved from looking at 1 bit of prefix to 2 bits of prefix now finance and computer science which was earlier together because I was looking at 1 bit now becomes different, but finance and physics both come to the same 1 0.

So, in the hash bucket address table 1 0 you have finance and physics coming in with Wu and Einstein records and computer science which has got 1 1 the Srinivasan record goes to a new bucket which comes from 1 1 here. Now the interesting fact is what happens to Mozart who was there if you remember the earlier structure this is we had only 1 here. So, this was going to Mozart this was 1 and Mozart was here because music had a prefix 0; now music has a prefix 0 0.

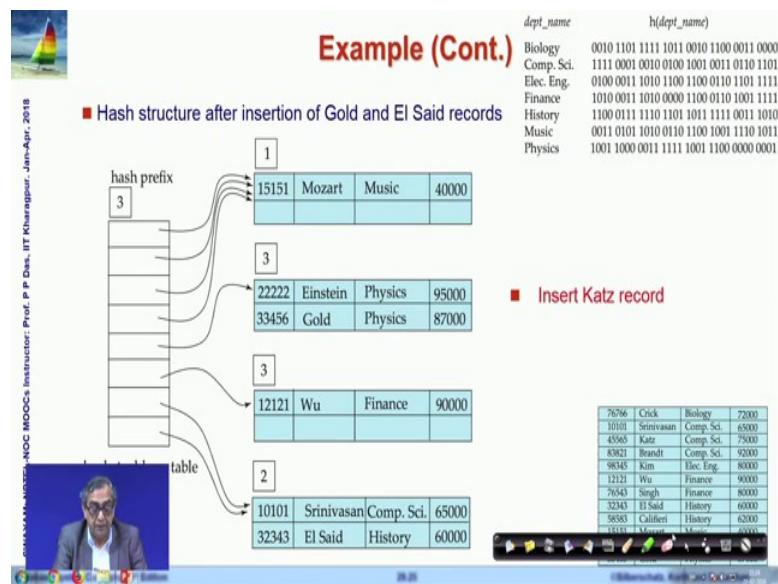
So, what he would have expected? You would have expected that therefore, since 0 0 has come and 0 1 has also come in. So, you would have expected that to have another bucket here which 0 1 is, but then you observe that actually that would be a quite a wasteful to do because you do not actually have a record which has a prefix 0 1.

Out of these two which are we are looking at two prefixes, but you do not really right now need to look at both the prefixes you can still resolve just by based on the first prefix 1. So, you do a simple trick you do not change the prefix level of the particular bucket you say it is 1. Because it is you just need to look at one bit to be able to come to the records in this bucket and the globally it has changed to 2 bits prefix, but locally you keep it as 1.

And with that what you have? You have 0 1 which has a bucket address table entry actually does not have a bucket because there is no records for that. So, you let that point to the same bucket. So, this is a very critical observation that these numbers are basically the local depth; the local information of how many bits in the prefix you need to look at to be able to resolve for coming to this bucket for the records that you currently have.

Whereas this is the global one this is a global maximum that you have. So, naturally local depth cannot exceed the global depth, but if it is equal then you have a unique mapping from the bucket address table entry to the bucket, but if the local depth as in here is less than the global depth; in terms of the number of prefix bits you are looking at then multiple bucket address table pointers actually end up in the same bucket and that is the main principle of this algorithm we can just continue further inserting gold and said into this.

(Refer Slide Time: 23:47)



So, as you try to insert gold and said gold is also from physics which we already had, so physics and said is from history which we did not have. So, history finance computers let me let me just mark them by the side. So, you have now computer science, finance, history, music and physics.

Now, you will find that you need to you now have physics is 1 0 and you have two records for that and music is continues to be 0 0 ah; obviously, history is 1 1 same as computer science. So, that has to go on this and finances on 1 0 now, but what happens is when you try to do this; you could not have inserted more records because you have run out of space in the buckets. So, again you have run out of that; so, you need to expand in terms of the number of bits that you look at.

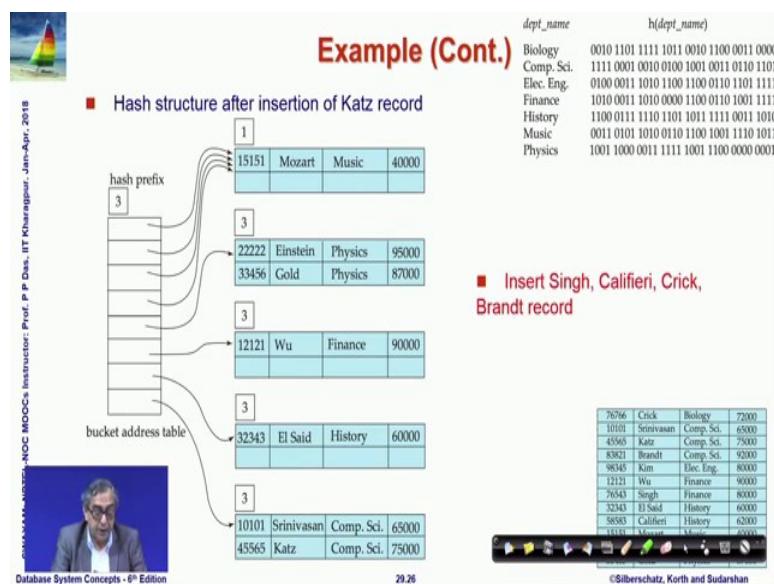
So, you increase that to 3 and now you have 0 0 0 to 1 1 1, but as I have explained not all buckets really need to look into. So, many bits Mozart this bucket continues to B with a

local depth of 1 because if you look into all these 4 different cases; then music is the only one which has a prefix 0, everyone else has a prefix 1. So, if I know that it is 0 then it comes to only this bucket and nowhere else consequently all these 4 bucket address pointers actually go to this bucket table.

Whereas these two for physics I have 1 0 and for finance we have 1 0 here and these come to. So, physics now is looking into 3; so, it is 1 0 0 finance is into 3 it is 1 0 1. So, both physics and finance go to different buckets; now coming to computer science it is 1 1 1 and there is no. So, computer science is 1 1 1 and there is no 1 1 0. So, the 1 1 0 bucket address table pointer continues to point to the same bucket and the local depth value is just 2 <3 in the global table.

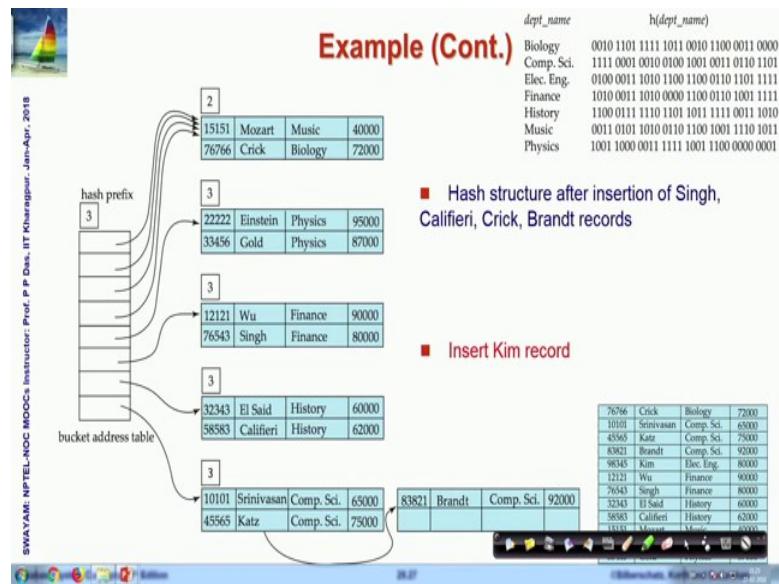
So, this is the basic process of doing dynamic hashing. So, I will not ah; so, the whole example in terms of this table I have given here worked out.

(Refer Slide Time: 26:49)



So, you can just go through every step and try to convince yourself.

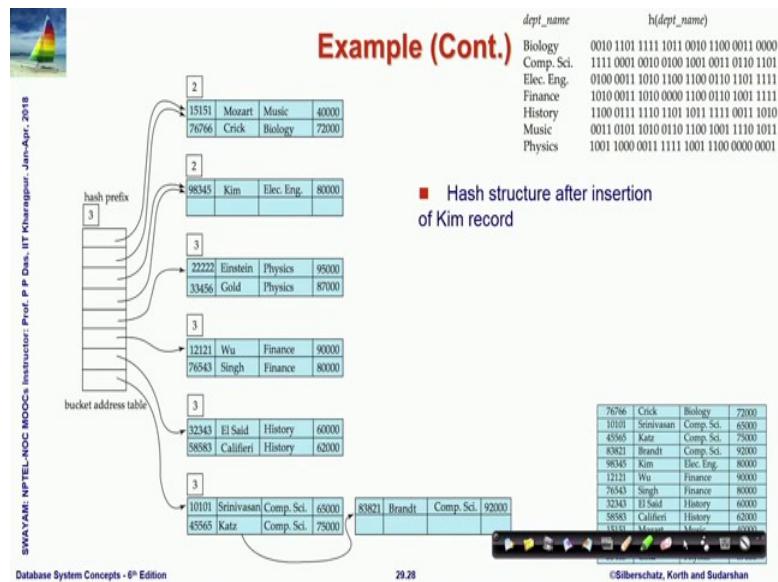
(Refer Slide Time: 26:54)



This is an interesting case that happens here where again you come to computer science professors to be entered. So, at level 3 of prefix you have all of them have prefix 1 1 1. So, you would have required to split or increase the prefix level globally the prefix level to 4, but assuming that there is an upper bound on the number of prefix levels; you can do which decides the size of the bucket address table. If that is given to be 3 you certainly cannot increase it further; so, all that you will have to do is actually do a kind of an overflow chain here as well.

So, all of them are 1 1 1 here which brings you to this you cannot find it you go to this and all 1 1 ones in future will have to be. So, it is a its kind of a tradeoff between what is the size of the global depth, how many prefixes globally you would like to look at what is the size of every bucket that you will have to maintain and what is the kind of chaining that you will have to accept because of that. So, this is what happens particularly.

(Refer Slide Time: 28:05)



So, you can continue in this way and this is a final table where all things have been hashed well. So, this is the basic extendable hashing scheme it has in this the performance does not degrade with the growth of the file and there is very minimal overhead of the space. But it does have disadvantages for example, there is a extra level of indirection to find the desired record because it the hash then come to the hash bucket address table and then go to the bucket address table itself may be very big because it is exponential in the size of the number of beds.

So, it could be larger than memory if that. So, much of you know a contiguous allocation may not be possible. So, you will need to have another possibly a B + tree structure to locate the desired record in the bucket address table first. And then changing the bucket address table will become a quite an expensive operation. So, the growth will become.

(Refer Slide Time: 28:13)

**Extendable Hashing vs. Other Schemes**

- Benefits of extendable hashing:
  - Hash performance does not degrade with growth of file
  - Minimal space overhead
- Disadvantages of extendable hashing
  - Extra level of indirection to find desired record
  - Bucket address table may itself become very big (larger than memory)
    - Cannot allocate very large contiguous areas on disk either
    - Solution: B\*-tree structure to locate desired record in bucket address table
  - Changing size of bucket address table is an expensive operation
- Linear hashing is an alternative mechanism
  - Allows incremental growth of its directory (equivalent to bucket address table)
  - At the cost of more bucket overflows

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition 29.29 ©Silberschatz, Korth and Sudarshan

So, there are several disadvantages that also this scheme has. So, another alternate is to use a linear hashing allows incremental growth of his directory at the cost of more bucket overflows of course,.

(Refer Slide Time: 29:25)

**COMPARATIVE SCHEMES**

PPD

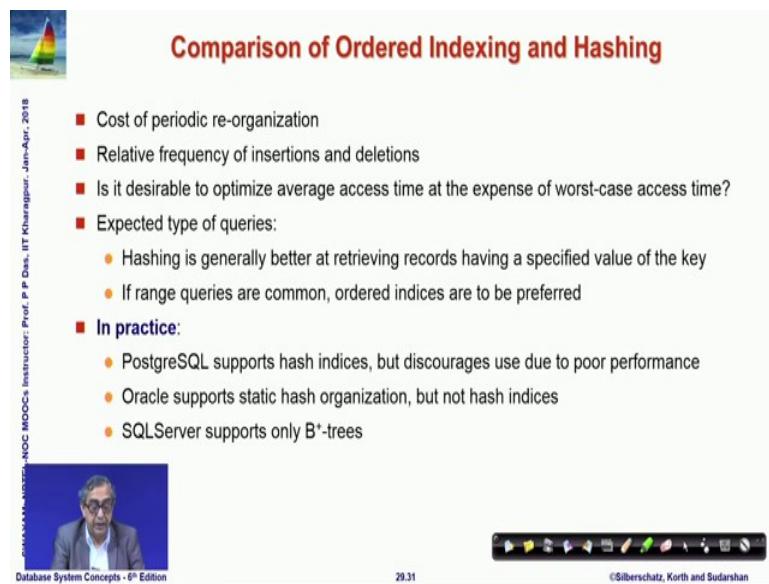
- Static Hashing
- Dynamic Hashing
- Comparison of Ordered Indexing and Hashing
- Bitmap Indices

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition 29.30 ©Silberschatz, Korth and Sudarshan

I would quickly try to compare the two major schemes that we have discussed.

(Refer Slide Time: 29:30)



The slide title is "Comparison of Ordered Indexing and Hashing". It features a small sailboat icon in the top left corner. The main content is a bulleted list comparing the two indexing methods:

- Cost of periodic re-organization
- Relative frequency of insertions and deletions
- Is it desirable to optimize average access time at the expense of worst-case access time?
- Expected type of queries:
  - Hashing is generally better at retrieving records having a specified value of the key
  - If range queries are common, ordered indices are to be preferred
- In practice:
  - PostgreSQL supports hash indices, but discourages use due to poor performance
  - Oracle supports static hash organization, but not hash indices
  - SQLServer supports only B+-trees

At the bottom left is a small video thumbnail showing a man speaking. The bottom right contains the text "Database System Concepts - 8<sup>th</sup> Edition", the page number "29.31", and the copyright notice "©Silberschatz, Korth and Sudarshan".

The ordered indexing and the hashing now naturally ordered indexing has suffers from the cost of periodic reorganization. And because the indexing will have to be maintained the hashing is better in terms of that relative you will have to look at the relative frequency of insertion deletion that decides much of the cost between going between these two schemes.

You will have to see is it desirable to optimize average access time at the expense of worst case access time. For example there could be several ways to organize; so, that your average become your worst case may be really really bad, but as long as your averages is very good you should be happy about it. So, those kind of hashing schemes should be more preferred. So, you also depends on the kind of expected type of query.

So, for example, hashing is better in terms of retrieving records which have a specific value of the key because you can directly map from that key to the bucket. And if range queries are common then as we have seen ordered indices would make it make much better sense because in terms of the ordering you can quickly get all the required records at the same physical location nearby physical location.

If you would like to understand as to what the industry practices are it is very mixed. And if you just look into 3 of the very common database systems PostgreSQL does support hash index, but recommends does not recommend it because of the poor performance oracle supports static hash organization, but not hash indices SQL server

supports only B + trees no hash index space scheme. So, of course, you can see that there as the community is mixed in terms of it is a reaction to whether its indexing or hashing, but hashing powerful at least in limited ways is a powerful technique to go with.

(Refer Slide Time: 31:35)

The slide has a header 'PPD' and a small sailboat icon. On the left, vertical text reads 'CHAMAKA MOOC NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018'. The main title 'BITMAP INDICES' is in red. Below it is a video frame showing a man speaking. The bottom right corner shows '©Silberschatz, Korth and Sudarshan' and the slide number '29.32'.

- Static Hashing
- Dynamic Hashing
- Comparison of Ordered Indexing and Hashing
- Bitmap Indices

The last two that I would like to just quickly remind I mean take you through is what is known as bitmap indexes.

(Refer Slide Time: 31:43)

The slide has a header 'Bitmap Indices' and a small sailboat icon. On the left, vertical text reads 'CHAMAKA MOOC NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018'. The main content is a bulleted list about bitmap indices. Below it is a video frame showing a man speaking. The bottom right corner shows '©Silberschatz, Korth and Sudarshan' and the slide number '29.33'.

- Bitmap indices are a special type of index designed for efficient querying on multiple keys
- Records in a relation are assumed to be numbered sequentially from, say, 0
  - Given a number  $n$  it must be easy to retrieve record  $n$ 
    - ▶ Particularly easy if records are of fixed size
- Applicable on attributes that take on a relatively small number of distinct values
  - E.g. gender, country, state, ...
  - E.g. income-level (income broken up into a small number of levels such as 0-9999, 10000-19999, 20000-50000, 50000- infinity)
- A bitmap is simply an array of bits

Bitmap indexing is a very simple idea. So, what you it is a special type of indexing which is designed for querying on multiple keys; the basic idea is that if let us assume

that all records in a relation are numbered from 0 to n and let us say that you are talking about attributes which can take very small number of distinct values.

So, bitmap indexing is not for any attribute. So, consider attributes such a very small number of distinct value say gender which has two possible values or few possible values the country state. So, take those or maybe you can you can nominally bucket a range of numbers source income level 5, 6, 10 income levels. So, small range of possibilities and bitmap is simply array of bits. So, take an array of possible array for the records and for the possible values you mark 1 or 2 0.

(Refer Slide Time: 32:39)

**Bitmap Indices (Cont.)**

- In its simplest form a bitmap index on an attribute has a bitmap for each value of the attribute
  - Bitmap has as many bits as records
  - In a bitmap for value v, the bit for a record is 1 if the record has the value v for the attribute, and is 0 otherwise

record number	ID	gender	income_level
0	76766	m	L1
1	22222	f	L2
2	12121	f	L1
3	15151	m	L4
4	58583	f	L3

©Silberschatz, Korth and Sudarshan

	Bitmaps for gender	Bitmaps for income_level
m	10010	L1 10100
f	01101	L2 01000
		L3 00001
		L4 00010
		L5 00000

So, this here is an example showing it. So, we are showing bitmap index for gender. So, you have a array for m the male gender and f female gender and if you look into the record 076766 has male under m gender m. And therefore, in the male gender bitmap index the first bit is 1 in f it is 0; so, actually m and f are complimentary.

Similarly, for the income levels you have 5 different bitmaps encoding; the 5 different possible levels in the income that you can have.

(Refer Slide Time: 33:21)

The slide has a title 'Bitmap Indices (Cont.)' in red at the top right. On the left, there is a small logo of a sailboat on water. The main content area contains a bulleted list of points about bitmap indices:

- Bitmap indices are useful for queries on multiple attributes
  - not particularly useful for single attribute queries
- Queries are answered using bitmap operations
  - Intersection (and)
  - Union (or)
  - Complementation (not)
- Each operation takes two bitmaps of the same size and applies the operation on corresponding bits to get the result bitmap
  - E.g.  $100110 \text{ AND } 110011 = 100010$
  - $100110 \text{ OR } 110011 = 110111$
  - $\text{NOT } 100110 = 011001$
  - Males with income level L1:  $100110 \text{ AND } 10100 = 10000$ 
    - Can then retrieve required tuples
    - Counting number of matching tuples is even faster

At the bottom, there is footer text: 'SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur', 'Database System Concepts - 8<sup>th</sup> Edition', '29.35', and '©Silberschatz, Korth and Sudarshan'. There is also a decorative footer bar with various icons.

Now the big advantage of bitmap indices are doing different queries on multiple attributes. And for example, the often queries have intersection union and they can be simply computed in terms of bitmapped operations. So, if you have two different values to be two conditions to check in terms of bitmap indices; then you can just make there and whatever satisfy.

So, say if you are looking at males at for example, here males at income level L 1, then you can you can just take the bitmap for gender and bitmap for income level and do the ending and you get that the first record has value 1.

(Refer Slide Time: 34:08)

The slide features a title 'Bitmap Indices (Cont.)' in red at the top right. On the left, there is a small logo of a sailboat on water. The main content area contains a bulleted list of points about bitmap indices:

- Bitmap indices generally very small compared with relation size
  - E.g. if record is 100 bytes, space for a single bitmap is 1/800 of space used by relation
    - ▶ If number of distinct attribute values is 8, bitmap is only 1% of relation size
- Deletion needs to be handled properly
  - Existence bitmap to note if there is a valid record at a record location
  - Needed for complementation
    - ▶  $\text{not}(A=v) = (\text{NOT bitmap-}A\text{-}v) \text{ AND ExistenceBitmap}$
- Should keep bitmaps for all values, even null value
  - To correctly handle SQL null semantics for NOT(A=v):
    - ▶ intersect above result with (NOT bitmap- $A\text{-Null}$ )

At the bottom, there is footer text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '29.36', and '©Silberschatz, Korth and Sudarshan'.

So, that is answer and you can quickly come to that. So, bitmap indices generally very I mean naturally they are they are they are small in compared to the relation size because you are doing bitmap indexing only if the attribute can take small number of distinct values.

Of course, the deletion has to be handled properly look at this and should keep bitmap for all values even if there are null values you must keep that otherwise you will lose track of that.

(Refer Slide Time: 34:33)

The slide features a title 'Efficient Implementation of Bitmap Operations' in red at the top right. On the left, there is a small logo of a sailboat on water. The main content area contains a bulleted list of points about efficient bitmap implementation:

- Bitmaps are packed into words; a single word and (a basic CPU instruction) computes and of 32 or 64 bits at once
  - E.g. 1-million-bit maps can be and-ed with just 31,250 instruction
- Counting number of 1s can be done fast by a trick:
  - Use each byte to index into a precomputed array of 256 elements each storing the count of 1s in the binary representation
    - ▶ Can use pairs of bytes to speed up further at a higher memory cost
    - Add up the retrieved counts
- Bitmaps can be used instead of Tuple-ID lists at leaf levels of B<sup>+</sup>-trees, for values that have a large number of matching records
  - Worthwhile if > 1/64 of the records have that value, assuming a tuple-id is 64 bits
  - Above technique merges benefits of bitmap and B<sup>+</sup>-tree indices

At the bottom, there is footer text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '29.37', and '©Silberschatz, Korth and Sudarshan'.

And there are several efficient implementations some information I have given, but is we do not want to go in much depth here.

(Refer Slide Time: 34:45)

**Module Summary**

- Explored various hashing schemes – Static and Dynamic Hashing
- Compared Ordered Indexing and Hashing
- Studies the use of Bitmap Indices for fast access of columns with limited number of distinct values

Database System Concepts - 8<sup>th</sup> Edition

29.38

©Silberschatz, Korth and Sudarshan

But several compression techniques are possible in terms of bitmaps; in the next module I will talk little bit more about how to use that. In this module to summarize we have talked about various hashing schemes static and dynamic hashing, compared the order indexing with hashing and introduced the notion of bitmap indices.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 30**  
**Indexing and Hashing/5 : Index Design**

Welcome to module 30 of Database Management Systems. We have been discussing about indexing and hashing and this is a concluding module on that.

(Refer Slide Time: 00:27)

**Module Recap**

- Static Hashing
- Dynamic Hashing
- Comparison of Ordered Indexing and Hashing
- Bitmap Indices

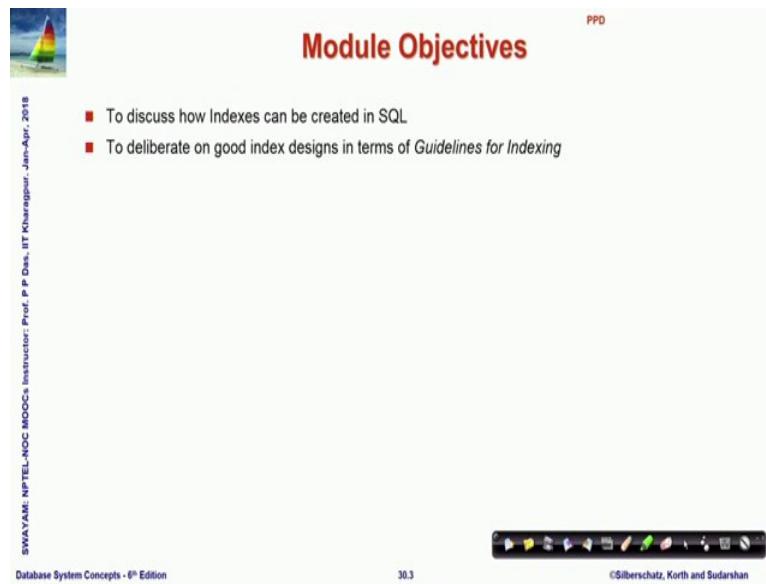
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr., 2018

PPD

Database System Concepts - 8<sup>th</sup> Edition 30.2 ©Silberschatz, Korth and Sudarshan

We have in the last module discussed about different hashing techniques static and dynamic and compare that in introduce bitmap indices.

(Refer Slide Time: 00:35)

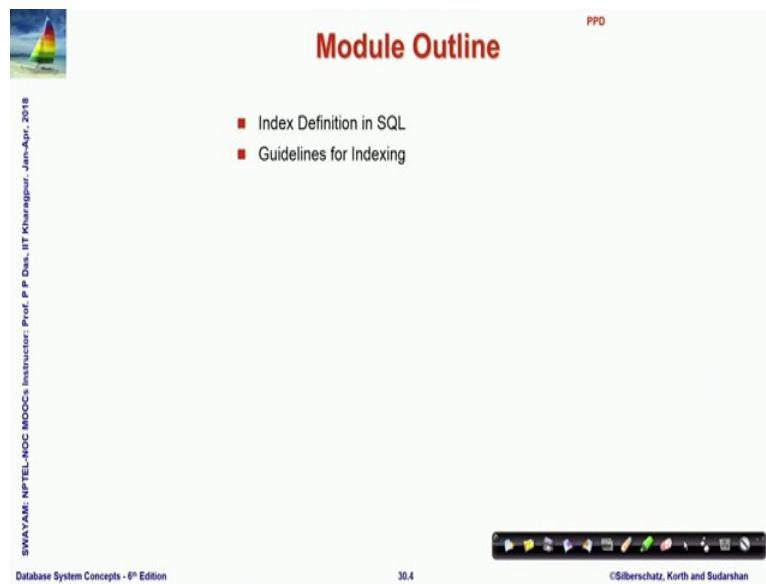


This slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. On the far left, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs", "Instructor: Prof. P P Deshpande, IIT Kharagpur", and "Jan-Apr, 2018". At the bottom left, it says "Database System Concepts - 8<sup>th</sup> Edition". The main content area contains two bullet points: "To discuss how Indexes can be created in SQL" and "To deliberate on good index designs in terms of *Guidelines for Indexing*". The bottom right corner includes the copyright notice "©Silberschatz, Korth and Sudarshan". A navigation bar with various icons is located at the bottom.

Now, in this module we would specifically look into the use cases, we will check as to how indexes can be created in SQL first. Because we will have to use it you have already known the theory of various different indices and so, on so, how do you actually tell the system to index.

And the second is the important thing as to when should you index and on what. So, we talked about a few guidelines for indexing.

(Refer Slide Time: 01:05)



This slide is titled "Module Outline" in red at the top right. It features a small sailboat icon in the top left corner. On the far left, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs", "Instructor: Prof. P P Deshpande, IIT Kharagpur", and "Jan-Apr, 2018". At the bottom left, it says "Database System Concepts - 8<sup>th</sup> Edition". The main content area contains two bullet points: "Index Definition in SQL" and "Guidelines for Indexing". The bottom right corner includes the copyright notice "©Silberschatz, Korth and Sudarshan". A navigation bar with various icons is located at the bottom.

So, that is that is what we want to learn.

(Refer Slide Time: 01:09)

The slide has a header 'Index Definition in SQL' and a footer with course details: SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr- 2018 Database System Concepts - 8<sup>th</sup> Edition Author: Silberschatz, Korth and Sudarshan Page: 30.6.

**■ Create an index**

```
create index <index-name> on <relation-name>
(<attribute-list>)
```

E.g.: `create index b-index on branch(branch_name)`

**■ Use `create unique index` to indirectly specify and enforce the condition that the search key is a candidate key**

- Not really required if SQL **unique** integrity constraint is supported – it is preferred

**■ To drop an index**

```
drop index <index-name>
```

**■ Most database systems allow specification of type of index, and clustering**

- You can also create an index for a cluster
- You can create a composite index on multiple columns up to a maximum of 32 columns
  - A composite index key cannot exceed roughly one-half (minus some overhead) of the available space in the data block

So, index can be defined in SQL in very similar syntax as you create a table. So, you say create index put a name for that index and then you say on which relation are you indexing and put the list of attributes on which you are indexing. So, if branch can relation can be indexed on branch name and I may call that b- index.

Now, there is a way to also express create unique index if you say create unique index and it will expect that the search key is a candidate key because I mean in the sense it all values of that will have to be distinct unique. Now, this used to be very common to do this kind of indexing earlier, but now it is more preferred that you can use unique integrity constraint in terms of the create table which we have already discussed. And that will ensure that you have that kind of a condition satisfied and you may not create unique index for that.

If you do not want an indexes actually do drop index and put the index name. So, most database system allow specification of the type of indexing and clustering that you want to do. So, you can create an index for a cluster also and you can create index for composite index for multiple columns.

(Refer Slide Time: 02:32)

The slide has a header 'Indexing Examples' and a footer 'Source: https://docs.oracle.com/cd/B10500\_01/appdev/920/a9651/index.htm'. It contains a bulleted list of examples:

- Create an index for a single column, to speed up queries that test that column:
  - CREATE INDEX emp\_ename ON emp\_tab(ename);
- Specify several storage settings explicitly for the index:
  - CREATE INDEX emp\_ename ON emp\_tab(ename)  
TABLESPACE users STORAGE (INITIAL 20K NEXT 20k PCTINCREASE 75)  
PCTFREE 0 COMPUTE STATISTICS;
- Create index on two columns, to speed up queries that test either the first column or both columns:
  - CREATE INDEX emp\_ename ON emp\_tab(ename, empno) COMPUTE STATISTICS;
- If a query is going to sort on the function UPPER(ENAME), an index on the ENAME column itself would not speed up this operation, and it might be slow to call the function for each result row
  - A function-based index precomputes the result of the function for each column value, speeding up queries that use the function for searching or sorting:
    - CREATE INDEX emp\_upper\_ename ON emp\_tab(UPPER(ename)) COMPUTE STATISTICS;

So, let us run through quickly couple of examples create an index for a single column to speed up queries the test that column. So, we are saying employee emp\_tab is a relation which has an attribute ename the employee name and we want to create an index on that if we do that then any search that is based on the employee name will become really fast.

Now, while you create the index you could also use optionally various other factors which relate to particularly the storage setting you can set what is you know what is the storage you want to keep for the index how you would like to increase increment that and so, on what is the table space. And very most interestingly you could say that compute statistics now this is something which is optional, but is very useful. For example, if you are not sure as to how your data is getting distributed in different relations and how really they are queried you would not know whether an index is good or it is inappropriate.

So, it is good to actually compute that statistics in terms of the index that you want to know that by doing this index what kind of accesses have happened? So, compute statistics will tell the database system to keep on computing this which you can subsequently refer to. You can create index on two columns also; so, here we are showing one where emp\_tab is indexed on employee name emp\_ename and employee

number together. So, you saying that you create an index on both of these and compute the statistics at the same time.

Now, other ways there are index that can be created on functions. So, suppose if there is a query which going to sort based on the uppercase writing of the e\_name. So, if I just index the e\_name then that itself would not speed up the operation because while you want to sort then the e\_name will have to be changed into the upper case by upper and that is every time will have to do that for every record and then actually apply the sorting comparisons.

So, that will become a slow process, but you can do a function based indexing where you can specify as you can see here the function based indexing where you say that you index based on upper e\_name. So, what will happen your actual values are in impossibly lower case or mixed case, but your index emp upper e\_name will get created on the in the order of the upper case of e\_name and will be very useful in terms of the sorting later on.

(Refer Slide Time: 05:36)

**PPD**

## Bitmap Index in SQL

- create bitmap index <index-name> on <relation-name>(<attribute-list>)
- Example:
  - Student (Student\_ID, Name, Address, Age, Gender, Semester)
  - CREATE BITMAP INDEX Idx\_Gender ON Student (Gender);
  - CREATE BITMAP INDEX Idx\_Semester ON Student (Semester);

STUDENT	STUDENT_ID	STUDENT_NAME	ADDRESS	AGE	GENDER	SEMESTER
Prof. P. P. Das, IIT Kharagpur - June-Aug., 2018	100	Joseph	Alabedon Township	20	M	1
SWAYAM: NPTEL-NOCO's Instructor: Prof. P. P. Das, IIT Kharagpur - June-Aug., 2018	101	Allen	Fraser Township	22	F	1
	102	Chris	Clinton Township	20	F	2
	103	Patty	Troy	22	F	4

Bitmask Index

M	1	0	0	0
F	0	1	1	1

First Row  
2nd Row  
3rd Row  
4th Row

1	1	1	0	0
2	0	0	1	0
3	0	0	0	0
4	0	0	0	1

- SELECT \* FROM Student WHERE Gender = 'F' AND Semester = 4;
- ↳ AND 0 1 1 1 with 0 0 0 1 to get the result

Source: <https://www.tutorialcup.com/dbms/bitmap-index>

Database System Concepts - 8<sup>th</sup> Edition      30.8      ©Silberschatz, Korth and Sudarshan

Now, you can like the normal index you can also create the bitmap index. So, you just say create bitmap index on the name and rest of the structure is similar. So, if there is a student relation which has these fields I can we can create an index on gender; we can create another index on semester these are very typical candidate for bitmap index

because gender can take 2 values here male and female semester can take 4 values 1, 2, 3, 4.

So, the bitmap are shown here and then if I want to do a select where the gender is F and semester is 4; then it is basically anding the bitmap of F which is 0 1 1 and the bitmap of semester 4 which is 0 0 1. So, if we if we and these two we will find that we have the result which is 0 0 0 (011 AND 001). So, which tells me that student id the fourth record of the student ID 103 is a result.

So, this is how bitmap indexing can be used in SQL.

(Refer Slide Time: 06:50)

The slide has a title 'Multiple-Key Access' in red at the top right. On the left, there is a small image of a sailboat on water. The main content area contains several bullet points and a code snippet:

- Use multiple indices for certain types of queries
- Example:

```
select ID
from instructor
where dept_name = "Finance" and salary = 80000
```
- Possible strategies for processing query using indices on single attributes:
  - Use index on `dept_name` to find instructors with department name Finance; test `salary = 80000`
  - Use index on `salary` to find instructors with a salary of 80000; test `dept_name = "Finance"`
  - Use `dept_name` index to find pointers to all records pertaining to the "Finance" department. Similarly use index on `salary`. Take intersection of both sets of pointers obtained

At the bottom, there is footer text: 'SWAYAM: NPTEL-NOC Instructor: Prof. P P Doshi, IIT Kharagpur - Jan-Apr., 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '30.9', and '©Silberschatz, Korth and Sudarshan'.

And actually this the whole thing can be used subsequently in multiple key access for example, if you are doing a query where it is you have department name is finance and salary is 8000, then there could be several strategies for processing this query using the index values for example, if you have single index on single attributes. So, use you can use the index on department name to find instructors which have department and finance. And then test if the salaries 80000 or you can use index on salary to find instructors with salary 80000 and then test if department name is finance.

Or you can use department name index to find pointers to all records that part in to finance department. Index on salary to find all records that part in to 80000 salary and then take intersection of the both sets to get the final result.

(Refer Slide Time: 08:04)

**Indices on Multiple Keys**

- Composite search keys are search keys containing more than one attribute
  - E.g. (*dept\_name*, *salary*)
- Lexicographic ordering:  $(a_1, a_2) < (b_1, b_2)$  if either
  - $a_1 < b_1$ , or
  - $a_1 = b_1$  and  $a_2 < b_2$

CHANDRAKANTAPUR NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition

30.10 ©Silberschatz, Korth and Sudarshan

So, multiple key access could be achieved in terms of various single indexing single attribute indexing also; When we are doing composite search keys then naturally if there are 2 then the indexing means that you will have to define a combined lexical order. So, department name salary means that either department it is it is ordered to indexes are ordered in terms of just the department name.

Or if the department name is same then they are ordered in terms of salary this ordering in which you write the attributes in the multi composite search key is very important because you can see that for the two if the department name is same then the salary will be compared, but not the other way around.

(Refer Slide Time: 08:50)

The slide has a header 'Indices on Multiple Attributes' with a sailboat icon. The text discusses indexing combined search-keys like (dept\_name, salary). It lists handling cases for WHERE clauses involving equality and less-than conditions, noting efficiency trade-offs. A video player interface at the bottom shows a thumbnail of a speaker and navigation controls.

Suppose we have an index on combined search-key  
(*dept\_name*, *salary*)

- With the **where** clause
  - where *dept\_name* = "Finance" **and** *salary* = 80000  
the index on (*dept\_name*, *salary*) can be used to fetch only records that satisfy both conditions.
    - Using separate indices is less efficient — we may fetch many records (or pointers) that satisfy only one of the conditions
  - Can also efficiently handle
    - where *dept\_name* = "Finance" **and** *salary* < 80000
  - But cannot efficiently handle
    - where *dept\_name* < "Finance" **and** *balance* = 80000
      - May fetch many records that satisfy the first but not the second condition

So, when you have index on multiple attributes say again going back to that same example of department naming finance and salary being 80000. So, using separate index is less efficient though we saw how that can be done, but we can also efficiently handle if we have this indexing on department name and salary. Then we can also easily handle queries like department name is finance and salary < 100; it is not because as you can easily figure out because if I can find the equality then I also know what is less.

But note that we cannot efficiently handle if I say that where department name is less than finance and balance I am sorry this should be salary is 80000. The reason is that the ordering of the attributes in this composite key is department name salary. So, if there is if department name is less than is there is no way to check for the equality of salary, but if the department name equals then there is a possibility of checking on the salary. So, because of this ordering this will may fetch many records that satisfy the first one, but not the second condition.

(Refer Slide Time: 10:16)

The slide has a title 'Privileges Required to Create an Index' at the top right. On the left, there is a small sailboat icon and a vertical column of text: 'CHAMAKAM NOC NOC INSTRUCTOR: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr- 2018'. The main content area contains a bulleted list of requirements:

- When using indexes in an application, you might need to request that the DBA grant privileges or make changes to initialization parameters
- To create a new index
  - You must own, or have the INDEX object privilege for, the corresponding table
  - The schema that contains the index must also have a quota for the tablespace intended to contain the index, or the UNLIMITED TABLESPACE system privilege
  - To create an index in another user's schema, you must have the CREATE ANY INDEX system privilege
- Function-based indexes also require the QUERY\_REWRITE privilege, and that the QUERY\_REWRITE\_ENABLED initialization parameter to be set to TRUE

At the bottom left, there is a small video thumbnail showing a man speaking. The bottom right corner shows the source URL: [https://docs.oracle.com/cd/B10500\\_01/appdev/920/a9651/](https://docs.oracle.com/cd/B10500_01/appdev/920/a9651/), page number 30.12, and copyright information: ©Silberschatz, Korth and Sudarshan.

Now, you should also remember that you need a special privilege to create an index because this is partly in the domain of the administrator's job. So, you need the specific privileges access rights to be able to do that. So, to create a new index either you have to own or own that those set of tables on which you are creating the index and or have the index object privilege for those tables or the schema that contains the index might also have a quota.

So, that you can because creating the index means you are will be using the temporary tablespace on a on a regular basis. And, but with this you will not be able to create index in some other user schema for that you need a global right which is the create any index kind of system privilege. So, also check if you are not being able to create an index check what is your privilege that exists function based indexes require other privileges; please check on that.

(Refer Slide Time: 11:28)

The slide features a small sailboat icon in the top left corner. In the top right, the text "PPD" is written above a bulleted list: "Index Definition in SQL" and "Guidelines for Indexing". The main title "GUIDELINES FOR INDEXING" is centered in large red capital letters. Below it is a video thumbnail showing a man speaking. At the bottom, there is a navigation bar with icons and the text "Database System Concepts - 8<sup>th</sup> Edition", "30.13", and "©Silberschatz, Korth and Sudarshan".

Now, let us. So, we have seen how to create index how to use that in terms of the SQL application SQL query system.

(Refer Slide Time: 11:43)

The slide has a similar layout to the previous one. It includes a sailboat icon, a bulleted list of topics, and a large title "Guidelines for Indexing". The list covers various aspects of database design and performance optimization. The video thumbnail, footer text, and navigation bar are also present.

- In Modules 16 to 20 (Week 4), we have studied various issues for a proper design of a relational database system. This focused on:
  - Normalization of Tables leading to
    - Reduction of Redundancy to minimize possibilities of Anomaly
    - Easier adherence to constraints (various dependencies)
    - Efficiency of access and update – a better normalized design often gives better performance
- The performance of a database system, however, is also significantly impacted by the way the data is physically organized and managed. These are done through:
  - Indexing and Hashing
- While normalization and design are startup time activities that are usually performed once at the beginning (and rarely changed later), the performance behavior continues to evolve as the database is used over time. Hence we need to continually:
  - Collect statistics about data (of various tables) to learn of the patterns, and
  - Adjust the indexes on the tables to optimize performance
- There is no sound theory that determines optimal performance. Rather, we take a quick look into a few common guidelines that can help you keep your database efficient.

Now, we will quickly take a look into why how should we index and where. So, if you recall in the modules 16 to 20 in the week four we have studied various issues of a proper design of relational database system, we focused on normalization of tables that can reduce redundancy and minimize anomaly how can we easily adhere to various

constraints how to improve the efficiency of access and update a better normalized design often gives better performance.

For example, we are optimizing the minimizing the requirement for computing join and all those. So, those advantages we have seen, but the actual performance of a database system is significantly impacted by the way the physical data is organized and managed which does not come across in terms of the logical design that we have seen. So, these are what are being achieved in terms of indexing and hashing. So, this is where we need to understand the actual boundary to the physical organization and that is what we have been trying to do.

So, if you think back while you are normalizing at the design level. So, those are the startup time activities; so, usually we will design and normalize and you know make the create table and do all that at the beginning of a database system. And it is really it will really be changed later because it will have severe implications, but the performance behavior will continue to evolve will continue to change because the design does not tell you exactly what the statistics of that data would be what the behavior of the data would be. So, it will evolve as data base is used over time.

Hence you will need to continuously collect statistics about the data of various tables to learn of the patterns as to which table is getting heavier which where what are the attributes on which more accesses are happening, where what kind of queries you are getting and you have to adjust the indexes on the tables to optimize the performance.

So, that is the whole requirement all about unfortunately unlike the functional dependency or multivalued dependency theories that we studied in the design space; there is no sound theory that determines optimal performance. So, all that have is more and expertise that you develop through experience. So, what I will take you through are a set of few common guidelines about how to keep your database agile while you are you go through the life cycle of different data coming in and going out.

(Refer Slide Time: 14:23)

**Guidelines for Indexing**

■ Rule 0: Indexes lead to Access – Update Tradeoff

- Every query (access) results in a 'search' on the underlying physical data structures
  - Having specific index on search field can significantly improve performance
- Every update (insert / delete / values update) results in update of the index files – an overhead or penalty for quicker access
  - Having unnecessary indexes can cause significant degradation of performance of various operations
  - Index files may also occupy significant space on your disk and / or
  - Cause slow behavior due to memory limitations during index computations
- Use informed judgment to index!

CHAMAKAM NOORI, NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018  
Database System Concepts - 8<sup>th</sup> Edition

30.15 ©Silberschatz, Korth and Sudarshan

So, the first rule I say rule 0 is the indexes lead to access update tradeoff we have already seen this at every query results in a search in the underlying physical data structure as we have understood. Having specific index on search certainly can improve performance, but as we have already noted every update with the be it insert, delete or update of values will result in update of the index files.

So, it is an overhead or penalty for quicker access that we are paying. So, having unnecessary indexes can cause significant degradation of performance. Index files will also occupy significant space on your disk and it may actually cause to slow down your behavior due to memory limitation during index computation. So, rule 0 indexes lead to this trade off always be watchful about that use judgment to index.

(Refer Slide Time: 15:26)

**Guidelines for Indexing**

■ Rule 1: Index the Correct Tables

- Create an index if you frequently want to **retrieve less than 15%** of the rows in a large table
  - The percentage varies greatly according to the relative speed of a table scan and how clustered the row data is about the index key
    - The faster the table scan, the lower the percentage
    - More clustered the row data, the higher the percentage
  - Index columns used for joins to improve performance on **joins of multiple tables**
  - Primary and unique keys automatically have indexes, but you might want to create an **index on a foreign key**
  - **Small tables** do not require indexes
    - If a query is taking too long, then the table might have grown from small to large

Rule 1 index the correct tables decide which tables are best candidates for index to creating an index if you frequently want to retrieve say less than 15 percent of the rows in a large table. Now first 15 percent is a ballpark number this can vary greatly according to the relative speed of the table scan ah.

Fast of the table scan you can use a lower percentage of cut off more cluster the row data you can use a higher percentage for cut up. Index tables index columns used for joining multiple tables if you have situations or multiple tables are used in joins on a on a moderately regular basis then the columns are used in the join in these tables; these tables should be indexed based on those.

The primary and unique keys automatically have indexes, but you might want to you have an index on the foreign key. So, consider that small tables do not require index if a query is requiring unnecessarily long time or unexpectedly long time; it is time to check if the table has become really big compared to small and it might be term to index that.

(Refer Slide Time: 16:45)

**Guidelines for Indexing**

PPD

■ Rule 2: Index the Correct Columns

- Columns with one or more of the following characteristics are candidates for indexing:
  - ▶ Values are relatively unique in the column
  - ▶ There is a wide range of values (good for regular indexes)
  - ▶ There is a small range of values (good for bitmap indexes)
  - ▶ The column contains many nulls, but queries often select all rows having a value. In this case, a comparison that matches all the non-null values, such as:
    - WHERE COL\_X > -9.99 \*power(10,125) is preferable to WHERE COL\_X IS NOT NULL
    - This is because the first uses an index on COL\_X (if COL\_X is a numeric column)
- Columns with the following characteristics are less suitable for indexing:
  - ▶ There are many nulls in the column and you do not search on the non-null values
  - ▶ LONG and LONG RAW columns cannot be indexed
- The size of a single index entry cannot exceed roughly one-half (minus some overhead) of the available space in the data block

Source: [https://docs.oracle.com/cd/B10500\\_01/appdev/920/a9651/](https://docs.oracle.com/cd/B10500_01/appdev/920/a9651/)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur Date: Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

30.17

©Silberschatz, Korth and Sudarshan

So, rule 1 index the correct tables and certainly related to that is index the correct columns. The columns with some of the characteristics I have just noted down are good candidates for indexing values are relatively unique in the column, then indexing will give you a good benefit. There is a wide range of values where your regular indexes will work well.

There is a small range of values where bitmap indexes will give you good results. So, use those in column contains many nulls, but queries often select all rows having a value. So, there are column have lot of null values, but whenever you do a query then you actually take out rows which have values. So, in those cases you can actually if you involve certain I mean if you write the SQL query by keeping the condition in a way. So, that the index can be used that is for example, you could put a condition such that only non null values will be matched.

Compared to that if you have just taken (Refer Time: 17:54) at non null check your first query would run faster; if you have an index on the COL\_X because the query would be able to work on that index. So, these are things that you should do in terms of the column. And if a column has the kind of characteristic that there are many nulls in the column and you typically do not search non null values; then it is good it is better not to index those columns long and long row columns cannot be indexed anyway. So, this remember the rule 2 index the correct columns.

(Refer Slide Time: 18:29)

**Guidelines for Indexing**

■ Rule 3: Limit the Number of Indexes for Each Table

- The more indexes, the more overhead is incurred as the table is altered
  - When rows are inserted or deleted, all indexes on the table must be updated
  - When a column is updated, all indexes on the column must be updated
- You must weigh the performance benefit of indexes for queries against the performance overhead of updates
  - If a table is primarily read-only, you might use more indexes; but, if a table is heavily updated, you might use fewer indexes

Source: [https://docs.oracle.com/cd/B10500\\_01/appdev/920/a9651/](https://docs.oracle.com/cd/B10500_01/appdev/920/a9651/)

30.18 ©Silberschatz, Korth and Sudarshan

Then the rule 3 limit the number of indexes for each table the more the index more overhead we have already seen this as a part of rule 2 rule 0 as well. When rows are inserted or deleted indexes of a table must be updated when columns are updated all the indexes on the column must be updated. So, there is a lot of cost; so, half as limited number of indices as will start with purposed.

So, you must regularly weigh the benefit of having the indexed for queries against the performance overhead of the updates. For example, if a table is primarily read only you might use more indexes because the overhead will be less, but if a table is heavily updated you might use fewer number of indices.

(Refer Slide Time: 19:14)

The slide has a header 'Guidelines for Indexing' in red. On the left, there's a small logo of a sailboat on water. On the right, there's a table titled 'Table VENDOR\_PARTS' with columns 'VEND ID', 'PART NO', and 'UNIT COST'. The table contains 10 rows of data. At the bottom of the slide, there's a navigation bar with icons for back, forward, search, and other presentation controls.

Rule 4: Choose the Order of Columns in Composite Indexes

- The order of columns in the CREATE INDEX statement can affect performance
  - Put the column expected to be used most often first in the index
  - You can create a composite index (using several columns), and the same index can be used for queries that reference all of these columns, or just some of them
- For the VENDOR\_PARTS table, assume that there are 5 vendors, and each vendor has about 1000 parts. Suppose VENDOR\_PARTS is commonly queried as:
  - SELECT \* FROM vendor\_parts WHERE part\_no = 457 AND vendor\_id = 1012;
  - Create a composite index with the most selective (with most values) column first
    - CREATE INDEX ind\_vendor\_id ON vendor\_parts (part\_no, vendor\_id);
- Composite indexes speed up queries that use the leading portion of the index:
  - So queries with WHERE clauses using only PART\_NO column also runs faster
  - With only 5 distinct values, a separate index on VENDOR\_ID does not help

Source: [https://docs.oracle.com/cd/B10500\\_01/appdev/920/a9651.htm](https://docs.oracle.com/cd/B10500_01/appdev/920/a9651.htm)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

30.19

©Silberschatz, Korth and Sudarshan

Rule 4 choose the order of columns in the composite index. So, you have already seen couple of slides back I talk to you to the impact of what is the impact of ordering in other columns in terms of composite index. So, the order of columns in the create index statement can affect performance. So, the column that you expected to be used most often put that as the first index.

Because it is also possible that you are actually not doing a query which takes the whole of the composite search key, but a part of it, but if you have a composite search key index you will still benefit if the query is using the attributes from the first part of the index. So, here I am showing some example say there is a vendors part table and let us say there are 5 vendors and let us say. So, there is vendor id part number and unit cost forget about unit cost for this consideration.

So, it is primarily part number and vendor id. So, let us say there are 5 vendors and each vendor has about 1000 parts. So, and let us say that it is queried like this that the part number is such and such and vendor id is such and such you get you select all all that matches.

Now, if you create a composite index then it should be on part number and vendor id not the other way around. Because if you do that then queries where only part number is used will also run faster, but the vendor id here is not a very good candidate for indexing as a first attribute because it has only five possible values very small

number of value. So, indexing really does not help here it cannot discriminate after indexing there will be lot of clusters still found.

(Refer Slide Time: 21:17)

The slide has a header 'Guidelines for Indexing' in red. On the left is a small sailboat icon. On the right is a video player showing a man speaking. The video player includes controls like play, pause, and volume, and displays the source URL 'https://docs.oracle.com/cd/B10500\_01/appdev/920/a965...' and page number '30-20'. The footer includes the text 'Database System Concepts - 8<sup>th</sup> Edition' and '©Silberschatz, Korth and Sudarshan'.

**Rule 5: Gather Statistics to Make Index Usage More Accurate**

- The database can use indexes more effectively when it has statistical information about the tables involved in the queries
- Gather statistics when the indexes are created by including the keywords COMPUTE STATISTICS in the CREATE INDEX statement
- As data is updated and the distribution of values changes, periodically refresh the statistics by calling procedures like (in Oracle):
  - DBMS\_STATS.GATHER\_TABLE\_STATISTICS and
  - DBMS\_STATS.GATHER\_SCHEMA\_STATISTICS

Rule number 5 gather statistics to make index usage more accurate that is a that is a very very important factor the database can use indexes more effectively if the statistical information is available. So, gather statistics learn from that we have already discussed how to gather statistics from the create index statement.

And then there are functions these are function names in oracle in your system you might want to check up what these functions are called. So, by that you can find out statistics about the tables and the schema that you have and use that information to subsequently optimize the index.

(Refer Slide Time: 21:58)

**Guidelines for Indexing**

■ Rule 6: Drop Indexes That Are No Longer Required

- You might drop an index if:
  - It does not speed up queries. The table might be very small, or there might be many rows in the table but very few index entries
  - The queries in your applications do not use the index
  - The index must be dropped before being rebuilt
- When you drop an index, all extents of the index's segment are returned to the containing tablespace and become available for other objects in the tablespace
- Use the SQL command DROP INDEX to drop an index. For example, the following statement drops a specific named index:
  - `DROP INDEX Emp_ename;`
- If you drop a table, then all associated indexes are dropped
- To drop an index, the index must be contained in your schema or you must have the `DROP ANY INDEX` system privilege

Source: [https://docs.oracle.com/cd/B10500\\_01/appdev/920/a96510/index.htm#i1000](https://docs.oracle.com/cd/B10500_01/appdev/920/a96510/index.htm#i1000)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kanpur Date: Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

30.21

©Silberschatz, Korth and Sudarshan

The last rule 6 is drop index that are no longer required. So, if an index might be dropped because for several reasons for example, it is it simply does not speed up the queries. So, table might have become too small there will be many rows in the table, but very few index increase right we have seen these are not the ideal.

So, it may not have been the case earlier and now it may be the case. So, in that case that index should be drop because it is not helping you the queries in your application do not use the index the query you have certain indexes and the queries are done on other attributes or other composite attributes. So, it is not the indexes of no value and; obviously, index must be dropped before being rebuilt if you are rebuilding if you are creating new index in a new way then. So, make a judgment and drop indexes which are no longer required as an when you observe that in drop indexes and improve the performance.

When you drop an index all extents of the indexes segment are return to the tablespace. So, this is basically the space management and SQL command for this you already know now please keep in mind that if you drop a table then all associated indexes are automatically dropped because; obviously, if the data is not there then how about their index. So, to drop an index you need the drop any index system privilege we talked about privileges earlier too.

(Refer Slide Time: 23:26)

**Module Summary**

- Learned to create Indexes in SQL
- Introduced a few rules for good index

SWAYAM: NPTEL-NOC MOOCs  
Instructor: Prof. P. P. Deshpande, IIT Kharagpur  
Date: Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition  
30.22  
©Silberschatz, Korth and Sudarshan

So, this summarizes our discussions on indexing and hashing. So, here in this particular module you have learned to create index in SQL and introduce few rules for good index. Overall in this week in all the 5 modules we have learnt about how to speed up query processing, how to speed up the execution of access insert delete queries in your database through the lifetime. And we have looked at various different indexing schemes, we have looked at hashing and made comparisons.

So, one take back that you can certainly have is the most important indexing scheme or indexing structure is the B + tree which can be used for data files as well as for index files and several like SQL server uses the B+ tree only. Now, hashing options we have looked at and we have seen that it has a varied acceptability it is a powerful technique, but not all systems use that equally strongly.

And we have then made understood that indexing a database or tables on different attributes is a very delicate responsibility which has to be done with a lot of judgment. And for that statistics must be rightly collected and good judgment in terms of the distribution of the data, access of the data nature of queries all these need to be considered carefully so, that you can really get good performance from the design that you have.

So, on top of the knowledge of good design that you acquired through the all the theory of normalization and you know redundancy removal; your good judgment in terms of

good appropriate index design will take you a long way in terms of making a very good database system engineer.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 31**  
**Transactions/1**

Welcome to module 31 of Database Management System. This module and the few following it will be on transactions. So, we have so far, through the modules that you have done we have so far looked at the first the schema of a database system, which is the plan the layout of how the data will be organized. Then we have looked at if the data is populated, then how we can query how we can manipulate the data.

We have looked at how the data is actually physically stored, and how it can be efficiently accessed through different mechanisms in the storage. Now we will focus on the actual execution time. We will focus on what goes on when the data in a database system is accessed, it is read, locally modified and then written back. In a very simple terms this is the operation that keeps on happening in the database systems, which we will identify which we say are transactions.

(Refer Slide Time: 01:40)

**Week 06 Recap**

PPD

<ul style="list-style-type: none"><li>▪ <b>Module 26: Indexing and Hashing/1 (Indexing/1)</b><ul style="list-style-type: none"><li>◦ Basic Concepts of Indexing</li><li>◦ Ordered Indices</li></ul></li><li>▪ <b>Module 27: Indexing and Hashing/2 (Indexing/2)</b><ul style="list-style-type: none"><li>◦ Balanced Binary Search Trees</li><li>◦ 2-3-4 Tree</li></ul></li><li>▪ <b>Module 28: Indexing and Hashing/3 (Indexing/3)</b><ul style="list-style-type: none"><li>◦ B+-Tree Index Files</li><li>◦ B-Tree Index Files</li></ul></li></ul>	<ul style="list-style-type: none"><li>▪ <b>Module 29: Indexing and Hashing/4 (Hashing)</b><ul style="list-style-type: none"><li>◦ Static Hashing</li><li>◦ Dynamic Hashing</li><li>◦ Comparison of Ordered Indexing and Hashing</li><li>◦ Bitmap Indices</li></ul></li><li>▪ <b>Module 30: Indexing and Hashing/5 (Index Design)</b><ul style="list-style-type: none"><li>◦ Index Definition in SQL</li><li>◦ Guidelines for Indexing</li></ul></li></ul>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr- 2018

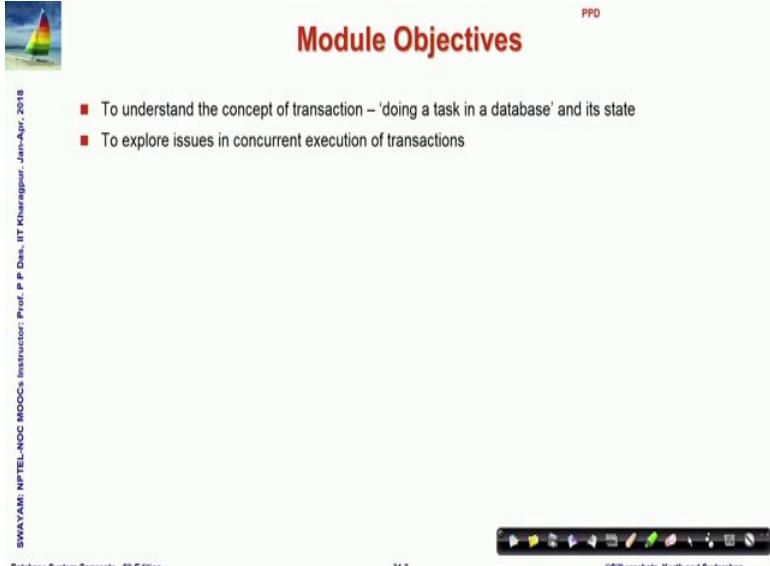
Database System Concepts - 8<sup>th</sup> Edition

31.2

©Silberschatz, Korth and Sudarshan

So, this is what we had done in the last week talking about index.

(Refer Slide Time: 01:46)



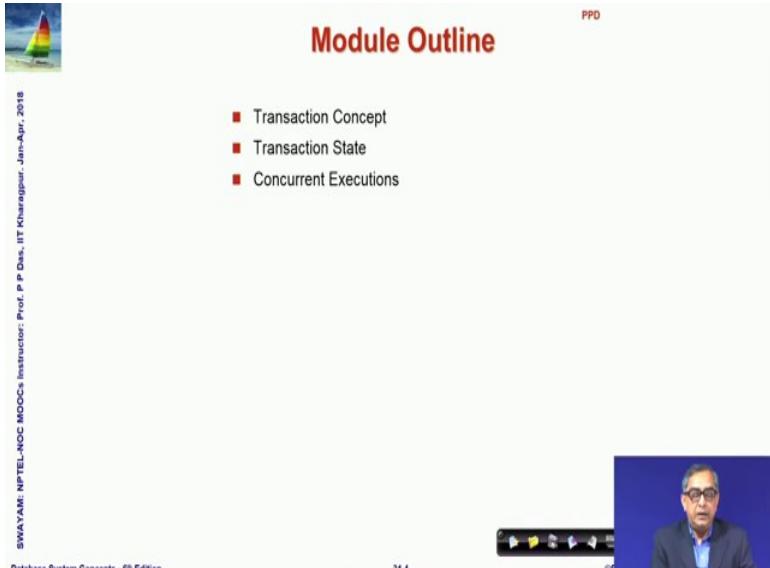
The slide is titled "Module Objectives" in red. It features a small sailboat icon in the top left corner and a "PPD" logo in the top right corner. A vertical sidebar on the left contains the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr. 2018". The main content area lists two objectives:

- To understand the concept of transaction – 'doing a task in a database' and its state
- To explore issues in concurrent execution of transactions

At the bottom, there is a navigation bar with icons for back, forward, search, and other presentation controls. The footer includes "Database System Concepts - 8<sup>th</sup> Edition", "31.3", and "©Silberschatz, Korth and Sudarshan".

And we now start with the understanding of this concept of transaction, and we explore various issues related to concurrent execution of transactions. So, we will explain in more detail what this mean.

(Refer Slide Time: 02:03)



The slide is titled "Module Outline" in red. It features a small sailboat icon in the top left corner and a "PPD" logo in the top right corner. A vertical sidebar on the left contains the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr. 2018". The main content area lists three topics:

- Transaction Concept
- Transaction State
- Concurrent Executions

At the bottom, there is a video player showing a man speaking, a navigation bar with icons, and the footer "Database System Concepts - 8<sup>th</sup> Edition", "31.4", and "©Silberschatz, Korth and Sudarshan".

And these are the 3 topics that we will focus on in this module.

(Refer Slide Time: 02:13)

The slide has a header 'Transaction Concept' with a sailboat icon. The content includes a list of points about transactions, a code snippet for a transfer, and a note about issues. It also includes copyright information and navigation icons.

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018

**Transaction Concept**

- A **transaction** is a *unit* of program execution that accesses and possibly updates various data items
- For example, transaction to transfer \$50 from account A to account B:
  1. `read(A)`
  2. `A := A - 50`
  3. `write(A)`
  4. `read(B)`
  5. `B := B + 50`
  6. `write(B)`
- Two main issues to deal with:
  - Failures of various kinds, such as hardware failures and system crashes
  - Concurrent execution of multiple transactions

Database System Concepts - 8<sup>th</sup> Edition      31.6      ©Silberschatz, Korth and Sudarshan

So, let us first take a look at what does a transaction mean. We say transaction is a unit of program execution that accesses and possibly updates, we data items.

So, it reads possibly makes some local changes and then it writes it back. So, here is an example of a transaction. So, without that detail unnecessary details what it looks at there are two accounts account A and B, and we want to transfer 50 dollars from account A to account B. So, we want to we need to debit account A by 50 dollars, and then credit account B by 50 dollars that will achieve the transfer. So, to start this process we first need to know what is the current balance of the account A.

So, that is done by read the first instruction. Then so, A after reading a has come into a local buffer into a temporary which exists in memory, if the current balance was 200 dollar, then that has not changed that remains to be 200 dollar a has become is a local variable which is taken the value 200 dollar now. So, we locally change it we debit 50 dollars from that. So, a becomes 150 dollar, and then we write it back. So, in the account balance where it was 200 dollar, it will now become 100 and 50 dollar with this 50 dollars debited.

Then we have to do the credit process to the account B. So, in the 4th instruction we read B. So, let us say the current balance of account B was 300 dollar, then read B will make the temporary variable B as 300 we credit 300; that is, we add 50 dollars to that it

becomes 350, and then we write B onto the account based balance. So, account-based balance will now change to 350 dollar.

So, this whole sequence of 6 instructions is called a transaction. And as you can understand that to achieve our target of transferring 50 dollars from account A to account B, all these six instructions have to execute in this order so that we can get the desired results. Now the question is; so, this is pretty simple, this is like a very simple low-level program. But there are two main issues that we have to deal with. First, what is the guarantee that once the instruction one starts? What is the guarantee that it will continue up to instruction 6?

There may be some failures in between the disk may fail the hardware may fail the system may crash. So, what will happen to the state of the database? What will happen to the values that exist in the database? What will happen to the target operation that we wanted to do achieved through this transaction if such failures happen. A second issue is, we need multiple transactions to execute concurrently. What it means that, suppose I am working on a net banking updating my account I am making transfers to another party whom I need to pay. At the same time, several other people are also doing operations on their accounts, respective accounts.

Lot of other operations might happen from the database itself. For example, while I am making a transfer at that same time the database may be crediting some quarterly interest to my account. All these transactions, actually execute concurrently, which means that, they all are independently executing. They use the same CPU, but they achieve the result at the same time. So, it is not that the transactions are actually happening on separate machines, the transactions have to take effect on the same database.

So, they have to occur in a concurrent manner, that is what we see is a concurrent manner because they occur together. And while this is going on, how do we ensure, but there is one CPU. So, the CPU is executing these instructions in some order. So, how do we ensure that this in the face of such concurrency the transactions will still give me correct result? So, these are the two major issues for which we are going to study about transactions, and what we in general say the transaction management systems.

(Refer Slide Time: 07:25)

The slide has a header 'Required Properties of a Transaction' with a sailboat icon. On the left, there is a vertical sidebar with text: 'SWAYAM-NIETL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr- 2018'. Below the sidebar is a video player showing a man speaking. The main content area contains a section titled 'Atomicity requirement' with three bullet points. It also includes a code snippet for a transaction transfer and copyright information.

- Atomicity requirement
  - If the transaction fails after step 3 and before step 6, money will be "lost" leading to an inconsistent database state
    - ▶ Failure could be due to software or hardware
  - The system should ensure that updates of a partially executed transaction are not reflected in the database

Transaction to transfer \$50 from account A to account B:

```
1. read(A)
2. A := A - 50
3. write(A)
4. read(B)
5. B := B + 50
6. write(B)
```

Database System Concepts - 8<sup>th</sup> Edition      31.7      ©Silberschatz, Korth and Sudarshan

So, we first set the targets we put some required properties of a transaction. The first requirement is atomicity.

Suppose again just look at the same transaction, suppose the system crashes there is a system failure after the first three instructions has happened and 4<sup>th</sup> instruction was about to happen. So, what will happen? The already the account A has been debited by 50 dollars and account B has not yet been credited with that 50 dollars. So, simply at this point if the transaction if the system failure happens, then simply 50 dollars will disappear from the system it will not exist. So, the basic requirement is that once a transaction start it should either completely happen it should either do all the 6 instruction as in this case or it should do nothing.

So, there is an all or none kind of requirement that is what we say it is like. So, transactions in a way are indivisible or atomic and this is the atomicity requirement.

(Refer Slide Time: 08:35)

The slide has a header 'Required Properties of a Transaction' with a sailboat icon. It includes a sidebar for 'SYNOPSIS: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr- 2018'. The main content is a section on 'Consistency requirement' with a bulleted list and a code example for a transaction transfer.

**Consistency requirement**

- In example, the sum of A and B is unchanged by the execution of the transaction
- In general, consistency requirements include
  - Explicitly specified integrity constraints
    - primary keys and foreign keys
  - Implicit integrity constraints
    - sum of balances of all accounts, minus sum of loan amounts must equal value of cash-in-hand
- A transaction, when starting to execute, must see a consistent database
- During transaction execution the database may be temporarily inconsistent
- When the transaction completes successfully the database must be consistent
- Erroneous transaction logic can lead to inconsistency

Transaction to transfer \$50 from account A to account B:

- read(A)
- $A = A - 50$
- write(A)
- read(B)
- $B = B + 50$
- write(B)

The second requirement is called consistency requirement. That is as the transactions are making changes to the database at, through these changes the integrity of the database the consistency of the values should not get affected.

So, if we there are certain specific integrity constraints we have talked of primary keys foreign keys and so on. And there could be implicit domain integrity constraints. For example, in this accounting case if we are making transfers, then while making a transfer from account A to account B the sum of the balances in account and account B before the transfer and after the transfer must be same.

So, money should not disappear, neither should get should it get generated. So, what we assume that a transaction when it starts to execute. It must start in a consistent database which is correct in every respect. During the transaction there may be temporary inconsistency. For example, if you look at instruction 4 or instruction 5 at this time, the account A has already been debited by 50 dollars the account B has not been credited by that 50 dollar. So, if you add instruction 4 if you try to see, what is the sum of the balance in account A and account B you will see that sum is 50 dollars less. But when the transaction completes, it completes the instruction 6, then again, the sum will be same as thus as it were at the beginning.

So, at the beginning of an execution, and at the end of a successful execution the database must be consistent in between there may be transient inconsistency. So, this is called the consistency requirement.

(Refer Slide Time: 10:28)

**Required Properties of a Transaction (Cont.)**

**Isolation requirement**

- If between steps 3 and 6 (of the fund transfer transaction) , another transaction **T2** is allowed to access the partially updated database, it will see an inconsistent database (the sum  $A + B$  will be less than it should be).

T1	T2
1. <code>read(A)</code> 2. <code>A := A - 50</code> 3. <code>write(A)</code>	<code>read(A), read(B), print(A+B)</code>  4. <code>read(B)</code> 5. <code>B := B + 50</code> 6. <code>write(B)</code>

- Isolation can be ensured trivially by running transactions **serially**
  - That is, one after the other
- However, executing multiple transactions concurrently has significant benefits

SWAYAM: NPTEL-NOC/NOCCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
31.9  
©Silberschatz, Korth and Sudarshan

The third is again first look at the example the same on the left is a transaction T1 which is the transaction we have been talking of. And suppose there is another transaction T2 which happens concurrently. If it happens concurrently the transaction T2 has let us say 3 instructions read (A) read (B) and print (A + B). So, it tries to read the balance of account A and B and prints their sum.

Obviously, if the transaction T2 is allowed these 3 instructions of transaction T2 if they are allowed to be executed; between instruction 3 and instruction 4 of transaction T1, then T2 will print a sum of  $A + B$  which is , than the  $A + B - 50$  at the beginning. So, it will become it will appear as if there is some inconsistency that has happened.

So, the isolation requirement says that when transactions occur concurrently, the net effect of the transactions should be as if they happen either first T1 happened and then T2 happen. Over first u or T2 executed and then T1 executed. The though even though they can may execute in a concurrent or mixed manner, the result of such inconsistent state of the database should not be available to the other transactions. So, this is called the isolation requirement.

So, that transactions need to be isolated appropriately; so that they can obviously if they execute serially then the isolation is trivially satisfied, but that will mean that your throughput, your performance will be very low. So, we need transactions to happen concurrently, but the isolation must be satisfied.

(Refer Slide Time: 12:24)

The slide has a title 'Required Properties of a Transaction' in red at the top right. To the left of the title is a small icon of a sailboat on water. On the far left, there is vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Date, IIT Kharagpur - Jan-Apr- 2018'. At the bottom left is the text 'Database System Concepts - 8<sup>th</sup> Edition'. In the center, under the title, is a section titled 'Durability requirement' with a bullet point: 'Once the user has been notified that the transaction has completed (i.e., the transfer of the \$50 has taken place), the updates to the database by the transaction must persist even if there are software or hardware failures'. Below this is a code block for a transaction transfer: 'Transaction to transfer \$50 from account A to account B:' followed by a numbered list of six steps: 1. `read(A)`, 2. `A := A - 50`, 3. `write(A)`, 4. `read(B)`, 5. `B := B + 50`, 6. `write(B)`. At the bottom right are navigation icons and the text '©Silberschatz, Korth and Sudarshan'.

The 4th is called the durability requirement, which says that if a transaction has finally completed successfully. Then the update the changes that the database that has been made by the transaction, that must persist even if there is some software or hardware failures in future so once.

This transaction of transferring 50 dollar from A to B has successfully completed with the six instructions having been executed and the money have been transferred, that will should persist even if subsequently some error some failures in the database will occur. So, it must be the changes must be durable.

(Refer Slide Time: 13:11)

## ACID Properties

A **transaction** is a unit of program execution that accesses and possibly updates various data items. To preserve the integrity of data the database system must ensure:

- **Atomicity:**
  - Either all operations of the transaction are properly reflected in the database or none are
- **Consistency:**
  - Execution of a transaction in isolation preserves the consistency of the database
- **Isolation:**
  - Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions
  - That is, for every pair of transactions  $T_i$  and  $T_j$ , it appears to  $T_j$  that either  $T_i$  finished execution before  $T_j$  started, or  $T_j$  started execution after  $T_i$  finished
- **Durability:**
  - After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018

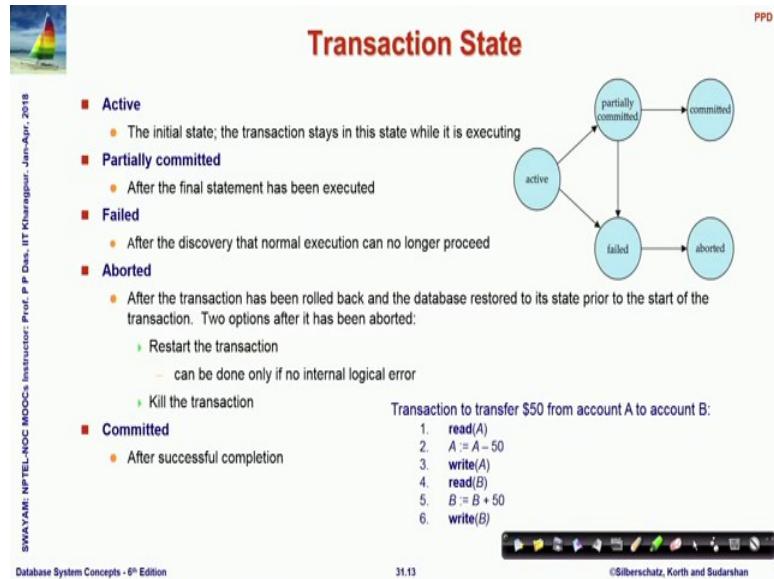
Database System Concepts - 8<sup>th</sup> Edition      31.11      ©Silberschatz, Korth and Sudarshan

So, these 4 properties are together called the acid properties of a transaction system. So, acid means a for atomicity that either all operation of the transaction are properly reflected in the database or none of them are reflected consistency c for consistency execution of a transaction in isolation preserves the consistency in the database.

The isolation requirement, that if multiple transactions are occurring concurrently, transaction  $T_i$ ,  $T_j$  occurring concurrently that is some instruction of  $T_i$  happen then some instructions of  $T_j$  happen then some instruction of  $T_i$  again happen and in this manner. Even then, the final result should look like as if  $T_i$  has happened followed by  $T_j$  or  $T_j$  has first executed followed by  $T_i$  the isolation i for isolation and finally, durability once the successfully transactions have completed the changes in the database should persist. So, A C I D the acid properties are the critical properties of the transaction system and must always be satisfied.

Next what we look at is as transactions go through each and every instruction.

(Refer Slide Time: 14:29)



The transaction happened to be in one of the different states. So, while the transaction as soon as the transaction starts and starts executing starting from the initial state it is in an active state. So, consider this same transaction is done read it is in active state it has decremented A by 50 it is in active state and so on. So, as long as it is executing, it is in active state, unless it has first let us talk about success.

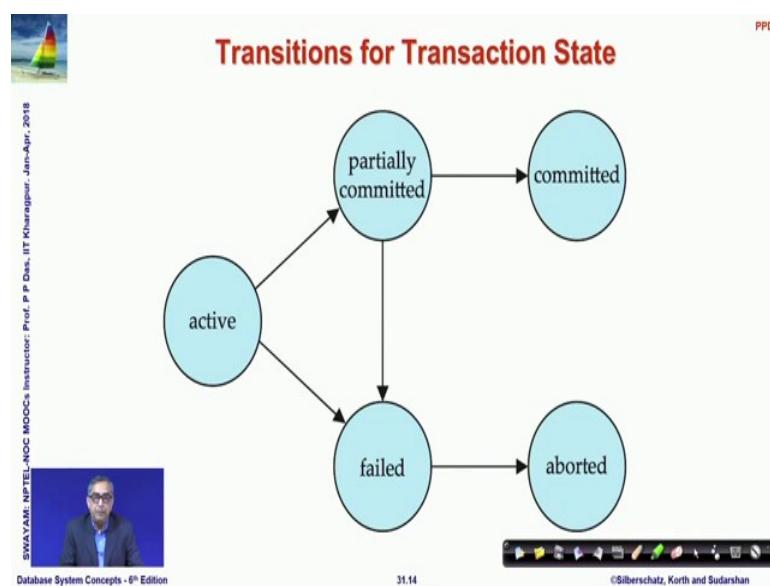
So, once it has executed the last treatment, last instruction that is instruction 6 here, it is in a state that is called partially committed. So, it has been able to successfully complete all the instructions. Or it might happen that during being in the active state, or being in the partially committed state some errors has happened so that the normal execution cannot proceed any further. Then, the transaction comes in to the failed state. A transaction which is in the failed state will eventually get aborted, because it is not known when the failure has happened.

So, naturally at the time of failure there could be an inconsistency failure could have happened in the 4<sup>th</sup> instruction in this transaction and as you have already noted. That A has already been debited by 50 dollars and B has not been credited that 50 dollars so, it is in an inconsistent state. Say failure if the transaction is in a failed state like this then we need to rollback, we need to undo the changes that we have done. We need to credit back the 50 dollars that was debited from A, so that we can reach a consistent state.

And once we have done that once we have done this rollback successfully, the transaction goes to an aborted state that is it could not take place, and after that you have 2 choices either you can restart the transaction, or you can totally kill the transaction do not do it at all depending on different situation that choice is made. In the other case if it is it were partially committed, then all instructions had completed, now the bookkeeping and other actions were required. If there is some failure during that time from partially committed it comes to fail state, and then goes to abort state as I have already explained. Or it actually commits all the changes correctly and it has completed successfully and it goes to a committed state where the transaction has successfully finished.

So, every transaction will go through this state at any point of time a transaction will be in one of the states, and depending on the status of execution it will continue to remain in that state or will change state.

(Refer Slide Time: 17:26)



So, this state transition diagram for transactions are very important, and you must thoroughly understand what is happening and remember this particular state transition ok. Now let us look into the actual concrete execution situations.

(Refer Slide Time: 17:44)

The slide has a header 'Concurrent Executions' with a sailboat icon. The main content lists advantages of concurrent executions and describes concurrency control schemes. A video player window shows a speaker, and the footer includes course information and navigation icons.

**Advantages of Concurrent Executions:**

- Multiple transactions are allowed to run concurrently in the system. Advantages are:
  - Increased processor and disk utilization**, leading to better transaction *throughput*
    - For example, one transaction can be using the CPU while another is reading from or writing to the disk
  - Reduced average response time** for transactions; short transactions need not wait behind long ones

**Concurrency control schemes** – mechanisms to achieve isolation

- That is, to control the interaction among the concurrent transactions in order to prevent them from destroying the consistency of the database

SWAYAM-NPTEL-NOOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
31.16 ©Silberschatz, Korth and Sudarshan

So, in the concurrent execution situation, what we have we have multiple transactions that run at the same time on the system. So, that will advantages it will increase throughput, it will increase processor and disk realization for example, when one transaction is doing some operations with on the CPU, some internal computations are going on the disk can still be accessed by another transaction to read or write some values.

So, the throughput will increase and also the average response time will reduce because there may be a short transaction which if it were serially done then it will have to wait for a very long transaction, which may already been executed executing, but if we allow concurrent execution then in between that long transaction few cycles may be taken to execute the short transaction and the average response time will improve.

So, that is our basic requirement. Naturally we need to do this in a controlled manner so that we ensure that the acid property is the consistency of the database and the acid properties are maintained.

(Refer Slide Time: 18:50)

The slide has a header 'Schedules' in red. On the left, there is a small sailboat icon. The main content is a bulleted list under the heading 'Schedule'. The footer contains the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '31.17', and '©Silberschatz, Korth and Sudarshan' along with a set of navigation icons.

- **Schedule** – a sequences of instructions that specify the chronological order in which instructions of concurrent transactions are executed
  - A schedule for a set of transactions must consist of all instructions of those transactions
  - Must preserve the order in which the instructions appear in each individual transaction
- A transaction that successfully completes its execution will have a **commit** instructions as the last statement
  - By default transaction assumed to execute commit instruction as its last step
- A transaction that fails to successfully complete its execution will have an **abort** instruction as the last statement

So, for doing this we create what is called a schedule? A schedule is a sequence of instructions that specify, the chronological, or the time wise order in which instructions of concurrent transactions are executed. So, what is what will the schedule will have? Scheduler will have for a set of it is defined for the set of transactions. And it must consist of all instructions of those transactions. And in a certain order, and what is the basic requirement that in this schedule in this ordering, the original order of instructions in any of this given transaction, you have an individual transaction must be preserved.

But the instructions from different transaction can be interleaved, intermixed in between to prepare the schedule. So, a transaction that successfully completes its execution will perform what is called a **commit** instruction we will more specifically say what is **commit a commit instruction**, which means successful completion as the last statement that should be the last statement if the committee is not given by default also transactions which have executed successfully are assumed to have executed commit, or if the transaction fails to successfully complete the execution; that means, we will do abort as a last statement ok.

(Refer Slide Time: 20:12)



### Schedule 1

PPD

■ Let  $T_1$  transfer \$50 from A to B, and  $T_2$  transfer 10% of the balance from A to B  
 ■ An example of a **serial** schedule in which  $T_1$  is followed by  $T_2$ :

$T_1$	$T_2$	A	B	A+B	Transaction	Remarks
read (A)		100	200	300	@ Start	
$A := A - 50$		50	200	250	$T_1$ , write A	
write (A)		50	250	300	$T_1$ , write B	@ Commit
read (B)		45	250	295	$T_2$ , write A	
$B := B + 50$		45	255	300	$T_2$ , write B	@ Commit
write (B)						
commit						
	read (A)					Consistent @ Commit
	$temp := A * 0.1$					Inconsistent @ Transit
	$A := A - temp$					Inconsistent @ Commit
	write (A)					
	read (B)					
	$B := B + temp$					
	write (B)					
	commit					

Database System Concepts - 8<sup>th</sup> Edition      31.18      ©Silberschatz, Korth and Sudarshan

So, let us take an example so, again we are going back to the same example. So, we have two transactions T1 and T2. T1 transfers 50 dollars from A to B as we have seen. And T2 transfers 10 percent of the balance from A to B. So, one transaction debits 50 dollars one transaction debits 10 percent of the account balance of A to B. So, if they are serially executed as you can see here we are serially executing them as in. So, first you first your whole of T1 is executing, and once this has committed, that is successfully ended then T2 is executing.

So, at the beginning if we assume this is just an assumption. If we assume at the beginning that A had 100 dollar and B had 200 dollar, then the sum was 300 dollar. So, if the A is read 100 dollar is read then it becomes 50 then you write this A. So, when you are here at this point, you can see, this is what you will have because A has changed from 100 to 50 because you have debited B nothing has happened on B so, sum is 250. So, you can see that is why I have shown different colors you can see at 250. This state of the database is temporarily inconsistent because the sum has become different from 300.

Then it reads B it reads 200 adds that 250 it writes B. You come to this point where after this write, when the commit is happening after the writing this B. Then T1 has actually completed, and 50 dollars has got transferred to account B, and the sum is again back to 300 so, consistency is preserved. Then transaction 2 starts so, A is read 50 dollars is read

in temporary you compute 10 percent of that 5 dollar you decrement a by 5 dollar and write it back.

So, when you have written it back, you write back 45 dollar. Again, naturally the sum becomes 5 dollar less, the 5 dollar that you have kept in this temp, and this becomes again transitively inconsistent in the process. Then you do the read B, ad that temporary 5 dollar back to B and then you finally, when you write B here you write back from 200 and 50 you have added 5 dollar to 255, and again the sum becomes 300 you are again back to the consistent state.

So, you can see, through this process that when transactions actually happen in a serial manner, this is how things will move on so, which is quite understandable.

(Refer Slide Time: 23:05)

**Schedule 2**

PPD

■ A serial schedule in which  $T_2$  is followed by  $T_1$ :

$T_1$	$T_2$	A	B	$A+B$	Transaction	Remarks
	read(A) temp := A * 0.1 A := A - temp write(A) read(B) B := B + temp write(B) commit	100	200	300	@ Start	
		90	200	290	$T_2$ , write A	
		90	210	300	$T_2$ , write B	@ Commit
		40	210	250	$T_1$ , write A	
		40	260	300	$T_1$ , write B	@ Commit

Consistent @ Commit  
Inconsistent @ Transit  
Inconsistent @ Commit

Values of A & B are different from Schedule 1 – yet consistent

SWAYAM: NPTEL-NOCOCS Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

31.19

©Silberschatz, Korth and Sudarshan

So, let us move on let us. So, this is a different schedule you can see, but this is also a serial schedule here what we have assumed that all instructions of T 2 are done first then all instructions of T 1. I am not going through the going through each step you can see what are the consistent, and the temporarily inconsistently states of the database, but at the end the database is in a consistent state.

And you can note that now the end value of a is 40 dollar, and n value of B is 260 dollar. In the previous schedule, the value was 45 dollar, and 255 dollar these two are different. But both of them are actually correct both of them are consistent, because when things

happen in this distributed manner, we have no control in terms of whether that whether first 50 dollars should be debited and then 10 should be debited transferred. Or whether first 10 should be transferred or 50 dollars where will be transferred after that, either of that is a correct consistent state.

So, the different schedules might give you different results that is not of any concern because both of them are possible valid results. But the question is it must finally, have a consistent state of the database so, both of these are consistent.

(Refer Slide Time: 24:20)



**Schedule 3**

PPD

**■ Let  $T_1$  and  $T_2$  be the transactions defined previously. The following schedule is not a serial schedule, but it is equivalent to Schedule 1**

$T_1$	$T_2$	$T_1$	$T_2$	A	B	$A+B$	Transaction	Remarks
read ( $A$ ) $A := A - 50$ write ( $A$ )		read ( $A$ ) $A := A - 50$ write ( $A$ )		100	200	300	@ Start	
	read ( $A$ ) $temp := A * 0.1$ $A := A - temp$	read ( $B$ ) $B := B + 50$	write ( $B$ )	50	200	250	$T_1$ , write $A$	
read ( $B$ ) $B := B + 50$	write ( $A$ )	write ( $B$ )	commit	45	200	245	$T_2$ , write $A$	
commit				45	250	295	$T_1$ , write $B$ @ Commit	
	read ( $B$ ) $B := B + temp$	read ( $A$ ) $temp := A * 0.1$ $A := A - temp$	write ( $B$ )	45	255	300	$T_2$ , write $B$ @ Commit	
	commit	write ( $A$ )	commit				Consistent @ Commit	
							Inconsistent @ Transit	
							Inconsistent @ Commit	

Schedule 3                      Schedule 1

SWAYAM: NPTEL-NOC MOOCs; Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018

Note – In schedules 1, 2 and 3, the sum “ $A + B$ ” is .....!

Database System Concepts - 8<sup>th</sup> Edition                      31.29                      ©Silberschatz, Korth and Sudarshan

Now, take an interesting example, where schedule 3 where in here if you, if you look at carefully there are few instructions of  $T_1$  which are executed. And then in the temporal order few other instructs, few instructions of  $T_2$  are executed, then again  $T_1$  then again  $T_2$ .

So, the instructions from two different transactions are getting interleaved. And this is what the execution status would be. So, you can see that when you are when  $T_1$  writes  $A$  this is where you are 50 dollars has been debited. Then when  $T_2$  writes  $A$  subsequently, another 5 dollar is debited so, it becomes 45. So, then you have  $T_1$  again executing and adding  $B$  on to that. And by that it is not only that it has gone into an inconsistent, it is it was already in an inconsistent state, but that was transient that was temporary. But now the transaction  $T_1$  has totally completed. It is completed his execution it is at it is commit, but your database is still in an inconsistent state.

So, this is something which is possible, because you are doing an interleaving of the instructions of the two transactions in the schedule. But once you allow the rest of the transaction B this part to complete that is B gets updated and you reach here. And that also has committed. So, your schedule comprised of transaction 1 and transaction 2, when both of them have completed you have again reached state which is consistent. And if you look at the results of what you have achieved you will immediately identify that this doing it doing the transactions according to schedule 3, which is interleaving in this manner is equivalent to this in this manner of interleaving is equivalent to doing them according to in this manner which is schedule 1.

So, you have got a schedule which is equivalent to schedule one. And it is therefore, so, this is just an example to show that it is actually possible to interleave the instructions of 2 transactions, and create a schedule which will still which might in the process have transient or even inconsistent commit states of the database. But finally, when the schedule ends it will it is possible that it will bring you to a consistent state.

(Refer Slide Time: 26:58)

**Schedule 4**

The following concurrent schedule does not preserve the sum of "A + B"

T <sub>1</sub>	T <sub>2</sub>	A	B	A+B	Transaction	Remarks
read (A)		100	200	300	@ Start	
A := A - 50	read (A)	90	200	290	T2, write A	
	temp := A * 0.1	90	200	290	T1, write A	
	A := A - temp	90	250	340	T1, write B	@ Commit
	write (A)	90	260	350	T2, write B	@Commit
write (B)						
read (B)						
B := B + 50						
write (B)						
commit						
	B := B + temp					
	write (B)					
	commit					

Consistent @ Commit  
Inconsistent @ Transit  
Inconsistent @ Commit

SWAYAM: NPTEL-MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 8<sup>th</sup> Edition

31.21

©Silberschatz, Korth and Sudarshan

Now, look at again for those transactions look at a different interleaving, a different schedule, again T 1, T2 are involved. But you have now tried to interleave them in a different order. So, earlier the interleaving was done after T1 has done write A, here it is done after he has been the locally debited by 50. And then this part is done and then the

write is happening. And now if you go through the steps I will leave it as an exercise for you in schedule 4.

Now, if you go through the state you will find that when transaction T1 commits ends here, you have an inconsistent state. And finally, even when the schedule ends that T2 has committed. There is A, you are in an inconsistent state somehow that sum of A and B which was 350 has become 300 has become 350 so, 50 dollars as what generated. So, this is so you can see that if you interleave the transactions, then it is quite possible that the transactions will may or may not actually give you a consistent data base.

(Refer Slide Time: 28:07)

**Module Summary**

- A task is a database is done as a transaction that passes through several states
- Transactions are executed in concurrent fashion for better throughput
- Concurrent execution of transactions raise serializability issues that need to be addressed
- All schedules may not satisfy ACID properties

SWAYAM: NPTEL NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr 2018

Database System Concepts - 8<sup>th</sup> Edition      31.22      ©Silberschatz, Korth and Sudarshan

So, here in this module, we have understood the basic tasks that a data base performs database executes; which is in form of a transaction. And we have seen that they must satisfy a set of properties typically called the acid properties, and atomicity, consistency, isolation and durability must be satisfied. And when the transactions are executed in concurrent fashion, we improve the throughput, but the concurrent execution of transaction raise issues of serializability; that is, the concurrent execution that the interleaved schedule of instruction of two or more transactions can give rise to certain effect which violate the acid properties.

And those need to be addressed that certainly inconsistent database is certainly never acceptable. And so that is the basic problem that we have identified which we will have to address in the coming modules.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 32**  
**Transactions/2: Serializability**

Welcome to module 32 of Database Management Systems, from the last module we are discussing about transactions and transaction management.

(Refer Slide Time: 00:27)

The slide has a header 'Module Recap' in red. On the left, there is a small image of a sailboat on water. A vertical sidebar on the left contains the text 'SWAYAM: NPTEL-NOOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. The main content area lists three bullet points: '■ Transaction Concept', '■ Transaction State', and '■ Concurrent Executions'. At the bottom, there is footer text: 'Database System Concepts - 8<sup>th</sup> Edition', '32.2', and '©Silberschatz, Korth and Sudarshan'. There is also a standard presentation navigation bar at the very bottom.

And we have technical look into the basic concept of a transaction the transaction state and the issues.

(Refer Slide Time: 00:35)

The slide is titled "Module Objectives" in red bold font at the top right. At the top left is a small sailboat icon. The title "Module Objectives" is centered above a bulleted list of three items. The footer contains course details and navigation icons.

**Module Objectives**

- To understand the issues that arise when two or more transactions work concurrently
- To introduce the notions of Serializability that ensure schedules for transactions that may run in concurrent fashion but still guarantee and serial behavior
- To analyze the conditions, called conflicts, that need to be honored to attain Serializable schedules

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur- 2018

Database System Concepts - 8<sup>th</sup> Edition

PPD

32.3

©Silberschatz, Korth and Sudarshan

In concurrent execution and in this module, we will look try to understand, what are the very specific issues that happen when two or more transactions work concurrently we have seen that now it is possible that they execute in a schedule, which would not let us preserve the acid properties.

So, we want to introduce the very basic concept of making sure that such concurrent execution schedules are acceptable, and those are the notions of serializability. And we will analyze different conditions called conflicts that need to be honored to attend the serializability of the schedules.

(Refer Slide Time: 01:17)

Module Outline

PPD

- Serializability
- Conflict Serializability

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Deshpande, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

32.4

©Silberschatz, Korth and Sudarshan

So, serializability is the main topic to discuss.

(Refer Slide Time: 01:21)

Serializability

PPD

- **Basic Assumption** – Each transaction preserves database consistency
- Thus, serial execution of a set of transactions preserves database consistency
- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule. Different forms of schedule equivalence give rise to the notions of:
  1. **conflict serializability**
  2. **view serializability**

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Deshpande, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

32.6

©Silberschatz, Korth and Sudarshan

So, to understand serializability we make a basic assumption, we make an assumption that every transaction by itself preserves the database consistency. That is, it starts in a consistent state of the database. And through the execution of its instructions in the order given it leaves the database in a consistent state, that is satisfied in each and every transaction. So, we can conclude that, if we serially execute a set of instruction set of transactions, then the consistency of the database will always be preserved.

Now, the problem happens, and as we have seen in the last module, that problems happen when possibly concurrent transactions happen. And we may execute may be executing the instruction in an order which leads to the violation of acid properties, the consistency in particular. So, we say that a concurrent schedule is serializable, if there is a there is some serial schedule, you say what is the serial schedule serial schedule is where the transactions are executed one after the other.

So, if you have 2 transactions in the concurrent system, then if I do T 1 then I do T 2 it is a serial schedule. If I do T2 and then I do T1 it is a serial schedule as well. So, if I have a concurrent schedule, if you refer back to the last module schedule 3; where the instructions of T1 and T2 are interleaved, then it is it will have to be equivalent to a serial schedule either T1 after T2 or T2 after T1. Different forms of schedule equivalence is used one is called conflict serializability, and the other is called view serializability. In the present module we will first discuss conflict serializability.

(Refer Slide Time: 03:19)

The slide has a header 'Simplified view of transactions' with a sailboat icon. The main content is a bulleted list:

- We ignore operations other than **read** and **write** instructions
  - Other operations happen in memory (are temporary in nature) and (mostly) do not affect the state of the database
  - This is a simplifying assumption for analysis
- We assume that transactions may perform arbitrary computations on data in local buffers in between reads and writes
- Our simplified schedules consist of only **read** and **write** instructions

At the bottom left is a video thumbnail of a speaker, and at the bottom right is a navigation bar.

Now, we make now a transaction may have all varied kinds of instructions, but we make an assumption that we will ignore anything other than any instruction other than the read and write instruction. Because other operations like we saw an operation where an account is debited by 50 or account is credited. So, you subtract 50 you add 50 you multiply by 0.1 or things like that are all operations that happen in the local buffer in the memory, and never temporary in nature and mostly they do not affect the state of the

database, because you have read the data do the changes write it back. So, it is a read and write that actually are important for that maintaining the consistency after database. So, that simplifies our process of analysis to a good extent.

So, this is so, we assume that between every read and write or read and read write and write and so on, the database the transactions may be doing arbitrary computations, which are all in the local buffer and do not affect the state. So, we can make this assumption that our shift schedules consist only of read and writing.

(Refer Slide Time: 04:31)

The slide has a title 'Conflicting Instructions' in red at the top right. On the left is a small image of a sailboat on water. The main content area contains the following text:

■ Let  $I_i$  and  $I_j$  be two instructions of transactions  $T_i$  and  $T_j$  respectively. Instructions  $I_i$  and  $I_j$  conflict if and only if there exists some item  $Q$  accessed by both  $I_i$  and  $I_j$ , and at least one of these instructions wrote  $Q$ .

- 1.  $I_i = \text{read}(Q)$ ,  $I_j = \text{read}(Q)$ .  $I_i$  and  $I_j$  don't conflict
- 2.  $I_i = \text{read}(Q)$ ,  $I_j = \text{write}(Q)$ . They conflict
- 3.  $I_i = \text{write}(Q)$ ,  $I_j = \text{read}(Q)$ . They conflict
- 4.  $I_i = \text{write}(Q)$ ,  $I_j = \text{write}(Q)$ . They conflict

■ Intuitively, a conflict between  $I_i$  and  $I_j$  forces a (logical) temporal order between them

- If  $I_i$  and  $I_j$  are consecutive in a schedule and they do not conflict, their results would remain the same even if they had been interchanged in the schedule

At the bottom left is a video thumbnail showing a man speaking. The bottom right shows navigation icons and the text 'Silberschatz, Korth and Sudarshan'.

Now, we say that suppose  $I_i$  and  $I_j$ , 2 instructions for belonging to transaction  $T_i$  and transaction  $T_j$ . So, there are two transactions  $T_i$  and  $T_j$ ,  $T_i$  has an instruction  $I_i$ ,  $T_j$  has an instruction  $I_j$  and we say that  $I_i$  ad  $I_j$  this instruction will conflict, if and only if there is some item  $Q$ , that is some data entity  $Q$ ; which both  $I_i$  and  $I_j$  are trying to access. And at least one of these instructions try to write.

So, these two instructions from true transactions are trying to manipulate the same data item, and at least one of them is trying to write. If that happens then we say that  $I_i$  and  $I_j$  these two instructions are conflicting. So, you can naturally enumerate the four possibilities, if both of them are reading their own conflict. If it is read write, write read, write All of them are cases of conflict.

So, naturally intuitively, you can figure out that since the write changes are value that if there is a conflict between these two instructions then there must be a fixed temporal order between them. So, if  $I_i$  and  $I_j$  are consecutive in a schedule and they do not conflict. Then we can interchange the temporal order of  $I_i$  and  $I_j$ , that will also not make a difference, because they do not conflict. But if they conflict I cannot make the change in their ordering.

(Refer Slide Time: 06:18)

The slide has a title 'Conflict Serializability' in red at the top right. To its left is a small sailboat icon. On the left side, there is vertical text: 'SWAYAM-NPTEL-NOJC-MOOCs Instructor: Prof. P. Deshpande, IIT Kanpur - Jan-Apr- 2018'. At the bottom left is a video player showing a man speaking. The bottom of the slide features a navigation bar with icons for back, forward, search, and other presentation controls. The footer includes the text 'Database System Concepts - 8<sup>th</sup> Edition', '32.10', and '©Silberschatz, Korth and Sudarshan'.

So, that gives rise to the notion of conflict serializability. So, we say if a schedule  $S$  can be transformed into another schedule  $S'$  by a series of swaps of non-conflicting instructions, then  $S$  and  $S'$  are conflict equivalent. So, what are you saying? That we have two one schedule  $S$ , and we will swap non-conflicting instruction, possibly since non-conflicting instructions that occur side by side. And if by doing this, if I can create the schedule  $S$  primed, then I will say  $S$  and  $S'$  are conflict equivalent. But if  $S$  and  $S'$  that, I cannot transform  $S$  into  $S'$  by just swapping non-conflicting instructions, then they are not conflict equivalent.

The second definition to keep in mind is a schedule  $S$  is conflict serializable, if it is conflict equivalent to a serial schedule, what is the serial schedule? Just to remind you serial schedule is one where the transactions are happened one after the other in a serial manner. So, all instructions of one transaction complete, then all instructions of the second transaction complete, then all instructions of the third transaction complete and so

on. So, if a schedule is conflict serializable; that is, if in a schedule. I can swap non-conflicting instructions. And make it into a serial schedule, and then I will say that the given schedule is a conflict serializable schedule ok.

(Refer Slide Time: 08:00)

**Conflict Serializability (Cont.)**

- Schedule 3 can be transformed into Schedule 6 – a serial schedule where  $T_2$  follows  $T_1$ , by a series of swaps of non-conflicting instructions.
  - Swap  $T_1.\text{read}(B)$  and  $T_2.\text{write}(A)$
  - Swap  $T_1.\text{read}(B)$  and  $T_2.\text{read}(A)$
  - Swap  $T_1.\text{write}(B)$  and  $T_2.\text{write}(A)$
  - Swap  $T_1.\text{write}(B)$  and  $T_2.\text{read}(A)$
- Therefore, Schedule 3 is conflict serializable:

$T_1$	$T_2$	$T_1$	$T_2$	$T_1$	$T_2$
read(A) write(A)	read(A) write(A)	read(A) write(A)	read(A) write(A)	read(A) write(A)	read(A) write(A)
read(B) write(B)	read(B) write(B)	read(B) write(B)	read(B) write(B)	read(B) write(B)	read(B) write(B)

**Schedule 3**

**Schedule 5**

**Schedule 6**

PPD

So now let us it is time for a number of examples to understand this better. So, we had seen schedule 3, will have to refer to the earlier module 4 schedule 3. Sir, no not I am sorry this is just abstracted form of that; not the actual one because in the in the earlier schedule 3 we had shown all the complete other computations also, but the read writes are the same.

Now, that this schedule 3 can be converted to so, this is where you have schedule 3, and you can easily see that the part of transaction  $T_1$  then a part of transaction  $T_2$ . So, schedule 3 is not a serial schedule, but if you can swap non conflicting instructions, then you are able to convert this into this schedule which if we are calling a schedule 6. Where all instructions of  $T_1$  is followed by all instructions of  $T_2$  which is a serial schedule.

So, since this can be done, we will say it is conflict serializable schedule 3 is conflict serializable and just to see how that happens. So, you start here let me erase this marks and start here. So, here if I look into these two instructions, which are the consecutive instructions in schedule 3 I can swap them; that is, I can do read B first and then do read A, I can swap read B and write A read B, and write A can be swapped.

Once I have done that, then I can swap read B with read A. It has become before right A can swap it with, because read B and write A, or write B read B and read A these do not conflict their non-conflicting instruction. Why read B and write A and non-conflicting, because they are not reading and writing to the same data item. Why read B and read A are non-conflicting, they are accessing the same data item, but both of them are read there is no write. So, I can swap so, this is the second one I can. So, once I do that read B will come here and write (A), read (A), write(A)will come down.

Then again, I can see that write B can be swapped with write A. Both are rights, but referring to different data items. Similarly, write B then can be swapped with read A, because they are again referring to different data items. So, I can do this and then these will also come up. So, I will eventually after these 4 swaps, this whole schedule 3 will transform into this serial 6, and we get a serial schedule.

So, we will say that schedule 3 is conflict serializable. That is the basic concept that we are trying to establish here.

(Refer Slide Time: 11:02)

$T_3$	$T_4$
read (Q)	write (Q)
write (Q)	read (Q)

■ We are unable to swap instructions in the above schedule to obtain either the serial schedule  $\langle T_3, T_4 \rangle$ , or the serial schedule  $\langle T_4, T_3 \rangle$

Just as very simple example suppose you had two transactions  $T_3$  and  $T_4$ , and you have this situation. Now is it conflict serializable it is not. Because to make it conflict serializable. I need to either swap write(Q) of  $T_3$  with write(Q) of  $T_4$  which is not possible because these are conflicting instructions, they both access the same data item Q and they both are write.

The other option is I could swap read(Q) in T<sub>3</sub> and write(Q) in T<sub>4</sub>, that they are also conflicting because they access the same data item and one of them is write. So, I cannot do either of this swaps which mean, that I cannot find a conflict equivalent schedule for this schedule; either to T<sub>3</sub> T<sub>4</sub> or to T<sub>4</sub> T<sub>3</sub>. It is not this schedule is not conflict equivalent to either one of them.

So, this schedule is not conflict serializable, this is the core concept. So, if you go through different examples and try to understand this at the very beginning, then in terms of the transaction management the whole study of transaction management you will have very easy progress.

(Refer Slide Time: 12:37)

**Example: Bad Schedule**

Consider two transactions:

<b>Transaction 1</b>	<b>Transaction 2</b>
UPDATE accounts	UPDATE accounts
SET balance = balance - 100	SET balance = balance * 1.005
WHERE acct_id = 31414	

In terms of read / write we can write these as:

- Transaction 1:  $r_1(A), w_1(A) // A$  is the balance for acct\_id = 31414
- Transaction 2:  $r_2(A), w_2(A), r_2(B), w_2(B) // B$  is balance of other accounts

Consider schedule S:

- Schedule S:  $r_1(A), r_2(A), w_1(A), w_2(A), r_2(B), w_2(B)$
- Suppose: A starts with \$200, and account B starts with \$100
- Schedule S is very bad! (At least, it's bad if you're the bank!) We \$100 from account A, but somehow the database has that our account now holds \$201!

Source: <http://www.cburch.com/cs/340/reading/serial/>

SWAYAM, NPTEL, NOC, INOC Instructor: Prof. P. P. Desai, IIT Kanpur - June-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

32.13

©Silberschatz, Korth and Sudarshan

So, let us let me show you number of other bad schedules, and let me a little bit more complex examples.

So, consider two transactions transaction 1 here. Update an account, where the account id is 31414 a specific account and balance is debited by 100. So, it is debiting 100 from the balance. Where in the transaction 2, you update accounts where balance is changed to balance times 1.005 which means that we are giving a 0.5 percent interest, and here there is no where clause. So, transaction 2 actually changes does this balance change in all the accounts, whereas, transaction 1 makes this debit in only one account.

Let see what will happen in terms of them. So, let us first try to write out transaction 1 and transaction 2, the first in the read write Abstracted form. So, transaction 1 it is working only on one account let us call it account A. So, what does it do? It has to set the balance to debit 100. So, it has to read so, this is  $r_1$  by  $w_1(A)$ , we mean that it is read the subscript here refers to the transaction number.

So,  $r_1$  stands for r stands for read, 1 stands for transaction 1. So, it is read by transaction 1. And what are we reading? We are reading the account balance A, let us arbitrarily we are calling it A. And then what we will have to do? After having debited that locally we will have to write it back so that the change has happened. So,  $r_1(A)$  followed by  $w_1(A)$  is transaction 1 which is being shown on the left. So, I have shown you from the actual sql statement, how can you make an abstraction of the read write that we use in terms of reasoning about the serializability.

In contrast, if you look at transaction 2, naturally transaction 2 does not have a where clause. So, it performs this balance update on each of the accounts. So, it will also perform this balance update on the account A or account balance A rather, that we assumed in the transaction 1. So, we model that by saying that naturally for changing the balance from balance times 1.005, we need to read A if the read is done in transaction 2. So, that is  $r_2(A)$  and write it back. So, that is  $w_2(A)$  and then I assume that B is some other account. There may be one more account there may be 100 thousand more account, but so far as serializability are concerned, these are all other different accounts from A. So, we symbolically just consider one; that is, some other account B other than the balance a and naturally to do the change here or do the update here will have to read B r to be and w to B. So, these are the two transactions in the simplified form that we have to analyze.

Now, let us consider A so, we have between transaction 1 and transaction 2 we have 6 instructions. So, we produce a schedule 6, where these 6 instructions are interleaved. And we satisfy the basic constraint that the instructions of every transaction occur in the same order in which they existed. So,  $r_1$  precedes  $w_1$ ,  $r_1$  precedes  $w_1$  in this schedule. R 2 A precedes  $w_2(A)$ ,  $w_2(A)$  precedes  $r_2(B)$ ,  $r_2(B)$  precedes  $w_2(B)$  and so on. So, their original ordering is maintained, but we have an interleaved schedule called S. And then on the on the write if you look at here on the write this is schedule S.

So, in the write we are saying, that let us say that A starts with 200 dollar, and B at the beginning is 100 dollar. So, what will happen you will read? You will read here this is the first thing  $r_1(A)$ . So, 200 is read then  $r_2(A)$ . So, what happens if  $r_2(A)$  is read Again 200 is read. And then you do  $w_1(A)$ . So, what is  $w_1(A)$ ?  $w_1(A)$  is the write of the transaction 1. So, transaction 1 writes after debiting this balance this is the intermediate computation.

So, when transaction 1 writes, it writes based on the value that it had read in  $r_1(A)$ ; which was 200 then 100 debited. So, it writes back 100. Then the next is  $w_2(A)$ . So, what will  $w_2(A)$  do?  $w_2(A)$  will write back the write back  $w_2(A)$  is here, will write back the result of the computation in transaction 2 based on what it read in the  $r_2(A)$ .  $r_2(A)$  had written had read 200, we hear and therefore, if you multiply 200 by this factor it becomes 201. So,  $w_2(A)$  will write 201.

So, naturally E as  $w_2(A)$  has changed the value of A after  $w_1(A)$  naturally the final value of A will be 201. Then you have r to be w to be which reads 100 makes this balance change by 1.005, it becomes 100.5. So, this is what we will have when actually this schedule completes. So, if I mean just this look at what has happened, it has I have debited 100 dollar from account A which was transaction 1, but and here I had started with 200 dollar. But at the end what I have according to the schedule is account A has a balance which is 201 dollar. Whereas, it should have had a balance which should have been 100 dollar, the balance in account B is fine. But it shows that 101 dollar more in account A.

So, naturally the bank is going to get bankrupt very soon if such scheduled are allowed. So, this schedule is an incorrect inconsistent schedule, it is a bad schedule, let us take other examples.

(Refer Slide Time: 19:18)

The slide features a small sailboat icon in the top left corner. The title 'Example: Bad Schedule' is in red at the top right. Below the title is a list of bullet points and a table comparing two transaction schedules.

**Ideal schedule is serial:** (A = \$200, B = \$100)

**Serial schedule 1:**  $r_1(A), w_1(A), r_2(A), w_2(A), r_2(B), w_2(B) // A = 100.50, B = 100.50$

**Serial schedule 2:**  $r_2(A), w_2(A), r_2(B), w_2(B), r_1(A), w_1(A) // A = 101.00, B = 100.50$

**We call a schedule serializable if it has the same effect as some serial schedule regardless of the specific information in the database.**

**As an example, consider Schedule T, which has swapped the third and fourth operations from S:**

- Schedule S:  $r_1(A), r_2(A), w_1(A), w_2(A), r_2(B), w_2(B)$
- Schedule T:  $r_1(A), r_2(A), w_2(A), w_1(A), r_2(B), w_2(B)$

**By first example, the outcome is the same as Serial schedule 1. But that's just a peculiarity of the data, as revealed by the second example, where the final value of A can't be the consequence of either of the possible serial schedules.**

**S nor T are serializable.**

	A is \$100 initially	A is \$200 initially
$r_1(A):$	100.00	200.00
$r_2(A):$	100.00	100.00
$w_2(A):$	100.50	201.00
$w_1(A):$	0.00	100.00
$r_2(B):$		100.50
$w_2(B):$	100.50	100.50

**Schedule T**

Source: <http://www.cburch.com/cs/340/reading/serial/>

SWAYAM/NPTEL NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

32.14

©Silberschatz, Korth and Sudarshan

Now let us ask what is the ideal schedule, what is ideally what should happen. Ideally naturally we can have we will have serial schedules, there are two transactions. So, there are two possible serial schedules that can happen that is first  $T_1$  happens, then whole of  $T_2$  happens. I am sorry, first  $T_1$  and then whole of  $T_2$ . Or first whole of  $T_2$  and then  $T_1$ . And if you go through the steps, assuming that A initially is 200 dollar and B is 100 dollar, these are the possible results that you see naturally. As I had mentioned earlier also, the different ordering different schedule might give you different results, but both of them are correct, because any one of them will happen, but both are consistent. Either debit has first happened, then the interest credit or interest credit it has first happened and then the debit.

So, either of these schedules are acceptable, but what we got as a schedule S in the last case are not acceptable. So, we will call it you will serializable, if it has the same effect, as some of the one of the two schedules that we have here. Then we will say that is it this is serializable schedule. So, again we create another example schedule T here. So, what we do? We take the schedule S which we saw was bad, and we interchange, these two we do  $w_2(A)$  first and  $w_1(A)$  next.

Now, you see very interesting things will happen. So now, you focus on this part, on the left part of schedule T where we are assuming that A and B both have 100 dollar to start with. And then go through these steps  $r_1$  is in transaction 1  $r_2$  is in transaction 2, then  $w_2$

happens so, the interest is credit 100.5. And then what has happened?  $w_2$  after that  $w_1(A)$  so, whatever was written here is debited and written back. So, whatever was read there is 100 dollar. So, you debit 100 dollar it becomes 0.

So, A has become 0, and then you have the B which goes on correctly. So, things look like that it appears that we are perfectly ok. So, by the first example the outcome is same as a serial schedule one. And so, we might just think that things have been good, but this is just incidental based on the particular values. Now let us consider another execution by the same schedule which makes use of this value 200 and 100.

Now, as it with 200 and 100 and we do  $w_2$  followed by  $w_1$ . So, when  $r_2(A)$  is followed by  $w_2$ ,  $r_2(A)$  100 read 200 and that 10, 1.005 or that kind of interest is given then it becomes 201. And then  $r_1$  then you have  $w_1$ . Now what does  $w_1(A)$  changes?  $r_1$  had read 200, and from that you have subtracted 100. So now, you have as  $w_1(A)$ , you have 100 input. And from this you have B certainly does not change.

So, if you look into that, now you can see that he has A value which is 100; which certainly if you look back. So, 200 and 100 are the values that we had assumed here, and you can see that in neither of the schedule A can have a value, which is 100 dollar as we have found here. It can either be 100.50 or it can be 101, but you have got a value 100. So, even though with some data, a schedule might look like serializable, but it actually is not and it needs to be properly established that schedule is serializable.

(Refer Slide Time: 24:00)

**Example: Good Schedule**

■ What's a non-serial example of a serializable schedule?

- We could credit interest to A first, then withdraw the money, then credit interest to B:
- Schedule U:  $r_2(A), w_2(A), r_1(A), w_1(A), r_2(B), w_2(B) // A = 101, B = 100.50$

■ Schedule U is conflict serializable to Schedule 2:

Schedule U:  $r_2(A), w_2(A), r_1(A), \underline{w_1(A)}, r_2(B), w_2(B)$   
 swap  $w_1(A)$  and  $r_2(B)$ :  $r_2(A), w_2(A), r_1(A), \underline{r_2(B)}, \underline{w_1(A)}, w_2(B)$   
 swap  $w_1(A)$  and  $w_2(B)$ :  $r_2(A), w_2(A), r_1(A), r_2(B), w_2(B), w_1(A)$   
 swap  $r_1(A)$  and  $r_2(B)$ :  $r_2(A), w_2(A), r_2(B), \underline{r_1(A)}, \underline{w_2(B)}, w_1(A)$   
 swap  $r_1(A)$  and  $w_2(B)$ :  $r_2(A), w_2(A), r_2(B), \underline{w_2(B)}, \underline{r_1(A)}, w_1(A)$ : Schedule 2

SWAYAM NPTEL-NOC NOC-2018 Instructor: Prof. P. Das, IIT Kharagpur - Jam-Apr-2018

Source: <http://www.cburch.com/cs/340/reading/serial/>

So, neither S naught T are serializable, yet another schedule U this again. So, you can see that transaction 1 is happening instruction of transaction 1 is happening somewhere in the middle. With transaction 2 and this is what you get. So, if you if you look back as to earlier case. You will find that this is same as scheduled 2. So, this is same as scheduled 2.

So, again my the data it looks like that this is correct, but we have to actually establish that this is correct. So, we can establish say that by proving that schedule 2 is, I am sorry, schedule 2 is conflict serialize, schedule U is conflict serializable to schedule 2. How we do that? We keep on swapping the non-conflicting instructions. This is one we start with, and we swap  $w_1$  with  $r_2(B)$  this is possible they are referring to two different data items. Then we swap  $w_1$  with  $w_2$  again different data items. Then we swap  $r_1$  with  $r_2$  again different data items and also, they are both of them are read. And finally, we swap  $r_1(A)$  with  $w_2(A)$ ,  $r_1(A)$  with  $w_2(B)$  and we get this, and now you can see that this is transaction 2 followed by transaction 1 which is scheduled 2. Which is indeed a serial schedule and we have been able to transform schedule U into a conflict equivalent schedule 2 which is serial.

So, we will say that while our earlier attempts on schedule S and schedule T were not serializable schedule U is serializable.

(Refer Slide Time: 26:09)

The slide has a title 'Serializability' in red at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list of points related to serializability:

- Are all serializable schedules conflict-serializable? No.
- Consider the following schedule for a set of three transactions.
  - $w_1(A), w_2(A), w_2(B), w_1(B), w_3(B)$
- We can perform no swaps to this:
  - The first two operations are both on A and at least one is a write;
  - The second and third operations are by the same transaction;
  - The third and fourth are both on B at least one is a write; and
  - So are the fourth and fifth.
  - So this schedule is not conflict-equivalent to anything – and certainly not any serial schedules.
- However, since nobody ever reads the values written by the  $w_1(A)$ ,  $w_2(B)$ , and  $w_1(B)$  operations, the schedule has the same outcome as the serial schedule:
  - $w_1(A), w_1(B), w_2(A), w_2(B), w_3(B)$

At the bottom, there is a footer with the text 'Source: <http://www.cburch.com/cs/340/reading/serial/>' and a navigation bar with icons for back, forward, search, etc.

So, naturally all serializable schedules are they conflict serializable. No, for example, here I have given. So, here what we are trying to highlight is a schedule may be serializable, but it may not be conflict serializable. So, conflict serializability is a stronger notion. So, here I have given a small example which I leave to you to go through in detail and understand where it is not possible to show that it is conflict serializable in the sense you cannot there are three transactions here  $w_1$   $w_2$  and  $w_3$ . And you cannot swap non-conflicting instructions in this schedule and convert it into a serial schedule.

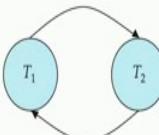
So, serial schedule here will mean,  $T_1, T_2, T_3, T_1, T_3, T_2, T_2, T_1, T_3$  like that. Any of the 6 possibilities, you cannot convert this in a conflict equivalent manner to any of those 6 serial schedules. But this actually is a serial schedule, because very interestingly even though there are multiple writes, but in between there are no reads. So, you can you can easily reason, that the values have actually not changed ok. So, this is on the basic notion of serializability.

(Refer Slide Time: 27:36)



**Precedence Graph**

- Consider some schedule of a set of transactions  $T_1, T_2, \dots, T_n$
- **Precedence graph**
  - A direct graph where the vertices are the transactions (names)
- We draw an arc from  $T_i$  to  $T_j$  if the two transactions conflict, and  $T_i$  accessed the data item on which the conflict arose earlier
- We may label the arc by the item that was accessed
- Example



SWAYAM-NPTEL-NCOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr. 2018
Database System Concepts - 6<sup>th</sup> Edition
32.17
©Silberschatz, Korth and Sudarshan

Now, the question naturally is how do I detect, if a schedule is serializable. So, the process is to construct a what is called a precedence graph. So, if I have a set of transactions, then I construct a graph is a directed graph where the vertices are the transactions their names. And we will draw an graph from  $T_i$  to  $T_j$ , that is my graph means there will be an edge is a directed edge. If these 2 transactions  $T_i$  and  $T_j$  are

conflicting. So, if  $T_i$   $T_j$  conflict there will be edge between them. And the edge will be from  $T_i$  to  $T_j$ , if  $T_i$  access the data item which conflict with  $T_j$ . So, if  $T_i$  is a head is earlier, then we will draw the arc from  $T_i$  to  $T_j$ , otherwise it will be from  $T_j$  to  $T_i$ . And we may also annotate label the arc by the item on which item that is being accessed.

(Refer Slide Time: 28:47)

**Testing for Conflict Serializability**

- A schedule is conflict serializable if and only if its precedence graph is acyclic
- Cycle-detection algorithms exist which take order  $n^2$  time, where  $n$  is the number of vertices in the graph
  - (Better algorithms take order  $n + e$  where  $e$  is the number of edges.)
- If precedence graph is acyclic, the serializability order can be obtained by a *topological sorting* of the graph
  - That is, a linear order consistent with the partial order of the graph.
  - For example, a serializability order for the schedule (a) would be one of either (b) or (c)

(a)

(b)

(c)

SWAYAM-NETEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr - 2018  
Database System Concepts - 8<sup>th</sup> Edition

32.18 ©Silberschatz, Korth and Sudarshan

So, this could be A so, possible what is called the precedence graph. So, it is a schedule is conflict serializable, if and only if it is precedence graph is acyclic. Naturally, if there is a cycle then; that means, that any of the like we have here, if there if it is a cyclic like this then it is possible that I can actually do a topological ordering of these nodes, and we can find a serial schedule. But if it has a cycle then naturally, I i cannot put any of the transactions on the cycle at the beginning and put the others on the later part things will; obviously, always conflict.

So, we can easily these are details of the algorithms, cycle detection can be done very easily. Either in  $n$  square time in a simple manner or when  $n$  plus  $E$  time where  $E$  is a number of edges. So, the precedence if the precedence graph is acyclic the serializability order will be obtained by simple topological sorting. I am not discussing what these algorithms are I would expect that you know if you do not please look up in algorithms book.

(Refer Slide Time: 30:07)

The slide has a header 'Testing for Conflict Serializability' with a small sailboat icon. On the left, there is vertical text: 'SWAYAM-NIETL-NOOC Instructor: Prof. P P Desai, IIT Kharagpur', '2018', and 'PPD'. The main content is a bulleted list of steps:

- Build a directed graph, with a vertex for each transaction.
- Go through each operation of the schedule.
  - If the operation is of the form  $w_i(X)$ , find each subsequent operation in the schedule also operating on the same data element  $X$  by a different transaction: that is, anything of the form  $r_j(X)$  or  $w_j(X)$ . For each such subsequent operation, add a directed edge in the graph from  $T_i$  to  $T_j$ .
  - If the operation is of the form  $r_i(X)$ , find each subsequent write to the same data element  $X$  by a different transaction: that is, anything of the form  $w_j(X)$ . For each such subsequent write, add a directed edge in the graph from  $T_i$  to  $T_j$ .
- The schedule is conflict-serializable if and only if the resulting directed graph is acyclic.
- Moreover, we can perform a topological sort on the graph to discover the serial schedule to which the schedule is conflict-equivalent.

At the bottom, there is a video player showing a man speaking, the text 'Database System Concepts - 8<sup>th</sup> Edition', the page number '32.19', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, to test for conflict serializability; the steps will be build the directed graph. Then go through each operation of shall, you look at each operation read or write. If the operation is a write, then find so, if it is  $w_i(X)$ , then find what is happening with data this data element  $X$  in different transactions that exists later on that instructions exist later on.

If there is some  $r_j(X)$  or some  $w_j(X)$ , either in this was in transaction  $I_j$ , in transaction some transaction  $j$  if there is a read  $X$  or if there is a write of  $X$ , then there will be a directed graph age from  $T_i$  to  $T_j$ . This is what I said earlier. On the other case if your operation is of the from  $r_i$  J if it is a read operation then all that you need to look for is only a right on this  $X$  on the different transaction. And then you will have a naturally if you if your current operation is read and you do not find a write there may be other reads on  $X$  then you do not add any conflict edge.

So, on this graph the schedule is conflict serializable if it is acyclic, and we will do topological sort to get that as I have.

(Refer Slide Time: 31:28)

**Testing for Conflict Serializability**

- Consider the following schedule:
  - $w_1(A), r_2(A), w_1(B), w_3(C), r_2(C), r_4(B), w_2(D), w_4(E), r_3(D), w_5(E)$
- We start with an empty graph with five vertices labeled  $T_1, T_2, T_3, T_4, T_5$ .
- We go through each operation in the schedule:
  - $w_1(A)$ :  $A$  is subsequently read by  $T_2$ , so add edge  $T_1 \rightarrow T_2$
  - $r_2(A)$ : no subsequent writes to  $A$ , so no new edges
  - $w_1(B)$ :  $B$  is subsequently read by  $T_4$ , so add edge  $T_1 \rightarrow T_4$
  - $w_3(C)$ :  $C$  is subsequently read by  $T_2$ , so add edge  $T_3 \rightarrow T_2$
  - $r_2(C)$ : no subsequent writes to  $C$ , so no new edges
  - $r_4(B)$ : no subsequent writes to  $B$ , so no new edges
  - $w_2(D)$ :  $C$  is subsequently read by  $T_2$ , so add edge  $T_3 \rightarrow T_2$
  - $w_4(E)$ :  $E$  is subsequently written by  $T_5$ , so add edge  $T_4 \rightarrow T_5$
  - $r_3(D)$ : no subsequent writes to  $D$ , so no new edges
  - $w_5(E)$ : no subsequent operations on  $E$ , so no new edges
- We end up with precedence graph
- This graph has no cycles, so the original schedule must be serializable. Moreover, since one way to topologically sort the graph is  $T_3-T_1-T_4-T_2-T_5$ , one serial schedule that is conflict-equivalent is
  - $w_3(C), w_1(A), w_1(B), r_4(B), w_4(E), r_2(A), r_2(C), w_2(D), r_3(D), w_5(E)$

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr., 2018

Source: <http://www.cs.cmu.edu/~cga/210/lesson10serial/>

Database System Concepts - 8<sup>th</sup> Edition      32.20      ©Silberschatz, Korth and Sudarshan

So, here what I have done is I have actually taken a little bigger example, where you can see that at the beginning here. I have given a schedule which has 5 transactions. And it has A B C D E, 5 different data elements, and variety of read write happening on them. So, based on that, you start with an empty graph having so, your graph will have 5 nodes because these are the transactions. And then you go through the schedule, you start with the very first one w 1 A. So, a is the data item you are looking at and then you see who is doing it. So, you see that well a is read by A is here read by T 2.

So, there is a conflict. So, you will add an edge  $T_1 T_2$  so, this edge gets added. Then is to have  $r_2(A)$ , and you find look for A, A, A, A, A there is no A so, there is no subsequent write. So, there is no new edge then you have w 1 B, w 1 B and if you look for you have  $r_4(B)$ ; so  $r_4$  that this transaction 4 is reading it later on. So,  $w_1(A)$  B subsequently read by  $T_4$  so, there is a conflict. So, you add the edge  $T_1, T_4, T_1, T_4$ . You proceed in this way, you can work it out in full. And when you come to the end you have constructed this particular graph which is the precedence graph. And you can very easily see that this precedence graph is acyclic there is no cycle here. And therefore, the original schedule is serializable. And what is that what is the order in which you find out what is the corresponding serial schedule for that you do a topological sort.

So, by topological sort which will mean this have no predecessor. So, any one of them can be the first node other one can be the next node. So, it could be  $T_3 T_1$  or  $T_1 T_3$ . So, let

us say  $T_3$   $T_1$  then  $T_3$   $T_1$  has happened. So, I can put any one of  $T_2$  or  $T_4$  after that. Here I put  $T_4$  then  $T_2$ . So, up to this and then finally,  $T_5$ . So, this is one possible serial schedule to which this given schedule is conflict serializable. And so, the actual serial schedule. So, if you do this schedule, you will get a result which is a result of this serial schedule which is  $T_3, T_1, T_4, T_2, T_5$ . It is also actually this channel is conflict serializable to several other schedule because you can do this topological sorting in various different manners, you could have started with  $T_1$  and then do  $T_3$  and then do the rest. You could have done  $T_3, T_1$ , and then instead of doing  $T_4, T_2$ , you could do  $T_2, T_4$ .

So, you will get a number of, but having one equivalent serial schedule. One conflict equivalent serial schedule is enough to prove the serializability of a schedule. Now so, based on that you say that this particular schedule is conflict serializable, and it will be safe to execute the interleaved instructions of the 5 different transactions in this manner in the schedule, and we will always have a consistent result.

(Refer Slide Time: 35:18)

**Module Summary**

- Understood the issues that arise when two or more transactions work concurrently
- Learned the forms of serializability in terms of conflict and view serializability
- Acyclic precedence graph can ensure conflict serializability

SWAYAM: NPTEL-NOC MOOCs. Instructor: Prof. P. P. Das, IIT Kharagpur. Jan-Apr - 2018

Database System Concepts - 6<sup>th</sup> Edition 32.21 ©Silberschatz, Korth and Sudarshan

So, here in this module, you have understood the issues that arise in terms of concurrency when two or more transactions work concurrently. And very specifically we have learnt about different forms of serializability. In this module we have talked of conflict serializability, view serializability we will take up later on. And we have seen an algorithm, simple algorithm, based on the acyclic precedence graph, which will allow you to ensure that a given schedule is conflict serializable or not.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 33**  
**Transactions/3 : Recoverability**

Welcome to module 33 of Database Management Systems. This is on transactions again there is a third and closing module on transactions and, we will discuss recoverability issues and some more of the serializability issues in this module.

(Refer Slide Time: 00:40)

The slide is titled "Module Recap" in red at the top right. On the left, there is a small image of a sailboat on water. The footer contains the text "SWAYAM: NPTEL-NOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr., 2018", "Database System Concepts - 8<sup>th</sup> Edition", "33.2", "©Silberschatz, Korth and Sudarshan", and a set of standard presentation navigation icons.

In the last module we have talked at length about serializability and specifically, we looked at what is known as conflict serializability and the algorithm to detect that. ah

(Refer Slide Time: 00:50)

The slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. On the right side, there is a vertical column of text: "PPD", "SWAYAM: NPTEL-MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur, Jan-Apr. 2018". The main content consists of two bullet points:

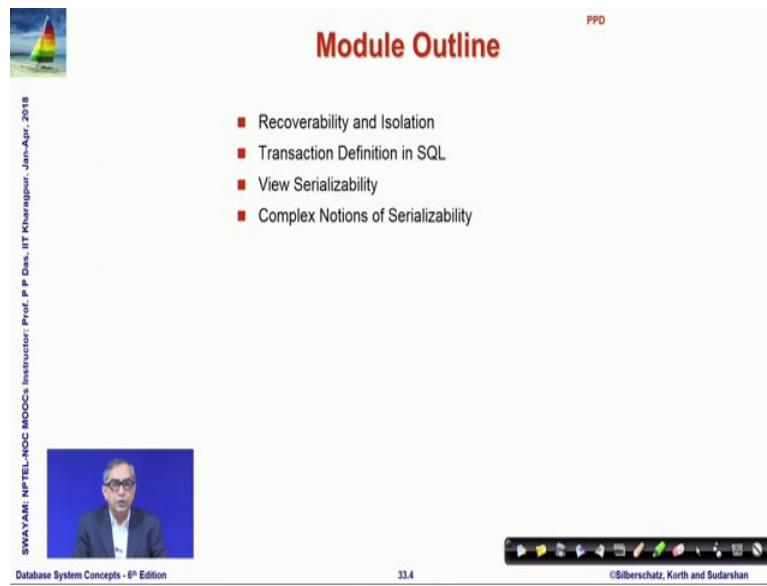
- What happens if system fails while a transaction is in execution? Can a consistent state be reached for the database? Recoverability attempts to answer issues in state and transaction recovery in the face of system failures
- Conflict serializability is a crisp concept for concurrent execution that guarantees ACID properties and has a simple detection algorithm. Yet only few schedules are Conflict serializable in practice. There is a need to explore – View Serializability – a weaker system for better concurrency

At the bottom left is a video thumbnail showing a man speaking. The bottom center contains the text "Database System Concepts - 8<sup>th</sup> Edition" and "33.3". The bottom right shows a navigation bar with icons and the text "©Silberschatz, Korth and Sudarshan".

Now, we would bring in another perspective is if while a transaction is in execution what if the system would fail, the failure may be due to hardware software, various different reasons power outage, disk crash and so on. So, why when that happens the database is likely to come into an inconsistent state. So, we would like to discuss how to recover from that inconsistent state and bring it back to a consistent state.

We would also look at that going forward from conflict serializability, what are the other notions of serializability, that can be used to serialize transactions and we will look at a weaker definition of serializability known as view serializability, which can serialize more schedules than what conflict serializability can give us.

(Refer Slide Time: 01:51)



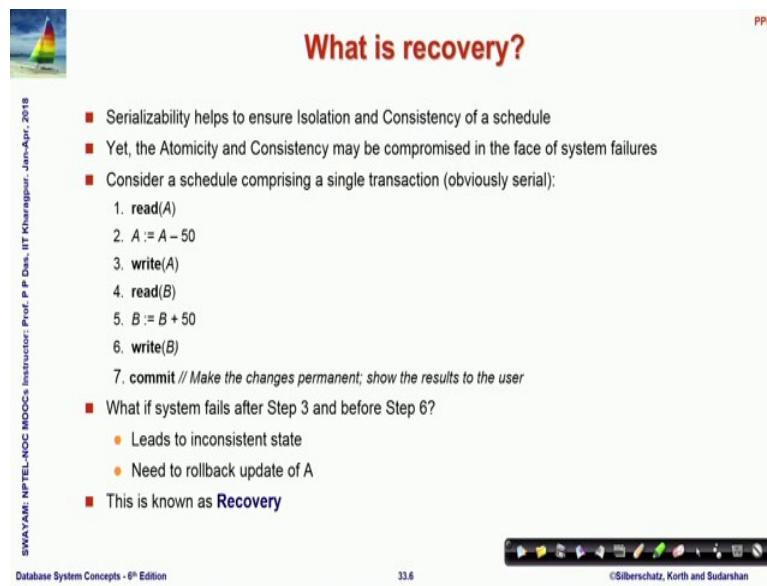
The slide is titled "Module Outline" in red at the top right. On the left, there is a small sailboat icon and a vertical sidebar with the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Desai, IIT Kanpur - Jan-Apr. 2018". The main content area contains a list of topics:

- Recoverability and Isolation
- Transaction Definition in SQL
- View Serializability
- Complex Notions of Serializability

At the bottom left is a video thumbnail showing a man speaking. The footer includes "Database System Concepts - 8<sup>th</sup> Edition", "33.4", and "©Silberschatz, Korth and Sudarshan". A toolbar is visible at the bottom.

So, these are the topics to discuss and we start with recoverability and isolation.

(Refer Slide Time: 01:57)



The slide is titled "What is recovery?" in red at the top right. On the left, there is a small sailboat icon and a vertical sidebar with the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Desai, IIT Kanpur - Jan-Apr. 2018". The main content area contains a list of points:

- Serializability helps to ensure Isolation and Consistency of a schedule
- Yet, the Atomicity and Consistency may be compromised in the face of system failures
- Consider a schedule comprising a single transaction (obviously serial):
  1. `read(A)`
  2. `A := A - 50`
  3. `write(A)`
  4. `read(B)`
  5. `B := B + 50`
  6. `write(B)`
  7. `commit // Make the changes permanent; show the results to the user`
- What if system fails after Step 3 and before Step 6?
  - Leads to inconsistent state
  - Need to rollback update of A
- This is known as **Recovery**

At the bottom left is a video thumbnail showing a man speaking. The footer includes "Database System Concepts - 8<sup>th</sup> Edition", "33.6", and "©Silberschatz, Korth and Sudarshan". A toolbar is visible at the bottom.

So, what we have done is we have seen the serializability help us, if we think in terms of the acid properties that we started by defining as the desirable properties of the transactions, we have seen that the serializability significantly helps us to achieve isolation and consistency of a schedule, yet the atomicity and consistency may be compromised, if there is a system failure.

So, we had talked about this example a bit earlier again let us take a look. So, this is a transaction where an amount of 50 dollar is being transferred from account A to account B. So, he first read debit and then write on account A and then read credit and write to account B and we have added a 7th instruction, which is commit and I will talk more about that in this module which makes that changes to A and B permanent and shows a result to the user as well.

Now, what happens if the system fails between step 3 and after step 3 when A has been written and before step 4 step 6 when B has finally, been written. So, naturally 50 dollars will simply disappear because what has been debited from A and will be available to be seen in account A will the corresponding credit will not be visible.

So, this leads to inconsistent state and to handle that what we need to do is to roll back the transaction, which means that we need to undo the changes that we have already done. So, we have to again go back to account A and write a new value which was the earlier value the value before the debit had happened. And this process of restoring the consistency back to the database is known as the recovery process.

(Refer Slide Time: 03:58)

**Recoverable Schedules**

$T_g$	$T_g$
read (A) write (A)	read (A) commit
read (B)	

If  $T_g$  should abort,  $T_g$  would have read (and possibly shown to the user) an inconsistent database state. Hence, database must ensure that schedules are recoverable.

SWAYAM: NPTEL-NOCO MOOCs Instructor: Prof. P. Desai, IIT Kanpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

33.7

©Silberschatz, Korth and Sudarshan

So, we say that a so, let us define a schedule to be recoverable if a transaction  $T_j$  reads A data previously written by a transaction  $T_i$ , then the commit operation of  $T_i$  must appear before the commit operation of  $T_j$ , if that happens then that is the earlier transaction which has written the data and  $T_j$  the later transaction which is reading the data the

earlier transaction has to commit that is make the changes permanent in the database before  $T_j$  actually reads it. If that happens, then we say that that schedule is a recoverable schedule.

So, consider a following schedule of transactions  $T_8$  and  $T_9$  where  $T_8$  has read and written A, but has not committed; that means, some more tasks in  $T_8$  are still pending it has not finished, but  $T_9$  then reads A which is in terms of serializability it is fine, but then  $T_9$  commits and then  $T_8$  is again trying to read B the continues. So, what happens is what if the transaction will fail the transaction  $T_9$  will fail immediately after the read operation.

So, what will happen I am sorry, if  $T_8$  aborts in between, then what will happen that  $T_9$  would have read because, say in read B or of  $T_8$   $T_8$  aborts that it fails, then  $T_9$  has already read the intermediate value of A and has committed which means it is possibly shown it to the user, but  $T_8$  since it has aborted sent it has failed, it has to be rolled back and the original value of A will be rolled back which is different from what has already been shown to the user and he will reach an inconsistent state.

(Refer Slide Time: 06:01)

**Cascading Rollbacks**

■ **Cascading rollback** – a single transaction failure leads to a series of transaction rollbacks. Consider the following schedule where none of the transactions has yet committed (so the schedule is recoverable)

$T_{10}$	$T_{11}$	$T_{12}$
read (A) read (B) write (A)  abort	read (A) write (A)	read (A)

■ If  $T_{10}$  fails,  $T_{11}$  and  $T_{12}$  must also be rolled back  
■ Can lead to the undoing of a significant amount of work

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Desai, IIT Kharagpur - Jan-Apr-2018

Database System Concepts - 8<sup>th</sup> Edition 33.8 ©Silberschatz, Korth and Sudarshan

So, this is an example of a schedule which is not recoverable. Now let us also observe that a single transaction failure not only means that one transaction needs to be rolled back, but it could have a cascading effect, that is a series of transaction may require a rollback. So, here is an example of  $T_{10}$ ,  $T_{11}$  and  $T_{12}$ . So,  $T_{10}$  reads A and B and writes A

and then  $T_{11}$  reads and writes A and  $T_{12}$  reads A and at that time if  $T_{10}$  fails if that aborts, then naturally it is not enough to simply roll back  $T_{10}$  because, if we roll back  $T_{10}$ , then we the value of a goes back to the original and  $T_{11}$  would have a wrong value which  $T_{10}$  had written, but has now been undone has now been rolled back.

So, it means that  $T_{11}$  will also have to be rolled back. Similarly if that is rolled back then naturally  $T_{12}$  also have to be rolled back and so on and when this rolling back goes from one transaction to the other we say this is the cascading roll back. And this can lead to a significant amount of work.

(Refer Slide Time: 07:15)

**Cascadeless Schedules**

- **Cascadeless schedules** — for each pair of transactions  $T_i$  and  $T_j$  such that  $T_j$  reads a data item previously written by  $T_i$ , the commit operation of  $T_i$  appears before the read operation of  $T_j$ .
- Every cascadeless schedule is also recoverable
- It is desirable to restrict the schedules to those that are cascadeless
- Example of a schedule that is NOT cascadeless

$T_{10}$	$T_{11}$	$T_{12}$
read (A) read (B) write (A)	read (A) write (A)	read (A)
abort		

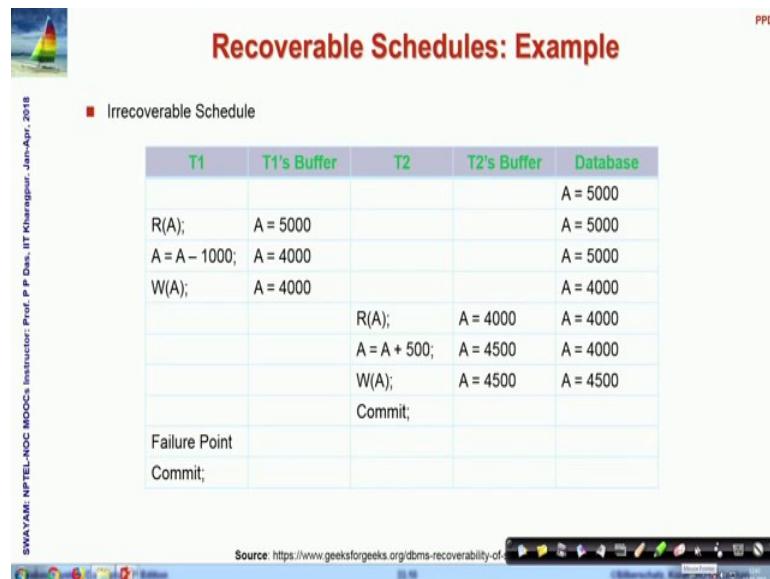
SWAYAM: NPTEL-NOC MOOCs; Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr 2018  
Database System Concepts - 8<sup>th</sup> Edition  
33.9 ©Silberschatz, Korth and Sudarshan

So, what we would prefer is if we could have schedules where such cascading roll back is not required. So, and there is a there is a condition through which you can achieve that. So, if we have a pair of transaction  $T_i$  and  $T_j$ . So, that  $T_j$  reads A data item previously written by  $T_i$ , then the commit operation of  $T_i$  has to happen before the read operation of  $T_j$  which means that said in other words that  $T_j$  should read only read values which are already committed and not read intermediate temporary values of other transactions.

So, every cascadelable schedule is also recoverable because, you can individually recover that and it is desirable to restrict schedules to those which are cascade less as far as possible, we will see that in non not all cases that is possible, but if it is possible you would like schedules which are cascade less. So, that covered a rollback work the extra

work can be minimized. So, here is an example which we had just seen which is not a cascadable schedule.

(Refer Slide Time: 08:25)



The slide has a header 'Recoverable Schedules: Example' and a subtitle 'Irrecoverable Schedule'. It contains a table with columns for T1, T1's Buffer, T2, T2's Buffer, and Database. The table shows the following sequence of operations:

T1	T1's Buffer	T2	T2's Buffer	Database
				A = 5000
R(A);	A = 5000			A = 5000
A = A - 1000; A = 4000				A = 5000
W(A); A = 4000				A = 4000
	R(A); A = 4000			A = 4000
	A = A + 500; A = 4500			A = 4000
	W(A); A = 4500			A = 4500
	Commit;			
Failure Point				
Commit;				

Source: <https://www.geeksforgeeks.org/dbms-recoverability-of-transaction-schedule/>

So, wait for word let us take a couple of examples of very similar transactions and, we would see when their schedules are irrecoverable, when their cascaded recovery is possible cascaded rollback is possible and, when cascade less rollback is possible. So, if you so, here what I have done is I have shown here the 2 transactions  $T_1$  and  $T_2$ . And this is what transaction  $T_1$  is doing and we assume that in the database the initial value of  $a$  is 5000.

So, what will happen is read here and this value is a different  $A$  this is in the buffer or the memory of  $T_1$  transaction, where  $A$  becomes 5000, then you subtract 1000 and then you write back the moment you write back in the database in between the value in the database is not changing, it is only that value is only in the buffer and, when you write back the value in the database has changed.

And then transaction  $T_2$  reads that value. So, in its local buffer  $A$  becomes 4000 it increments by 500 and then writes it back and when that happens, then in the database also the value has changed to 4500 and then  $T_2$  commits and at this point let us assume that there was if there was a failure. So, this is the point where there was a failure there were other instructions in  $T_1$  as well which is not of our interest right now, and then  $T_1$  would have committed, but what happens if the failure happens at this point naturally the

$T_1$  needs to roll back  $T_1$  needs to undo this and set the this value 5000 back into the database.

But that would mean that what  $T_2$  has committed  $T_2$  has already committed this value 4500 in the database and therefore, that has been probably been used in other places and shown to the user that will create an inconsistency in the database. So, these are this is a schedule of  $T_1$  and  $T_2$  which cannot be recovered from. So, let us and so what it has what has been violated that  $T_2$  has actually read A value which was in transit and, then it has already committed based on that read value.

(Refer Slide Time: 10:57)

Recoverable Schedules: Example

■ Recoverable Schedule without cascading rollback

T1	T1's Buffer	T2	T2's Buffer	Database
				A = 5000
R(A);	A = 5000			A = 5000
A = A - 1000; A = 4000				A = 5000
W(A); A = 4000				A = 4000
Commit;				
		R(A); A = 4000	A = 4000	A = 4000
		A = A + 500; A = 4500	A = 4500	A = 4000
		W(A); A = 4500	A = 4500	A = 4500
		Commit;		

Source: <https://www.geeksforgeeks.org/dbms-recoverability-of-schedule/>

Now, let us look into the next. So, what has been done here that all the changes are the same, but the only point that we have done is we have changed the point where the commit happens again still the  $T_2$  is reading the same value in our in a and is making the updates 4500, but the commit happens at a later point of time after the commit of this transaction  $T_1$  has taken place.

So, this is recoverable, but if we want to recover  $T_1$  naturally; that means, that for  $T_1$  to be recovered, I also need to recover  $T_2$  because  $T_2$  is used a value which is not going to be the value in after the rollback of  $T_1$  has happened  $T_2$  has used 4000, but after the rollback the value in the database will be back to 5000. So, it is the rollback is required for  $T_1$  as well as in  $T_2$ . So, this is a case of cascaded cascading roll back that has

happened. So, some more work is being done and that has happened because T 2 now here the rollback is possible because T 2 is committing after T 1.

So, the transaction it is reading from it is actually committing the changes after that source transaction has committed. So, that satisfies the condition of recoverable schedule. So, you are able to recover, but it still required the cascading because T 1 had read A value in here of A which was not yet committed. So, if we would have committed that, then we would have been able to actually create a schedule which is cascade less as we see in the next slide.

So, now I what the change that has happened is a commit is done, right after writing the value of A and T<sub>2</sub> reads that only after that commit has happened, earlier it was reading before that commit has happened. So, once T<sub>2</sub> reads it after this commit. So, if there is some there is some requirement of if there is some situation of rollback, then only T<sub>1</sub> needs to be rolled back and T<sub>2</sub> does not need to be a rollback because, it has used a value which is already committed.

So, this is the basic through the example you can clearly see, what is how the rollback can happen and in a later module, we will discuss the processes of how to do this kind of rollback the cascading and non cascading both kinds and show how to go ahead with that, but now for now what we learned is schedules need to be recoverable and, preferably cascade less rollback recovery schedules are preferred in case of database transactions.

Now, let us move on and talk little bit about what is available in SQL language in terms of handling transactions.

(Refer Slide Time: 14:12)

The slide has a header 'Transaction Definition in SQL' with a sailboat icon. The main content is a bulleted list:

- Data manipulation language must include a construct for specifying the set of actions that comprise a transaction
- In SQL, a transaction begins implicitly
- A transaction in SQL ends by:
  - **Commit work** commits current transaction and begins a new one
  - **Rollback work** causes current transaction to abort
- In almost all database systems, by default, every SQL statement also commits implicitly if it executes successfully
  - Implicit commit can be turned off by a database directive
    - ▶ For example in JDBC, connection.setAutoCommit(false);

On the left margin, there is vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018'. At the bottom left is a photo of the speaker, and at the bottom right are navigation icons.

So, SQL we have seen the kind of DDL data definition and data manipulation language paths and those were discussed in terms of our interactive session as well. As a part of data manipulation it is also possible to specify certain specific transaction events. So, a transaction in SQL typically begins implicitly and, it ends by a commit work which says that let us, you commit the current transaction that is make all the changes permanent, in the database make it visible to the user and begin a new work, or it could roll back the transaction which means that all the changes that you had done are rolled back and the transaction basically aborts.

So, in almost all systems by default every SQL statement commits implicitly and, if it has been able to execute successfully, otherwise it rolls back and this implicit commit can be controlled also, it can be in different system there are different ways to control that and say that I do not want implicit commit I would only want commit to be done explicitly.

(Refer Slide Time: 15:22)

The slide has a header 'PPD' in the top right corner. The title 'Transaction Control Language (TCL)' is in red at the top center. On the left, there is a small sailboat icon. The main content area contains two bulleted lists under red square icons:

- The following commands are used to control transactions.
  - **COMMIT** – to save the changes
  - **ROLLBACK** – to roll back the changes
  - **SAVEPOINT** – creates points within the groups of transactions in which to ROLLBACK
  - **SET TRANSACTION** – Places a name on a transaction
- Transactional control commands are only used with the **DML Commands** such as
  - INSERT, UPDATE and DELETE only
  - They cannot be used while creating tables or dropping them because these operations are automatically committed in the database

At the bottom left, it says 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018'. In the center, it says 'Source: [http://www.tutorialspoint.com/sql/sql\\_transactions.htm](http://www.tutorialspoint.com/sql/sql_transactions.htm)'. At the bottom right, it says '©Silberschatz, Korth and Sudarshan'.

So, for that purpose a part of SQL called the transaction control language has different instructions commit to save the changes roll back to roll back, the changes undo the changes and also to do some do, it in some controlled way by defining savepoint and you can also set the a particular name to a transaction and it is behavior.

So, let us look at examples for doing that soon and these TCL commands are used with specific DML commands they are meaningful in terms of insert update and delete only for example, if you are creating a database or you are doing a select to data retrieval, then these instructions have no role in those transactions.

(Refer Slide Time: 16:09)

**TCL: COMMIT Command**

PPD

■ The COMMIT is the transactional command used to save changes invoked by a transaction to the database  
■ The COMMIT saves all the transactions to the database since the last COMMIT or ROLLBACK command  
■ The syntax for the COMMIT command is as follows:  
   ● SQL> DELETE FROM Customers WHERE AGE = 25;  
   ● SQL> COMMIT;

SQL> SELECT \* FROM Customers;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000
2	Khilan	25	Delhi	1500
3	kaushik	23	Kota	2000
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bhopal	8500
6	Komal	22	MP	4500
7	Muffy	24	Indore	10000

SQL> SELECT \* FROM Customers;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000
3	kaushik	23	Kota	2000
5	Hardik	27	Bhopal	8500
6	Komal	22	MP	4500
7	Muffy	24	Indore	10000

Source: [http://www.tutorialspoint.com/sql/sql\\_transactions.htm](http://www.tutorialspoint.com/sql/sql_transactions.htm)

Before DELETE After DELETE

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Des., IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

33.16

©Silberschatz, Korth and Sudarshan

So, **COMMIT** is a transaction command which is used to save changes and make them permanent based on what has been invoked. So, here you see the example of a customer database and, what I am showing is if you this is the initial state of that table and, before any value has been deleted and if you do select star from customers these 7 records is what you get to see, in view of that you do a delete and then you commit the delete.

So, we say that I have deleted and make that deletion permanent. So, deleting based on age. So, this record is supposed to be get deleted and this record is supposed to get deleted and, after I have done the commit then again if I do the same data retrieval. And now I get to see 5 records only the 2 record number 2 and record number 4 have been permanently deleted. So, this is the way you can explicitly do commit and make the changes permanent.

(Refer Slide Time: 17:11)

**TCL: ROLLBACK Command**

- The ROLLBACK is the command used to undo transactions that have not already been saved to the database
- This can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued
- The syntax for a ROLLBACK command is as follows:
  - SQL> DELETE FROM Customers WHERE AGE = 25;
  - SQL> ROLLBACK;

SQL> SELECT \* FROM Customers;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000
2	Khilan	25	Delhi	1500
3	kaushik	23	Kota	2000
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bhopal	8500
6	Komal	22	MP	4500
7	Muffy	24	Indore	10000

Before DELETE

SQL> SELECT \* FROM Customers;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000
2	Khilan	25	Delhi	1500
3	kaushik	23	Kota	2000
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bhopal	8500
6	Komal	22	MP	4500
7	Muffy	24	Indore	10000

After DELETE

Source: [http://www.tutorialspoint.com/sql/sql\\_transactions.htm](http://www.tutorialspoint.com/sql/sql_transactions.htm)

PPD

In terms of rollback it is a command which is used to undo transactions that is the changes that have already not been saved to the database you can roll back.

So, you can roll back or undo transactions only back up in history up to the last commit, or the last rollback command was issued on this. So, again looking at the same example this is the initial state and, then you did a delete as we did last time. So, these 2 records are to be deleted, but then instead of commit we have given a rollback. So, as you give rollback these deletion operations get undone. So, these two records are again back to the table and so, after the rollback if I again do the select I will get to see the 2 records back in my list. So, this is the purpose of the rollback command.

(Refer Slide Time: 18:09)

The slide has a header 'TCL: SAVEPOINT / ROLLBACK Command' with a small logo of a sailboat on the left. On the right, there is a small 'PPD' watermark. The main content is divided into two columns:

- SAVEPOINT:**
  - A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction
  - The syntax for a SAVEPOINT command is:
    - SAVEPOINT SAVEPOINT\_NAME;
  - This command serves only in the creation of a SAVEPOINT among all the transactional statements.
  - The ROLLBACK command is used to undo a group of transactions
  - The syntax for rolling back to a SAVEPOINT is:
    - ROLLBACK TO SAVEPOINT\_NAME;
- Example:**
  - SQL> SAVEPOINT SP1;
    - Savepoint created.
  - SQL> DELETE FROM Customers WHERE ID=1;
    - 1 row deleted.
  - SQL> SAVEPOINT SP2;
    - Savepoint created.
  - SQL> DELETE FROM Customers WHERE ID=2;
    - 1 row deleted.
  - SQL> SAVEPOINT SP3;
    - Savepoint created.
  - SQL> DELETE FROM Customers WHERE ID=3;
    - 1 row deleted.

Source: [http://www.tutorialspoint.com/sql/sql\\_transactions.htm](http://www.tutorialspoint.com/sql/sql_transactions.htm)

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr., 2018

Prof. P. Das is shown in a video thumbnail on the left side of the slide.

Now, you can see transactions often could be long. So, within the transaction you may want to mark certain points. So, that in case you roll back or you need to roll back, you can roll back to that particular point and those points are in the transaction are known as the **SAVEPOINT**. So, this is the format use a save point and give it a name and, then later on you can use those save points for your purpose of rollback.

So, you are again if you are doing a rollback, then you instead of just doing rollback, you now use the save point ID that you had used in naming that particular point up to which you want to roll back and, do a rollback and that will happen only up to that point. So, let us look at an example so, here it is a series of instructions in a DML transaction. So, I initially set SP one as a save point that is I may want to roll back to the beginning, when I delete one record say ID 1. So, 1 record gets deleted, then I again save another save point another save point SP 2 this was SP 1 and, then delete a second record another save point delete another record.

So, now I have a control to undo at this point have a control to undo to 3 points for example, if I do a rollback to SP 3 I will roll back to this point, where only this record will be deletion of this record will be undone, but the first 2 records will still look show as deleted, but if I roll back to save point SP 2, then 2 records ID 2 and ID 3 that were deleted their deletion will be undone and only 1 deletion will look up. Similarly if I roll back to SP 1, it will show that no deletion as it all happened.

(Refer Slide Time: 20:14)

**TCL: SAVEPOINT / ROLLBACK Command**

PPD

At the beginning

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000
2	Khilan	25	Delhi	1500
3	kaushik	23	Kota	2000
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bhopal	8500
6	Komal	22	MP	4500
7	Muffy	24	Indore	10000

SQL> SELECT \* FROM Customers;

After ROLLBACK

ID	NAME	AGE	ADDRESS	SALARY
2	Khilan	25	Delhi	1500
3	kaushik	23	Kota	2000
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bhopal	8500
6	Komal	22	MP	4500
7	Muffy	24	Indore	10000

SQL> SELECT \* FROM Customers;

Source: [http://www.tutorialspoint.com/sql/sql\\_transactions.htm](http://www.tutorialspoint.com/sql/sql_transactions.htm)

So, if I do that on the this is the initial state on the left to the initial state of the database 3 records have been deleted and, then I do undo off the first deletion of the first 2 and I roll back to SP 2.

So, then when I undo the deletion of the last 2 records, then the what I see is the records which are marked as ID 2 and ID 3, which were done after SP 2 was marked which were deleted after SP 2 are marked, they are back into the table whereas, the deletion of SP 1 is still in effect and therefore, deletion that was none after SP 1 that is of record ID 1 is still missing and in this way you can control and roll back to any specific point in a database in a database transaction.

(Refer Slide Time: 21:13)

The slide is titled "TCL: RELEASE SAVEPOINT Command". It features a list of bullet points:

- The RELEASE SAVEPOINT command is used to remove a SAVEPOINT that you have created
- The syntax for a RELEASE SAVEPOINT command is as follows:
  - RELEASE SAVEPOINT SAVEPOINT\_NAME;
- Once a SAVEPOINT has been released, you can no longer use the ROLLBACK command to undo transactions performed since the last SAVEPOINT

Source: [http://www.tutorialspoint.com/sql/sql\\_transactions.htm](http://www.tutorialspoint.com/sql/sql_transactions.htm)

Database System Concepts - 8<sup>th</sup> Edition      33.20      ©Silberschatz, Korth and Sudarshan

You can once you have marked a safe point you can also, release the safe point that is you can choose to forget that safe point. Once a safe point has been released you cannot roll back to that safe point naturally.

(Refer Slide Time: 21:26)

The slide is titled "TCL: SET TRANSACTION Command". It features a list of bullet points:

- The SET TRANSACTION command can be used to initiate a database transaction
- This command is used to specify characteristics for the transaction that follows
  - For example, you can specify a transaction to be read only or read write
- The syntax for a SET TRANSACTION command is as follows:
  - SET TRANSACTION [ READ WRITE | READ ONLY ];

Source: [http://www.tutorialspoint.com/sql/sql\\_transactions.htm](http://www.tutorialspoint.com/sql/sql_transactions.htm)

Database System Concepts - 8<sup>th</sup> Edition      33.21      ©Silberschatz, Korth and Sudarshan

You can use **SET TRANSACTION** command to initiate a database transaction also and, it is typically used to specify the characteristics of the transaction, particularly if you want to say whether a transaction is a read only transaction or a read write transaction,

then you can do it in this way, you can say set transaction and give a read or write flag read or write or read only flag for that.

Let us quickly take a look at a different form of serializability besides the conflict serializability is called view serializability.

(Refer Slide Time: 21:59)

The slide has a title 'View Serializability' in red at the top right. To the left of the title is a small image of a sailboat on water. On the far left edge of the slide, there is vertical text that reads 'SWAYAM: NPTEL-NOOCs Instructor: Prof. P. P. Dham, IIT Kharagpur - Jan-Apr. 2018'. At the bottom left is the text 'Database System Concepts - 8<sup>th</sup> Edition'. In the center, there is a bulleted list of three conditions for view equivalence:

- Let  $S$  and  $S'$  be two schedules with the same set of transactions.  $S$  and  $S'$  are **view equivalent** if the following three conditions are met, for each data item  $Q$ ,
  1. If in schedule  $S$ , transaction  $T_i$  reads the initial value of  $Q$ , then in schedule  $S'$  also transaction  $T_i$  must read the initial value of  $Q$ .
  2. If in schedule  $S$  transaction  $T_j$  executes **read( $Q$ )**, and that value was produced by transaction  $T_k$  (if any), then in schedule  $S'$  also transaction  $T_j$  must read the value of  $Q$  that was produced by the same **write( $Q$ )** operation of transaction  $T_k$ .
  3. The transaction (if any) that performs the final **write( $Q$ )** operation in schedule  $S$  must also perform the final **write( $Q$ )** operation in schedule  $S'$ .
- As can be seen, view equivalence is also based purely on **reads** and **writes** alone

At the bottom right of the slide, there is a navigation bar with icons for back, forward, search, and other presentation controls. The page number '33.23' is also visible near the bottom center.

So, in terms of view serializability we again define what is known as when are 2 transaction schedules defined to be view equivalent, earlier you remember we define 2 schedules to be conflict equivalent, now we are defining view equivalent. So, there are 3 conditions the conditions are simple what conditions say is a to try a schedules are view equivalent, if the transaction the initial value that a transaction reads is same in both these schedules, for every transaction the initial value that it reads must be the same between the 2 schedules.

Similarly, the third condition says that the final write that is done, final value that it writes every transaction writes in both the schedules must be the same the same writes should operate. And the second conditions is a read write pair that every transaction when it performs a read on the data item, it must read from the write corresponding write in the other schedule in by the same by the transaction that which did the write.

So, I always initialize start with the same initial values for every data item in both schedules, I always read from the corresponding right in the same schedule in the 2

schedules and, I must write the final in every transaction every data item must be written in the same way in the 2 schedules.

So, this is again and the key balance is based purely on read write alone as is the case of conflict equivalence also.

(Refer Slide Time: 23:39)

**View Serializability (Cont.)**

SWAYAM: NPTEL-NOOCs Instructor: Prof. P P Dham, IIT Kharagpur, Jan-Apr, 2018

- A schedule S is **view serializable** if it is view equivalent to a serial schedule
- Every conflict serializable schedule is also view serializable
- Below is a schedule which is view-serializable but *not* conflict serializable

$T_{27}$	$T_{28}$	$T_{29}$
read (Q)		write (Q)
write (Q)		write (Q)

- What serial schedule is above equivalent to?
  - $T_{27} - T_{28} - T_{29}$
  - The one read(Q) instruction reads the initial value of Q in both schedules and
  - $T_{29}$  performs the final write of Q in both schedules
  - $T_{28}$  and  $T_{29}$  perform write(Q) operations called **blind writes**, without having performed a read(Q) operation
  - Every view serializable schedule that is not conflict serializable has **blind writes**

Database System Concepts - 8<sup>th</sup> Edition      33.24      ©Silberschatz, Korth and Sudarshan

So, given the definition of view equivalence, we can say schedule is the view serializable, if it is view equivalent to a serial schedule earlier which said that a schedule is conflict serializable, if it is conflict equivalent to a serial schedule. Now we are defining the view serializability, with a little bit of thought you can convince yourself that every conflict serializable schedule is also view serializable, but the reverse is not true.

So, here is a schedule which is view serializable, but it is not conflict serializable, you know this is not conflict serializable because, certainly you cannot make it into a serial schedule make it equivalent to a serial schedule because, you cannot move this right Q above the right Q of  $T_{28}$  or of  $T_{29}$ , but you cannot move this either. So, given that but if you in terms of the view equivalence we balance, then you will say that this is equivalent to a serial schedule and what should be the serial schedule; obviously, there are 6 choices because there are 3 schedules.

So, there are 6 possible permutations which give you 6 different serial schedules and if in that so our first condition says that I must read from the same value so; obviously,  $T_{27}$  reads the initial value of Q. So,  $T_{27}$  has to be the first transaction, if the third condition says that I must do the same right  $T_{29}$  does the final write here. So, the in the serial schedule also  $T_{29}$  must be the last one. So,  $T_{28}$  has to be the middle one.

So, the serial schedule that this is equivalent to is  $T_{27} T_{28} T_{29}$  and, the one reads and the other 2 writes and  $T_{29}$  performs a final write. So, you can see that this is a this is not a conflict serializable, but this is view serializable and if you note the view serializability moment, you have you view serializability and you may not have conflict serializability, then you must be having certain blind rights, these are called blind rights this is a blind right, in the sense that here you are writing the value of Q in  $T_{28}$  without having read it is current or previous value. So, you have just blindly you had just computed some value and you are writing to that.

So, if a schedule is not conflict serializable, but is view serializable it must have performed some blind rights where it has written data without actually reading it. So, this is a weaker form of serializability that is possible.

(Refer Slide Time: 26:20)

The slide has a title 'Test for View Serializability' in red at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list of points:

- The precedence graph test for conflict serializability cannot be used directly to test for view serializability
  - Extension to test for view serializability has cost exponential in the size of the precedence graph
- The problem of checking if a schedule is view serializable falls in the class of *NP*-complete problems
  - Thus, existence of an efficient algorithm is *extremely unlikely*
- However, practical algorithms that just check some **sufficient conditions** for view serializability can still be used

At the bottom left, there is a small video thumbnail showing a person speaking. The footer contains the text 'SWAYAM: NPTEL-NOCO MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018', 'Database System Concepts - 6<sup>th</sup> Edition', '33.25', and '©Silberschatz, Korth and Sudarshan' along with a set of navigation icons.

Now the question is similar to conflict serializability, where we saw that it schedule can be conflict serializable, if it is corresponding precedence graph is a cyclic. So, we would

like to extend find out similar test for view serializability, but as it turns out that trying to find out this is exponential in cost in terms of the size of the precedence graph.

So, it has been proved that the of checking, whether a schedule is view serializable is in the class of NP complete problem. So, if you are good in algorithms. So, you will know what NP problems are and when are problems called NP complete, in very simple terms even if you are not familiar with that depth of algorithms, you can simply issue note that if an algorithm is NP complete, then it is extremely unlikely that there exists an efficient algorithm for.

If there exists any kind of polynomial time algorithm, it is extremely unlikely still not it is still an open problem in computer science, whether a tall polynomial algorithm exists for NP complete problems, but it is extremely unlikely that an efficient algorithm will exist, you may have some approximate algorithms which can give sufficiency conditions which can say that well if these conditions are satisfied, then necessarily a schedule is view serializable, but not a sufficient condition are not a necessary condition, that is in other words that there may be some schedules which do not satisfy the sufficient condition, but are still view serializable.

(Refer Slide Time: 27:59)

**View Serializability: Example 1**

- Check whether the schedule is view serializable or not?
  - S : R2(B); R2(A); R1(A); R3(A); W1(B); W2(B); W3(B);
- Solution:
  - With 3 transactions, total number of schedules possible =  $3! = 6$ 
    - <T1 T2 T3>
    - <T1 T3 T2>
    - <T2 T3 T1>
    - <T2 T1 T3>
    - <T3 T1 T2>
    - <T3 T2 T1>
  - Final update on data items:
    - A :-
    - B : T1 T2 T3
    - Since the final update on B is made by T3, so the transaction T3 must execute after transactions T1 and T2.
    - Therefore,  $(T1, T2) \rightarrow T3$ . Now, Removing those schedules in which T3 is not executing at last:
      - <T1 T2 T3>
      - <T2 T1 T3>

Source: <http://www.edugrabs.com/how-to-check-for-view-serializability/>

Database System Concepts - 6<sup>th</sup> Edition      33.26      ©Silberschatz, Korth and Sudarshan

So, using view serializability have certain problems. So, here I have worked out a longer problem in terms of the view serializability to check that. So, it is kind of a brute force algorithm. So, if you see this is the schedule given there are two data items A and B and

there are 3 transactions  $T_1$   $T_2$   $T_3$ . Since there are three transactions, then if I want to prove if it is view serializable, then what I will have to do I will have to find a one of the possible serial schedules which is view equivalent to this?

So, first I list out all the serial schedules given 3 transactions, there are 6 serial schedules and then I first start with condition three which is who is doing the last update. So, there are writes are only on B. So, and last of that are being done in all the 3 transactions. So, there is no write on A so, the list of final update on A is empty and for B the order is  $T_1$   $T_2$   $T_3$  so,  $T_3$  does the last.

So, it must whatever schedule this whatever serial schedule this given schedule S has to be view equivalent to must have  $T_3$  as the last transaction to execute. So, only these two are the candidates which may be view equivalent to this schedule S.

(Refer Slide Time: 29:35)

**View Serializability: Example 1**

- Check whether the schedule is view serializable or not?
  - S : R2(B); R2(A); R1(A); R3(A); W1(B); W2(B); W3(B);
- Solution:
  - Initial Read + Which transaction updates after read?
    - A : T2 T1 T3 (initial read)
    - B : T2 (initial read); T1 (update after read)
    - The transaction T2 reads B initially which is updated by T1. So T2 must execute before T1.
    - Hence,  $T_2 \rightarrow T_1$ . Removing those schedules in which T2 is executing before T1:
      - $<T_2\ T_1\ T_3>$
  - Write Read Sequence (WR)
    - No need to check here
    - Hence, view equivalent serial schedule is:
      - $T_2 \rightarrow T_1 \rightarrow T_3$

Source: <http://www.edugrabs.com/how-to-check-for-view-serializability/>

So, we reduce down and now we have only to decide whether these two any of these two are view equivalent to the given schedule S. So, moving on with that now next we check condition 1 and condition 2 together.

So, condition one checks that they must read the same value in both the schedule. So, we see that these are the reads that are happening on A. So, we see that on A there are reads happening, I am sorry this is these are the three that is reading A. So, it happens in the order of  $T_2$   $T_3$   $T_1$  and  $T_3$ .

So, this is what you find and in terms of B we find that transaction 2 reads B and writes it. So, it has to be in that order. So, it reads it does an initial read in terms of  $T_2$  and, then the first right of that read value is happening in the transaction  $T_1$  after the update of the read.

So, that means, that whatever schedule we look for in terms of view equivalence, they must have in that schedule  $T_1$  must follow  $T_2$ . So,  $T_2$  must happen first because it needs to read the initial value and, then that initial value is used by then there is a right on by  $T_1$ . So,  $T_2$  has to come before  $T_1$  so; that means, we are already in terms of only 2 we have seen that there are two possible candidates based on condition 3, it is  $T T_1 T_2$ .

So, in these two we only can have this one which is satisfying the other conditions and there is no read write sequence. So, we conclude that indeed  $T_2 T_1 T_3$  satisfies all the three conditions of initial read write after read and the final write conditions and therefore, this given schedule S is actually view equivalent to a serial schedule and, it is a view serial schedule and can be used safely for the transaction.

(Refer Slide Time: 31:45)

The slide is titled "View Serializability: Example 2". It features a logo of a sailboat on the left and a "PPD" watermark in the top right. The main content area contains the following text:

- Check whether the schedule is Conflict serializable and view serializable or not?
  - S : R1(A); R2(A); R3(A); R4(A); W1(B); W2(B); W3(B); W4(B)
- Solution is given in the next slide (hidden). First try to solve it and then check the solution.

At the bottom, there is a video player showing a person speaking, with the text "SWAYAM: NPTEL-NOC" and "Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018". The video player also shows the source URL "Source: http://www.edugrabs.com/how-to-check-for-view-serializability", the duration "33:28", and copyright information "©Silberschatz, Korth and Sudarshan".

There is another example given here, where there are four transactions R1 R 2 R 3 and R 4 and there are 2 data items A and B and, you have to find out establish whether this is view serializable or not, I am not working out this one this is worked out in the presentation slide, but I will not show it here you are you should first try it out and, then

once you have been able to do it or you are unable to do that, then you check the solution from the presentation slide.

(Refer Slide Time: 32:26)

**More Complex Notions of Serializability**

The schedule below produces the same outcome as the serial schedule  $\langle T_1, T_5 \rangle$ , yet is not conflict equivalent or view equivalent to it

$T_1$	$T_5$
read (A) $A := A - 50$ write (A)	read (B) $B := B + 10$ write (B)
read (B) $B := B + 50$ write (B)	read (A) $A := A + 10$ write (A)

- If we start with  $A = 1000$  and  $B = 2000$ , the final result is 960 and 2040
- Determining such equivalence requires analysis of operations other than read and write

SWAYAM: NPTEL-NOCOCS Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

33.32

©Silberschatz, Korth and Sudarshan

There are different other complex motions of serializability also for example, if you look at this particular schedule this actually is a serializable schedule, this is the effect that it produces will be same as the serial schedule of  $T_1$   $T_5$ , but if you go through the definitions of conflict equivalence and, view equivalence you will be able to show that this schedule is neither conflict conflict serializable nor view serializable, but yet given the particular.

So, if you just look at the read write this is not a serializable schedule in terms of conflict or view equivalence, but given the fact that it actually performs simple add subtract operations on these variables, using the properties of add subtract operations you would be able to you can actually see that this particular schedule actually is a serializable schedule and, you will get whatever initial values you start with the value that you will achieve through this schedule and the value that will achieve with the serial schedule  $T_1$   $T_5$  are indeed same in every case.

But this is determining this requires the understanding of other instructions other operations, besides the read and write. So, this is just to show you that using the read write model and conflict and view equivalents and the only not the only ways of getting

to serializability there are more complex models, but we will not go into the depth of these complex serializability aspect.

(Refer Slide Time: 33:56)

**Module Summary**

- With proper planning, a database can be recovered back to a consistent state from inconsistent state in the face of system failures. Such a recovery is done via cascaded or cascadeless rollback
- View Serializability is a weaker serializability system for better concurrency. However, testing for view serializability is NP complete

SWAYAM: NPTEL-NOCs Instructor: Prof. P P Das, IIT Kanpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

33.33

©Silberschatz, Korth and Sudarshan

So, we have shown that with proper planning, a database can be recovered back to a consistent state from an inconsistent state, in case of system failure.

And this such a recovery can be through cascaded or cascadeless rollback and, we have also introduced a simpler model of serializability in terms of the view serializable, but testing for view serializability is np complete. So, as an effective algorithm it is not that powerful.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 34**  
**Concurrency Control /1**

Welcome to module 34 of Database Management Systems, in this module and the next we will talk about Concurrency Control a very key concept of database transactions.

(Refer Slide Time: 00:28)

The slide has a header 'Module Recap' in red. On the left, there is a small image of a sailboat on water. A vertical sidebar on the left contains the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. The main content area lists four bullet points under the heading 'Module Recap': '■ Recoverability and Isolation', '■ Transaction Definition in SQL', '■ View Serializability', and '■ Complex Notions of Serializability'. At the bottom, there is footer text: 'Database System Concepts - 8<sup>th</sup> Edition', '34.2', and '©Silberschatz, Korth and Sudarshan'. There is also a decorative black bar at the bottom with various icons.

So, in the last module we have talked about continuing on the transactions, we had talked about recoverability of databases, how to satisfy the **ACID** properties the basic transaction in SQL and we have introduced a second form of serializability in terms of the view serializability.

(Refer Slide Time: 00:44)

The slide has a header 'Module Objectives' in red. On the left, there is a small sailboat icon and some vertical text: 'SWAYAM: NPTEL-NOC NOCC's Instructor: Prof. P. P. Doshi, IIT Kharagpur - Jan-Apr., 2018'. In the top right corner, it says 'PPD'. Below the title, there are two bullet points:

- Concurrency Control through design of serializable schedule is difficult in general. Hence we take a look into locking mechanism and Lock-Based Protocols
- We need to understand how locks may be implemented

At the bottom, there is footer text: 'Database System Concepts - 8<sup>th</sup> Edition', '34.3', and '©Silberschatz, Korth and Sudarshan'.

Now, here in this we will talk more on the different aspects of concurrency control because, it is good that if two schedules are given, we can we may try to prove if they conflict serializable, if their view serializable and then we can use them, but doing that in general while the database is in execution is an extremely difficult problem because, who is going to give the who is who will be able to give the all possible different types of transactions that may happen hundreds of them that may be going on in a in the database at any given point of time.

So, how do you prove that or how do you know whether they are conflicts serial, or which of them conflict serializable sets are of view serializable sets and so, on. So, we introduce a different kind of need to have a different kind of mechanism and that is the mechanism of lock that is used.

(Refer Slide Time: 01:40)

The slide has a header 'Module Outline' in red. On the left, there is a small sailboat icon and a vertical sidebar with the text 'SWAYAM: NPTEL-MOC Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. The main content area contains a bulleted list: 'Concurrency Control', 'Lock-Based Protocols', and 'Implementing Locking'. Below the list is a video frame showing a man in a suit. At the bottom, it says 'Database System Concepts - 8<sup>th</sup> Edition', '34.4', and '©Silberschatz, Korth and Sudarshan'.

So, we will discuss about those aspects issues and the lock based mechanisms.

(Refer Slide Time: 01:45)

The slide has a header 'Concurrency Control' in red. On the left, there is a small sailboat icon and a vertical sidebar with the text 'SWAYAM: NPTEL-MOC Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. The main content area contains a bulleted list of requirements for concurrency control: 'A database must provide a mechanism that will ensure that all possible schedules are both: Conflict serializable, Recoverable and preferably cascadeless', 'A policy in which only one transaction can execute at a time generates serial schedules, but provides a poor degree of concurrency', 'Concurrency-control schemes tradeoff between the amount of concurrency they allow and the amount of overhead that they incur', 'Testing a schedule for serializability after it has executed is a little too late! Tests for serializability help us understand why a concurrency control protocol is correct', and 'Goal – to develop concurrency control protocols that will assure serializability'. Below the list is a video frame showing a man in a suit. At the bottom, it says 'Database System Concepts - 8<sup>th</sup> Edition', '34.5', and '©Silberschatz, Korth and Sudarshan'.

So, a database must provide a mechanism that will ensure all possible schedules are both conflict serializable, that is a basic requirement and are recoverable and preferable in a cascade less manner. So, that is the basic requirements that we have seen.

Naturally if we have everything as serial that will happen by default, but that will have very poor degree of concurrency and very low throughput. So, concurrency control

schemes will trade off the amount of concurrency that is allowed and, the amount of overhead. So, what will I have to be ensured is I should be able to for example, if I say that the schedules are always serial then the overhead of ensuring concurrency is the minimum, but naturally the benefit is also minimum, we get a very poor throughput.

The more we would like to allow for more and more concurrency in the system, but at the same time we will need to have to see what is the overhead of that what is the cost of that what how do we have to ensure those and, naturally as we I have already said testing for a scheduled to be serializable after it has happened is; obviously, too late and the question is beforehand how do I get to know it in a general setting.

So, we need to have a certain protocol through which that, transactions might be written, a protocol through which the transactions must operate so, that we can achieve good concurrency in the system. So, here our objective is to develop concurrency control protocols that will ensure serializability and if possible cascadeless recovery.

(Refer Slide Time: 03:28)

**Concurrency Control**

- One way to ensure isolation is to require that data items be accessed in a mutually exclusive manner; that is, while one transaction is accessing a data item, no other transaction can modify that data item
  - Should a transaction hold a lock on the whole database
    - ↳ Would lead to strictly serial schedules – very poor performance
- The most common method used to implement locking requirement is to allow a transaction to access a data item only if it is currently holding a **lock** on that item

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

34.7

©Silberschatz, Korth and Sudarshan

So, naturally what you do you try to see whenever we have conflict, the basic problem of serializability is conflict that is what are you reading the right data and, what happens if you inadvertently make changes in a data that has already been read by someone else and so on. So, the why we need to achieve isolation of the transactions would be to make the accesses as mutually exclusive as possible.

So, naturally one way it could be to do it using locks. So, we by the basic concept of the lock is you say that this data item is in use. So, others should not use it. Now what should be the data item, should it be the whole database, it can be the whole database we say this database is in use other transaction cannot use it, which boils down to saying almost that you have a serial schedule.

So, at any point of time only one transaction can operate on the database naturally your concurrency will be very poor. So, that is not what is what is acceptable. So, we need locking mechanisms, or a mechanism to control exclusivity in terms of holding locks on smaller items possibly at a record level at a value level and so on. So, that gives rise to a whole lot of lock based protocol some of which we are going to discuss.

(Refer Slide Time: 04:53)

The slide has a title 'Lock-Based Protocols' in red. Below the title is a bulleted list of points about locks:

- A lock is a mechanism to control concurrent access to a data item
- Data items can be locked in two modes :
  1. *exclusive (X) mode*. Data item can be both read as well as written. X-lock is requested using **lock-X** instruction
  2. *shared (S) mode*. Data item can only be read. S-lock is requested using **lock-S** instruction
- A transaction can unlock a data item Q by the **unlock(Q)** Instruction
- Lock requests are made to the concurrency-control manager by the programmer
- Transaction can proceed only after request is granted

At the bottom left is a small video thumbnail of a man speaking. At the bottom right is a navigation bar with icons for back, forward, search, etc. The footer contains the text 'SWAYAM: NPTEL-NOCO: Instructor: Prof. P P Das, IIT Kanpur - Jan-Apr- 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '34.9', and '©Silberschatz, Korth and Sudarshan'.

So, lock is a mechanism to control concurrent access and to start with there are a variety of locks that exist in a database system variety of types, but to start with we are talking about two locking modes one is exclusive mode, which is designated as X and other is shared mode and which is designated as S.

Naturally the exclusive in the exclusive mode, the data item can be read and written both and such a lock is obtained by doing it **lock-X** instruction and in the shared mode the data item can only be read. So, as you can understand why is it exclusive, when you do read write because if two transactions try to write the same item at the same time, then

you do not know what is who has been successful and who is the last right and what is the actual final value they will become indeterminate.

But if I have a value and multiple transactions read that at the same time certainly there is no problem because, all of them necessarily will read the same data. So, that is what is called shared and a shared lock or a shared mode lock can be obtained in terms of the **lock-X** instruction. And transaction when it has a lock it can unlock that by an unlock on the same data item.

So, there is a concurrency control manager to whom the lock requests are made, whether it is a request to grant, or it is a request to release and a transaction can proceed only after the request has been granted.

(Refer Slide Time: 06:26)

The slide is titled "Lock-Based Protocols". It includes a "Lock-compatibility matrix" table:

	S	X
S	true	false
X	false	false

Below the table is a list of rules:

- A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions
- Any number of transactions can hold shared locks on an item,
  - But if any transaction holds an exclusive on the item no other transaction may hold any lock on the item
- If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted
- Transaction  $T_i$  may unlock a data item that it had locked at some earlier point
- Note that a transaction must hold a lock on a data item as long as it accesses that item
- Moreover, it is not necessarily desirable for a transaction to unlock a data item immediately after its final access of that data item, since serializability may not be ensured

So, let us look into finer details this is what is known as a lock compatibility matrix. So, if there are multiple lock modes which is what is expected, then you try to see which locks can be held or operated simultaneously.

So, this is shown in terms of our present assumption that has shared an exclusive lock naturally, it transact two transactions can hold a shared lock simultaneously on the same data item, but all other combinations that is no two transactions can hold a shared and an exclusive, or two exclusive locks on the same data item at the same time. So, which means that two transactions can read a value at the same time, but two transactions

cannot one is reading the value and other is writing, the value is not possible the reverse is also not possible and two transactions writing, the value is not possible those are called said to be the incompatible modes of loss.

So, if the transaction is granted a lock, if it is compatible with the lock that is already held by another transaction, you cannot get an incompatible lock granted to you. And any number of transactions certainly can hold the shared lock and an item, but if any transaction wants to have an exclusive lock on the item, then no other transaction may hold any lock on that item. So, if I want to write that as a transaction I must be the only transaction who is who has to have that exclusive lock I must be the only transaction who is trying to write, but when I want to read many transactions can simultaneously read.

So, if a lock cannot be granted that if I want a lock either a shared lock, or an exclusive lock and if it cannot be granted, then the transaction has to wait till the all incompatible locks have been released and, only then this lock can be requested lock in being granted. And certainly a transaction who is holding a lock on a data item can unlock it at some point, after its purpose of accessing the data item is over and, a transaction must hold a lock on the data item as long as it is accessing the item that is the basic protocol.

So, you must request first get a grant of that lock do the operations that you want and then you unlock, this is a basic process that has to happen, usually it is said that as soon as you are done with the operations of the data item you mean unlock that, you may want to wait for a little longer for the ensuring the serializability these details we will see subsequently.

(Refer Slide Time: 09:14)

**Lock-Based Protocols: Example**

■ Let A and B be two accounts that are accessed by transactions T1 and T2.

- Transaction T1 transfers \$50 from account B to account A.
- Transaction T2 displays the total amount of money in accounts A and B, that is, the sum  $A + B$ .
- Suppose that the values of accounts A and B are \$100 and \$200, respectively

T1: lock-X(B); read(B); $B := B - 50;$ write(B); unlock(B); lock-X(A); read(A); $A = A + 50;$ write(A); unlock(A);	T2: lock-S(A); read(A); unlock(A); lock-S(B); read(B); unlock(B); display(A + B)
--------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------

■ If these transactions are executed serially, either as T1, T2 or the order T2, T1, then transaction T2 will display the value \$300

SWAYAM: NPTEL-NOC INOC's Instructor: Prof. P. P. Dass, IIT Kharagpur - Jan-Apr- 2018  
Database System Concepts - 8<sup>th</sup> Edition  
34.11  
©Silberschatz, Korth and Sudarshan

So, let us come to an example. So, here are two transactions  $T_1$  and  $T_2$ . So, this is a the two transactions  $T_1$ ,  $T_2$  the transaction  $T_1$  does this operation it transfers 50 dollar from account B to account A.

So, it debits B here credits A here. So, it transfers and transaction  $T_2$  displays the total sum of money in the accounts A and B. So, it reads A reads B displays and say initially the transaction initially let us say these accounts have values 100 and 200. So, what this transaction will do it needs to do the transfer. So, it needs to read B debit and write and what it has to do since it has to read it must have a shared lock. Since it has to write it must have a exclusive lock and, if it has got an exclusive lock it will also be able to read that data. So, what it does it performs an exclusive lock.

So, it requests for an exclusive lock and only on getting that it can do this and, when this is over the purpose is over it unlocks B. Similarly to update a it takes an exclusive lock on a updates and, then releases a lock. Transaction  $T_2$  what it does it has to read and display. So, it does not need an exclusive lock it takes the shared lock reads and unlocks, it again takes a shared lock on B reads and unlocks and finally, displays the two data.

Now, if these transactions are executed serially that is  $T_1$  after  $T_2$  or  $T_2$  after  $T_1$ , then the transaction  $T_2$  will always display the value 300 because, 300s is the initial value that we will be able to see if  $T_2$  runs first and  $T_1$  300 is all so, the final value because only 50

dollar has been transferred from B to A. So, the sum remains same. So, you will be able to see that if  $T_2$  runs after  $T_1$ . So, the consistency of the database is maintained.

(Refer Slide Time: 11:32)

	$T_1$	$T_2$	concurrency-control manager
	lock-X(B) read(B) $B := B - 50$ write(B) unlock(B)	lock-S(A) read(A) unlock(A) lock-S(B) read(B) unlock(B) display(A + B)	grant-X(B, $T_1$ ) grant-S(A, $T_2$ ) grant-S(B, $T_2$ ) grant-X(A, $T_1$ )
$T_1:$	lock-X(B); read(B); $B := B - 50;$ write(B); unlock(B); lock-X(A); read(A); $A := A + 50;$ write(A); unlock(A);	lock-S(A); read(A); unlock(A); lock-S(B); read(B); unlock(B); display(A + B)	
$T_2:$		lock-X(A) read(A) $A := A - 50$ write(A) unlock(A)	

Schedule 1

Now, let us see let us consider a schedule written here as schedule 1 and the the transactions are executing concurrently. So, then this is a possible schedule and let us see what will happen. So, what it does this is where the lock exclusive lock on B is held and B is updated, then A is read then B is read display is done and then this update on a has happened. And this is where we are showing that how the grants are happening.

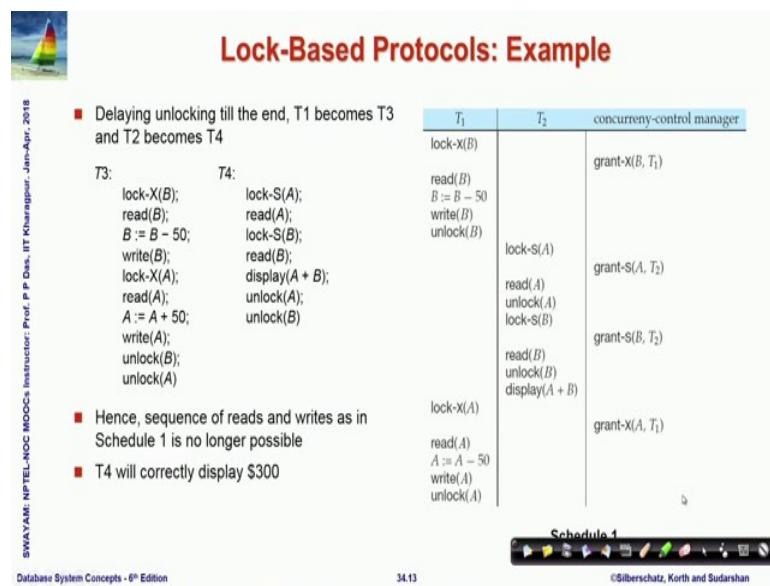
So, as the lock is requested then the request goes to the system that a exclusive lock on B is requested by transaction  $T_1$ . So, that subsequently gets granted and only when the grant has happened the corresponding axis can start, but it can be any indeterminate amount of time between the request of the lock which is here and, the actual grant of the lock, but this operation can happen only after the grant has happened.

So, in every case that is what has to be observed right. Now what happens in this schedule what will be the consequence. So, in this schedule if we look at the transaction  $T_2$  will display only 250 dollar, it will not display 300 dollar why because, if you if you look at this carefully, if you look at this carefully this is where B has got updated. So, B has become 50 dollar less. And then the whole of A and B have been read and displayed. So, naturally the total sum is 50 dollar less.

So, even though we have used a lock it has not been able to achieve the required even though, we have used the lock we have not been able to achieve the required serializability. And it is possible to create a schedule where inconsistent data is getting generated  $T_2$  is actually reading an inconsistent data. So, you have seen inconsistent state in terms of here. Why did it happen? This happened because if we look carefully this has happened because,  $T_1$  has unlocked to prematurely  $T_1$  has unlocked as soon as the update to be was over.

So, it was possible for  $T_2$  to read that value of  $B$  which is not what is desirable and we will see that we might want to delay the unlocking till the end, let us see what happens if we do that.

(Refer Slide Time: 14:43)



So, we are here now it is the same transaction in terms of the notion, but  $T_1$  has been made to  $T_3$  here, where you have seen that unlocking is been pushed to the end  $T_2$  has been made into  $T_4$ , where the unlocking is pushed to the end. And now naturally if you look into this, if you wanted to do a schedule 1 you cannot do this kind of a schedule 1 that schedule will not be permissible because, you will not be able to get the locks,  $T_4$  will not be able to get the locks that  $T_2$  could get in the sequence of reads and writes in schedule 1 is no longer possible.

(Refer Slide Time: 15:52)

**Lock-Based Protocols: Example**

- Given, T<sub>3</sub> and T<sub>4</sub>, consider Schedule 2 (partial)
- Since T<sub>3</sub> is holding an exclusive mode lock on B and T<sub>4</sub> is requesting a shared-mode lock on B, T<sub>4</sub> is waiting for T<sub>3</sub> to unlock B
- Similarly, since T<sub>4</sub> is holding a shared-mode lock on A and T<sub>3</sub> is requesting an exclusive-mode lock on A, T<sub>3</sub> is waiting for T<sub>4</sub> to unlock A
- Thus, we have arrived at a state where neither of these transactions can ever proceed with its normal execution
- This situation is called **deadlock**
- When deadlock occurs, the system must roll back one of the two transactions.
- Once a transaction has been rolled back, the data items that were locked by that transaction are unlocked
- These data items are then available to the other transaction, which can continue with its execution

T <sub>3</sub>	T <sub>4</sub>
lock-X(B); read(B); $B := B - 50;$ write(B);	lock-S(A); read(A); lock-S(B); read(B); lock-X(A); read(A); $A := A + 50;$ write(A); unlock(B); unlock(A)
lock-X(A)	lock-S(A) read(A) lock-S(B)

Schedule 2

SWAYAM: NPTEL-NOC NOOCs Instructor: Prof. P. P. Doshi, IIT Kharagpur; Jan-Apr., 2018  
Database System Concepts - 6<sup>th</sup> Edition  
34.14  
©Silberschatz, Korth and Sudarshan

So, whatever way we actually do the schedule T<sub>4</sub> will always correctly show, that the sum is three hundred dollar. So, here we are again showing T<sub>3</sub> T<sub>4</sub> this is a schedule given schedule 2, which is just given partially. And since T<sub>3</sub> is holding an exclusive lock on B and T<sub>4</sub> is requesting a shared lock. So, if I hold it this is T<sub>3</sub> is holding an exclusive lock and T<sub>4</sub> is requesting for a shared lock.

So, T<sub>4</sub> has to wait for T<sub>3</sub> to unlock B before it can actually do that operation. Similarly you will find if you look further T<sub>4</sub> has already got a shared lock to read A. And T<sub>3</sub> needs a shared lock on I am sorry T<sub>3</sub> needs an exclusive lock on A to be able to proceed. So, this one is here. So, T<sub>4</sub> cannot go beyond this point because T<sub>3</sub> has the lock on B and, one is this T<sub>3</sub> cannot go beyond this point because T<sub>4</sub> has that shared lock.

So, what we situation are we getting into. So, we are getting into a situation where, neither of T<sub>3</sub> or T<sub>4</sub> can actually proceed the normal execution, T<sub>3</sub> is waiting for exclusive lock on A and which T<sub>4</sub> has and T<sub>4</sub> is waiting for the shared lock on B, which T<sub>3</sub> already holds as an exclusive manner. And this in so, this is kind this is what is called deadlock, if you have a studied operating system, then you have must be knowing deadlock very well, and there the deadlock happens to different other issues of sharing resources here it is because of the lock.

So, moment you use locks there is a possible danger of having deadlock. And once you have a deadlock there is no other way than to unroll one or more of the transaction, then start all over again, it has to roll back one of the two transactions to be able to proceed. And once the transactions are rolled back the data items that were locked by the transactions will also be unlocked. So, please understand this in view of the earlier discussion we had in terms of transact TCL commands.

So, when you actually which we are roll back we had at that point could only say that your value of the data item in the database will be rolled back, but certainly as now you can understand that, if you roll back also the locks that you have required we also get unlocked so, that other transactions can get those locks and proceed. So, then the data items become available for other transactions and, that can continue the execution.

(Refer Slide Time: 19:03)

The slide has a title 'Lock-Based Protocols' in red. To the left is a small logo of a sailboat on water. On the right is a decorative footer bar with icons. The main content is a bulleted list of nine points about locking and deadlocks.

■ If we do not use locking, or if we unlock data items too soon after reading or writing them, we may get inconsistent states  
■ On the other hand, if we do not unlock a data item before requesting a lock on another data item, deadlocks may occur  
■ Deadlocks are a necessary evil associated with locking, if we want to avoid inconsistent states  
■ Deadlocks are definitely preferable to inconsistent states, since they can be handled by rolling back transactions, whereas inconsistent states may lead to real-world problems that cannot be handled by the database system  
■ A **locking protocol** is a set of rules followed by all transactions while requesting and releasing locks  
■ Locking protocols restrict the set of possible schedules  
■ The set of all such schedules is a proper subset of all possible serializable schedules  
■ We present locking protocols that allow only conflict-serializable schedules, and thereby ensure isolation

SWAYAM NPTEL-NOC MOOC Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr-2018

Database System Concepts - 8<sup>th</sup> Edition

34.15

©Silberschatz, Korth and Sudarshan

So, if we do not so, so we are saying that we wanted to use locks to get better control on the serializability and so on. And it was partly possible, but then we are getting into different other kinds of different problems.

So, if you do not use locking or, if we unlock data items very early then after reading, or writing them then we may get inconsistent state this is what you have seen, on the other hand, if we do not unlock a data item before requesting a lock on another data item that is if we hold it on for a very long time, then deadlock may occur. So, if we do it too soon

we do not use the lock that we have a problem of inconsistent state, if we do it hold it for too long then there could be problem of deadlock.

Now, deadlocks are necessarily evil of locking, if you do locking you will always face a deadlock, if we want to avoid inconsistent states. Now between these two; obviously, we would prefer deadlock the reason, we will prefer deadlock to inconsistent state is the fact that, if we have deadlock we still have the option of rolling back and we can take different strategies to decide what to rollback and how much to rollback and so on whereas, inconsistent states may lead to real world problems that cannot be handled by the database system.

In fact, in some in many cases I may get into some inconsistent state which is very difficult to even recognize that it is an inconsistent state. So, we will continue and prefer deadlocks over inconsistent states and, we will define we will try to define different locking protocols a set of rules that the transactions should follow, while the request and release locks to make our life relatively easier.

So, locking protocols necessarily will restrict the set of possible schedules because, we will put in some discipline in terms of how we look and how we release them, and the set of all such schedules is a proper subset of possible serializable schedules that is easy to understand. And we will present locking protocols that allow only conflict serializable schedule which ensures isolation.

(Refer Slide Time: 21:23)

The Two-Phase Locking Protocol

- This protocol ensures conflict-serializable schedules
- Phase 1: Growing Phase
  - Transaction may obtain locks
  - Transaction may not release locks
- Phase 2: Shrinking Phase
  - Transaction may release locks
  - Transaction may not obtain locks
- The protocol assures serializability. It can be proved that the transactions can be serialized in the order of their **lock points**
  - That is, the point where a transaction acquired its final lock

SWAYAM: NPTEL-NOCOCS Instructor: Prof. P. P. Doshi, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

34.16

©Silberschatz, Korth and Sudarshan

So, let us look at the most widely used protocol this is called the two phase locking protocol which guarantees conflict serializability, it does a simple thing it has two phases a growing phase, where it transaction may obtain locks and may not release any lock. And a shrinking phase which the transaction may release locks and may not obtain any law. So, you are just separating out the you know the grant or the access of locks holding of locks and the releasing of locks into two different phases you do not mix them up.

And that is the two phrases phases of the locking protocol. And this ensures so, we are we will not do the proof, but you can look it up in the book or, but you can see through examples that it can be shown that transactions can be serialized in the order of the points where they do the locking. So, these are known as lock points and, that is where the transaction actually acquired it is final lock.

(Refer Slide Time: 22:25)

The Two-Phase Locking Protocol (Cont.)

- There can be conflict serializable schedules that cannot be obtained if two-phase locking is used
- However, in the absence of extra information (e.g., ordering of access to data), two-phase locking is needed for conflict serializability in the following sense:
  - Given a transaction  $T_i$  that does not follow two-phase locking, we can find a transaction  $T_j$  that uses two-phase locking, and a schedule for  $T_i$  and  $T_j$  that is not conflict serializable

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Dass, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

34.17

©Silberschatz, Korth and Sudarshan

So, there can be conflict serializable schedules that cannot be obtained, if two phase locking is used. So, what this is saying if you use two phase locking you are guaranteed to have conflict serializable schedule, but there are conflicts serializable schedules for which you may not be able to honor the two phase locking protocol. So, two phase locking protocol is kind of a sufficiency condition.

(Refer Slide Time: 22:50)

Lock Conversions

- Two-phase locking with lock conversions:
  - First Phase:
    - can acquire a lock-S on item
    - can acquire a lock-X on item
    - can convert a lock-S to a lock-X (upgrade)
  - Second Phase:
    - can release a lock-S
    - can release a lock-X
    - can convert a lock-X to a lock-S (downgrade)
- This protocol assures serializability. But still relies on the programmer to insert the various locking instructions

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Dass, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

34.18

©Silberschatz, Korth and Sudarshan

So, you could also refine the two phase locking with what is known as lock conversion that is you can acquire in the growing phase, or the first phase you can acquire a

exclusive lock, a shared lock, or you can convert a shared lock that you already have into an exclusive lock which is called the lock upgrade process.

Similarly, in the shrinking phase you can release a shared lock release an exclusive lock, or you are holding an exclusive lock you can make it a shared lock. So, you can download. So, you can understand that upgrade and downgrade are strategies to only use that much of restriction that you need, to impose on others and to allow others to access the data to the based possible way. This protocol again issuers serializability and the it certainly depends on the programmer as to how the programmer inserts the various locking instructions.

(Refer Slide Time: 23:46)

The slide has a title 'Automatic Acquisition of Locks: Read' in red. On the left is a small sailboat icon. The main content is a pseudocode algorithm:

```
■ A transaction  $T_i$  issues the standard read/write instruction, without explicit locking calls  
■ The operation  $\text{read}(D)$  is processed as:  
    if  $T_i$  has a lock on  $D$   
        then  
            read( $D$ )  
        else begin  
            if necessary wait until no other transaction has a lock-X on  $D$   
            grant  $T_i$  a lock-S on  $D$ ;  
            read( $D$ )  
        end
```

At the bottom left is a photo of a man, and at the bottom right is a navigation bar with icons.

Now, you will have to when you want to do read or write, you may acquire locks automatically the database systems will allow that. So, this is a very simple algorithm. So, if you want to read a data, if you have a lock already on that either shared, or exclusive you can simply read it, if you do not have that then if you may have to wait until no other transaction has an exclusive lock on that because, you know that read or shared lock is not compatible with the exclusive lock.

So, you may have to wait till all are the in no other transaction the transaction that was having exclusive lock possibly has released it and, then take a grant of the shared lock on this item and, then read it is a very simple algorithm to automatically acquire locks.

(Refer Slide Time: 24:37)



## Automatic Acquisition of Locks: Write

■ `write(D)` is processed as:

```
if  $T_i$  has a lock-X on D
  then
    write(D)
  else begin
    if necessary wait until no other transaction has any lock on D,
    if  $T_j$  has a lock-S on D
      then
        upgrade lock on D to lock-X
      else
        grant  $T_i$  a lock-X on D
    write(D)
  end;
■ All locks are released after commit or abort
```

SWAYAM: NPTEL-NOC INOC's Instructor: Prof. P. P. Dass, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

34.20

©Silberschatz, Korth and Sudarshan

Write is little bit more complex because to be able to write either, you already have an exclusive lock on D, then you write or you may have to wait till no other transaction has any log because, exclusive lock is not compatible with shared lock or with other exclusive lock.

So, as long as some transaction has a lock on D you cannot proceed, but once you come to a state, that you already if no other transaction has a lock, then you see whether you yourself have a shared lock on D, if you have a shared lock then you upgrade it to an exclusive lock, if you do not have a shared lock, then you they take a grant of the exclusive lock and then you can go and write. So, it is if you follow the two phases these algorithms become very simple. And when you commit or the abort the transaction, then naturally all locks are get will get released.

(Refer Slide Time: 25:32)



## Deadlocks

■ Two-phase locking does not ensure freedom from deadlocks

	$T_3$	$T_4$
lock-X(B);	lock-S(A);	lock-x (B)
read(B);	read(A);	read (B)
$B := B - 50;$	lock-S(B);	$B := B - 50$
write(B);	read(B);	write (B)
lock-X(A);	display(A + B);	
read(A);	unlock(A);	lock-s (A)
$A := A + 50;$	unlock(B);	read (A)
write(A);		lock-s (B)
unlock(B);		
unlock(A);		

■ Observe that transactions  $T_3$  and  $T_4$  are two phase, but, in deadlock



SWAYAM: NPTEL-HOC MOOCs Instructor: Prof. P P Dhas, IIT Kharagpur - Jan-Apr, 2018  
Database System Concepts - 8<sup>th</sup> Edition  
34.21 ©Silberschatz, Korth and Sudarshan

So, the two phase protocol we have already seen that does not ensure freedom from deadlock, you can may follow two phase locking protocol here is an example, but you may still have schedules which will have deadlocks. So, this is one example you can just convince yourself.

(Refer Slide Time: 25:52)



## Starvation

■ In addition to deadlocks, there is a possibility of **starvation**

■ **Starvation** occurs if the concurrency control manager is badly designed. For example:

- A transaction may be waiting for an X-lock on an item, while a sequence of other transactions request and are granted an S-lock on the same item
- The same transaction is repeatedly rolled back due to deadlocks

■ Concurrency control manager can be designed to prevent starvation



SWAYAM: NPTEL-HOC MOOCs Instructor: Prof. P P Dhas, IIT Kharagpur - Jan-Apr, 2018  
Database System Concepts - 8<sup>th</sup> Edition  
34.22 ©Silberschatz, Korth and Sudarshan

There is another problem that can happen, in addition to deadlock this is a code there is a possibility of what is known as starvation; starvation, occurs usually it occurs when the control concurrency control manager is not a efficient one.

So, what did we see in terms of automatic locks in read and write operation is you may have to wait because, someone else is holding a lock on an item. Now holding an exclusive lock on the item, now it is possible that like the current transaction there may be couple of other transactions who are also waiting for a lock on that item and, when the opportunity comes that there is no log being held by any transaction, one of the waiting transactions must be given the lock you cannot if it is an exclusive lock you cannot give it to more than one transaction, but say three transactions were waiting for the exclusive lock and one of them get, that and that transaction can proceed the other transactions have to rollback because, they are not getting the lock.

So, now you again start you again come to the point where you wanted the exclusive lock on that item and at that time somebody is holding it and there are other transactions who are also requesting for exclusive lock. And when you come back and when finally, the exclusive lock is released by all other transactions, then again it is possible that while you are waiting some other transaction that was waiting who gets that exclusive lock and you do not get that so, you roll back and this could repeatedly could keep on happening.

So, if you have a weak strategy in terms of concurrency control, you have you will see that you have had infinite possibilities infinite occurrences, where you could have got that exclusive lock, but you are not being able to get that and therefore, you starve on the data and this is known as a data starvation problem which will also have to be checked while we do the concurrency control policies.

(Refer Slide Time: 27:57)

The slide has a header 'Cascading roll-back' with a sailboat icon. On the left, there is a list of bullet points. On the right, there is a table showing transaction schedules for T<sub>5</sub>, T<sub>6</sub>, and T<sub>7</sub>.

**List of points:**

- The potential for deadlock exists in most locking protocols. Deadlocks are a necessary evil
- When a deadlock occurs there is a possibility of cascading roll-backs
- Cascading roll-back is possible under two-phase locking
- In the schedule here, each transaction observes the two-phase locking protocol, but the failure of T<sub>5</sub> after the read(A) step of T<sub>7</sub> leads to cascading rollback of T<sub>6</sub> and T<sub>7</sub>.

**Table:**

T <sub>5</sub>	T <sub>6</sub>	T <sub>7</sub>
lock-X(A) read(A) lock-S(B) read(B) write(A) unlock(A)		
	lock-X(A) read(A) write(A) unlock(A)	lock-S(A) read(A)

SWAYAM: NPTEL-NOC INOCCS Instructor: Prof. P. P. Dabholkar, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
34.23  
©Silberschatz, Korth and Sudarshan

There is a the potential for deadlock exists in most locking protocols, as we have seen and when a deadlock occurs there is a possibility of cascading roll back because, when it deadlock happens then naturally you will have to roll back. So, you may have to do a cascading roll back as this example is showing. And it is possible for a two phase locking protocol have we have in the example is shown here, where all the transactions are following cascading roll back has to as following two phase locking protocol, but if T<sub>5</sub> fails after the read step of T<sub>7</sub> after the read step of T<sub>7</sub>, if T<sub>5</sub> fails then it leads to a cascading rollback T<sub>7</sub> T<sub>5</sub> has to be rolled back. So, T<sub>6</sub> will have to be rolled back, so T<sub>7</sub> will have to be rolled back and so on. ah

(Refer Slide Time: 28:54)

The slide has a header 'More Two Phase Locking Protocols' with a sailboat icon. The content includes two bullet points:

- To avoid Cascading roll-back, follow a modified protocol called **strict two-phase locking**
  - a transaction must hold all its exclusive locks till it commits/aborts
- **Rigorous two-phase locking** is even stricter.
  - All locks are held till commit/abort. In this protocol transactions can be serialized in the order in which they commit

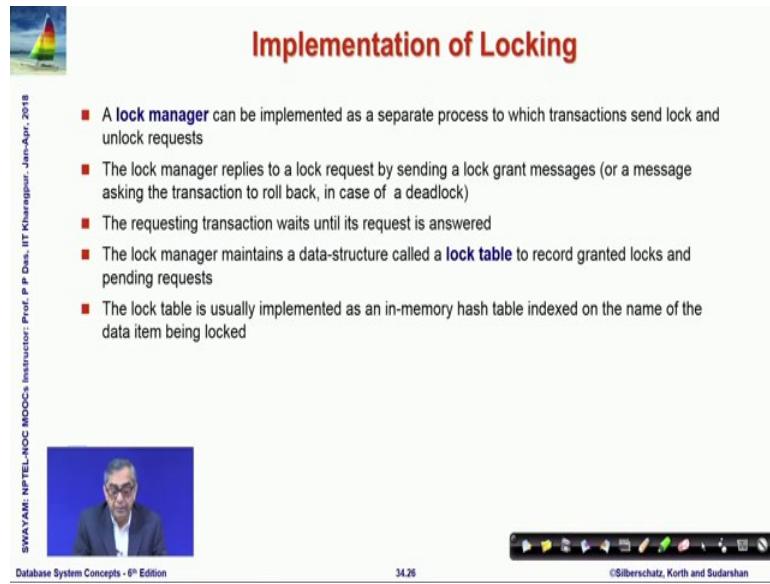
On the left margin, vertical text reads: SWAYAM, NPTEL-NOC, INOC-CC, Instructor: Prof. P. P. Doshi, IIT Kharagpur, Jan-Apr., 2018.

At the bottom, there is a video frame showing a man speaking, the text 'Database System Concepts - 8<sup>th</sup> Edition', the number '34.24', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

Interestingly there are several other protocols and particularly two more two phase locking protocol 1 is called strict two phase locking, which avoids cascading roll back, where a transaction must hold all exclusive locks till it finally, commits and aborts naturally you can figure out that you are making the time for the transaction to hold lock longer. So, naturally the level of concurrency will go down that is all possible serializable schedules will be smaller, but this guarantees that you will not have a cascading roll back. And there is an even stricter rigorous two phase locking where all locks are held till commit or abort.

In the strict 1 only exclusive locks are held till commit or abort there is a till the end of that transaction, but in rigorous two phase locking all locks are held till the committed abort, in this protocol transaction can be serialized, in the order in which they do the commit and in that way this is a serializable protocol, which also avoids the cascading roll back up. Now finally, before you close this module a quick word in terms of how do you implement locking.

(Refer Slide Time: 30:09)



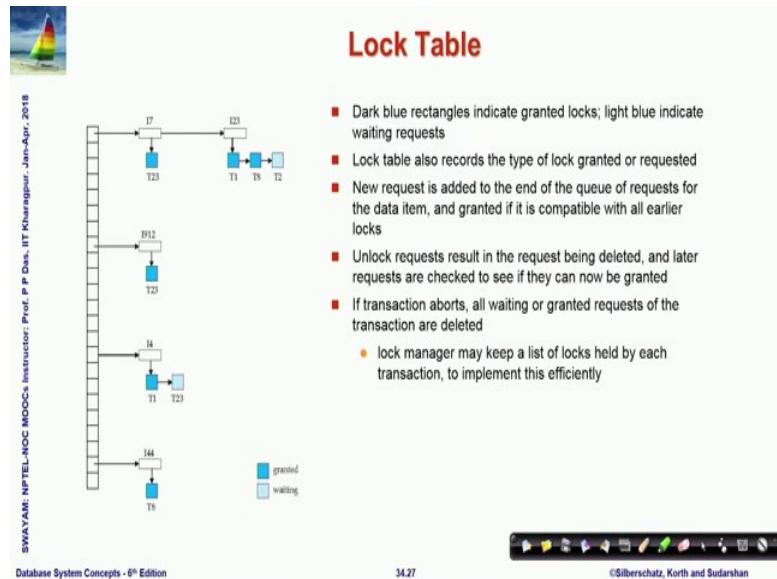
The slide has a title 'Implementation of Locking' in red at the top right. On the left, there is a small sailboat icon. The main content is a bulleted list of five points about lock managers:

- A **lock manager** can be implemented as a separate process to which transactions send lock and unlock requests
- The lock manager replies to a lock request by sending a lock grant messages (or a message asking the transaction to roll back, in case of a deadlock)
- The requesting transaction waits until its request is answered
- The lock manager maintains a data-structure called a **lock table** to record granted locks and pending requests
- The lock table is usually implemented as an in-memory hash table indexed on the name of the data item being locked

At the bottom left, it says 'SWAYAM: NPTEL-NOC INOC's Instructor: Prof. P. P. Dabholkar, IIT Kharagpur - Jun-Apr., 2018'. In the center, there is a video frame showing a man speaking. At the bottom right, there is a navigation bar with icons and the text 'Database System Concepts - 8<sup>th</sup> Edition', '34.26', and '©Silberschatz, Korth and Sudarshan'.

It is the lock there is a lock manager, which implements the locking the lock manager itself runs on a different process to which every transactions end lock and unlock requests. And the lock manager maintains a data structure to maintain what are the transactions, who are holding different locks on different items and based on that the grant messages are queued on that data structure and, these messages actually release the locks and, otherwise the transaction has to wait the lock manager maintains this as a lock table. And this is typically a in memory hash table because, it needs to naturally be very fast and is in the name of the data item being locked.

(Refer Slide Time: 31:01)



So, let us just show you and so, these are the different this is an instance of a lock table and, the nodes are different data items. So,  $I_7$ ,  $I_9$ ,  $I_{23}$ ,  $I_4$ ,  $I_{44}$ ,  $I_{23}$  are different data items this is a hash table. So, you can see that on  $I_7$  and  $I_{23}$  there is a collision and there is collate state chain happening on that. And then for every item you have you maintain a list of locks that are granted to different transactions and the list of requests that are waiting, the dark blue here shows the grant and the light blue shows a waiting status here.

So, it takes it naturally says what type of lock is granted and requested and based on this therefore, when you get a request to put it in the you come and put it you hash it to that data item, put that request on that queue and based on the current status you can decide, whether it can be granted or it has to wait. So, it is added new requests are added at the end of that queue, it is first in first out and whenever a release happens, then naturally a granted node is removed and a waiting node might get a chance to block that item, if the transaction reports all waiting, or granted requests of the transactions certainly will get deleted ok.

(Refer Slide Time: 32:30)

The slide is titled "Module Summary" in red at the top right. On the left, there is a small sailboat icon. The main content area contains two bullet points:

- Understood the locking mechanism and protocols
- Realized that deadlock is a peril of locking and needs to be handled through rollback

On the far left, vertical text reads: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Doshi, IIT Kharagpur - Jan-Apr., 2018". At the bottom, it says "Database System Concepts - 8<sup>th</sup> Edition". The bottom right corner features a copyright notice: "©Silberschatz, Korth and Sudarshan". A decorative footer bar with various icons is at the very bottom.

So, this is a simple way to manage the locks. So, in this module on concurrency control we have understood the basic locking mechanism and protocols, we have specifically looked at the lock compatibility matrix and the strategies of granting and releasing locks and, we have seen the consequent danger of having deadlock and in some cases starvation, which we have agreed to live with. So, if deadlock happens we will have to roll back one or more transactions and then restart again and, but we cannot take the risk of not having serializable transactions because, that might lead to inconsistent state of the database which is not acceptable.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 35**  
**Concurrency Control/2**

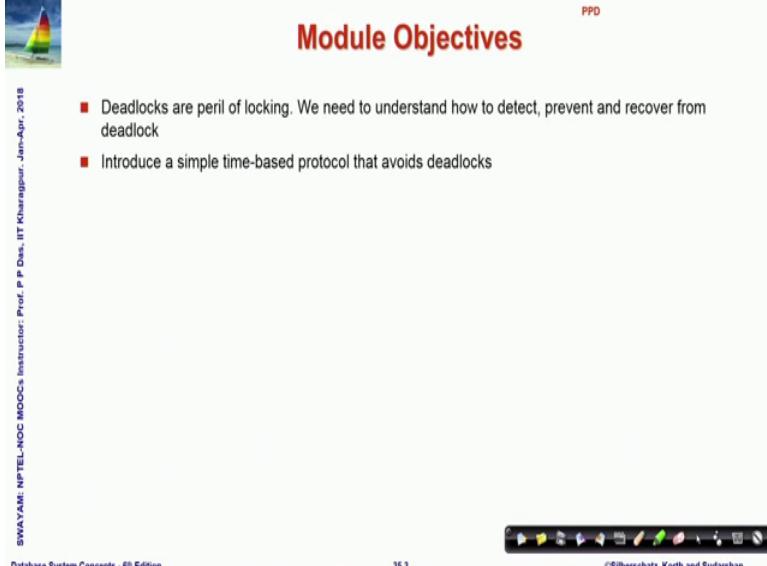
Welcome to module 35 of Database Management Systems. We have been discussing about concurrency control, this is a second and concluding module on that.

(Refer Slide Time: 00:29)

The slide has a header 'Module Recap' in red. On the left, there is a small image of a sailboat on water. The footer contains copyright information: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur. © 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '35.2', and '©Silberschatz, Korth and Sudarshan' along with a navigation bar.

So, in the last module, we have talked about the basic issues in concurrency control; and particularly talked about lock based protocol and how to implement locking in very simple terms.

(Refer Slide Time: 00:39)



The slide has a header "Module Objectives" in red. On the left, there is a small sailboat icon and some vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018". On the right, it says "PPD". Below the header is a bulleted list:

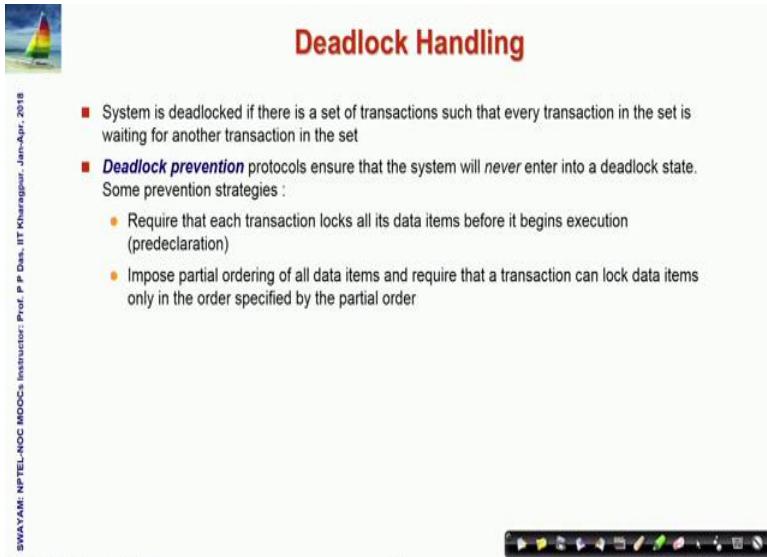
- Deadlocks are peril of locking. We need to understand how to detect, prevent and recover from deadlock
- Introduce a simple time-based protocol that avoids deadlocks

At the bottom, it shows "Database System Concepts - 8<sup>th</sup> Edition", "35.3", and "©Silberschatz, Korth and Sudarshan". There is also a decorative footer bar with various icons.

As we have seen that deadlocks of the perils of locking I mean we cannot do without locking and certainly if we lock then deadlocks are inevitable almost to happen. So, here first we try to understand how since dead locks are inevitable.

So, there has to be mechanisms to detect deadlocks and recover from them. And also we would like to look at if it is possible to create strategies which can prevent deadlock from happening at all. And so after having studied that we would like to understand take a look into a simple time-based protocol that can avoid deadlock.

(Refer Slide Time: 01:27)



The slide has a header "Deadlock Handling" in red. On the left, there is a small sailboat icon and some vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018". Below the header is a bulleted list:

- System is deadlocked if there is a set of transactions such that every transaction in the set is waiting for another transaction in the set
- **Deadlock prevention** protocols ensure that the system will never enter into a deadlock state. Some prevention strategies :
  - Require that each transaction locks all its data items before it begins execution (predeclaration)
  - Impose partial ordering of all data items and require that a transaction can lock data items only in the order specified by the partial order

At the bottom, it shows "Database System Concepts - 8<sup>th</sup> Edition", "35.6", and "©Silberschatz, Korth and Sudarshan". There is also a decorative footer bar with various icons.

So, deadlock handling. So, system is deadlock if there is the again just to recap the simple idea is if there is a set of instructions such that every transaction in the set is waiting for another transaction in the set and therefore none of them can actually proceed. So, deadlock prevention protocol ensures that the system will never enter into the deadlock state.

So, the question is can we make some strategy. So, why are we getting into the deadlock, because transactions are making requests for different locks and those are granted. And then some more requests come and we come to a state where A is waiting for B, B is waiting for C, C is waiting for a kind of a situation and we get into a deadlock.

So, can we have strategies so that the requests and releases are done in a way, so that the deadlock will not happen at all. So, I mean fortunately such number of such strategies exist. For example, one strategy which is called a pre-declaration which required that each transaction locks all debt items before it begins its execution that can be shown that that ensures that you will never have deadlock because where in very simple terms you will not be able to start before you have got all the locks.

And once you have got all the locks naturally you have every access to all possible data items and therefore, you will be able to proceed. Naturally, the flip side of this is this will delay the beginning of the transactions to a great extent in many cases, and particularly will bring down the level of concurrency that you can have.

The other which is smarter is what it does is imposes a kind of partial ordering of all the data items that a transaction and all the data items that exist. And it requires that the transaction can lock the items in only in that specific order. So, the important thing here is a partial order among the data items.

And the fact that you locked data items in that order that is specified by the partial order, you cannot lock out of order. And if you can do that then it can be shown that the deadlock will get prevented. We cannot we do not have time to go into the details of how that works, but I just want you to know that such strategies of prevention exist.

(Refer Slide Time: 03:47)

The slide has a header 'Deadlock Prevention' with a sailboat icon. The main content lists two schemes: 'wait-die' and 'wound-wait'. The 'wait-die' scheme is non-preemptive, while the 'wound-wait' scheme is preemptive. A small video window shows a professor speaking. The footer includes course details and copyright information.

Deadlock Prevention

- Following schemes use transaction timestamps for the sake of deadlock prevention alone
- **wait-die** scheme — non-preemptive
  - Older transaction may wait for younger one to release data item. (older means smaller timestamp)
    - Younger transactions never wait for older ones; they are rolled back instead
  - A transaction may die several times before acquiring needed data item
- **wound-wait** scheme — preemptive
  - Older transaction *wounds* (forces rollback) of younger transaction instead of waiting for it
    - Younger transactions may wait for older ones
  - May be fewer rollbacks than *wait-die* scheme

SWAYAM: NPTEL-NOC INOC's Instructor: Prof. P. P. Doshi, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

35.7

©Silberschatz, Korth and Sudarshan

So, the other possible prevention schemes that we will we would like to look at little bit more depth is the fact that I can use timestamps for the transaction. And use those timestamps that is which will tell me which is an earlier transaction in which, a later transaction for preventing deadlock and several strategies for that exist. We will discuss two of them. First is what is known as wait and die with scheme, which is non-preemptive. Non-preemptive means that well in this no one preempts anyone else.

So, what do you do in wait and die, the older transactions because you assume that every transaction has a timestamp. So, smaller the timestamp, older is a transaction. So, older transaction may wait for the younger one to release the item. So, if two transactions are conflicting then the older one will wait; and the younger transaction will never wait for the older one, they are rolled back instead.

So, if there is a conflict, then the younger one in that will always roll back, and the older one will wait. So, a transaction may die several times before acquiring the needed data item kind of starvation may happen, but certainly there will not be a deadlock. Because my A waiting on B, and B waiting on A, cannot happen because out of A and B one must be older has to be older, and that only will wait, the other one will abort, abort and roll back on.

The other is a preemptive scheme where which is called wound and wait scheme, where the older transaction wounds up or forces a rollback of the younger transaction instead of waiting for it that is why this is preemptive. So, the older transaction is preempting the young that transaction to continue to wait and forces it to roll back to abort and that, but the younger transaction may wait for the older one.

So, by doing this preemptive one also it is possible to have a fewer roll backs than the other scheme. So, it is a preemptive scheme, but it the advantages it might allow you fewer roll backs to happen. And with these two kind of timestamp based schemes it is possible to actually prevent deadlocks and for that reason these kind of schemes are often preferred in many context.

(Refer Slide Time: 06:21)

The slide has a title 'Deadlock Prevention' in red at the top right. On the left is a small sailboat icon. The main content area contains two bullet points:

- Both in *wait-die* and in *wound-wait* schemes, a rolled back transaction is restarted with its original timestamp. Older transactions thus have precedence over newer ones, and starvation is hence avoided
- **Timeout-Based Schemes:**
  - a transaction waits for a lock only for a specified amount of time. If the lock has not been granted within that time, the transaction is rolled back and restarted,
  - Thus, deadlocks are not possible
  - simple to implement; but starvation is possible. Also difficult to determine good value of the timeout interval

At the bottom left is vertical text: 'SWAYAM: NPTEL-NOC's Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018'. At the bottom center: 'Database System Concepts - 6<sup>th</sup> Edition' and '35.8'. At the bottom right: '©Silberschatz, Korth and Sudarshan'.

So, both in wait and die, and wound and wait scheme, the rollback transaction is restarted with its original timestamp. This is a very very important point to note. When you restart, so your rollback so you have to restart that transaction you restart the transaction you do not put the timestamp of when it is being restarted, you put the timestamp of its original time; The time when it was started and had to be aborted and rollback.

So, the older transactions have precedence over the newer ones and that starvation will get avoided.

So, now what becomes you are you are actually a new candidate because you have been rolled back in and started again, but you carry your older timestamp. So, your precedence has gone higher because in wait and die, and wound and wait in both actually the older one has a precedence.

So, by carrying your older timestamp, you inherently bring in a higher precedence in the system. And in this way there is a precedence based ordering that will naturally always happen. So, this will not only avoid deadlock, but this will also ensure that starvation is avoided; So, very simple and nice scheme.

So, in this you usually have time out basically my transaction waits for a lock only for a specified amount of time. If the lock has not been granted within that time the transaction is rolled back and restarted, and therefore, the deadlock is not possible. It is simple to implement, but starvation can happen in the timeout based scheme. And it is also difficult to determine what is a good time interval to wait.

If you wait too short, then you will spend a lot of time in the in the rollback and restart. If you wait for too long, then your throughput will go down because several transactions are basically waiting on logs. So, theoretically it does avoid deadlock, but in terms of starvation and in terms of the practicality, this there are critical things to decide on this.

(Refer Slide Time: 08:24)

The slide has a header 'Deadlock Detection' with a sailboat icon. The main content is a bulleted list of points about deadlock detection:

- Deadlocks can be described as a *wait-for graph*, which consists of a pair  $G = (V,E)$ ,
  - $V$  is a set of vertices (all the transactions in the system)
  - $E$  is a set of edges; each element is an ordered pair  $T_i \rightarrow T_j$ .
- If  $T_i \rightarrow T_j$  is in  $E$ , then there is a directed edge from  $T_i$  to  $T_j$ , implying that  $T_i$  is waiting for  $T_j$  to release a data item
- When  $T_i$  requests a data item currently being held by  $T_j$ , then the edge  $T_i \rightarrow T_j$  is inserted in the wait-for graph. This edge is removed only when  $T_j$  is no longer holding a data item needed by  $T_i$
- The system is in a deadlock state if and only if the wait-for graph has a cycle. Must invoke a deadlock-detection algorithm periodically to look for cycles

Small text on the left edge: SWAYAM: NPTEL-NOCO's Instructor: Prof. P. P. Desai, IIT Kharagpur - Jam-Aut. 2018

Navigation icons at the bottom right include arrows, a magnifying glass, and other presentation controls.

The second issue in terms of deadlock that we must be able to answer is well hundreds of transactions are going on in the system. Now, how do you know that a deadlock has happened? Because if a deadlock has happened and if you are not using a preventive scheme to ensure that the deadlock will not will never happen theoretical proof; If you are allowing say two phases locking kind of protocol where deadlocks can happen then you must know what the must be able to detect that a deadlock has happened and then take care of it to rollback the transaction.

So, for doing this, we again create a graph, which is wait for graph which is very similar to the precedence graph we saw earlier which the nodes are the transactions and the edges are ordered pair of transactions. So, what do you put an edge from  $T_i$  to  $T_j$ , you put this edge in what it means is  $T_i$  is waiting for  $T_j$ . So, if we have a conflict, then certainly one transaction is holding the lock and other is other has requested for that lock.

So, what you do you put an edge for from the one that is waiting for the lock to the one that is already holding the lock for the release of the lock, and in this way the graph gets built up. So, naturally when  $T_i$  requested it item currently being held by  $T_j$ , then the edge  $T_i-T_j$  is inserted in the in this graph. And when the release happens then this edge is removed because  $T_j$  is no more holding the item that  $T_i$  had actually required.

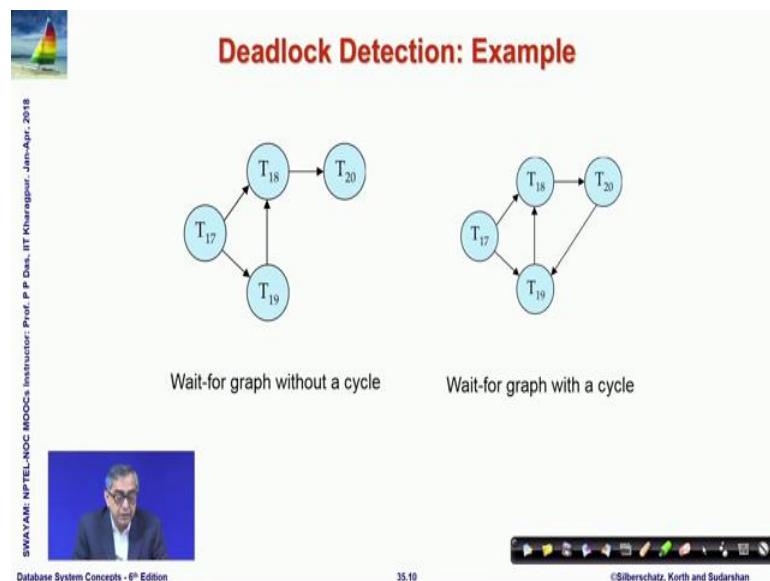
So, this is how this is kind of a dynamic graph the wait for graph is a kind of dynamic graph which will regularly keep getting updated. Now, naturally from the description of this graph, you can understand that a deadlock if a deadlock has to happen then this graph must have a cycle. So, if at any instant the graph has a cycle then there is a deadlock; otherwise the graph will grow and shrink grow and shrink it will keep on happening that way.

So, it is important to ensure that this graph remains acyclic which now this is dynamically happening hundreds of transactions, transactions are getting created, they are getting committed, aborted, they are requesting locks they are releasing locks and so on. So, how do you ensure that the graph at every stages is remaining a cyclic or a cycle has happened and therefore, a deadlock is actually happening.

So, what you will need to do is periodically run another process which invokes the deadlock-detection in the graph that is it looks for the cycles, and the cycle is there, then

you have to do some strategy to roll back about one of the transactions, and break the cycle and then so that the other transaction can proceed.

(Refer Slide Time: 11:29)



So, these are examples of the wait for graph. For example, here on the left as you see if you if I may point out in the left, if we see that T<sub>17</sub> is waiting for T<sub>18</sub> and T<sub>19</sub>, T<sub>18</sub> is waiting for T<sub>20</sub>. So, eventually and T<sub>20</sub> is waiting for none. So, at some point of time T<sub>20</sub> will be done and when that is done then T<sub>18</sub> would be able to proceed. And if T<sub>18</sub> is able to proceed then T<sub>19</sub> would be able to proceed, and then T<sub>17</sub> would be able to proceed.

So, there is no possibility of a deadlock, whereas in here if you look in the graph on right, then you can see that between these three they are waiting on each other. So, no matter how long you wait this will this deadlock will never be broken, and the deadlock-detection system has to detect this cycling and decide to abort one of these transactions and so that the rest of the transactions can progress. So, this is a simple deadlock-detection mechanism.

(Refer Slide Time: 12:44)

Deadlock Recovery

- When deadlock is detected :
  - Some transaction will have to be rolled back (made a victim) to break deadlock. Select that transaction as victim that will incur minimum cost
  - Rollback -- determine how far to roll back transaction
    - ↳ Total rollback: Abort the transaction and then restart it
    - ↳ More effective to roll back transaction only as far as necessary to break deadlock
  - Starvation happens if same transaction is always chosen as victim. Include the number of rollbacks in the cost factor to avoid starvation

SWAYAM: NPTEL-NOC's Instructor: Prof. P. P. Doshi, IIT Kharagpur, Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

35.11

©Silberschatz, Korth and Sudarshan

So, when the deadlock is detected there has to be a recovery. So, trump transactions will have to be rolled back to break the deadlock. And so there is a there is a strategy required to select which transaction must rollback; naturally that should be done based on the minimum cost that is you do not want because if you roll back then the recomputation naturally because you have to restart and do that transaction again.

So, you have to in terms of rollback, you have to determine how far to rollback that transaction. There can be a total one, so that you roll back the whole transaction abort and then restart it or you can roll back to a previous point, we discussed notions of safe point in the transaction program.

And so it is be more effective to roll back transaction only as far as necessary to break the deadlock, you may not need to roll back everything. Maybe this is this transaction is participating in the deadlock, because it is holding some exclusive lock which it took three instructions before. But before that it has done 300 instructions it is not necessary to rollback the whole of the 300 instructions, you can just roll back up to the point where it took that exclusive lock which is creating the problem.

So, that those are some of the strategies which can improve the throughput and minimize the possibility of starvation.

Starvation will again happen if the same transaction is chosen as a victim to be rolled back every time, which the possibility exists. And so the number of roll backs is also usually kept as a cost factor. So, when you roll back a transaction, you also keep a number saying that how many times this transaction has been rolled back.

So, higher that cost becomes then you would like to avoid doing the rollback for that transaction because so that it does not wait infinitely in terms of (Refer Time: 14:45) starvation. So, these are some of the simple strategies that roll back the deadlock recovery can be done.

(Refer Slide Time: 15:04)

The slide has a title 'Timestamp-Based Protocols' in red at the top right. On the left is a small image of a sailboat on water. The main content area contains a bulleted list of four points:

- Each transaction is issued a timestamp when it enters the system. If an old transaction  $T_i$  has time-stamp  $TS(T_i)$ , a new transaction  $T_j$  is assigned time-stamp  $TS(T_j)$  such that  $TS(T_i) < TS(T_j)$ .
- The protocol manages concurrent execution such that the time-stamps determine the serializability order
- In order to assure such behavior, the protocol maintains for each data item two timestamp values:
  - **W-timestamp(Q)** is the largest time-stamp of any transaction that executed **write(Q)** successfully
  - **R-timestamp(Q)** is the largest time-stamp of any transaction that executed **read(Q)** successfully

At the bottom left is a small video thumbnail showing a person speaking. The footer contains the text 'SWAYAM: NPTEL-NOC's Instructional Prof. P. P. Das, IIT Kharagpur - Jan-Apr - 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '35.13', and '©Silberschatz, Korth and Sudarshan'.

So, having talked about the prevention detection and recovery from deadlocks let us quickly look at a simple time-based protocol in contrast to the two phase locking protocol we had earlier. This protocol does not lead to deadlock. So, what you do in here is each transaction is issued a timestamp when it enters the system. So, hold at the transaction, less is the value of the timestamp so that is a simple. So, time goes in the increasing order.

Now, the protocol manages a concurrent execution such that timestamps determine they themselves will determine the serializability order, they will determine in which order the transaction should occur. And for that for each data item two timestamp values are maintained; one is a right time-stamp on the data item queue, another is a read

timestamp. So, this is the latest read write and read times for the data item. So, w timestamp Q is the largest time stem of any transaction that executed a write Q successfully. So, naturally what it means the largest timestamp means the latest write that has happened. Similarly, it keeps it latest read.

(Refer Slide Time: 16:15)

**Timestamp-Based Protocols**

- The timestamp ordering protocol ensures that any conflicting **read** and **write** operations are executed in timestamp order
- Suppose a transaction  $T_i$  issues a **read(Q)**
  1. If  $TS(T_i) \leq W\text{-timestamp}(Q)$ , then  $T_i$  needs to read a value of  $Q$  that was already overwritten.
    - Hence, the **read** operation is rejected, and  $T_i$  is rolled back
  2. If  $TS(T_i) \geq W\text{-timestamp}(Q)$ , then the **read** operation is executed, and **R-timestamp(Q)** is set to  $\max(R\text{-timestamp}(Q), TS(T_i))$

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr-2018

Database System Concepts - 6<sup>th</sup> Edition      35.14      ©Silberschatz, Korth and Sudarshan

Now, using that you build up this protocol, so it is again looks only at conflicting read and write operations, and they are executed in timestamp order. So, let us suppose that let us consider the case of read. So, a transaction  $T_i$  has issued a read.

Now, if that the timestamp of  $T_i \leq W\text{-timestamp}(Q)$  which means that **W-timestamp(Q)** is the latest write. And  $TS(T_i)$  is a timestamp of the transaction. So, the transaction is older than the latest write. So, the transaction  $T_i$  needs to read a value that was already overwritten because the latest write has happened after the transaction. So, this read operation can be rejected and  $T_i$  will be rolled back.

If it is in contrast, if the timestamp of the transaction is greater than the latest right time  $W\text{-timestamp}(Q)$  then the read operation is executed and since we are doing a read operation. So, this becomes the latest read operation and therefore, the read timestamp **R-timestamp(Q)** is set to the maximum of the current read timestamp and the timestamp of the transaction. Mind you here we the timestamp one confusion that may come to your mind is are we looking at the exact time when the read has happened or when the write

has happened, no, we are all of this reasoning is happening with the timestamp of the transaction.

So, whenever it started. So, it is a older transaction and newer transition that we are reasoning with. So, when you update R-timestamp then you are the R-timestamp already has a value which is the timestamp of the latest transaction that has read that value and  $TS(T_i)$  is the timestamp of the transaction that is read it now.

So, it is not always that since this read is the last read you will update this. So, by the sense of latest what I mean is the latest in the sense of the timestamp of the transaction that is reading it. So, you will compute that in terms of finding the maximum of the current read timestamp and the timestamp of this transaction.

(Refer Slide Time: 18:48)

The slide is titled "Timestamp-Based Protocols (Cont.)" in red at the top right. On the left, there is a small logo of a sailboat on water. The main content area contains a bulleted list under a heading: "Suppose that transaction  $T_i$  issues **write(Q)**". The list includes three numbered points:

1. If  $TS(T_i) < R\text{-timestamp}(Q)$ , then the value of  $Q$  that  $T_i$  is producing was needed previously, and the system assumed that that value would never be produced
  - Hence, the **write** operation is rejected, and  $T_i$  is rolled back
2. If  $TS(T_i) < W\text{-timestamp}(Q)$ , then  $T_i$  is attempting to write an obsolete value of  $Q$ 
  - Hence, this **write** operation is rejected, and  $T_i$  is rolled back
3. Otherwise, the **write** operation is executed, and  $W\text{-timestamp}(Q)$  is set to  $TS(T_i)$

At the bottom of the slide, there is footer text: "SWAYAM: NPTEL-NOCO Instructor: Prof. B P Das, IIT Kharagpur - Jan-Apr. 2018", "Database System Concepts - 6<sup>th</sup> Edition", "35.15", and "©Silberschatz, Korth and Sudarshan". There is also a navigation bar with icons for back, forward, search, and other presentation controls.

Write is a little bit more complex, but we can reason in the same way. So, if  $T_i$  issues a write ( $Q$ ). And if the timestamp of  $T_i < R\text{-timestamp}(Q)$  that is if this transaction is it is less so it is older than the read timestamp that is it is older than the transaction that read  $Q$  last, the youngest transaction that read the value of  $Q$ .

So, then the value  $Q$  that  $T_i$  is producing was needed earlier, it is trying to write, but already a newer transaction has used the value. And the system assumed that what the  $T_i$  was supposed to write was not available was not produced, hence the write operation is rejected  $T_i$  does not should not write this and  $T_i$  will be rolled back.

Second case if the transaction  $T_i$  has a timestamp which is less than the write timestamp. So, which means that this transaction is older than the transaction that has done the last write; So,  $T_i$  is attempting to write an absolute value of  $Q$ , and hence this write operation is again rejected and  $T_i$  is rolled back. Otherwise, in other cases, the write operation is executed and the write timestamp will be set to the timestamp of this transaction which has written it. So, this is a very simple protocol for read and write.

(Refer Slide Time: 20:36)

**Example Use of the Protocol**

A partial schedule for several data items for transactions with timestamps 1, 2, 3, 4, 5

$T_1$	$T_2$	$T_3$	$T_4$	$T_5$
read (Y)	read (Y)			read (X)
		write (Y) write (Z)		read (Z)
read (X)	read (Z) abort		read (W)	
		write (W) abort		write (Y) write (Z)

SWAYAM: NPTEL-NOC's Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr 2018  
Database System Concepts - 6<sup>th</sup> Edition  
35.16 ©Silberschatz, Korth and Sudarshan

And here is an example shown in terms of this protocol. For example, this wanted to do a read and this so this is the that this is a time where the transaction had started so and this was the write that. So, when this read is happening this is the so this is naturally this is at hold at transaction than  $T_3$ . So, this at this point, the write timestamp is that of  $T_3$  and this is an older one, so it was trying to read that, so this was aborted.

Similarly as you see here if I clean and start again if we look at write W this is trying to do read and  $T_4$ , so read will this read has a timestamp which is of  $T_4$  which is later than the timestamp of  $T_3$ . So, this gets aborted. So, you will need to spend a little bit of time to convince yourself that this will never actually ensure never actually lead to any deadlock, and it is a very effective serializable and simple strategy to ensure serializability while it avoids the deadlock.

(Refer Slide Time: 22:08)

**Correctness of Timestamp-Ordering Protocol**

■ The timestamp-ordering protocol guarantees serializability since all the arcs in the precedence graph are of the form:

```
graph LR; A((transaction with smaller timestamp)) --> B((transaction with larger timestamp))
```

Thus, there will be no cycles in the precedence graph

- Timestamp protocol ensures freedom from deadlock as no transaction ever waits
- But the schedule may not be cascade-free, and may not even be recoverable

SWAYAM-NPTEL-NOC INOC's Instructor: Prof. P. P. Doshi, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

35.17

©Silberschatz, Korth and Sudarshan

At all the timestamp ordering protocol itself guarantees the serializability, so the transaction with the smaller timestamp will lead to transaction with larger timestamp, because those are the more recent transaction. So, since the ordering is always in this manner, there cannot be any cycle in this precedence graph, because if they are cycle then naturally, somewhere you are you will be coming from newer to an older transaction which is not allowed in this protocol.

So, there cannot be a cycle and so this ensures that there cannot be deadlock in this time-based protocol, but the schedules that they produce they may not be cascade free. And actually examples can be shown that they may not even be recoverable there may be some irrecoverable schedules that get produced through this time-based protocol.

(Refer Slide Time: 23:10)

The slide is titled "Module Summary" in red at the top right. On the left, there is a small sailboat icon. The main content area contains two bullet points:

- Explained how to detect, prevent and recover from deadlock
- Introduced a time-based protocol that avoids deadlocks

On the far left, vertical text reads: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Doshi, IIT Kharagpur - Jan-Apr. 2018". At the bottom, it says "Database System Concepts - 8<sup>th</sup> Edition", "35.18", and "©Silberschatz, Korth and Sudarshan". A decorative footer bar with various icons is at the bottom.

So, to summarize we have tried to take a look into explaining what are the different ways to prevent deadlock; Some of the strategies and specifically we focused on the time-based strategy. So, there are some strategies which are based on the order of accesses the data items ordering of data items and so on. And we have specifically focused on time-based strategies.

And from that there are multiple time-based strategies which can prevent deadlock. And in case the deadlock has happened then we have discussed a simple wait for graph data structure and algorithm to be able to detect that the deadlock has happened. And if once this has been detected, we have talked about basic strategy to recover from that that is how do you decide what are the what is a good candidate victim transaction which should be rolled back, which should be aborted.

And on that study we have presented a simple time-based protocol which maintains the timestamp of transactions to decide the ordering in terms of read and write and deciding as to whether you should continue doing a read or write or you should abort the read and write attempt; And thereby ensuring that deadlocks do not happen in the system though there may be other problems in this in terms of having cascading rollback or having some irrecoverable schedules at times.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 36**  
**Recovery/1**

Welcome, to module – 36 of Database Management Systems. In this module and the next, we will talk about recovery in databases.

(Refer Slide Time: 00:27)

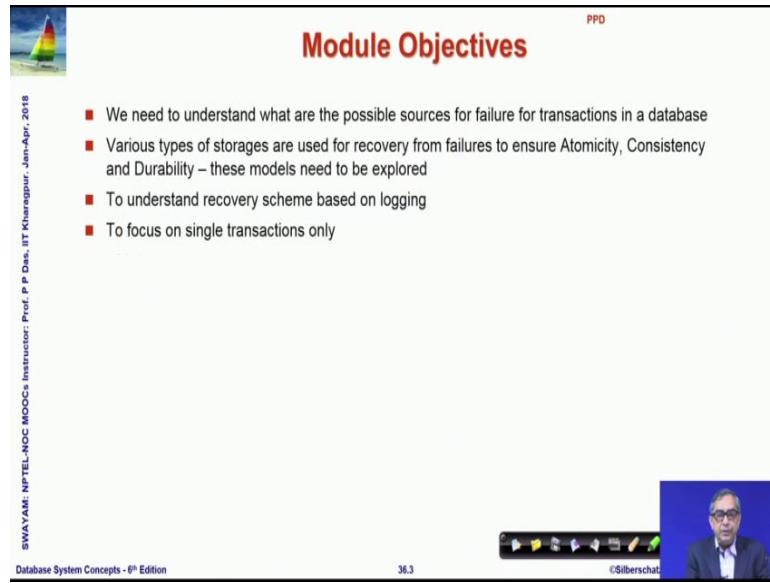
The slide is titled "Week 07 Recap" in red at the top right. It features a small sailboat icon in the top left corner. The main content is organized into two columns of bullet points:

<ul style="list-style-type: none"><li>▪ <b>Module 31: Transactions/1</b><ul style="list-style-type: none"><li>○ Transaction Concept</li><li>○ Transaction State</li><li>○ Concurrent Executions</li></ul></li><li>▪ <b>Module 32: Transactions/2: Serializability</b><ul style="list-style-type: none"><li>○ Serializability</li><li>○ Conflict Serializability</li></ul></li><li>▪ <b>Module 33: Transactions/3: Recoverability</b><ul style="list-style-type: none"><li>○ Recoverability and Isolation</li><li>○ Transaction Definition in SQL</li><li>○ View Serializability</li><li>○ Complex Notions of Serializability</li></ul></li></ul>	<ul style="list-style-type: none"><li>▪ <b>Module 34: Concurrency Control/1</b><ul style="list-style-type: none"><li>○ Lock-Based Protocols</li><li>○ Implementing Locking</li></ul></li><li>▪ <b>Module 35: Concurrency Control/2</b><ul style="list-style-type: none"><li>○ Deadlock Handling</li><li>○ Timestamp-Based Protocols</li></ul></li></ul>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

At the bottom left, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018". At the bottom right, there is a copyright notice: "©Silberschatz, Korth and Sudarshan". A navigation bar with various icons is at the very bottom.

In the last week, we have talked at length in terms of the transactions and the concurrency control. And we will see how the acid properties of the transaction can be fulfilled using the different recovery schemes.

(Refer Slide Time: 00:41)



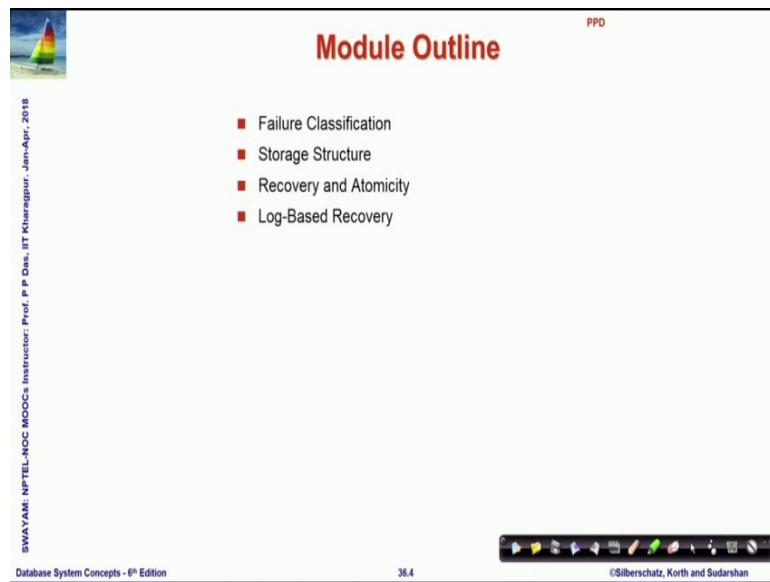
The slide features a sailboat icon in the top left corner and the text "PPD" in the top right corner. The title "Module Objectives" is centered in red. A vertical sidebar on the left contains the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018". The main content area lists four objectives:

- We need to understand what are the possible sources for failure for transactions in a database
- Various types of storages are used for recovery from failures to ensure Atomicity, Consistency and Durability – these models need to be explored
- To understand recovery scheme based on logging
- To focus on single transactions only

At the bottom right is a video frame showing a man, identified as Prof. Silberschatz, and the text "©Silberschatz". The footer includes "Database System Concepts - 8<sup>th</sup> Edition", "36.3", and a set of navigation icons.

To, specifically we will try to understand the different sources of failure and how the recovery can be facilitated by different storage structures particularly those different models of volatile and nonvolatile storages and we will take a look into recovery schemes that are based on logging mechanism and for this module we will focus only on single transactions.

(Refer Slide Time: 01:11)



The slide features a sailboat icon in the top left corner and the text "PPD" in the top right corner. The title "Module Outline" is centered in red. A vertical sidebar on the left contains the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018". The main content area lists five topics:

- Failure Classification
- Storage Structure
- Recovery and Atomicity
- Log-Based Recovery

At the bottom right is a video frame showing a man, identified as Prof. Silberschatz, and the text "©Silberschatz, Korth and Sudarshan". The footer includes "Database System Concepts - 8<sup>th</sup> Edition", "36.4", and a set of navigation icons.

So, these are the topics that will cover.

(Refer Slide Time: 01:17)

**Database System Recovery**

- All database reads/writes are within a transaction
- Transactions have the "ACID" properties
  - Atomicity - all or nothing
  - Consistency - preserves database integrity
  - Isolation - execute as if they were run alone
  - Durability - results are not lost by a failure
- Concurrency control guarantees I, contributes to C
- Application program guarantees C
- Recovery subsystem guarantees A & D, contributes to C

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

PPD

36.6 ©Silberschatz, Korth and Sudarshan

So, what we have looked at is all database writes and reads are within a transaction and transactions must satisfy the ACID properties and in terms of the concurrency control we have already seen that concurrency in a controlled guarantees isolation of transactions and in a certain way it contributes to achieving maintaining consistency. Application programs are heavily responsible for guaranteeing consistency, but to really guarantee the atomicity and durability of the data that the transactions reads and write the recovery sub system is required and it also contributes to the consistency property.

(Refer Slide Time: 01:56)

**Failure Classification**

- **Transaction failure:**
  - **Logical errors:** transaction cannot complete due to some internal error condition
  - **System errors:** the database system must terminate an active transaction due to an error condition (for example, deadlock)
- **System crash:** a power failure or other hardware or software failure causes the system to crash
  - **Fail-stop assumption:** non-volatile storage contents are assumed to not be corrupted as result of a system crash
    - ▶ Database systems have numerous integrity checks to prevent corruption of disk data
- **Disk failure:** a head crash or similar disk failure destroys all or part of disk storage
  - Destruction is assumed to be detectable
    - ▶ Disk drives use checksums to detect failures

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

36.7 ©Silberschatz, Korth and Sudarshan

So, let us look at if we are talking about recovery and the phase of failure. So, let us look at what are the generic types of failures that can happen one is the type transactions can fail. A transaction can fail due to logical error due to some internal error or it might fail due to some system error. So, that the system must terminate the transaction, we have talked about several situations where deadlock might happen and the transaction needs to be rolled back that is a kind of transaction failure error.

The second possible error can happen if there is a crash in the system. A system can crash due to hardware failure power failure software failure. So, we try to make fail stop assumptions that nonvolatile contents are assumed to be eh corrupted and database systems consequently have to involve a number of integrity checks to prevent the corruption of data.

And, the third broad category of failures happen with disk failure a disk might itself fail it is hardware may fail the head may crash and when that happens then the destruction is assumed to be detectable we must be able to detect such failures. There are checksums and other mechanisms for detecting failures, but broadly these are the three types of failures that a database system can go through.

(Refer Slide Time: 03:23)

The slide has a title 'Recovery Algorithms' in red at the top right. On the left, there is a small logo of a sailboat on water. The main content is a bulleted list of points:

- Consider transaction  $T_i$  that transfers \$50 from account A to account B
  - Two updates: subtract 50 from A and add 50 to B
- Transaction  $T_i$  requires updates to A and B to be output to the database
  - A failure may occur after one of these modifications have been made but before both of them are made
  - Modifying the database without ensuring that the transaction will commit may leave the database in an inconsistent state
  - Not modifying the database may result in lost updates if failure occurs just after transaction commits
- Recovery algorithms have two parts
  1. Actions taken during normal transaction processing to ensure enough information exists to recover from failures
  2. Actions taken after a failure to recover the database contents to a state that ensures atomicity, consistency and durability

At the bottom left, there is vertical text: 'SWAYAM: NPTEL-NOCO MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jam-Apr- 2018'. At the bottom right, there are navigation icons and the text 'Database System Concepts - 8<sup>th</sup> Edition' and 'Silberschatz, Korth and Sudarshan'.

So, in view of that ifs one or more of this failures happen then we need mechanisms to recover from that let us consider a very simple situation of a transaction which we saw earlier to that a transaction  $T_i$  transfers dollar 50 from account A to account B and

therefore, two updates have to happen; A has to get debited and B has to get credited. So, the transaction  $T_i$  requires updates to A and B that are happening that must be written that must be output to the database in a permanent manner. So, if failure may occur after one of these modifications have happened and before both of them are made, so that is one possibility. One possibility is we can get we have modified the database without ensuring that the transaction will necessarily commit, but the database has been checked transaction may not have committed. So, that will leave the database inconsistent because the transaction will have to be rolled back or it may so happen that database has not been modified and the, but the transaction has committed.

So, if the failure occurs at that point then there will be some lost updates. So, the recovery algorithms strategy has to primarily take care of two things; one is during the normal transactions it has to collect enough information so that the recovery from failures can be done. So, one is what we need to do while in normal transaction is going on because during that time we need to have enough data so that we can recovery in the phase of failure and the second set of actions are actions that can once a failure has happened to recover the database so that we can go back to a consistent state and ensure the atomicity consistency and durability of the transaction.

So, before we get into these discussions of the different recovery algorithms let us quickly look into this storage model that we are assuming.

(Refer Slide Time: 05:30)

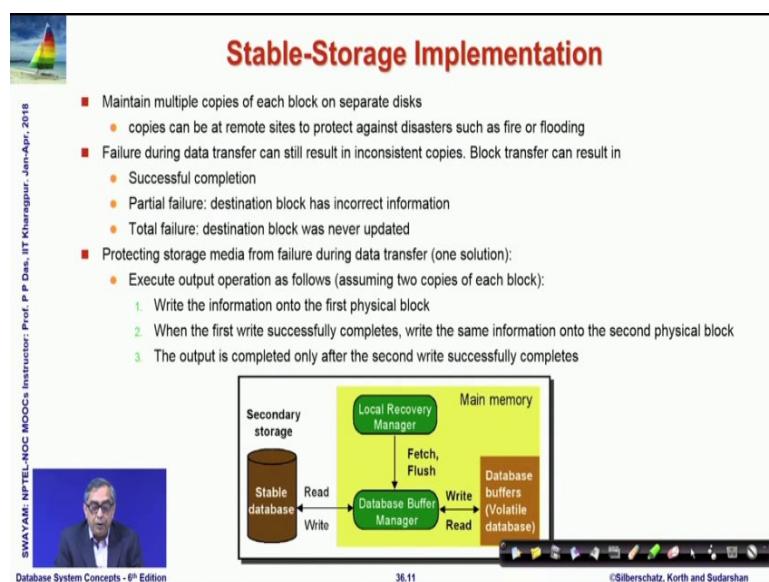
The slide has a header 'Storage Structure' in red. On the left, there is a small image of a sailboat on water. The right side contains three bulleted sections: 'Volatile storage', 'Nonvolatile storage', and 'Stable storage'. Each section includes a list of characteristics. At the bottom, there is a video player showing a man speaking, and the footer includes the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018' and 'Database System Concepts - 8<sup>th</sup> Edition'.

- **Volatile storage:**
  - does not survive system crashes
  - examples: main memory, cache memory
- **Nonvolatile storage:**
  - survives system crashes
  - examples: disk, tape, flash memory, non-volatile (battery backed up) RAM
  - but may still fail, losing data
- **Stable storage:**
  - a mythical form of storage that survives all failures
  - approximated by maintaining multiple copies on distinct nonvolatile media

We know there is volatile storage which we have discussed about that we know this nonvolatile storage which disk, tape, flash and all that volatile storage disappears whenever system crashes and non-volatile storage is supposed to survive the system crash, but it may still fail it may still cause loss of data.

So, we also consider a third kind of storage which is notionally known as stable storage. It is called a mythical form of storage where we assume that it will survive all kinds of failures. Now, naturally this in ideality this can never happen, but we can approximate this by maintaining multiple copies of the same data on distinct non-volatile media and the stable storage would be assumed to be one available component in the database system for making the recovery systems work.

(Refer Slide Time: 06:21)



So, as you can see in this diagram below. So, we are trying to explain more of what is the stable storage. So, this is on the secondary storage so, you have a stable database. So, kind of approximates that it will never fail whereas, on a routine basis things happen in terms of database buffers which are basically volatile databases, the volatile memory. So, now the fig so, what we do is we maintain multiple copies of each block of data and keep them on separate disk. So, even if one disk fail that is possible to recover from other disk. There are different kinds of the multiplicity that can be done even it can be located at a remote location so that even if there is a fire or flooding the database can be

recovered, but in principle will assume that the multiple copies are not all copies can fail at the same time.

So, it can now this will ensure the data has already been written then it is guarantee that it will stay we have talked about rate systems, but what happens if the failure happens during the data transfer where the result is still in transient state in it will live with transient copies. So, block transfer in general can result in either in successful completion or in partial failure, where the destination block actually has in current information or total failure where destination block could not be updated at all. So, to protect against the media again such failures during the data transfer the one possible solution could be and we assume that there are only two copies of each block it could you could have multiple copies to give you more resiliency against failure.

So, if we have two copies and the strategy could go like this that write the information onto the first physical block then once that is successfully completed then you write the same information on the second or physical block and the output is completed only after the second write the physical block is successfully completed. So, that is what we need to guarantee.

(Refer Slide Time: 08:45)

The slide has a title 'Stable-Storage Implementation (Cont.)' in red. Below the title is a sub-section header 'Protecting storage media from failure during data transfer (cont.):'. A bulleted list follows:

- Copies of a block may differ due to failure during output operation. To recover from failure:
  1. First find inconsistent blocks:
    - 1. *Expensive solution:* Compare the two copies of every disk block
    - 2. *Better solution:*
      - Record in-progress disk writes on non-volatile storage (Non-volatile RAM or special area of disk)
      - Use this information during recovery to find blocks that may be inconsistent, and only compare copies of these
      - Used in hardware RAID systems
  2. If either copy of an inconsistent block is detected to have an error (bad checksum), overwrite it by the other copy
  3. If both have no error, but are different, overwrite the second block by the first block

At the bottom left is a small video thumbnail showing a person speaking. At the bottom right are navigation icons and the text '©Silberschatz, Korth and Sudarshan'.

Now, to protect against that happens if during this transfer with during this write if some output operation is some failure happens. So, to recover from that you need to find out what are the blocks which are inconsistent, because we have kept two or more

copies so you have to compare two copies of every disk block have kept them at separate disks and see which one whether there has been some inconsistency. Now, this is theoretically ok, but this is very expensive because there are so many different blocks.

So, what is typically done a better solution is while you are actually doing the disk write where you are actually doing the output on a in the process of doing the output then you record these writes on a nonvolatile storage say non-volatile ram or special area of the disk and use this information during the recovery to find the blocks that are inconsistent and only compare those copies. So, that will be naturally much faster because memory as you know is much faster to access than the disk and these are strategy which is typically used in the rate system we have discussed earlier.

So, if either of either copy of a inconsistent block is detected to have some kind of an error, to checksums to the error then over write by the other copy, but if both have no error, but are different then overwrite the second one by the first one. So, this will make sure that you always have even if there is a transient failures you can take care of that you know what is wrong and you can take care of and correct that.

(Refer Slide Time: 10:25)

The slide has a title 'Data Access' in red at the top right. On the left is a small sailboat icon. The main content is a bulleted list of points about data access:

- **Physical blocks** are those blocks residing on the disk
- **System buffer blocks** are the blocks residing temporarily in main memory
- Block movements between disk and main memory are initiated through the following two operations:
  - **input( $B$ )** transfers the physical block  $B$  to main memory
  - **output( $B$ )** transfers the buffer block  $B$  to the disk, and replaces the appropriate physical block there
- We assume, for simplicity, that each data item fits in, and is stored inside, a single block

At the bottom left is a video frame showing a man speaking, with the text 'SWAYAM-NPTEL-NCCOCS Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. At the bottom center is the text 'Database System Concepts - 6<sup>th</sup> Edition' and '36.13'. At the bottom right is the copyright notice '©Silberschatz, Korth and Sudarshan'.

Now, to make this kind of a mechanism work we resort to a very simple module of data access. We assume that there are physical blocks on the disk that are on the non-volatile permanent storage and that is where finally, you want your data to decide,

but you also assume that there are system buffer blocks; the blocks that we decide temporarily in the main memory, so, they can be used in the in transit.

So, when you move the block between the disk and the main memory that is initiated by an input operation. So, you are doing an input so, all the physical block a that is disk a block physical block B is brought into the main memory or you have a output operation which transfers first a buffer block B to the disk and replaces the appropriate physical block there. So, these operations when you move physical blocks with the disk you call them as input and output. So, and we are making some assumption that the data that we want to write is small enough so that it fits into a block otherwise there are several schemes of or you know spread your data over multiple blocks.

(Refer Slide Time: 11:40)

The slide has a header 'Data Access (Cont.)' in red. On the left is a small logo of a sailboat. The content is organized into sections with bullet points:

- Each transaction  $T_i$  has its private work-area in which local copies of all data items accessed and updated by it are kept
  - $T_i$ 's local copy of a data item  $X$  is denoted by  $x_i$
  - $B_X$  denotes block containing  $X$
- Transferring data items between system buffer blocks and its private work-area done by:
  - **read( $X$ )** assigns the value of data item  $X$  to the local variable  $x_i$
  - **write( $X$ )** assigns the value of local variable  $x_i$  to data item  $\{X\}$  in the buffer block
- Transactions
  - Must perform **read( $X$ )** before accessing  $X$  for the first time (subsequent reads can be from local copy)
  - The **write( $X$ )** can be executed at any time before the transaction commits
- Note that **output( $B_x$ )** need not immediately follow **write( $X$ )**. System can perform the **output** operation when it deems fit

At the bottom left is vertical text: 'SWAYAM: NPTEL-NOCO's Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr-2018'. At the bottom center: 'Database System Concepts - 8<sup>th</sup> Edition' and '36.14'. At the bottom right: '©Silberschatz, Korth and Sudarshan'.

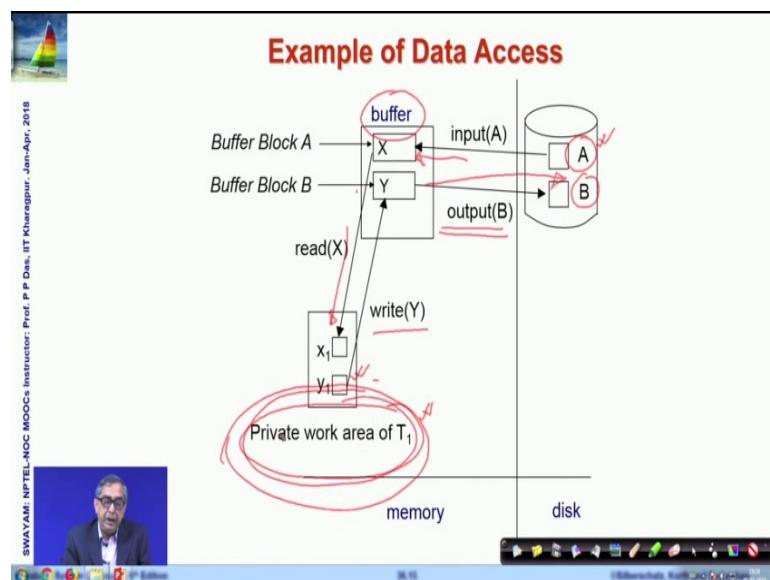
Now, the other part is each transaction on the other side is assumed to have a private work area. So, in the private work area that transaction actually gives local copies and these local copies say you have a data item X, so, you say that for transaction  $T_i$  the copy of that data item X is  $x_i$  and say,  $B_x$  is the block that contains X. So,  $B_x$  is the physical block and then you can transfer data between the transactions private area and this buffer block in terms of read and write operations.

So, we have two kinds of operation; one is input output which is between the memory and the physical block that is the disk and the other is read write operation which is between the transactions private area and the system buffered blocks. So, the transaction

must perform read before accessing X for the first time and once it is done that it has a local copy now and therefore, subsequent reads can happen from the local copy and the write can be executed at any time before the transaction actually commits.

So, let us look at also it is a fact is that the when I want to actually output the block that contains X, I mean your item X to be finally, written to disk the output B x need not immediately happen after you write. So, you are doing it in two stages, from the transactions private area to the system buffer, to the system buffer to the disk. So, first is write the next is output, but this may not actually follow immediately once the data exists in the system buffer it may be actually output on a at a later time whenever it is then fit to do that.

(Refer Slide Time: 13:32)



So, let us take a very quick looks schematically here to make things some simple understandable. So, they are data items A and B on the disk that we are talking of. So, if I do an input operation then I am actually trans and this is the buffer this is the system buffer. So, this is kind of a common buffer where you can keep data, we will see how to manage this system buffer and this is the private work area of a transaction say T<sub>1</sub>.

So, to process of read I mean if T<sub>1</sub> wants to if T<sub>1</sub> wants to read A then the that read initiation will bring A onto the buffer area as X and then it will read X as x<sub>1</sub> in its private area, it will this is where it will do the work. This is a private area where T<sub>1</sub> will do the work and possibly it has generated a write item y with which needs to go back to the

disk. So, again we will do a write to the buffer area and then at a later point there will be an output which will take this Y back to the disk.

So, this will ensure that the transaction can after reading the transaction can independently do the write to the buffers and outputs can happen independently of that either before that transaction commits or even after the transaction commits there are difference in situation there are different protocols that are followed and we will see through, but this is the basic simple model that will regularly be used.

So, please keep in mind we will talk about often will talk about three areas work area the private work area of a transaction this is an memory and the system buffer blocks where the data is the temporarily deciding on the way of being read or on the way of being written and the system disk where the physical block exists. And, this is the path way through this system buffer that the read writes will output will happen and please remember that we will use the term read write when it is between the private work area of a transaction and the system buffer block and will talk about input output when it is between in terms of the physical block with the disk.

So, the data access you can I have already explained. So, in terms of the data access these are the steps that the transaction will do to read or write as I have already explained. Now, in terms of now, let us see that how will in the background of such a storage access how will the recovery happen and how will the atomicity be guaranteed.

(Refer Slide Time: 16:20)

The slide has a title 'Recovery and Atomicity' in red at the top right. On the left, there is a small image of a sailboat on water. A vertical sidebar on the left contains the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018'. The main content area lists several bullet points:

- To ensure atomicity despite failures, we first output information describing the modifications to stable storage without modifying the database itself
- We study **log-based recovery mechanisms** in detail
  - We first present key concepts
  - And then present the actual recovery algorithm
- Less used alternative: **shadow-paging**
- In this Module we assume serial execution of transactions
- In the next Module, we consider the case of concurrent transaction execution

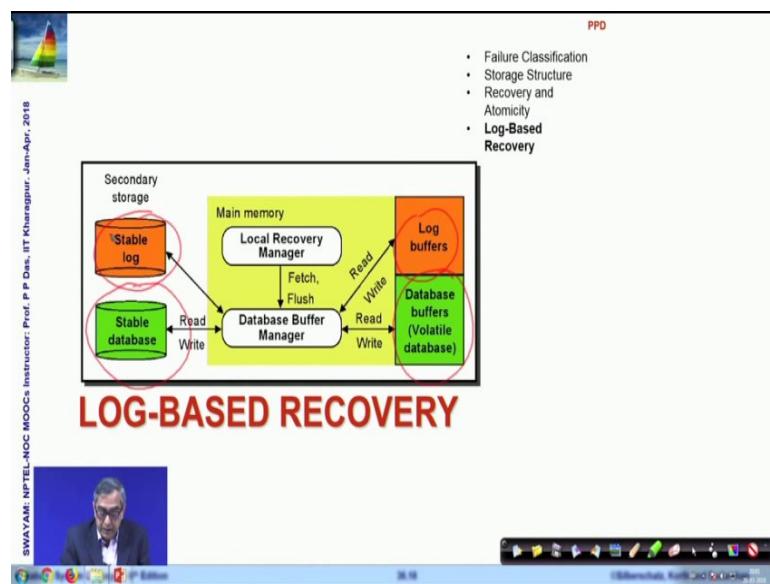
At the bottom left is a video player showing a person speaking, with the text 'Database System Concepts - 8<sup>th</sup> Edition'. At the bottom right is a navigation bar with icons for back, forward, search, and other presentation controls. The footer also includes the text '©Silberschatz, Korth and Sudarshan'.

So, to ensure atomicity in the phase of failure we need to output information describing the modifications to stable storage with input modifying database itself. So, what we are saying that to be able to recover that we should write the changes to the stable storage you recall that stable storage something that is assumed to be not failing without actually modifying database.

Now, we do a very simple mechanism which is called a log based recovery mechanism. So, we will first talk about this log based recovery mechanism what are the key concepts of logging and redo undo redo kind of operations and present the actual recovery algorithm. There are other alternatives also like shadow paging we will not discuss about that and I would like to again remind you that in this module we are talking about single transactions at a time, serial execution.

In the next module we will talk about concurrency of the; I mean the behavior of recovery algorithms and in the case of concurrent transactions.

(Refer Slide Time: 17:29)



So, now let us talk about the log based recovery mechanism. So, in the log based recovery mechanism you can see this is the basic this is your stable database which you want to make use of, these are your buffers you talked off and we will have certain logs the information of what have been doing in terms of the log buffers and the also is stable log which is a log that is written in the stable database. So, once we understand what is logging you will understand this, but I just wanted to show you that

like the data there are buffer copies as well as stable database copies in terms of log also there will be buffer copies as well as stable, log copies.

(Refer Slide Time: 18:18)

The slide has a title 'Log-Based Recovery' at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list of points about log-based recovery:

- A **log** is kept on stable storage
  - The log is a sequence of **log records**, which maintains information about update activities on the database
- When transaction  $T_i$  starts, it registers itself by writing a record  $\langle T_i, \text{start} \rangle$  to the log
- Before  $T_i$  executes **write(X)**, a log record  $\langle T_i, X, V_1, V_2 \rangle$  is written, where  $V_1$  is the value of  $X$  before the write (the **old value**), and  $V_2$  is the value to be written to  $X$  (the **new value**)
- When  $T_i$  finishes its last statement, the log record  $\langle T_i, \text{commit} \rangle$  is written
- Two approaches using logs
  - Immediate database modification
  - Deferred database modification

At the bottom left, there is a small video thumbnail showing a man speaking, with the text 'SWAYAM-NIITL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr, 2018'. At the bottom center, it says 'Database System Concepts - 8<sup>th</sup> Edition'. At the bottom right, it shows a Mac OS X dock with various icons and the text '36.19 ©Silberschatz, Korth and Sudarshan'.

So, a log is kept in the stable storage, it is a sequence of records. So, log is basically a record of what and. So, it is like if am doing some task we have always every task I do I keep a record of what am actually be doing and that is called the logging. So, when a transaction starts I write a log record which puts the transaction ID say  $T_i$  and then puts a keyword start to the log. So, that indicates that the transaction  $T_i$  has started and when it is about to execute a write, so, ma before it has actually executed the write, then I write a log record which looks like this which is.

So, here if we look carefully this is the idea of the transaction  $X$  is the data item that you want in to write  $V_1$  is the current value of the data item which kind of we can say is the old value and  $V_2$  is the value that we want to actually write. So, here you can see that we are clearly keeping a track of what we are writing and in that process what is the original value that would get changed. So, that is the main important factor of this logging that every with every write you remember as to what value was originally there and what value we have actually changed it to in that transaction.

Now, in this process finally, when that the transaction finishes the last statement of the log record is  $T_i$  commit. So, that actually is a meaning of committing a transaction when

this log record is written out. So, that is. So, a log will have start then different write log records and then finally, a commit log record.

So, there are basically two approaches of using log, one is called immediate database modification this is what we would follow here and there is a differed database modification.

(Refer Slide Time: 20:19)

**Database Modification**

- The **immediate-modification** scheme allows updates of an uncommitted transaction to be made to the buffer, or the disk itself, before the transaction commits
- Update log record must be written **before** a database item is written
  - We assume that the log record is output directly to stable storage
- Output of updated blocks to disk storage can take place at any time before or after transaction commit
- Order in which blocks are output can be different from the order in which they are written
- The **deferred-modification** scheme performs updates to buffer/disk only at the time of transaction commit
  - Simplifies some aspects of recovery
  - But has overhead of storing local copy
- We cover here only the immediate-modification scheme

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jun-Apr, 2018

Database System Concepts - 8<sup>th</sup> Edition

36.20

©Silberschatz, Korth and Sudarshan

In the immediate modification scheme the ma it allows updates of an uncommitted transaction to be made to the buffer or the disk itself before the transaction commit. So, before the transaction has committed that is before the  $\langle T_i, \text{commit} \rangle$  log record has been written at that point itself you allow the updates of the transaction to be made to the buffer or the disk and the update log record must be written before the database is actually written. So, you must first write the log and then actual database item. So, and we assume that the log record is output directly to the stable storage. So, that it is not there is no possibility of is getting lost.

Now, output of the updated blocks to disk storage can take place, that is the final actual output this is where we have written the log that that am doing this change, but the actual change we can take place any time before the transaction commits or even after the transaction commits. If you follow this ma protocol then you say you are in the immediate modification scheme and in fact, the order in which the blocks are output that finally, written to the disk may be different from the order in which they were originally

written, but the log records the will have to be written before these each one of this output are done.

In the deferred modification scheme the change updates are performed to buffer and disk only at the time of transaction commit not any time before that. So, that simplify some aspects of recovery, but it has other issues. So, we will not talk about this scheme, just know that there is an alternate scheme for doing things.

(Refer Slide Time: 22:03)

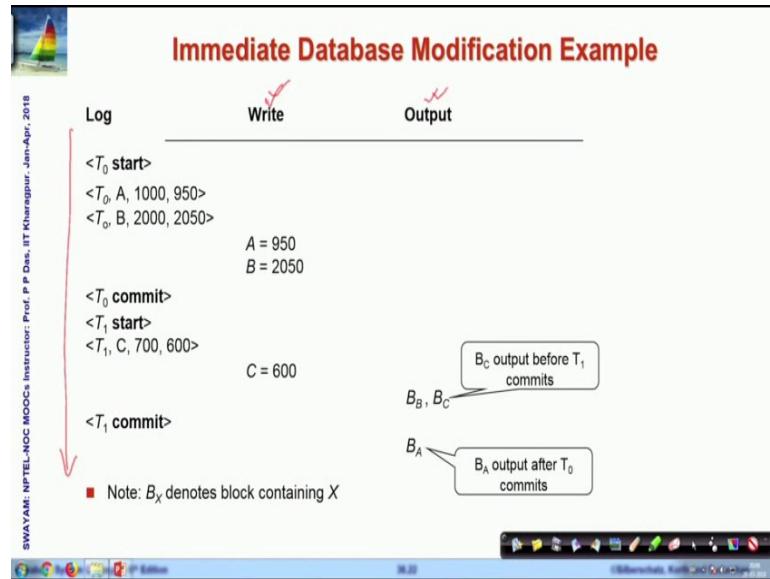
The slide has a title 'Transaction Commit' in red at the top right. To the left of the title is a small image of a sailboat on water. On the far left edge of the slide, there is vertical text that reads: 'SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr, 2018'. At the bottom left is a video frame showing a man with glasses speaking. The bottom of the slide contains navigation icons and the text 'Database System Concepts - 8<sup>th</sup> Edition' on the left, '36.21' in the center, and '©Silberschatz, Korth and Sudarshan' on the right.

- A transaction is said to have committed when its commit log record is output to stable storage
  - All previous log records of the transaction must have been output already
- Writes performed by a transaction may still be in the buffer when the transaction commits, and may be output later

So, now formally speaking what is transaction commit? A transaction commit is said transaction is said to have committed if its commit log record is output to the stable storage. That is  $T_i$  commit has gone to the stable storage is the meaning of the transaction has been committed. Obviously, all previous log records of the transactions must have been outputted already because those commit those outputs will have happen in the same order in which the actions are taken.

Now, the writes performed by the transaction may still be in the buffer. So, you have transaction is committed everything is done, but your actual writes that are performed may not have been outputted. They are they may still be in the buffer when the transaction commits and those may be output at a later point of time.

(Refer Slide Time: 22:52)



So, let us take an example here. Let us look at an example. So, here you see the log records and here is the sequence of write and output  $A_1$  is happening. So, in the log record the transaction starts here. So, you have a log record of start, what is the meaning of this? The meaning of this is transaction  $T_0$  is trying to write  $A$  and the current value is 1000 and it wants to change it to 950. So, this log record is written and you can see that the actual write actual write has not happened here, actual write is not done, but it is already has must like in the immediate database modification scheme it must write the log record before actually writing the output, actually doing the output or doing the write. So, this has happened here.

Similarly, the next one is another update transaction for  $B$  and actual writes have happened. So, which means the data has been written from the transactions private work area to the system buffer and then the transaction has transaction  $T_0$  has done commits. So, at this point if you go up to this then the commit of the transaction is already completed and another transaction  $T_1$  starts you please remember that we have said that we will we are using serial ma schedules only. So, only now another transaction can commit that has started and that has written log record for updating  $C$  from 700 to 600. So, there is a write for 600 then  $T_1$  has commit.

In the meanwhile and at this stage, in the meanwhile these blocks have been output. So, they have actually been written the disk and you can understand that this block  $B_B$  is a

block that contains the data of data item the updated value of data item B and ma this B C has the updated value of data item C. So, you can have see that actually these output of B is happening after the transactions is  $T_0$  has committed whereas, for update of data you can see the output is happening af before the  $T_1$  has committed. So, here it is happening after the commit, but here it is happening before the commit.

So, both of these are permitted both of these are allowed in terms of the protocol that we are following. And, you can also see that in terms of the order in which they were written A was written earlier, but A is output at a later point of time because that is a different sequence in which the system might decide for writing the buffer onto the disk.

So, this is the immediate database modification scheme through which we can write the logs.

(Refer Slide Time: 25:51)

**Undo and Redo Operations**

- Undo of a log record  $<T_i, X, V_1, V_2>$  writes the old value  $V_1$  to  $X$
- Redo of a log record  $<T_i, X, V_1, V_2>$  writes the new value  $V_2$  to  $X$
- Undo and Redo of Transactions
  - $\text{undo}(T_i)$  restores the value of all data items updated by  $T_i$  to their old values, going backwards from the last log record for  $T_i$ 
    - Each time a data item  $X$  is restored to its old value  $V$  a special log record (called redo-only)  $<T_i, X, V>$  is written out
    - When undo of a transaction is complete, a log record  $<T_i, \text{abort}>$  is written out (to indicate that the undo was completed)

SWAYAM-NIPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018

Now, the question is we have written the logs. So, what is the use of those logs? Naturally, the use of those logs are in terms of two operations to which we say are undo operation and redo operation and undo operation is one which basically undoes the operation the effect of an update. So, while ma you have done you have if this is a log record then undoing, so, this meant that  $X$  was changed it had a value  $V_1$  and it was changed to  $V_2$ . If you undo that then the old value comes back to this old value comes back to  $X$ . So, that is if I undo this particular action the which was put in the log record then  $X$  will get back it is original value and redo is doing the same thing over again if I

redo for this log record then the value of  $V_2$  will again reset on X. So, these are the two simple undo and redo operations which will help us achieve the recovery systems input.

So, what is meant by undo redo of transactions let us understand. So, when I undo a transaction  $T_i$  that restores the values of all data items updated by  $T_i$  to their old values. So, the values have been updated in this forward order. So, when you go to undo you will actually have to do that in the reverse order, because it is quite possible that X got 1 here then at some point it was updated to 17 then at some later point it was updated to 13. So, this update possibly had happened from 0, this update had happened from 1, this update had happened from 3. So, all those transactions records are there then you going backwards. So, you will first restore X back to 17 because this is then this back restoring back to 1 then going back to 0, in this order it will go on.

And, every time you restore you write that you write that out as a record which is known as redo, redo only record. So, you can see that here you are not trying to remember the original value you are just writing the value that you have written out in terms of the undo operation that is the old value and the going in this manner undo operation will terminate when you have come across the beginning of this one process, when it is complete then a log record  $\langle T_i, \text{abort} \rangle$  is written out which says that the undo is actually over. So, this is the undo operation.

(Refer Slide Time: 28:36)

The slide has a title 'Undo and Redo Operations' in red. To the left is a small logo of a sailboat on water. On the right is a video frame showing a person speaking. The slide content is as follows:

■ Undo of a log record  $\langle T_i, X, V_1, V_2 \rangle$  writes the **old** value  $V_1$  to X  
■ Redo of a log record  $\langle T_i, X, V_1, V_2 \rangle$  writes the **new** value  $V_2$  to X  
■ Undo and Redo of Transactions

- $\text{undo}(T_i)$  restores the value of all data items updated by  $T_i$  to their old values, going backwards from the last log record for  $T_i$ 
  - Each time a data item X is restored to its old value V a special log record (called **redo-only**)  $\langle T_i, X, V \rangle$  is written out
  - When undo of a transaction is complete, a log record  $\langle T_i, \text{abort} \rangle$  is written out (to indicate that the undo was completed)
- $\text{redo}(T_i)$  sets the value of all data items updated by  $T_i$  to the new values, going forward from the first log record for  $T_i$ 
  - No logging is done in this case

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018

For redo you said that the redo is doing the transactions doing the same instructions of the transactions in the same manner, it was done earlier. So, that unlike undo which goes backwards redo goes forward and it starts from the first log record of this transaction and goes on till the end and for this there is no separate logging for this operation.

(Refer Slide Time: 29:04)

The slide has a title 'Undo and Redo Operations (Cont.)' at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list of points about undo and redo operations. At the bottom left, there is a video player showing a person speaking. The bottom right corner shows the copyright information '©Silberschatz, Korth and Sudarshan'.

**SWAYAM-NIPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018**

**Undo and Redo Operations (Cont.)**

- The **undo** and **redo** operations are used in several different circumstances:
  - The **undo** is used for transaction rollback during normal operation
    - ▶ in case a transaction cannot complete its execution due to some logical error
  - The **undo** and **redo** operations are used during recovery from failure
- We need to deal with the case where during recovery from failure another failure occurs prior to the system having fully recovered

Database System Concepts - 8<sup>th</sup> Edition

36.24

©Silberschatz, Korth and Sudarshan

Now, how will the undo redo operations be used? There are two major situations in which they are used one is undo is used transactions roll back have to roll back during normal operation. That is nothing has I mean there is no system failure or there is no data disk failure anything, but if the transaction has a normal failure that it cannot complete it is execution due to some logical error or because it has to roll back because of deadlock or something, then you what you do you just undo the whole effect of the transaction go backwards and keep on undoing. But, when the there is a failure there is a failure and you have to recover from that then undo and redo operations both will be required as we will soon see.

So, we also need to deal with the case where the recovery from failure while you are recovering from failure another failure happens. So, what do you do in that case that is more complicated will talk about that later?

(Refer Slide Time: 30:04)

The slide has a title 'Transaction rollback (during normal operation)' at the top right. On the left, there is a small sailboat icon. The main content is a bulleted list of steps:

- Let  $T_i$  be the transaction to be rolled back
- Scan log backwards from the end, and for each log record of  $T_i$  of the form  $\langle T_i, X_j, V_1, V_2 \rangle$ 
  - Perform the undo by writing  $V_1$  to  $X_j$
  - Write a log record  $\langle T_i, X_j, V_1 \rangle$ 
    - ▶ such log records are called **compensation log records**
- Once the record  $\langle T_i, \text{start} \rangle$  is found stop the scan and write the log record  $\langle T_i, \text{abort} \rangle$

At the bottom left, there is a photo of a man, and the text 'SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P.P. Desai, IIT Kharagpur - Jan-Apr. 2018'. At the bottom center, it says 'Database System Concepts - 8<sup>th</sup> Edition' and '36.25'. At the bottom right, it says '©Silberschatz, Korth and Sudarshan'.

So, first let us discuss what happens when you roll back a transaction during normal operation. So, let  $T_i$  be the transaction. So, you have to naturally do the undo because you have to undo the effect that it has already created. So, you will scan the log records from the end and for each log record which is kind of an update like this you will perform an update to restore the original value the old value and write out a redo only log record or which is called compensation log record which says that this has been undone to the value  $V_1$  which is the original value of the transaction original value of the data item sorry.

Now, in going in this process backwards at some point of time you will reach come across  $\langle T_i, \text{start} \rangle$  log record when you face come across that you write log record  $\langle T_i, \text{abort} \rangle$  indicating that the undo of that transaction is over. So, this is the basic process of undoing the transactions during rollback.

(Refer Slide Time: 31:07)

**Undo and Redo on Recovering from Failure**

When recovering after failure:

- Transaction  $T_i$  needs to be undone if the log
  - contains the record  $\langle T_i, \text{start} \rangle$ ,
  - but does not contain either the record  $\langle T_i, \text{commit} \rangle$  or  $\langle T_i, \text{abort} \rangle$
- Transaction  $T_i$  needs to be redone if the log
  - contains the records  $\langle T_i, \text{start} \rangle$
  - and contains the record  $\langle T_i, \text{commit} \rangle$  or  $\langle T_i, \text{abort} \rangle$
- It may seem strange to redo transaction  $T_i$  if the record  $\langle T_i, \text{abort} \rangle$  record is in the log
  - To see why this works, note that if  $\langle T_i, \text{abort} \rangle$  is in the log, so are the redo-only records written by the undo operation. Thus, the end result will be to undo  $T_i$ 's modifications in this case. This slight redundancy simplifies the recovery algorithm and enables faster overall recovery time
  - such a redo redoes all the original actions including the steps that restored old value – known as **repeating history**

SWAYAM, NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
36.26 ©Silberschatz, Korth and Sudarshan

In the other case if you are recovering from a failure if there has been a failure then you do something which needs to be understood carefully. So, the transaction  $T_i$  needs to be undone if the log contains the record start  $\langle T_i, \text{start} \rangle$ , but it does not contain either  $\langle T_i, \text{commit} \rangle$  or  $\langle T_i, \text{abort} \rangle$ . So, perform  $\langle T_i, \text{start} \rangle$  you will know that it has started, but because of failure it could not complete, because if it could complete or if before that if it had to roll back because of the normal execution then it would have written  $\langle T_i, \text{commit} \rangle$  or  $\langle T_i, \text{abort} \rangle$ , but because of system failure you could not write any one of them. So, the transaction has to be rolled back.

The other case is the case where the transaction needs to be redone is when then it contains the record  $\langle T_i, \text{start} \rangle$ , but in addition it also contains the record  $\langle T_i, \text{commit} \rangle$  or  $\langle T_i, \text{abort} \rangle$ . So, this is the transaction which had completed successfully, did the start, it did the commit or it rolled back the whole thing happened successfully, but because of system failure changes have not been able to take place and therefore, you will have to again execute that transactions. So, that is why you do a redo. In the earlier case it is undone you want to undo the effect, here the effects were given, but they somehow could not be made durable the database have become inconsistent. So, you need to redo that whole thing.

So, ma it may sound little bit awkwardness that if the it contains the  $\langle T_i, \text{abort} \rangle$  why should you actually redo the transaction this is a just to keep things simple so that you

can just trace back the original history. So, you do not try to really optimize, but you just trace back the original history and do whatever had happened in the way and then that simplifies your algorithm significantly. And, then if there is a certain things which have been done by the undo operation you also want to go through those and maintain that status.

(Refer Slide Time: 33:23)

The slide has a title 'Immediate Modification Recovery Example' with a sailboat icon. It contains three log entries labeled (a), (b), and (c) showing transactions  $T_0$  and  $T_1$  interacting with objects A, B, and C.

	$<T_0 \text{ start}>$	$<T_0 \text{ start}>$	$<T_0 \text{ start}>$
(a)	$<T_0, A, 1000, 950>$	$<T_0, A, 1000, 950>$	$<T_0, A, 1000, 950>$
	$<T_0, B, 2000, 2050>$	$<T_0, B, 2000, 2050>$	$<T_0, B, 2000, 2050>$
	$<T_0 \text{ commit}>$	$<T_0 \text{ commit}>$	$<T_0 \text{ commit}>$
	$<T_1 \text{ start}>$	$<T_1 \text{ start}>$	$<T_1 \text{ start}>$
	$<T_1, C, 700, 600>$	$<T_1, C, 700, 600>$	$<T_1, C, 700, 600>$
			$<T_1 \text{ commit}>$

Recovery actions in each case above are:

- undo ( $T_0$ ): B is restored to 2000 and A to 1000, and log records  $<T_0, B, 2000>$ ,  $<T_0, A, 1000>$ ,  $<T_0, \text{abort}>$  are written out
- redo ( $T_0$ ) and undo ( $T_1$ ): A and B are set to 950 and 2050 and C is restored to 700. Log records  $<T_1, C, 700>$ ,  $<T_1, \text{abort}>$  are written out
- redo ( $T_0$ ) and redo ( $T_1$ ): A and B are set to 950 and 2050 respectively. Then C is set to 600

Navigation icons and copyright information are at the bottom.

So, here are some examples here they transaction we are showing it is failure recovery action at every case. So, if in case a the transaction has started and we made changes to A and B and at that time the failure happens. So, naturally the start is there and at commit is not there or abort is no there. So, these has to be undone. So, this will be undone A will get back the value thousand and B we will get back the value 2000 and to such record  $<T_0, B, 2000>$  and  $<T_0, A, 2000>$  that two compensation log records will be and then it  $<T_0, \text{abort}>$  will be written.

Now, if you look at second transaction transaction just a second states of as in b then you will see that  $T_0$  has actually started and committed and  $T_1$  has started after that which could not complete after updating C. So, in the case of b since  $T_0$  has start and commit both you have to redo that because you have lost all these changes we have to redo again and for that you do not log anything and then  $T_1$  could not complete because it has start and does not have the abort or commit. So, you would log record for undoing it, undoing  $T_1$  and you write  $<T_1, C, 100>$  and  $<T_1, \text{abort}>$ .

In the third case ma both transaction T 0 and transaction T 1 has commit start and commit and both have completed. So, you have to redo both of them. So, these are the basic different cases strategies that you have in place.

(Refer Slide Time: 34:59)

The slide has a header 'Checkpoints' with a small sailboat icon. The content lists several points about checkpoints:

- Redoing/undoing all transactions recorded in the log can be very slow
  - Processing the entire log is time-consuming if the system has run for a long time
  - We might unnecessarily redo transactions which have already output their updates to the database
- Streamline recovery procedure by periodically performing **checkpointing**
- All updates are stopped while doing checkpointing
  1. Output all log records currently residing in main memory onto stable storage
  2. Output all modified buffer blocks to the disk
  3. Write a log record <checkpoint L> onto stable storage where L is a list of all transactions active at the time of checkpoint

Navigation icons and slide details are visible at the bottom.

Now, the question is if you have to do this for all transactions when a failure happens and a failure may have happened say after 1 year or after 8, 9, 10 months and so on. So, there will be a huge you know set of redo, undo operations that you will have do it will run for a very long time. So, what we do is we create something like a check pointing where we said, ok. We will periodically choose a point of time where we will make sure that all updates have actually been consistently put in the disk and the database is surely on a consistent state and that is called check pointing.

So, whatever is done is a at a chosen point of check pointing, time of check pointing all updates are stopped in database. So, there is no changing changes happening in all transaction are no new transactions are allowed what is happening as.

So, you make sure that all records that are currently residing in your buffer is flushed on to the stable storage all modified buffer blocks which were not output we have also outputted and then you write that this is a write a log record saying the check point L on to the stable storage, where L is basically the transactions that were active at the time of checkpoint in the transactions that have already completed you do not need to remember because they have they are changes by the process of outputting all log records and all

modified buffer on to disk, you make sure that all completed transactions are fully secured now, they are consistent, they are you would have to, but those which are which were still continuing you keep the list and write that out in terms of the checkpoint log record and take it into the stable storage.

(Refer Slide Time: 36:52)

The slide has a header 'Checkpoints (Cont.)' with a small sailboat icon. The main content is a bulleted list:

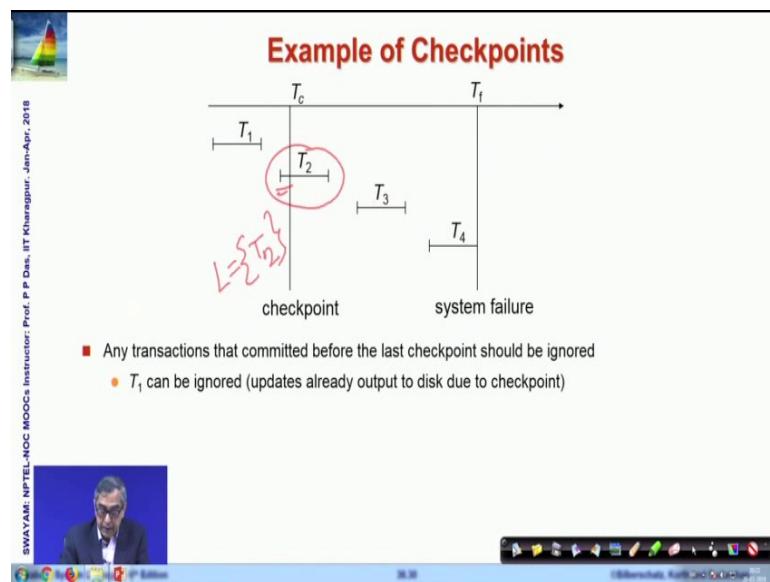
- During recovery we need to consider only the most recent transaction  $T_i$  that started before the checkpoint, and transactions that started after  $T_i$ 
  - Scan backwards from end of log to find the most recent <checkpoint L> record
  - Only transactions that are in  $L$  or started after the checkpoint need to be redone or undone
  - Transactions that committed or aborted before the checkpoint already have all their updates output to stable storage
- Some earlier part of the log may be needed for undo operations
  - Continue scanning backwards till a record < $T_i$ , start> is found for every transaction  $T_i$  in  $L$
  - Parts of log prior to earliest < $T_i$ , start> record above are not needed for recovery, and can be erased whenever desired

At the bottom left is a video frame showing a man speaking. On the right is a navigation bar with icons. The footer includes text: 'SWAYAM-NIPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr- 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '36.29', and '©Silberschatz, Korth and Sudarshan'.

So, during recovery what we need to consider is we have to now we not have to go back to the last time the disk fail we just need to go back to the last time we did check pointing and when you go back to the check pointing you already know that ma what are the transactions that are live at the time of check pointing. So, you can scan backwards and check out what were they had started. So, you need to and undo redo those are transactions and then you see what are the transactions that have committed aborted have already there in the output in the stable storage.

So, some of the earlier part of the log may need may be needed for undo operations. So, you continue scanning backwards till you find in < $T_i$ , start> and then you take care of that.

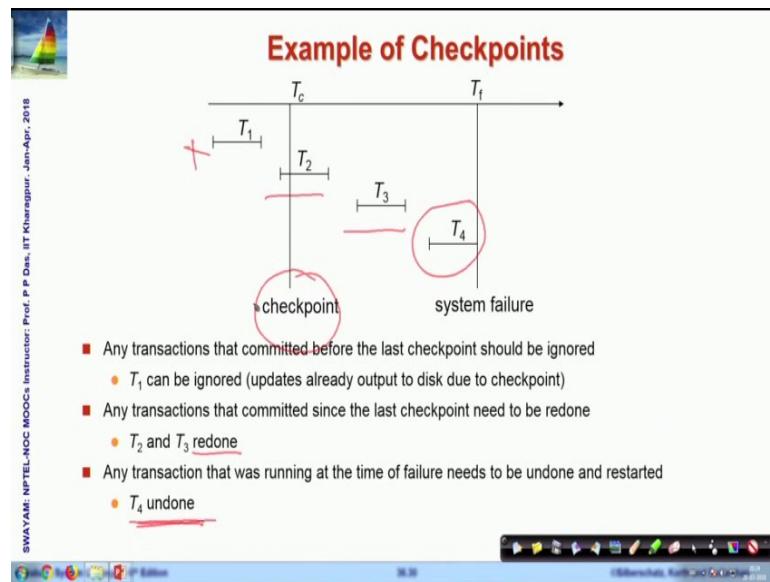
(Refer Slide Time: 37:48)



Let me just explain through an example. So, let us say that this is the checkpoint where you froze everything and did not allow any further updates to happen and rolled back all the data. So, what has happened is in this transaction  $T_1$  which is committed before the last checkpoint naturally there was no obtained pending for that. So, you have made sure that all the updates in terms of the log as well as the system buffer has been written on to have been output on to the disk at the time of check pointing. So, you do not remember need to remember this transaction at all, so this can simply be ignored.

Now, at the checkpoint you can see that transaction  $T_1$  was in execution. So, certain things had happened. So, at the checkpoint the part that has already happened the log records for that as well as the output for that this has already been firmly put into that because these are checkpoints because you are writing everything, but this transaction is still in execution. So, you will put this transaction in the checkpoint log list. So, we will say this is this is the  $T_2$  and this will need to be looked at. If you so, let us see what is you will do with this.

(Refer Slide Time: 39:02)

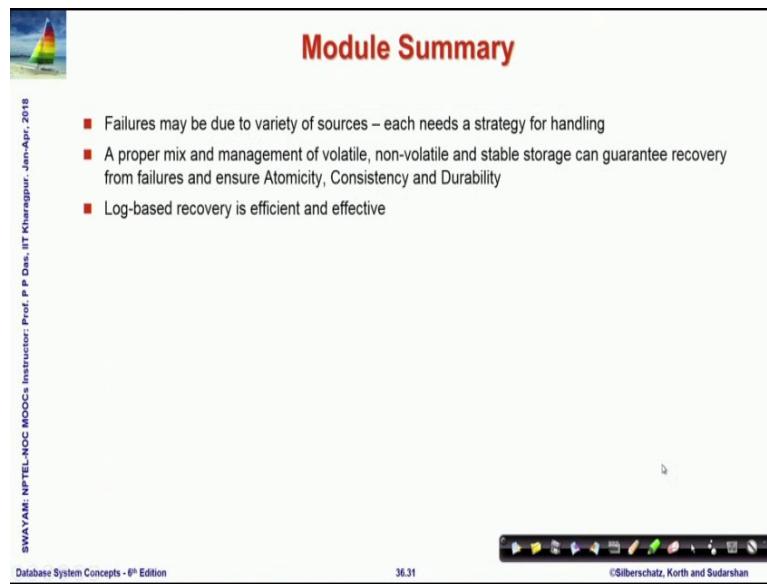


So, if we look at then naturally  $T_2$  if you have a failure at this point if you have a failure at this point as you have here then naturally this is the last stable point you know where everything was written to the disk in a consistent manner. So, what you will need to do you will have to execute transaction  $T_2$  once more to make sure that it is you know up to this point this being done, so, you need to redo this part. Similarly,  $T_3$  if you look at it started after the checkpoint and it committed before the system failure. So, you need to redo that as well.

And, if you look into the  $T_4$  if you look into  $T_4$  you can see that it started before the system failure of course, after the checkpoint and it was still running when the failure happens. So, you do not know what are the final results of that, so, what you will need to do? You will need to undo this transaction. So, this has no impact you can just ignore these cases you have to redo the transaction and in this case, in case of  $T_4$  you have to undo the transaction.

So, by check pointing and you obviously, the point you choose for check pointing has to be judiciously done it may not be very frequent and then it will there be a lot of overhead at the same time if you do it in a very long period of time then naturally you will not get the benefits, but check pointing is a very critical feature of doing the recovery in the databases.

(Refer Slide Time: 40:36)



The slide has a header 'Module Summary' with a sailboat icon. It contains a bulleted list of three items. The footer includes course details, a navigation bar, and copyright information.

**Module Summary**

- Failures may be due to variety of sources – each needs a strategy for handling
- A proper mix and management of volatile, non-volatile and stable storage can guarantee recovery from failures and ensure Atomicity, Consistency and Durability
- Log-based recovery is efficient and effective

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P.P. Desai, IIT Kanpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition      36.31      ©Silberschatz, Korth and Sudarshan

So, to summarize we have seen that there are many different types of failures and different strategies are required for handling them. And we have also seen that we use different kinds of storage structures and their judicious mix and then arrangement of these structures can guarantee recovery from failures. And we have taken a brief look into the log base recovery mechanism which is efficient as well as effective and I will remind you that all this discussion was done for serial transactions only.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 37**  
**Recovery/2**

---

Welcome to module 37 of Database Management Systems. We have been discussing about Database Recovery. This is the second and concluding part of the Database Recovery.

(Refer Slide Time: 00:25)

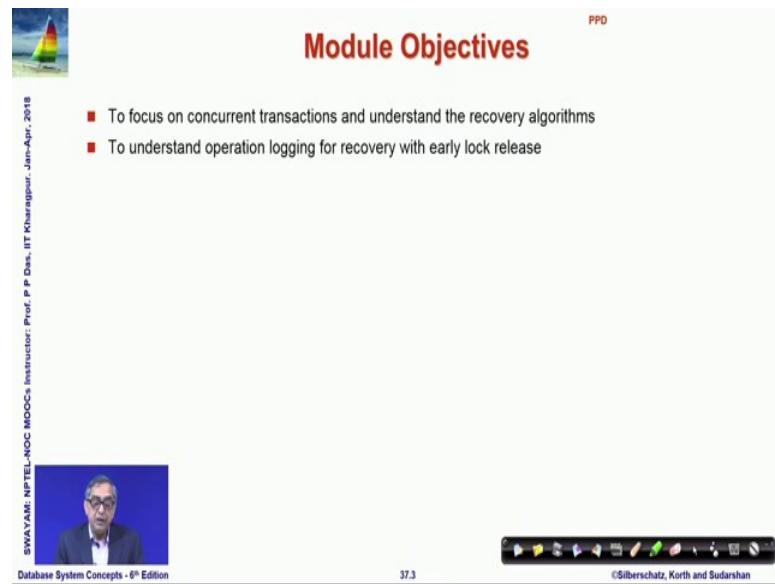
The slide is titled "Module Recap" in red at the top right. On the left, there is a small image of a sailboat on water. The footer contains copyright information: "SWAYAM: NPTEL-MOOCs Instructor: Prof. P P Das, IIT Kharagpur", "Database System Concepts - 8<sup>th</sup> Edition", "37.2", and "©Silberschatz, Korth and Sudarshan". A navigation bar with various icons is at the bottom right.

**Module Recap**

- Failure Classification
- Storage Structure
- Recovery and Atomicity
- Log-Based Recovery

We have earlier discussed about failure classification, storage structures and significantly the log based recovery mechanism.

(Refer Slide Time: 00:34)

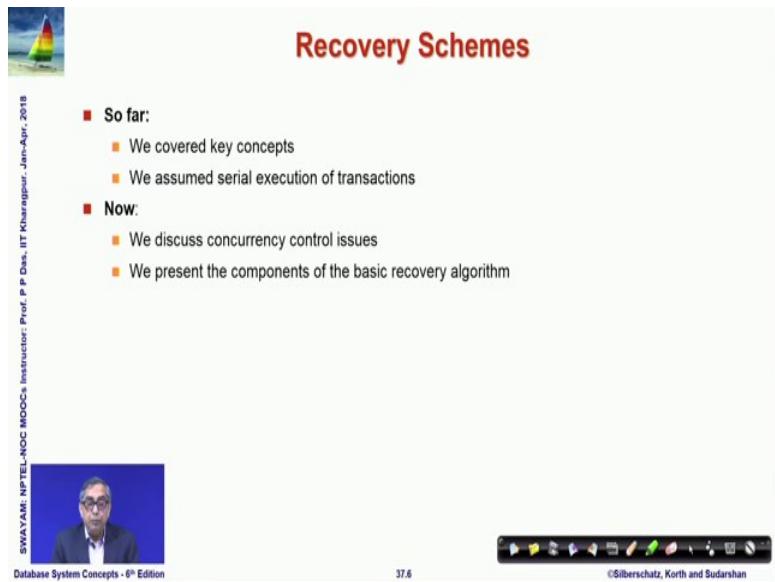


The slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. On the left side, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs", "Instructor: Prof. P. P. Das, IIT Kharagpur", and "Jan-Apr, 2018". In the center, there is a video frame showing a man speaking. At the bottom left, it says "Database System Concepts - 6<sup>th</sup> Edition". The bottom right contains a navigation bar with icons and the text "©Silberschatz, Korth and Sudarshan". The slide number "37.3" is at the bottom center.

- To focus on concurrent transactions and understand the recovery algorithms
- To understand operation logging for recovery with early lock release

In this, we will focus on concurrent transactions and understand the recovery algorithm for them and we will understand the operation of logging for recovery with early lock release. We will learn about another kind of logging mechanism; so, the recovery algorithm.

(Refer Slide Time: 00:51)



The slide is titled "Recovery Schemes" in red at the top right. It features a small sailboat icon in the top left corner. On the left side, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs", "Instructor: Prof. P. P. Das, IIT Kharagpur", and "Jan-Apr, 2018". In the center, there is a video frame showing a man speaking. At the bottom left, it says "Database System Concepts - 6<sup>th</sup> Edition". The bottom right contains a navigation bar with icons and the text "©Silberschatz, Korth and Sudarshan". The slide number "37.6" is at the bottom center.

- So far:
  - We covered key concepts
  - We assumed serial execution of transactions
- Now:
  - We discuss concurrency control issues
  - We present the components of the basic recovery algorithm

So, what we have seen so far are we have learned the basic concept of recovery and logging and we have assumed the serial execution of transactions and now we discussed

the Concurrency Control issues. So, now, we will assume that there are multiple transactions operating at the same time and the components that are required for the recovery of those.

(Refer Slide Time: 01:10)

The slide has a title 'Concurrency Control and Recovery' in red. To the left of the title is a small icon of a sailboat on water. Below the title is a video player window showing a man in a suit speaking. The video player has a progress bar and several control icons. On the left side of the slide, there is vertical text that reads 'SVAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018'. At the bottom left is the text 'Database System Concepts - 8<sup>th</sup> Edition'. At the bottom right is the text '©Silberschatz, Korth and Sudarshan'.

- With concurrent transactions, all transactions share a single disk buffer and a single log
  - A buffer block can have data items updated by one or more transactions
- We assume that *if a transaction  $T_i$  has modified an item, no other transaction can modify the same item until  $T_i$  has committed or aborted*
  - That is, the updates of uncommitted transactions should not be visible to other transactions
    - ▶ Otherwise how do we perform undo if  $T_1$  updates A, then  $T_2$  updates A and commits, and finally  $T_1$  has to abort?
  - Can be ensured by obtaining exclusive locks on updated items and holding the locks till end of transaction (strict two-phase locking)
- Log records of different transactions may be interspersed in the log

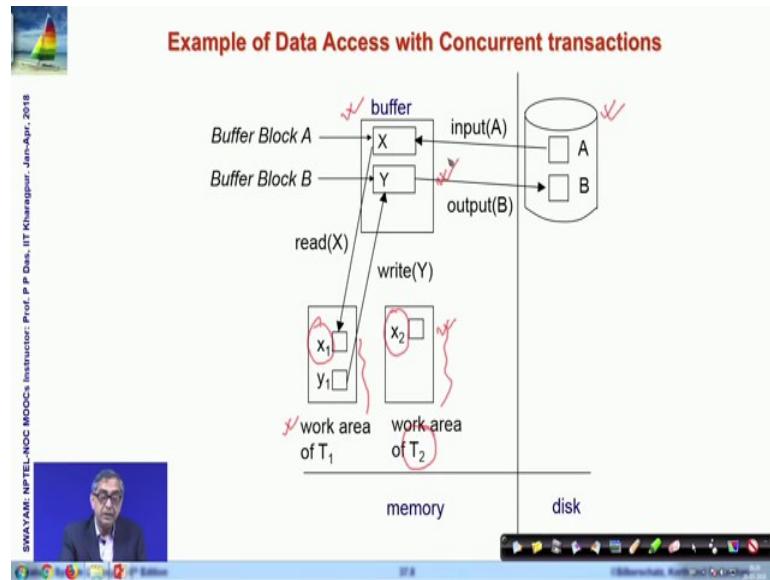
So, with Concurrent Transaction, all transactions share we already know that every transaction is a private work area that assumption stays, but we talked about a system buffer area.

So, that system buffer area the that area would be common for all different transactions, also the log area would be common for all transactions. So, now, the in the buffer area the, data rights are or reads or writes are done for different transactions and in the log the different logs of different transactions are fixed up. So, we make one assumption that if a transaction has modified an item, then no other transaction can modify that same item unless that transaction is committed or aborted.

So, which means that kind of when the transaction modifies the item it holds a lock and that lock is held till the end of the transaction and this is a I mean if we if we think back in terms of our locking protocol, this is a strict locking protocol that we are talking of. This is important for recovery because if we did not have this, then it is possible that multiple updates to the same rate item are done by multiple transactions. So, if we have

to undo that then we will not know we were which one it to be undone with. So, that is the basic problem. So, we will assume an exclusive lock in this case and log records will be written interspersed as we have already saw.

(Refer Slide Time: 02:43)



So, in terms of our storage access mechanism, the same eh earlier diagram, this is here is a disk, here is a buffer, the buffer is common and the private work area. So, now, in addition to  $T_1$ , we have another transaction  $T_2$  with its own buffer area, but so, the  $x$  has been written in  $T_1$ , has been read in  $T_1$  as  $x_1$ ,  $x$  has also been read in  $T_2$  as  $x_2$  and each are concurrently making changes in that private work area, but they are using the same system buffer area for the for writing the output back to the disk or reading directly from the disk. So, this is a model that we will go with.

(Refer Slide Time: 03:30)

The slide features a title 'Recovery Algorithm' in red at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list under the heading 'Logging (during normal operation):'. The list includes:

- $\langle T_i \text{ start} \rangle$  at transaction start
- $\langle T_i, X_p, V_1, V_2 \rangle$  for each update, and
- $\langle T_i \text{ commit} \rangle$  at transaction end

At the bottom left, there is a video player showing a man speaking, with the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr- 2018'. At the bottom right, it says 'Database System Concepts - 8<sup>th</sup> Edition', '37.9', and '©Silberschatz, Korth and Sudarshan'.

So, what is the recovery algorithm, first is logging and the logging structure remains same; the start transaction log, the update transaction log and the commit transaction log as before.

(Refer Slide Time: 03:43)

The slide features a title 'Recovery Algorithm (Contd.)' in red at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list under the heading 'Transaction rollback (during normal operation)'. The list includes:

- Let  $T_i$  be the transaction to be rolled back
- Scan log backwards from the end, and for each log record of  $T_i$  of the form  $\langle T_i, X_p, V_1, V_2 \rangle$ 
  - perform the undo by writing  $V_1$  to  $X_p$
  - write a log record  $\langle T_i, X_p, V_1 \rangle$ 
    - such log records are called **compensation log records**
- Once the record  $\langle T_i \text{ start} \rangle$  is found stop the scan and write the log record  $\langle T_i \text{ abort} \rangle$

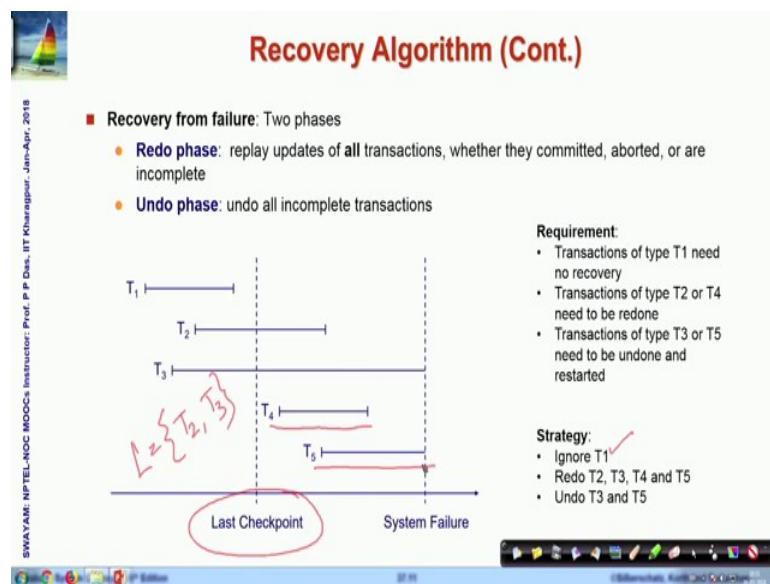
At the bottom left, there is a video player showing a man speaking, with the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr- 2018'. At the bottom right, it says 'Database System Concepts - 8<sup>th</sup> Edition', '37.10', and '©Silberschatz, Korth and Sudarshan'.

When you have to do a transaction rollback during normal operation; so, for that transaction  $T_i$  to be rolled back, what we will need to do is a rollback. So, undo has to happen. So, scan will scan the log backwards from the end and for each log record

update log record, we will restore the original value for which was written over and we will write a compensation log record as before and going backwards in this way when we come across the start log record, then we will stop that scan and write a abort log record in that place.

So, it is exactly same to what we did.

(Refer Slide Time: 04:21)



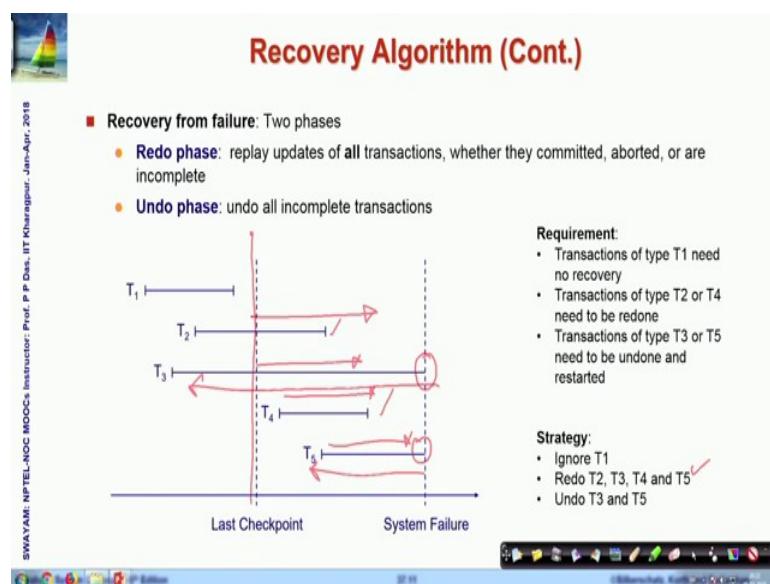
So, now let us look into the actual Recovery Algorithm. So, the transaction rollback has no difference. So, in the Recovery Algorithm, what we do we have a recovery phase and where we replay updates of all transactions. So, we make sure that all transactions whatever they did they those are done again. So, after the failure we recover from the failure. So, we up do all that again whether they are committed, whether they are aborted, whether they are incomplete in every case and then we keep track of what are the transactions which did not complete and for them we do an undo phase. So, here I am showing another example here.

So, this is the last checkpoint where eh all updates I mean freezing the updates, everything was output to the disk the log as well as the data item updates were put to the disk and the set of transactions that are live during that time well execution in that time

were recorded. So, if we look at that set L in this case, then it will be  $T_2$ ,  $T_3$  these two transactions.

So, we can we have already seen that our strategy would be that we will ignore  $T_1$  because it had completed before the last checkpoint.  $T_2$  and  $T_3$  were ongoing and then  $T_4$  has started after checkpoint and committed before that,  $T_5$  stared after checkpoint, but was also active was also in execution when system failed. So, our strategy would be, that we will assume as if this, this whole thing as is redone.

(Refer Slide Time: 06:06)



So,  $T_2$ ,  $T_3$ ,  $T_4$ ,  $T_5$  all these log records exist. So, we will follow through them and redo all of them. If we redo all of them then naturally we come across  $T_3$  and  $T_5$  which cannot proceed further because the system had could not proceed further because. So, we do not know in terms of the log what would have happened to them because the system had failed.

So, after having done this then, we do an undo phase where we undo this, but naturally the effect of these will remain. Now you can question that this could have been done in a more smart way, do we really need to redo everything and then undo some parts of that, that is a override in terms of that which is true, but this just makes the whole algorithm simple and over it actually is not very hard.

(Refer Slide Time: 06:55)

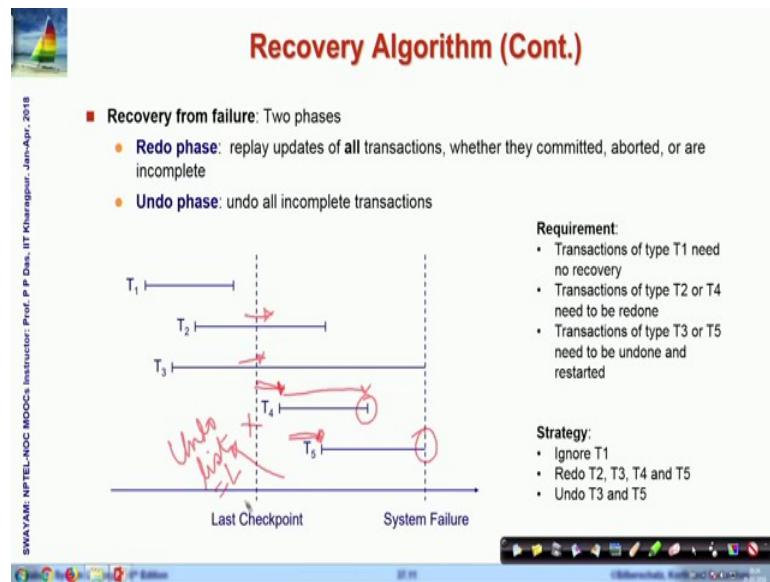
The slide has a header 'Recovery Algorithm (Cont.)' with a sailboat icon. The main content is a bulleted list under 'Redo phase':

- 1. Find last <checkpoint L> record, and set undo-list to L
- 2. Scan forward from above <checkpoint L> record
  - 1. Whenever a record < $T_j X_j V_1 V_2$ > is found, redo it by writing  $V_2$  to  $X_j$
  - 2. Whenever a log record < $T_i$  start> is found, add  $T_i$  to undo-list
  - 3. Whenever a log record < $T_i$  commit> or < $T_i$  abort> is found, remove  $T_i$  from undo-list

Navigation icons and text at the bottom include: SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr., 2018; Database System Concepts - 8<sup>th</sup> Edition; 37.12; ©Silberschatz, Korth and Sudarshan.

So, we are doing the redo phase, even the redo phase you will find the check point and you will scan forward from the checkpoint record and as you scan forward from the checkpoint record; if you have an update, you will simply redo which means  $V_2$ , will again be written to  $X_j$  and when you find a start transaction, then you do not know. Just look at this point carefully; if you find a start transaction for example, when you are working on this, suppose you come across a start transaction here, you will come across the start transaction transactions start here.

(Refer Slide Time: 07:31)



So, whenever you get that, then you put this into the undo list. Initially your undo list is L because they were going on. So, you do not know that they could finish all that still need to be undone and when you come across a new start, you add that to the undo list and then the rest of it is simple. So, you keep on going this way, if you find that the commit has happened or abort has happened, you remove that from the undo list, but if you do not find that then that stays in the undo list. So, you know if you, if you proceed from in this direction in the redo phase, you know and that way when you have scanned the whole log, you know what are the transactions which still need to be undone. So, that is a simple strategy that is followed.

(Refer Slide Time: 08:26)

The slide features a small sailboat icon in the top left corner. The title 'Recovery Algorithm (Cont.)' is centered at the top in red. Below the title, a section titled '■ Redo phase:' is listed with three steps:

1. Find last <checkpoint L> record, and set undo-list to L
2. Scan forward from above <checkpoint L> record
  1. Whenever a record < $T_i, X_j, V_1, V_2$ > is found, redo it by writing  $V_2$  to  $X_j$
  2. Whenever a log record < $T_i, \text{start}$ > is found, add  $T_i$  to undo-list
  3. Whenever a log record < $T_i, \text{commit}$ > or < $T_i, \text{abort}$ > is found, remove  $T_i$  from undo-list

On the right side of the slide, there is a decorative footer bar with various icons. The left edge of the slide shows vertical text: 'SWAYAM: NPTEL-NOC MOOCs', 'Instructor: Prof. P. P. Desai, IIT Kharagpur', and 'Date: Jan-Apr- 2018'.

So, ma whenever you have a log record start, then you put it to the undo list and whenever you get a log record which is committed abort which says that before the system failure the transaction actually had either committed that it finished everything or you had to roll back, then you remove that from the undo list. So, what will be left out, at the end will be the undo list of transactions that need to be undone subsequently.

(Refer Slide Time: 08:53)

The slide features a small sailboat icon in the top left corner. The title 'Recovery Algorithm (Cont.)' is centered at the top in red. Below the title, a section titled '■ Undo phase:' is listed with three steps:

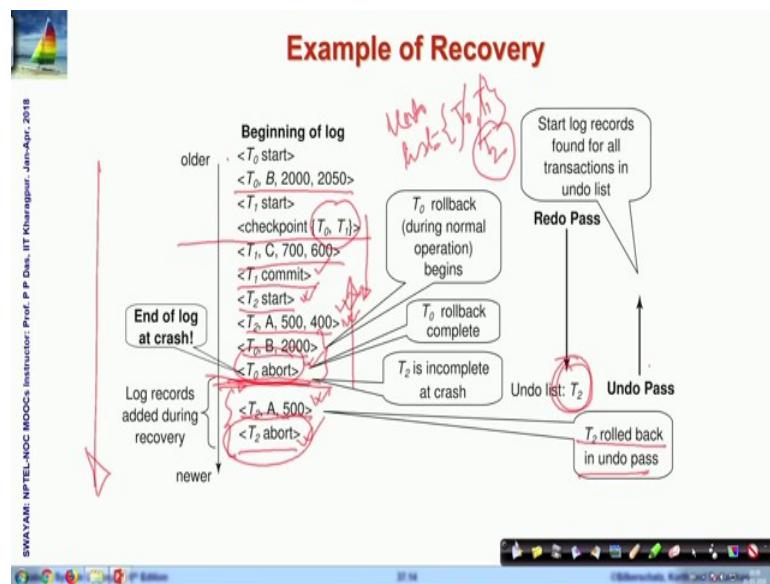
1. Scan log backwards from end
  1. Whenever a log record < $T_i, X_j, V_1, V_2$ > is found where  $T_i$  is in undo-list perform same actions as for transaction rollback:
    1. Perform undo by writing  $V_1$  to  $X_j$
    2. Write a log record < $T_i, X_j, V_1$ >
  2. Whenever a log record < $T_i, \text{start}$ > is found where  $T_i$  is in undo-list,
    1. Write a log record < $T_i, \text{abort}$ >
    2. Remove  $T_i$  from undo-list
  3. Stop when undo-list is empty
    - That is, < $T_i, \text{start}$ > has been found for every transaction in undo-list
    - After undo phase completes, normal transaction processing can commence

On the right side of the slide, there is a video frame showing a person speaking, a decorative footer bar with various icons, and the text 'Database System Concepts - 8<sup>th</sup> Edition' at the bottom left. The left edge of the slide shows vertical text: 'SWAYAM: NPTEL-NOC MOOCs', 'Instructor: Prof. P. P. Desai, IIT Kharagpur', and 'Date: Jan-Apr- 2018'.

In the undo phase, in the undo phase, you go backwards because it is undo. So, what you will do is a very similar. So, if you have an update record, then you undo in the main transaction which is in undo list then you do exactly in terms of transaction rollback that you write the old value and write a redo only log record. Now when you find going backwards, when you find  $\langle T_i, \text{start} \rangle$ ; so you know that this is a starting point of the transaction and the transaction is in undo list. So, it came across because it could not be it was on the undo list. So, which means that it could not be completed and therefore, you have found the start. So, this is where your undo operation is over. So, you write a abort log record and once you have written that, then you are done with the transaction. So, you remove that from the undo list and in this way, you will continue till your undo list is becomes empty.

Once it becomes empty, so, then you have found that  $\langle T_i, \text{start} \rangle$  for all transactions in the undo list and there is nothing more to do. So, after undo phase completes normal transaction processing can comment. So, your failure recovery from the failure is already taken care of.

(Refer Slide Time: 10:14)



So, here are certain examples which you could check out, here are a start. So, this is the how that this is the order in which the transactions are going and this is the crash point, these is where it failed and mind you. So, this is where and this is where our checkpoint

is. So, at checkpoint you can see that  $T_0$  and  $T_1$  are; what are your candidates? So, when you start in the redo phase, you start from this point because before that everything has been done. You naturally, you come across this. So, you redo this which means you again actually change C from its old value 700 to 600 and then  $T_1$  commits. As  $T_1$  commits, you know that this transaction is done with.

So, you remove that. So, your undo list at the beginning is  $T_0, T_1$ , but going in the forward direction when you come across **<T<sub>1</sub>, commit>**, you naturally from your undo list, then you come across **<T<sub>2</sub> , start>**. So, you know that another transaction is starting now. So, it may be you do not know whether it could complete or you could not. So, you add that to the undo list then give effect to this update, then if for  $T_0$  we have you have a rollback record that is because  $T_0$  actually you can see that  $T_0$  has aborted. So, the change that  $T_0$  had done earlier this had to be rolled back, this rollback is a normal transaction rollback, this is not because of the failure. So, this mm rollback had happened and this is where the rollback is complete.

So,  $T_2$  e,  $T_0$  is also completed. So,  $T_0$  after this is taken care of, then in the redo phase  $T_0$  is also complete and this is where you reach the crash point. So, your undo list is left with only  $T_2$ . So, now, when you have done this, so when you have taken done the redo here that  $T_2$  which is ongoing is there, then you write this log record. So, these log records are written during recovery not during the original transaction and  $T_2$  had to abort because of the system failure.

So,  $T_0$  support was due to the transaction rollback, but **<T<sub>2</sub> , abort>** is because of the system failure. So,  $T_2$  is rolled back in the redo phase. So, once this has been done, then you do the undo phase starting with  $T_2$  and then you go backwards as you go backwards here. So, you will undo this, this is what you write you come across  $T_2$  and naturally you have rollback. So, you write **<T<sub>2</sub>, abort>**. This is how the actual rollback can happen and you can see that now the with this redo undo phase you can always bring back the database to a consistent state and these transactions are executing concurrently and therefore, your log record is a intermix of the log record of different transactions. Now the last that we would like to talk about is Recovery with Early Lock Release.

(Refer Slide Time: 13:55)

The slide has a header 'Recovery with Early Lock Release' with a sailboat icon. On the left, there's vertical text: 'SWAYAM: NPTEL-NOC MOOCs', 'Instructor: Prof. P. P. Deshpande', 'IIT Kharagpur', and 'Jan-Apr., 2018'. The main content is a bulleted list:

- Support for high-concurrency locking techniques, such as those used for B\*-tree concurrency control, which release locks early
  - Supports "logical undo"
- Recovery based on "repeating history", whereby recovery executes exactly the same actions as normal processing

At the bottom left is a video player interface showing a man speaking, with the text 'Database System Concepts - 8th Edition'. The bottom right shows slide navigation icons and the text '©Silberschatz, Korth and Sudarshan'.

What this means is well, so far we have talked about recovery which is only in terms of data update, single data updates. So, I mean if I want to recover I can just you know write back the old data, but this is not true in case of many other situations for example, if you are inserting a record in a B-tree, then it is not enough only to undo that because you cannot undo and get back the same.

As you can understand that if you make inserts or deletes in the B-tree, if you are made an insert then the structure of the B-tree itself has changed and after that several other inserts, deletes may have happened. So, if you now want to just go back and undo this particular insert in terms of values, it is not possible to do that. So, when you want to do that, so you cannot do really kind of repeating the history kind of strategy.

(Refer Slide Time: 14:53)

The slide has a header 'Logical Undo Logging' with a sailboat icon. The main content is a bulleted list under a red square icon:

- Operations like B+-tree insertions and deletions release locks early
  - They cannot be undone by restoring old values (**physical undo**), since once a lock is released, other transactions may have updated the B+-tree
  - Instead, insertions (resp. deletions) are undone by executing a deletion (resp. insertion) operation (known as **logical undo**)
- For such operations, undo log records should contain the undo operation to be executed
  - Such logging is called **logical undo logging**, in contrast to **physical undo logging**
    - ▶ Operations are called **logical operations**
  - Other examples:
    - ▶ delete of tuple, to undo insert of tuple
      - allows early lock release on space allocation information
    - ▶ subtract amount deposited, to undo deposit
      - allows early lock release on bank balance

At the bottom left is a video thumbnail of a professor, at the bottom center is the page number '37.17', and at the bottom right is the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, what you have to do is do some kind of an undo which is logical. So, so far the undo was physical that, you wrote this, you change this value by this value. So, your undo is a physical. So, you restore the original value and your undo is done here. It is logical that is for the operation that you have performed, you try to find out a matching operation which creates the similar effect as of undo. So, if you have inserted, then your undo is a corresponding delete of that record. If you have incremented by 10 then you can say that your corresponding undo is a decrement by 10. So, that is what is known as the logical undo and it is logical undo is a very good option in case of delete of, insert delete of people.

So, if you have deleted a people undo to insert, if you have subtracted then undo to undo deposit to go forward and so on.

(Refer Slide Time: 15:55)

The slide has a header 'Physical Redo' with a sailboat icon. The main content lists advantages of physical redo:

- Redo information is logged **physically** (that is, new value for each write) even for operations with logical undo
  - Logical redo is very complicated since database state on disk may not be "operation consistent" when recovery starts
  - Physical redo logging does not conflict with early lock release

Navigation icons and slide details are at the bottom.

So, a redo information is logged physically, so new values for each right even for operations which are logically, which has logical undo. So, you do not do a logical redo I mean, I will not go into the details of why this is not done, but it simply makes things very complicated. So, physical redo is always physical and you can show that physical redo does not prohibit this kind of operations that we are trying to do, but the logical redo is not used. We will only use logical undo operation.

(Refer Slide Time: 16:40)

The slide has a header 'Operation Logging' with a sailboat icon. It describes the process of operation logging:

- Operation logging is done as follows:
  - When operation starts, log  $\langle T_p, O_p, \text{operation-begin} \rangle$ . Here  $O_p$  is a unique identifier of the operation instance
  - While operation is executing, normal log records with physical redo and physical undo information are logged
  - When operation completes,  $\langle T_p, O_p, \text{operation-end}, U \rangle$  is logged, where  $U$  contains information needed to perform a logical undo operation

Example: insert of (key, record-id) pair (K5, RID7) into index I9

Log sequence:

```
<T1, O1, operation-begin>
...
<T1, X, 10, K5>
<T1, Y, 45, RID7>
}
<T1, O1, operation-end, (delete I9, K5, RID7)>
```

Navigation icons and slide details are at the bottom.

So, how do you log for such a logical undo operation, what you do is instead of now. So, now, it is an operation it may not be a single value update. So, it is not captured in terms of one you know log record, but it could be a number of log records which have actually done three, four different changes to make that operation happen and you want to actually define an undo for that operation. So, when you start this. So, you start with a log which says that what is the transaction and what is the operation. So, you put an identifier to the operation and then you write operation begin.

So, you know this is where operation has started, then all the things that are happening for this operation while the operation is executing then you write normal log records with physical redo physical information. All these logs are written and when this operation ends mind you, this is a particular operation you are talking of. So, not the whole transaction whole transaction will continue when that particular operation ends, then you write an operation in record and along with that you write, what is a logical, what is a logical undo information you put that in.

So, let us have a look at the example. So, suppose your operation is insert of a key record pair, so, let us say this is the key record pair and into index I9. So, this operation starts here and then there are several steps to be done; for example, you will have to say if X is on the key value which had 10 and is becoming K5, you will have a physical update undo record of this. If Y is the record id which is RID 7, then it Y changes from 45 to. So, these are all physical redo steps in insert. So, these are the different instructions in terms of this broad operation and when you are done with all that then your operation ends and you write this undo information. So, insert of, so you had insert of this record with index 9.

So, now you do write your what will be the undo, to delete that from index I9, to delete this key record ID pair. So, this is a whole locking that we do. So, you can make use of this undo operation in terms of your recovery process.

(Refer Slide Time: 19:22)

The slide has a title 'Operation Logging (Cont.)' in red at the top right. On the left, there is a small image of a sailboat on water. The main content is a bulleted list under three main points:

- If crash/rollback occurs before operation completes:
  - the **operation-end** log record is not found, and
  - the physical undo information is used to undo operation
- If crash/rollback occurs after the operation completes:
  - the **operation-end** log record is found, and in this case
  - logical undo is performed using  $U$ ; the physical undo information for the operation is ignored
- Redo of operation (after crash) still uses physical redo information

At the bottom left, it says 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018'. In the center, it says 'Database System Concepts - 8<sup>th</sup> Edition'. At the bottom right, it says '37.20 ©Silberschatz, Korth and Sudarshan'.

So, if the crash or rollback occurs before the operation completes, then operation and log record is not found you will not find it. So, you do not know what is the undo operation. So, in that case the physical undo information is will be used to undo, but if we have a crash on rollback that happens after an operation completes, then the operation end log will be available and in this case we will use the undo operation that is given in the operation end log record and do a logical undo. And we will ignore all the physical undo information that the operation that that we will find in the log records. Redo of course will still use the physical redo information which is there.

(Refer Slide Time: 20:10)

**Transaction Rollback with Logical Undo**

Rollback of transaction  $T_i$ , scan the log backwards

1. If a log record  $\langle T_p, X, V_1, V_2 \rangle$  is found, perform the undo and log  $\langle T_p, X, V_1 \rangle$
2. If a  $\langle T_p, O_p, \text{operation-end}, U \rangle$  record is found
  - Rollback the operation logically using the undo information  $U$ 
    - Updates performed during roll back are logged just like during normal operation execution
    - At the end of the operation rollback, instead of logging an **operation-end** record, generate a record  $\langle T_p, O_p, \text{operation-abort} \rangle$
  - Skip all preceding log records for  $T_i$  until the record  $\langle T_p, O_p, \text{operation-begin} \rangle$  is found
3. If a redo-only record is found ignore it
4. If a  $\langle T_p, O_p, \text{operation-abort} \rangle$  record is found:
  - skip all preceding log records for  $T_i$  until the record  $\langle T_p, O_p, \text{operation-begin} \rangle$  is found
5. Stop the scan when the record  $\langle T_p, \text{start} \rangle$  is found
6. Add a  $\langle T_p, \text{abort} \rangle$  record to the log

**Note:**

- Cases 3 and 4 above can occur only if the database crashes while a transaction is being rolled back
- Skipping of log records as in case 4 is important to prevent multiple rollback of the same operation

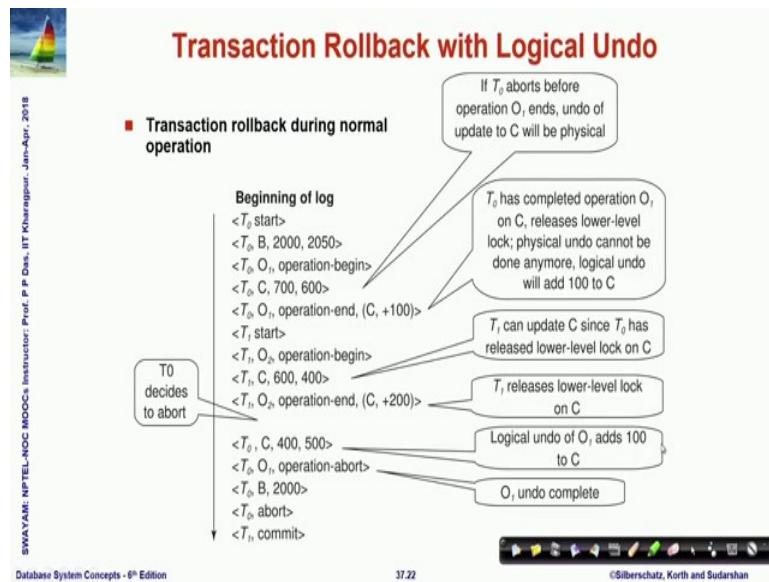
SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018

Database System Concepts - 8<sup>th</sup> Edition      37.21      ©Silberschatz, Korth and Sudarshan

So, if we look into the actual if we if we look into the transaction rollback with logical undo. So, if I have an update record which is naturally physical, and then we can perform the undo which is as we did last time the creating a redo only record. If I find an operation end record, then to rollback we will pick up the logical undo information from you and we will perform that operation. At the end of that we will certainly write the operation abort record to show to mark that this operation has been aborted.

So, if we have a redo only record, then we will ignore it and if we find an operation abort, then we will skip all the records that were found till the beginning. Naturally, you can you can understand that 3 and 4 will not happen in a normal course of transaction, it will happen only when the failures have happened during recovery and at the end we will add  $\langle T_i, \text{abort} \rangle$  record to the log. So, the critical thing to remember in this that whenever we are doing operation hmm unlogging, we are doing undoing based on the operation logging then since once we get the operation ends since we know what the undo information is, we have to make sure that through the undo process we actually ignore the physical undo records that exist in the log and just go with the operation case. So, this these are the notes I just mentioned it ok.

(Refer Slide Time: 22:15)



So, this is an example which you will have to spend some time and understand with care. So, you can see that a transaction  $T_0$  has started, this is where it has done a physical update, is a physical undo record and then it does an operation. Of course, it is a simple operation which changes the value of  $C$  from 700 to 600. So, naturally, so it has decremented by 100. So, your undo operation here is incrementing by 100.

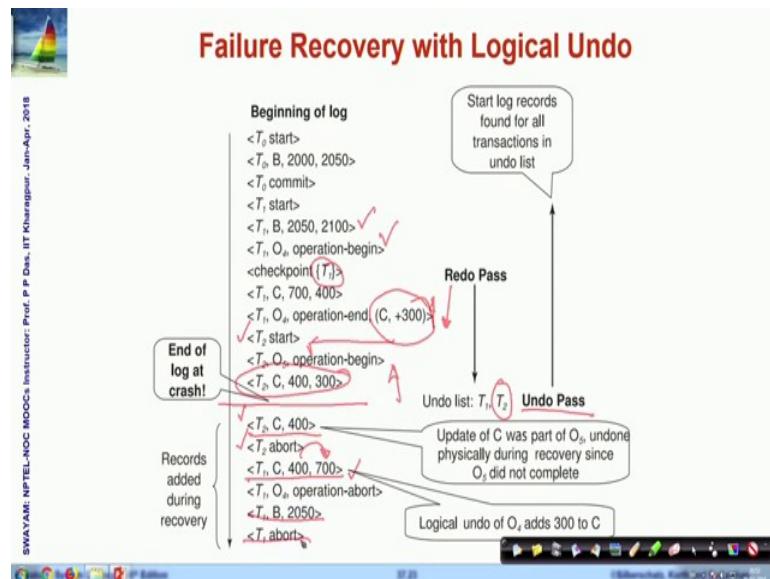
So, if  $T_0$  aborts here, if  $T_0$  aborts somewhere here you know before your operation end has happened then naturally the undo will have to be based on this physical undo structure. So, you will have to replace 600 by 700, but if it happens after this, then this is the case if it is completed the operation and then the failure happens, then you will do the logical undo that is whatever the value of  $C$  is you will just logically add 100 to that. But that means, that when you go backwards from here to find the begin, you will actually have to ignore this physical undo record because you have already given effect to that in terms of the operation undo that you are doing.

So, this is the basic difference. Here are different subsequent examples on that and you can see that well here after the operation end has happened, then possibly it has released the  $T_0$  has released a lock on  $C_1$ . So,  $T_1$  has again taken the log.  $T_1$  has again done the updates. So, then it releases that and  $T_0$  at this point might decide to abort; if  $T_0$  aborts, then this logical undo of  $O_1$  this operation will add, it had to add 100. So, it adds

now this C had become 400, now it is adding 100 back. So, C becomes 500 and then the operation is finished. So, you write operation abort and  $O_1$  undo of  $O_1$  gets completed, but you still have after going backwards in this, you still have this record which was directly updated.

So, these are the undo transactions, undo lock record for that where B is being restored from 2050 to 2000 and you record the transaction abort for  $T_0$ ,  $T_1$  eventually has committed here. So, this is how the transaction rollback will happen when logical undo is also used and this is a very powerful way to take care of that.

(Refer Slide Time: 25:13)



This is similarly another illustration for doing the failure recovery for with logical undo. So, here is the undo is a re redo phase that you are seeing here, this is where the end of log at the crash these are redo phase because these are check point where  $T_1$  was there. So, at the end of redo  $T_1$  if you if you. So, you are starting to redo from here. So, you have done operation end.  $T_1$  has not finished  $T_2$  has started. So, you have added  $T_2$  to the undo list and when the crash has happened both of them are on the undo list. So, they have to go through that undo pass. So, we undo  $<T_2, C, 400>$ . So, this is what this is this is how you will go.

So, this is the first thing you undo and then naturally you have come to the beginning of **<T2 , start>**. So, you abort and you are going back again and you are trying to do this. Why are you doing this because when you go back to undo from this point you come across this operation end which tells you that the undo operation has to happen by incrementing C by 300. So, C which had become 400 is now incremented by 300. You come to the check point which is the end here in terms of the operation begin and naturally you declare operation abort and going back further this is what you had when transaction T<sub>1</sub> had started.

So, you undo that. That is again a physical undo and finally, T<sub>1</sub> aborts. So, this is how in both cases of transaction rollback as well as in terms of the failure the recovery can be done with the logical undo process.

(Refer Slide Time: 27:10)

**Transaction Rollback: Another Example**

- Example with a complete and an incomplete operation

```

<T1, start>
<T1, O1, operation-begin>
...
<T1, X, 10, K5>
<T1, Y, 45, RID7>
<T1, O1, operation-end, (delete I9, K5, RID7)>
<T1, O2, operation-begin>
<T1, Z, 45, 70>
    ← T1 Rollback begins here
<T1, Z, 45> ← redo-only log record during physical undo (of incomplete O2)
<T1, Y, ... ,> ← Normal redo records for logical undo of O1
...
<T1, O1, operation-abort> ← What if crash occurred immediately after this?
<T1, abort>

```

SWAYAM: NPTEL-NOCO MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jam-Apr- 2018

Database System Concepts - 8<sup>th</sup> Edition      37.24      ©Silberschatz, Korth and Sudarshan

Here I have given another example. I will not go through it step by step. So, at a arts that you go through that following the same logic and convince yourself that you understand that how this transaction rollbacks with physical undo as well as logical undo is taking place.

(Refer Slide Time: 27:27)

The slide features a sailboat icon in the top left corner. The title 'Recovery Algorithm with Logical Undo' is centered at the top in a red header. Below the title, a vertical column of text provides a brief overview and the steps of the recovery algorithm. A small video frame of a professor is visible on the right side.

Basically same as earlier algorithm, except for changes described earlier for transaction rollback

1. (Redo phase): Scan log forward from last <checkpoint L> record till end of log
  1. Repeat history by physically redoing all updates of all transactions,
  2. Create an undo-list during the scan as follows
    - undo-list is set to L initially
    - Whenever < $T_i$  start> is found  $T_i$  is added to undo-list
    - Whenever < $T_i$  commit> or < $T_i$  abort> is found,  $T_i$  is deleted from undo-list

This brings database to state as of crash, with committed as well as uncommitted transactions having been redone

Now undo-list contains transactions that are incomplete, that is, have neither committed nor been fully rolled back

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur, Jan-Apr., 2018  
Database System Concepts - 8<sup>th</sup> Edition  
37.25 ©Silberschatz, Korth and Sudarshan

So, ma with this Recovery Algorithm with logical undo will look very similar to what we have already done with the physical undo redo and though that is what we have stated here, there is no nothing significantly new. So, I expect that you should be able to go through these steps.

(Refer Slide Time: 27:52)

The slide features a sailboat icon in the top left corner. The title 'Recovery with Logical Undo (Cont.)' is centered at the top in a red header. Below the title, a vertical column of text continues the explanation of the recovery process. A small video frame of a professor is visible on the right side.

Recovery from system crash (cont.)

2. (Undo phase): Scan log backwards, performing undo on log records of transactions found in undo-list
  - Log records of transactions being rolled back are processed as described earlier, as they are found
    - Single shared scan for all transactions being undone
  - When < $T_i$  start> is found for a transaction  $T_i$  in undo-list, write a < $T_i$  abort> log record.
  - Stop scan when < $T_i$  start> records have been found for all  $T_i$  in undo-list

■ This undoes the effects of incomplete transactions (those with neither commit nor abort log records). Recovery is now complete

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur, Jan-Apr., 2018  
Database System Concepts - 8<sup>th</sup> Edition  
37.26 ©Silberschatz, Korth and Sudarshan

And those will be clear to you again we have a two phase recovery of redo phase and the undo phase and we make use of the operation undo, logical undo as and when it is

possible and when that is and when we do that, we ignore all physical undo records and when it is not possible, then we lose the physical undo records and that is how the recovery can be achieved.

(Refer Slide Time: 28:19)

The slide has a title 'Module Summary' in red at the top right. Below it is a list of two bullet points: 'Studies the recovery algorithms for concurrent transactions' and 'Recovery based on operation logging supplements log-based recovery'. On the left edge of the slide, there is vertical text that reads 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018'. At the bottom of the slide, there is footer text 'Database System Concepts - 8<sup>th</sup> Edition' and '37.27' next to a progress bar. To the right of the progress bar is the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, in this module we have exposed ourselves with the Recovery Algorithms now for concurrent transactions as well and we have shown that how recovery can be done using operational logging, operations logging and making sure that really the database may not need to hold on to a lock for a long time on the data item and delay other transactions, but if it can define the undo operation on the data item, then it can release that log early and use that logging mechanism operation logging mechanism to recover the data.

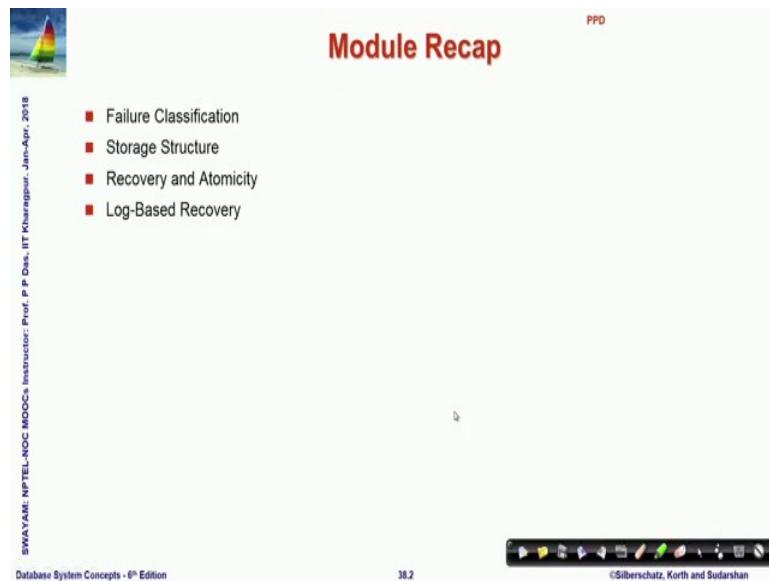
**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 38**  
**Query Processing and Optimization/1: Processing**

---

Welcome to module 38 of database management systems, in this module and the next we will talk about query processing and optimization of that in the current module we will talk about query processing.

(Refer Slide Time: 00:29)

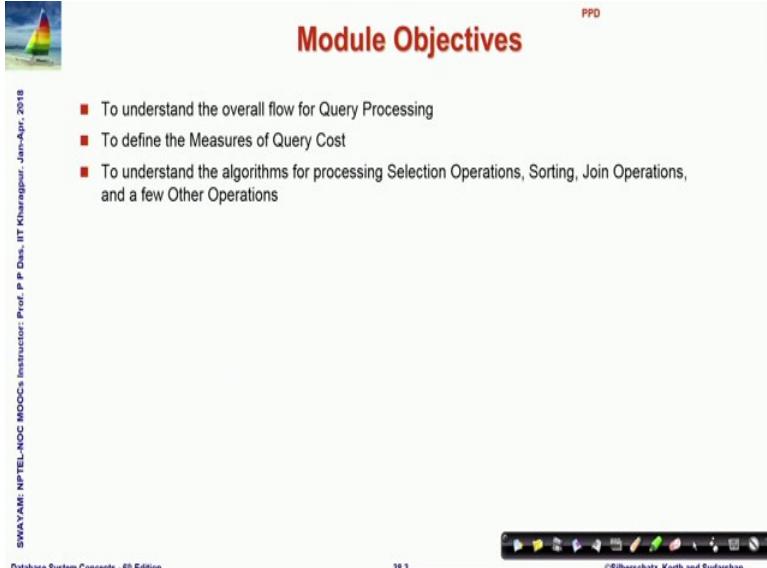


The slide is titled "Module Recap" in red at the top right. It features a small sailboat icon on the left. A vertical watermark on the left side reads "SWAYAM: NPTEL-NOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018". The main content is a bulleted list of four items: "Failure Classification", "Storage Structure", "Recovery and Atomicity", and "Log-Based Recovery". At the bottom, there is a navigation bar with icons for back, forward, search, and other presentation controls. The page number "38.2" is at the bottom center, and the copyright notice "©Silberschatz, Korth and Sudarshan" is at the bottom right.

- Failure Classification
- Storage Structure
- Recovery and Atomicity
- Log-Based Recovery

So, in the last module we had done talked about database recovery.

(Refer Slide Time: 00:36)



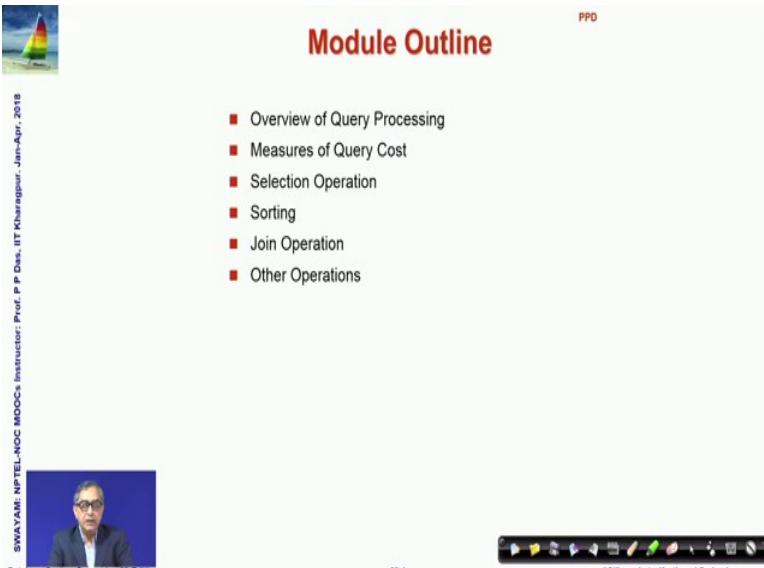
The slide title is "Module Objectives". It features a small sailboat icon in the top left corner and the acronym "PPD" in the top right corner. The main content is a bulleted list of objectives:

- To understand the overall flow for Query Processing
- To define the Measures of Query Cost
- To understand the algorithms for processing Selection Operations, Sorting, Join Operations, and a few Other Operations

On the left side, there is vertical text: "SWAYAM: NPTEL-MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur. Jan-Apr., 2018". At the bottom, it says "Database System Concepts - 6<sup>th</sup> Edition" and "38.3". The footer includes a navigation bar and the copyright notice "©Silberschatz, Korth and Sudarshan".

And now we will try to understand the overall flow of processing queries. So, if I fire a query like select from where how will that actually access the database files the B trees and indexes and so, on and compute the result is what we would like to discuss. And a query can be processed in multiple ways giving rise to different kinds of costs in terms of the time required for processing that query. So, we will define certain measures of query cost and then we will take a quick look into a set of sample algorithms for processing simple selection operation, sorting, joint operation and few of the other operations like aggregation.

(Refer Slide Time: 01:26)



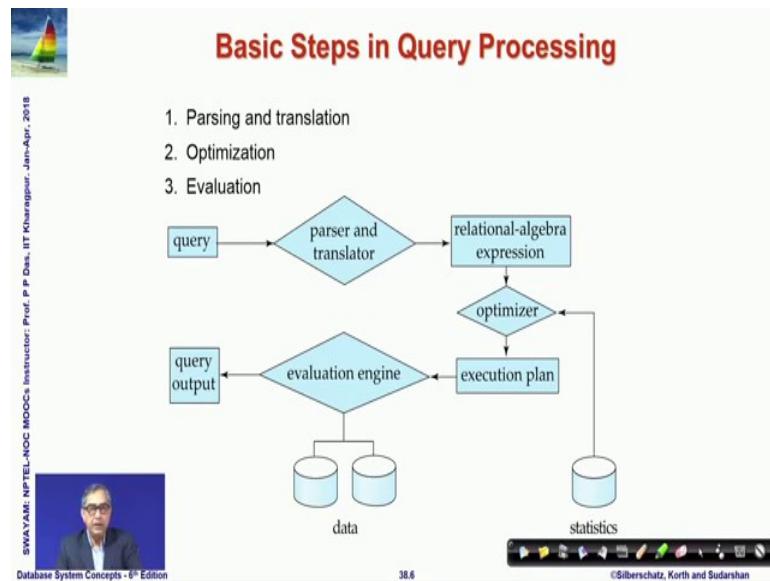
The slide title is "Module Outline". It features a small sailboat icon in the top left corner and the acronym "PPD" in the top right corner. The main content is a bulleted list of topics:

- Overview of Query Processing
- Measures of Query Cost
- Selection Operation
- Sorting
- Join Operation
- Other Operations

On the left side, there is vertical text: "SWAYAM: NPTEL-MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur. Jan-Apr., 2018". In the bottom left corner, there is a video thumbnail showing a person's face. At the bottom, it says "Database System Concepts - 6<sup>th</sup> Edition" and "38.4". The footer includes a navigation bar and the copyright notice "©Silberschatz, Korth and Sudarshan".

So, first let us so, these are the topics to talk about and first we will take a look at the overall query processing algorithm.

(Refer Slide Time: 01:35)



So, this is the flow so, this is a way the a query will get processed. So, here what you have is this is where the input query comes in naturally it is written in terms of a in terms of SQL which is kind of a programming language.

So, you need a parser and translator. So, the it is parse translated and it is translated into a relational algebra expression as we have shown at the very beginning discussed at the very beginning that SQL is basically derived or developed based on relational algebra. So, corresponding to every SQL query there is a one or more relational algebra expression. So, you express in terms of that, then you optimize you try to see how the relational algebra expression can be made efficient and to optimize this we might use some information about the statistics.

Statistics in terms of we might use the information past history information of say what is the what are the attributes on which more often the where conditions are port we might want to use statistics like how many tuples actually exist in the relation right now and so, on. And based on that we will decide on an execution plan, execution plan is how we actually want to what are the actions that we actually want to do in terms of accessing the different indexes and the different B + tree nodes to evaluate the query and that is the job

of the evaluation engine is you can see that it will access the data for that and finally, out of that the query output will be generated.

So, in this module and the next we will take a quick look into so, it is glimpses of these steps one by one and try to understand how query processing and optimization can happen.

(Refer Slide Time: 03:40)

**Basic Steps in Query Processing (Cont.)**

- Parsing and translation
  - translate the query into its internal form
    - ▶ This is then translated into relational algebra
  - Parser checks syntax, verifies relations
- Evaluation
  - The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Dass, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition

So, beyond a parsing and translation there is evaluation as we have talked of.

(Refer Slide Time: 03:48)

**Basic Steps in Query Processing : Optimization**

- A relational algebra expression may have many equivalent expressions
  - E.g.,  $\sigma_{\text{salary} < 75000}(\Pi_{\text{salary}}(\text{instructor}))$  is equivalent to  $\Pi_{\text{salary}}(\sigma_{\text{salary} < 75000}(\text{instructor}))$
- Each relational algebra operation can be evaluated using one of several different algorithms
  - Correspondingly, a relational-algebra expression can be evaluated in many ways
- Annotated expression specifying detailed evaluation strategy is called an **evaluation-plan**.
  - E.g., can use an index on *salary* to find instructors with  $\text{salary} < 75000$ ,
  - or can perform complete relation scan and discard instructors with  $\text{salary} \geq 75000$

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Dass, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition

Now, if we take in terms of say a query a where  $\sigma$  from where clause has been translated. So, for example, if this is we can we can simply write out say if we have select and what are we selecting here? We are selecting salary and where are we selecting it from? The from is instructor and under what conditions are we doing that? Where, where is **salary < 75000**.

So, if you had a query like this then you know then it will be it will get translated to some kind of a relational algebra expression like this where you do a selection on the salary and then you do you do a projection on the salary and you do a selection based on that condition. Now, it is also clear to say that this particular relational algebra expression can be equivalently written by swapping that these two conditions, that is we can first do a selection and then do the projection they are actually equivalent and that could be multiple equivalents.

So, we know that given a query there could be multiple relational algebra expressions and then the relational algebra expression the operations can be evaluated also in one of the by using one or more different algorithms.

So, basically what you have you have different options given a SQL query, you have different possible relational algebra expressions that are equivalent given every relational algebra expressions you have different strategies different algorithms to actually execute and evaluate that. And we would like to based on these we would like to annotate the expression we would like to mark out as to whether from the above to say whether we first project on salary and then do the selection or we first do the selection on salary less than 75000 and then project.

And with that annotation we will build up a total evaluation strategy which is known as the evaluation plan and for example, here we can have different strategies like we can use an index on salary and if you use that that will be it will be pretty efficient to find tuples which satisfy salary less than 75000 or we can scan the whole relation and discard all those instructors for which salary is greater than equal to 75000. So, there could be different ways in which we can do this evaluation and that is what optimally has to be decided in every case.

(Refer Slide Time: 06:50)

The slide has a header 'Basic Steps: Optimization (Cont.)' with a sailboat icon. The content is organized into sections:

- **Query Optimization:** Amongst all equivalent evaluation plans choose the one with lowest cost
  - Cost is estimated using statistical information from the database catalog
    - ▶ e.g. number of tuples in each relation, size of tuples, etc.
- In this module we study
  - How to measure query costs
  - Algorithms for evaluating relational algebra operations
  - How to combine algorithms for individual operations in order to evaluate a complete expression
- In the next module
  - We study how to optimize queries, that is, how to find an evaluation plan with lowest estimated cost

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition 38.9 ©Silberschatz, Korth and Sudarshan

So, in terms of query optimization out of all these equivalent evaluation plans we try to choose the one that gives some minimum cost the lowest cost evaluation.

So, the cost will have to be estimated based on certain statistical information from the database catalog for example, number of tuples in the relation, the size of the tuples, the attributes on which frequently condition are tested and so, on. So, this is what we would in totality try to understand out of that in this module we will first define what is the measures of cost and look at the algorithms for evaluating some of the relational algebra operations and then you can combine them to do bigger operations and in the next module we will talk about optimization.

So, first let us see how we define the cost because if we want to say that we can do it you say in 2-3 different ways, evaluate the same query in 2-3 different ways then we must assess as to what is the best way of doing it the best way is whatever gives us the least cost.

(Refer Slide Time: 08:02)

The slide has a header 'Measures of Query Cost' with a sailboat icon. The content lists factors contributing to query cost:

- Cost is generally measured as total elapsed time for answering query
  - Many factors contribute to time cost
    - disk accesses, CPU, or even network communication
- Typically disk access is the predominant cost, and is also relatively easy to estimate
- Measured by taking into account
  - Number of seeks \* average-seek-cost
  - Number of blocks read \* average-block-read-cost
  - Number of blocks written \* average-block-write-cost
    - Cost to write a block is greater than cost to read a block
      - data is read back after being written to ensure that the write was successful

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
38.11  
©Silberschatz, Korth and Sudarshan

So, measures of cost will be in absolute terms it is in terms of the elapsed time how much time does it take and there could be many factors which ma dictate that because in terms of evaluating this we will have to access the disk. So, access time of the disk will be involved, the computing time in CPU may be involved even some network communication may get involved.

So, out of these if we assume that there is no network communication cost just for simplicity that is everything is connected to a very you know high speed network then between the disk cost and the accesses and the CPU processing cost the disk access is a predominant cost. Because and it is relatively easy to estimate that because as we have looked at the storage structure we know that it is a typically a magnetic disk which where the head has to move to the correct cylinder to find the block where the records can be located. So, there is this process is called the seek.

So, we will need to find out how many estimate how many seek operations we need and multiply that by the average cost of seeking a block. Similarly, while we are reading that we need to estimate how many blocks to read and average cost to read a block, number of blocks to write average cost to write the block, cost to write the block is usually greater than the cost to read actually often when we write a write some data after writing we also usually read it back to make sure that the write was successful.

(Refer Slide Time: 09:53)

The slide has a title 'Measures of Query Cost (Cont.)' at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list of points:

- For simplicity we just use the **number of block transfers from disk and the number of seeks** as the cost measures
  - $t_T$  – time to transfer one block
  - $t_S$  – time for one seek
  - Cost for b block transfers plus S seeks  
$$b * t_T + S * t_S$$
- We ignore CPU costs for simplicity
  - Real systems do take CPU cost into account
- We do not include cost to writing output to disk in our cost formulae

At the bottom left, there is a small video thumbnail showing a person speaking. The bottom right corner shows the copyright information: ©Silberschatz, Korth and Sudarshan.

So, these are the typical cost factors that will dominate. So, if we say that if we just count the number of block transfers and the number of seeks and if the time to transfer one block is  $t_T$  and time for seek is one seek is  $t_S$ , then the cost of transferring  $b$  blocks doing and doing  $S$  seek will be given by this expression you can easily understand that.

So, every block transfer is  $t_T$  and the  $b$  blocks being transferred. So, this is the transfer cost and if there are  $S$  number of seeks and every seek time is  $t_S$ , then this is the seeking cost and adding them together we get the total cost of seek and transfer. For simplicity we will ignore the CPU cost and we will also for now not consider the cost of finally, writing the result back to the disk we will simply check as to what will it take to actually compute the result.

(Refer Slide Time: 10:54)

The slide has a title 'Measures of Query Cost (Cont.)' at the top right. On the left, there is a small image of a sailboat on water. The main content area contains two bulleted lists under red square bullet points:

- Several algorithms can reduce disk I/O by using extra buffer space
  - Amount of real memory available to buffer depends on other concurrent queries and OS processes, known only during execution
  - We often use worst case estimates, assuming only the minimum amount of memory needed for the operation is available
- Required data may be buffer resident already, avoiding disk I/O
  - But hard to take into account for cost estimation

At the bottom left, there is a small video thumbnail showing a person speaking. The bottom right corner includes the text 'SWAYAM-NPTEL-MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018' and 'Database System Concepts - 8<sup>th</sup> Edition'. The bottom center has the number '38.13' and the bottom right has the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, there are also it has to be noted that there are also several algorithms to reduce the disk I/O we can do that by using extra buffer space for example, one block has been read in and we are just using one record of that if in the next operation we have to use some record which is already existing in that block and if that block is maintained in that buffer then we do not need to go back to the disk and actually read the block once again.

So, the more of the buffer space that we can provide naturally the performance would become better, but certainly; that means, that the memory required for keeping the buffer would be higher and it is also often difficult to decide as to I mean estimate a query as to if I am looking for a particular block whether it is already there in the buffer.

So, that the I/O can be avoided or it needs to be actually read back from the disk, but these are some of the you know cost measures that are used in a more sophisticated cost function, but we will simply use the seek and read cost from the disk in terms of blocks to estimate our cost of the different operation. So, let us look at sample algorithms for different basic SQL operation. So, the first and most common SQL operation is selection as you all know.

(Refer Slide Time: 12:29)

The slide has a header 'Selection Operation' with a sailboat icon. It contains a list of bullet points under the heading 'File scan'. A handwritten note 'b\_r \* lrt + ts' is written next to the cost calculation. The footer includes the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018' and a small video thumbnail of the professor.

- File scan
- Algorithm A1 (**linear search**). Scan each file block and test all records to see whether they satisfy the selection condition
  - Cost estimate =  $b_r$  block transfers + 1 seek
    - ▶  $b_r$  denotes number of blocks containing records from relation  $r$
  - If selection is on a key attribute, can stop on finding record
    - ▶ cost =  $(b_r/2)$  block transfers + 1 seek
  - Linear search can be applied regardless of
    - ▶ selection condition or
    - ▶ ordering of records in the file, or
    - ▶ availability of indices
  - Note: binary search generally does not make sense since data is not stored consecutively
    - ▶ when there is an index available, binary search requires more seeks than index search

So, the selection for selection we discuss in multiple algorithms for different situations the first algorithm is here we are calling it as algorithm A1 is a linear search. So, what we do we just need to do some selection. So, we scan the say we are looking for a result of couple of records and or a single record then we just scan the file from one end to the other we look for all the records and check whether they satisfy the selection condition.

So, the cost for that would be  $b_r$  block transfers if there are if  $b_r$  is a number of blocks containing records from relation  $r$  then  $b_r$  blocks have to be read and one seek has to happen. Now, if the selection is on a key attribute and we can stop find on finding the record and on the average we can expect that we will be able to find it by having read half of the record. So,  $(b_r / 2) * \text{block transfers} + 1 \text{ seek}$  so, if I if I write it in the notation that we had used earlier this  $(b_r / 2) * \text{block transfer cost} + 1 \text{ seek cost}$ .

So, this should give us the cost of the finding out the particular record from any file if we are doing a linear search if we are doing a linear scan. So, the advantage of this is it can be applied irrespective of the condition, ordering of the records whether or not the index is available and so, on.

So, this could be the fallback in any case when we want to do that and just you may note that in memory when we search we say that we will keep the data sorted and binary search is efficient, but that is not the case for us here because as you know the data is not stored sequentially it is in terms of a tree structure. So, when the index is available we

will do the index based search otherwise we will have to do some kind of a linear scan alone. So, this was the first algorithm that we can think of.

(Refer Slide Time: 14:44)

The slide has a title 'Selections Using Indices' at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list of algorithms and their details:

- **Index scan** – search algorithms that use an index
  - selection condition must be on search-key of index
  - $h_i$  = height of B+ Tree
- **A2 (primary index, equality on key)**. Retrieve a single record that satisfies the corresponding equality condition
  - $Cost = (h_i + 1) * (t_T + t_S)$
- **A3 (primary index, equality on nonkey)** Retrieve multiple records
  - Records will be on consecutive blocks
    - ▶ Let  $b$  = number of blocks containing matching records
  - $Cost = h_i * (t_T + t_S) + t_S + t_T * b$

At the bottom left, there is a small video thumbnail showing a person speaking. The bottom right shows a standard Windows taskbar with icons for Start, Task View, File Explorer, Edge, Mail, Photos, and others.

Now, if now let us assume that it is the situation is such that we have some index on the B tree B + tree that we are using to going to do the selection on and let us assume that  $h_i$  is the height of that B+ tree. So, the second algorithm is good if we are using a primary index and we are looking for equality on a key that whether it matches certain key. So, what will have to do we know that in B+ tree if the if the height is  $h_i$  then we will be able to find the leaf node surely by  $h_i$  number of block transfers because we will be a  $h_i$  is the height of the tree.

So, this is a number of nodes the number of blocks that we will need to read. So, if each one of that will take a transfer time plus a seek time because they are not consecutively located. So, everything they will have to be will need to seek them. So, that will be  $h_i$  times  $t_T + t_S$  and we will need one additional block transfer to actually get the data get the record read it. So, that will give us cost that we have shown here.

In a variant of this algorithm A3 we may be using a primary index, but we are looking for equality on a non key. So, if you are looking for equality on a non key since is a non key then certainly in the result we may have multiple records, but the records will be on consecutive blocks because we are using a primary index.

So, if  $b$  is the number of blocks containing matching records then we will need to have say this is a cost to find out the first one and then they from consecutively they are on. We will need to locate the next record and the  $b$  blocks for transferring all the matching records this is the kind of cost that will need to go through natural you can see that in these cases all these cost expressions are better than what we were getting in terms of doing a linear search.

(Refer Slide Time: 17:08)

The slide has a header 'Selections Using Indices' with a sailboat icon. The main content is a bullet-point list under '■ A4 (secondary index, equality on nonkey)'.

- Retrieve a single record if the search-key is a candidate key
  - Cost =  $(h_i + 1) * (t_T + t_S)$
- Retrieve multiple records if search-key is not a candidate key
  - each of  $n$  matching records may be on a different block
  - Cost =  $(h_i + n) * (t_T + t_S)$ 
    - Can be very expensive!

On the left margin, vertical text reads: SWAYAM, NPTEL-NOOC, MDOC-4 Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018. At the bottom, there's a small video thumbnail of a man, the text 'Database System Concepts - 8<sup>th</sup> Edition', slide number '38.17', and copyright information '©Silberschatz, Korth and Sudarshan'.

If we look into a few of the other situations for example let us say instead of primary index if I have a secondary index and you are looking for a equality or non key then you can retrieve a single record if the search key is candidate key. If it is a candidate key then we know that even though it is not a primary key, but certainly two tuples can never match on them and still exist. So, it is a candidate key we will need to have we will get only a single record and therefore, this is a cost expression that you will get, but if it is not a candidate key then there could be multiple records that will have to be finally, retrieved.

So, if there are  $n$  records then first you will need  $h_i$  to locate the first record and times of course,  $(h_i + 1) * (t_T + t_S)$  cost and if there are  $n$  records then you will need to every time each one of them because they are on secondary index and non key. So, you have to retrieve them one by one and every time you will need a search you will need seek and

transfer time and this can as you can understand could be quite expensive if n turns out to be large which will often be the case.

(Refer Slide Time: 18:18)

**Selections Involving Comparisons**

- Can implement selections of the form  $\sigma_{A \leq v}(r)$  or  $\sigma_{A \geq v}(r)$  by using
  - a linear file scan,
  - or by using indices in the following ways:
- **A5 (primary index, comparison).** (Relation is sorted on A)
  - For  $\sigma_{A \geq v}(r)$  use index to find first tuple  $\geq v$  and scan relation sequentially from there
    - Cost =  $h_i * (t_f + t_s) + b * t_r$
  - For  $\sigma_{A \leq v}(r)$  just scan relation sequentially till first tuple  $> v$ ; do not use index
    - Cost =  $t_s + b * t_r$

SWAYAM: NITTEL-NOC: Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr- 2018

Database System Concepts - 6<sup>th</sup> Edition

38.18

©Silberschatz, Korth and Sudarshan

We can implement different this is a very common selection condition where we have these kind of conditions that we are selecting on less than equal to or greater than equal to kind of condition.

So, we can implement this using a linear file scan or by using indices in a certain way using indices we will certainly have a better performance. So, if we have a if we have a primary index and we are using comparison then what we will we can certainly decide is we need to find out the first tuple which matches this condition that is a is greater than equal to v and then once we have found that in a primary index the following ones will all be ordered in that manner. So, I can scan sequentially from there and get them.

So, finding the first one will give me finding the first one will give will take this cost because I am doing a search on the primary index and then if there are b blocks containing the result records then this is the cost that will need. Whereas, we can do something different also we can just start sequentially based on the primary index from the beginning and check for the till we get a tuple which is greater than v and then we do not use the index we can simply we know because these are all ordered in terms of the primary index. So, that will be the search result that can be easily produced. So, here we are using a linear scan and that itself will give a good result.

(Refer Slide Time: 20:03)

The slide has a title 'Selections Involving Comparisons' at the top right. On the left, there is a small logo of a sailboat on water. The main content area contains two bullet points:

- Can implement selections of the form  $\sigma_{A \leq V}(r)$  or  $\sigma_{A \geq V}(r)$  by using
  - a linear file scan,
  - or by using indices in the following ways:
- A6 (secondary index, comparison).
  - For  $\sigma_{A \geq V}(r)$  use index to find first index entry  $\geq v$  and scan index sequentially from there, to find pointers to records
    - Cost =  $(h_i + n) * (t_f + t_S)$
  - For  $\sigma_{A \leq V}(r)$  just scan leaf pages of index finding pointers to records, till first entry  $> v$ 
    - In either case, retrieve records that are pointed to
      - requires an I/O for each record
      - Linear file scan may be cheaper

At the bottom left, there is a small video thumbnail showing a man speaking. The footer contains the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr. 2018', 'Database System Concepts - 8<sup>th</sup> Edition', '38.19', and '©Silberschatz, Korth and Sudarshan'.

But if we are doing a similar operation based on a secondary index we are doing composition on a secondary index then we will again use the index to find the first index entry greater than equal to the key value and then scan the index sequentially.

So, we will get a cost which is similar to what we saw earlier and in the other condition we can just scan the leaf pages of index finding the pointers to record still the first entry so, we are doing more a sequential one. So, in either case retrieving records that are pointed to requires I/O for each record because they are on a secondary index. So, they are not necessarily consecutive and residing on the same block. So, they may be all distributed across different blocks and in such cases it may turn out that actually doing a simple linear scan may turn out to be cheaper.

(Refer Slide Time: 20:59)

Implementation of Complex Selections

- **Conjunction:**  $\sigma_{\theta_1} \wedge \theta_2 \wedge \dots \wedge \theta_n(r)$
- **A7 (conjunctive selection using one index)**
  - Select a combination of  $\theta_i$  and algorithms A1 through A6 that results in the least cost for  $\sigma_{\theta_i}(r)$
  - Test other conditions on tuple after fetching it into memory buffer
- **A8 (conjunctive selection using composite index)**
  - Use appropriate composite (multiple-key) index if available
- **A9 (conjunctive selection by intersection of identifiers)**
  - Requires indices with record pointers
  - Use corresponding index for each condition, and take intersection of all the obtained sets of record pointers
  - Then fetch records from file
  - If some conditions do not have appropriate indices, apply test in memory

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur Date: Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition      38.20      ©Silberschatz, Korth and Sudarshan

Often we have select conditions which are conjunction. So, it could be conjunctive select and may be using only one index in that case it depends on if there are n conditions then we will need to depending on the combination of this condition  $\Theta_n$  and the algorithms that we have seen here we can evaluate as to which strategy will give that least cost for this condition.

So, we will do the access based on that and then once we have accessed that tuple then we will try out the other conditions on the tuples that have been fetched into the memory buffer. You can also do conjunctive selection using composite index we can there are depending on the attributes involved in  $\Theta_1, \Theta_2, \Theta_n$  we may have a multi key index and that decision of course, as to whether I have a multi key index or what is that multi key index is of course, dependent on the earlier statistics.

But if we have some multi key index which are appropriate composite index then we can use that and more directly get the result which will be more efficient. Or we can do conjunctive selection by intersection of identifiers which require indices with record pointers and will use corresponding index for each condition and then fetch the records which is simple to understand.

(Refer Slide Time: 22:29)

The slide has a header 'Algorithms for Complex Selections' with a sailboat icon. It lists several selection algorithms:

- **Disjunction:**  $\sigma_{\theta_1 \vee \theta_2 \vee \dots \vee \theta_n}(r)$ .
- **A10 (disjunctive selection by union of identifiers)**
  - Applicable if *all* conditions have available indices
    - ▶ Otherwise use linear scan
  - Use corresponding index for each condition, and take union of all the obtained sets of record pointers
  - Then fetch records from file
- **Negation:**  $\sigma_{\neg \theta}(r)$ 
  - Use linear scan on file
  - If very few records satisfy  $\neg \theta$ , and an index is applicable to  $\theta$ 
    - ▶ Find satisfying records using index and fetch from file

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kanpur Date: Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition      38.21      ©Silberschatz, Korth and Sudarshan

Disjunction this that was conjunction if we want to do disjunction then if we have all conditions all of these conditions have index on them if it is available index then we can do something better. Otherwise if we do not have that then it is better to do a linear scan because the conditions all triples which satisfies  $\Theta_1$  will be there in the result, those which satisfy  $\Theta_2$  may or may not satisfy the others will also be there and so, on.

So, what we can do is if we have index on each one of these based on each one of these conditions then they can use corresponding index for each condition get the results and take their union and then fetch these records. So, these are some of the so, I just gave you a quick outline in terms of some of the different algorithms that selection could use. Negation of a condition could also be done, but it usually requires a linear scan on the file that is there is not much optimization that you can think of here. The next operation which is often required may not be explicitly, but in terms of doing other operations is sorting.

(Refer Slide Time: 23:47)

The slide is titled "Sorting" in red. It contains the following bullet points:

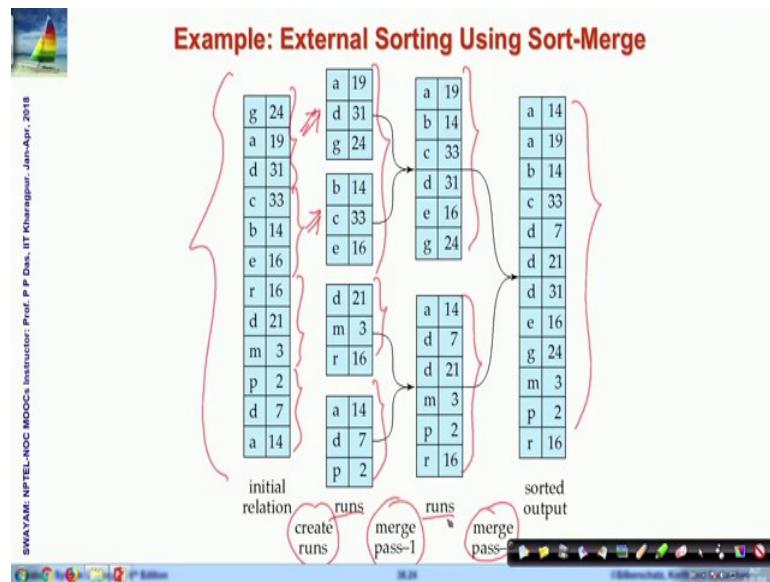
- We may build an index on the relation, and then use the index to read the relation in sorted order
  - May lead to one disk block access for each tuple
- For relations that fit in memory, techniques like quicksort can be used
- For relations that do not fit in memory, **external sort-merge** is a good choice

On the left side of the slide, there is a vertical sidebar with the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P.P. Desai, IIT Kanpur Date: Jan-Apr. 2018". At the bottom left, it says "Database System Concepts - 8<sup>th</sup> Edition". On the right side, there is a navigation bar with icons for back, forward, search, and other presentation controls. The page number "38.23" is at the bottom center, and the copyright notice "©Silberschatz, Korth and Sudarshan" is at the bottom right.

So, if we may build an index on the relation then we can use that index to read the relation in sorted order, this is what we have already discussed that B + tree in the in order traversal will always give you the sorted order. So, that may lead to one disk access for each tuple at times. Now that if the relation can totally fit all the records can totally fit into the memory then we can use some in memory algorithm like quicksort, but often that will not be the case relations are much bigger.

So, what will have to do is will have to take recourse to external sort and merge strategy which is a very old strategy, but very effective.

(Refer Slide Time: 24:30)



So, just to illustrate that suppose sorry so, suppose these are this is the initial relation so, what we do certainly we cannot read that whole relation in terms of memory into a memory. So, what we do we take different parts and say we are taking in groups of three just for illustration and make them and sort them in memory. So, take them so, take that many records which you can fit into the memory and sort them.

So, once you have sorted them then you can these are two sorted sub lists of the original set of records. So, now, you can merge them according to the merge strategy so, this is the sample merge saw strategy and again you write this back you do the similar things again here write them back. So, now, you have two bigger short sorted lists so, so these are called runs. So, the first step creates the runs and now after you have done merging once you get longer runs then again you merge them into a bigger run and depending on the on the actual size of the file and the size of the memory that directly fits in you might be doing multiple such runs till you get to the sorted output.

(Refer Slide Time: 25:52)

The slide has a decorative background featuring a sailboat on water. The title 'External Sort-Merge' is at the top right. The text 'Let  $M$  denote memory size (in pages)' is followed by a numbered list:

1. **Create sorted runs.** Let  $i$  be 0 initially  
Repeatedly do the following till the end of the relation:
  - (a) Read  $M$  blocks of relation into memory
  - (b) Sort the in-memory blocks
  - (c) Write sorted data to run  $R_i$ ; increment  $i$ .Let the final value of  $i$  be  $N$
2. **Merge the runs (next slide)....**

At the bottom left is the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018'. At the bottom center are page numbers 'Database System Concepts - 8<sup>th</sup> Edition' and '38.25'. At the bottom right is the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, that is a very external sort merge is a very effective strategy that the databases will always use and the efficiency of that or the cost of that depends on the size of the memory in terms of pages as to what can fit a complete one run data. So, this is whatever I have described is simply given here in steps of algorithm.

(Refer Slide Time: 26:16)

The slide has a decorative background featuring a sailboat on water. The title 'External Sort-Merge (Cont.)' is at the top right. The text '2. Merge the runs (N-way merge). We assume (for now) that  $N < M$ ' is followed by a numbered list:

1. Use  $N$  blocks of memory to buffer input runs, and 1 block to buffer output. Read the first block of each run into its buffer page
2. **repeat**
  1. Select the first record (in sort order) among all buffer pages
  2. Write the record to the output buffer. If the output buffer is full write it to disk.
  3. Delete the record from its input buffer page  
If the buffer page becomes empty **then**  
read the next block (if any) of the run into the buffer
3. **until** all input buffer pages are empty:

At the bottom left is the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr- 2018'. At the bottom center are page numbers 'Database System Concepts - 8<sup>th</sup> Edition' and '38.26'. At the bottom right is the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, that is a sort that is that here is a merge so, you can go through that and convince yourself that this is what algorithm is actually doing.

(Refer Slide Time: 26:24)



## External Sort-Merge (Cont.)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018

- If  $N \geq M$ , several merge passes are required
  - In each pass, contiguous groups of  $M - 1$  runs are merged.
  - A pass reduces the number of runs by a factor of  $M - 1$ , and creates runs longer by the same factor
    - ▶ E.g. If  $M=11$ , and there are 90 runs, one pass reduces the number of runs to 9, each 10 times the size of the initial runs
  - Repeated passes are performed till all runs have been merged into one

Database System Concepts - 8<sup>th</sup> Edition 38.27 ©Silberschatz, Korth and Sudarshan

And there are two cases you have to consider whether your data fits into the memory otherwise if your it does not fit into the memory then multiple passes are required and these are the steps of the algorithm that will be followed. Now next to sorting certainly we have often talked about that join is a very required operation in relational database in terms of SQL.

(Refer Slide Time: 26:56)



## Join Operation

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018

- Several different algorithms to implement joins
  - Nested-loop join
  - Block nested-loop join
  - Indexed nested-loop join
  - Merge-join
  - Hash-join
- Choice based on cost estimate
- Examples use the following information
  - Number of records of student: 5,000 takes: 10,000
  - Number of blocks of student: 100 takes: 400



Database System Concepts - 8<sup>th</sup> Edition 38.29 ©Silberschatz, Korth and Sudarshan

So, let us see what will it take to do a join so, first we talk about so, the join could be done in several ways nested loop join, block nested loop join, indexed nested loop join,

merge join, hash join. So, these are different strategies of doing join we will just illustrate the algorithms for the first three strategies.

(Refer Slide Time: 27:15)

The slide has a title 'Nested-Loop Join' in red at the top right. On the left is a small sailboat icon. The main content is a bulleted list of points about the Nested-Loop Join algorithm:

- To compute the theta join  $r \bowtie_{\theta} s$   
for each tuple  $t_r$  in  $r$  do begin  
  for each tuple  $t_s$  in  $s$  do begin  
    test pair  $(t_r, t_s)$  to see if they satisfy the join condition  $\theta$   
    if they do, add  $t_r \cdot t_s$  to the result.  
  end  
end
- $r$  is called the **outer relation** and  $s$  the **inner relation** of the join
- Requires no indices and can be used with any kind of join condition
- Expensive since it examines every pair of tuples in the two relations

At the bottom left is a small video thumbnail of a professor. The footer includes the text 'SWAYAM: NPTEL-NOCOCS Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018', 'Database System Concepts - 6<sup>th</sup> Edition', '38.30', and '©Silberschatz, Korth and Sudarshan'.

So, nested loop join what we are trying to do is very simple we have two relations we have two relations here  $r$  and  $s$  and we have a condition  $\Theta$  and we are doing a  $\Theta$  join. So, what needs to be done in terms of  $\Theta$  join in the relational algebra what do we do we do a Cartesian product and then in the Cartesian product we check out this  $\Theta$  condition.

So, the basic Cartesian product is all records of  $r$  will have to be matched will have to be connected to all record of  $s$ . So, that naturally can be done using a nested for loop so, for each tuple in our you try out each tuple in  $s$  take the  $t_r, t_s$  pair and if they satisfy the condition  $\Theta$  then they go to the output otherwise you leave that otherwise you discard that and here we say  $r$  is the outer relation and this  $s$  is the inner relation. So, naturally since you have to examine every pair this could be quite expensive to perform and the cost may be quite high.

(Refer Slide Time: 28:19)

The slide has a header 'Nested-Loop Join (Cont.)' with a sailboat icon. The content includes a list of points about nested-loop joins:

- In the worst case, if there is enough memory only to hold one block of each relation, the estimated cost is  
$$n_r * b_s + b_r$$
 block transfers, plus  
$$n_r + b_r$$
 seeks
- If the smaller relation fits entirely in memory, use that as the inner relation.
  - Reduces cost to  $b_r + b_s$  block transfers and 2 seeks
- Example of join of students and takes:
  - Number of records of student: 5,000 takes: 10,000
  - Number of blocks of student: 100 takes: 400
- Assuming worst case memory availability cost estimate is
  - with student as outer relation:
    - $5000 * 400 + 100 = 2,000,100$  block transfers,
    - $5000 + 100 = 5100$  seeks
  - with takes as the outer relation:
    - $10000 * 100 + 400 = 1,000,400$  block transfers and 10,400 seeks
- If smaller relation (student) fits entirely in memory, the cost estimate will be 500 block transfers
- Block nested-loops algorithm (next slide) is preferable

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P.P. Desai, IIT Kanpur Date: Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition      38.31      ©Silberschatz, Korth and Sudarshan

So, if we look at what could be the possible cost. So, if  $n_r$  is the number of records in relation  $r$  and  $b_r$  is the number of blocks in which they exist then for every record you have to actually access all the blocks of the other every for every tuple of relation  $r$  you have to actually access all the blocks of relation  $s$ . So, you get this and you have to access all the blocks of relation  $r$ .

So, this is the kind of block transfer that you will get you will require and naturally you will require so many seek because every time you have to find out you have to go and seek for that. So, one optimization that is very common is what you can do is if the smaller relation can entirely fit into the memory then you do not need to do this repeated read for that both the relations.

So, if it fits into that then the cost will significantly reduce to  $b_r + b_s$  block transfers because you want the smaller one has already fit. So, you just need to access one relation once you need to read the smaller relation and put it in the memory and then you just need to read the other relation one after the other. So,  $b_r$  blocks of that and so, you are seeking only twice one for reading  $r$ , one for reading  $s$  there is a 2 seeks.

So, here I have just shown a simple example of computing the join of student and takes let us say student has 5000 records and spread over 100 blocks takes relation has 10000 records spread over 400 blocks then if you apply the formula above you will find that if student is the outer relation you have so, many block transfer and so, many seeks.

Whereas, if takes is the outer relation then you have so many block transfers and so many seeks.

So, you can understand you can see here that if you make student as a outer relation then you have much larger number of block transfers though you need to do less number of seek, but taking, but using takes as a outer relation you have much less block transfers, but more number of seek usually seek is less expensive than less costly than the block transfer.

So, will possibly in with this kind of a statistics if it is available then we will possibly take takes as outer relation and student as the inner one. You can refine this.

(Refer Slide Time: 31:07)

**Block Nested-Loop Join**

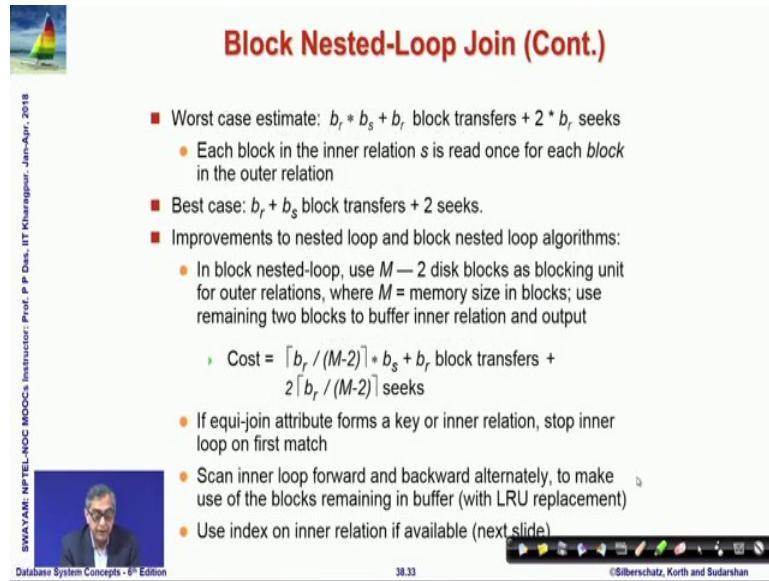
■ Variant of nested-loop join in which every block of inner relation is paired with every block of outer relation

```
for each block  $B_r$  of  $r$  do begin
    for each block  $B_s$  of  $s$  do begin
        for each tuple  $t_r$  in  $B_r$  do begin
            for each tuple  $t_s$  in  $B_s$  do begin
                Check if  $(t_r, t_s)$  satisfy the join condition
                if they do, add  $t_r \cdot t_s$  to the result.
            end
        end
    end
end
```

SWAYAM: NPTEL-NOC's Instructor: Prof. P. P. Dand, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
38.32  
©Silberschatz, Korth and Sudarshan

Strategy by doing a block nesting that is instead of taking every tuple of the relation you can take every block of the relation. So, for every block of relation  $r$  you try to match with you try to combine with every block of relation  $s$  and then within every block of relation  $r$  the block  $b_r$  you take tuple and within  $b_s$  you take  $t_s$  and then you do whatever we are doing earlier, but naturally you get a much better performance.

(Refer Slide Time: 31:46)



**Block Nested-Loop Join (Cont.)**

■ Worst case estimate:  $b_r * b_s + b_r$  block transfers +  $2 * b_r$  seeks

- Each block in the inner relation  $s$  is read once for each block in the outer relation

■ Best case:  $b_r + b_s$  block transfers + 2 seeks.

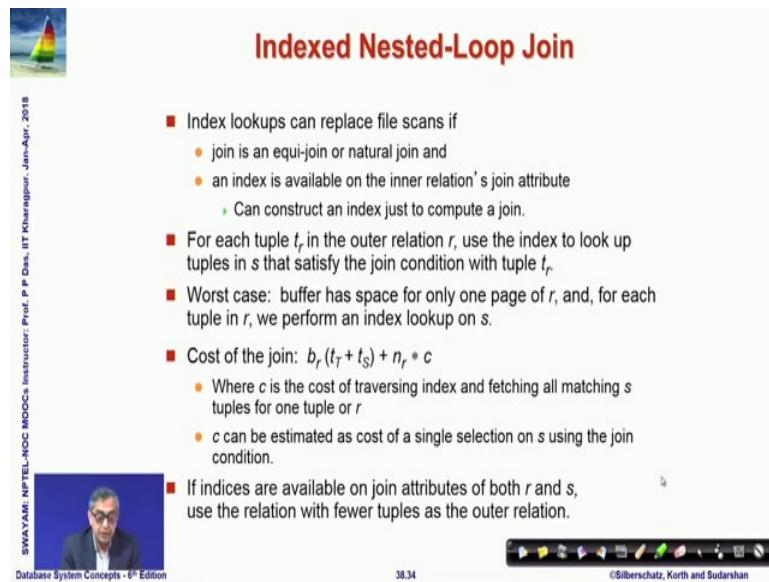
■ Improvements to nested loop and block nested loop algorithms:

- In block nested-loop, use  $M - 2$  disk blocks as blocking unit for outer relations, where  $M$  = memory size in blocks; use remaining two blocks to buffer inner relation and output
  - Cost =  $\lceil b_r / (M-2) \rceil * b_s + b_r$  block transfers +  $2 \lceil b_r / (M-2) \rceil$  seeks
- If equi-join attribute forms a key or inner relation, stop inner loop on first match
- Scan inner loop forward and backward alternately, to make use of the blocks remaining in buffer (with LRU replacement)
- Use index on inner relation if available (next slide)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr., 2018  
Database System Concepts - 8<sup>th</sup> Edition  
38.33 ©Silberschatz, Korth and Sudarshan

Because you are now optimizing based on the block reads only you are not reading every tuple every time you need. So, I will not go through this you know simple algebra to show that if you have a memory size of  $M$  M blocks that your cost will significantly decrease and, but the larger the  $M$  your cost will come down by a factor of this  $M$ . So, block nested loop join will usually be far more efficient than the simple nested loop join.

(Refer Slide Time: 32:14)



**Indexed Nested-Loop Join**

■ Index lookups can replace file scans if

- join is an equi-join or natural join and
- an index is available on the inner relation's join attribute
  - Can construct an index just to compute a join.

■ For each tuple  $t_r$  in the outer relation  $r$ , use the index to look up tuples in  $s$  that satisfy the join condition with tuple  $t_r$ .

■ Worst case: buffer has space for only one page of  $r$ , and, for each tuple in  $r$ , we perform an index lookup on  $s$ .

■ Cost of the join:  $b_r(t_r + t_s) + n_r * c$ 

- Where  $c$  is the cost of traversing index and fetching all matching  $s$  tuples for one tuple of  $r$
- $c$  can be estimated as cost of a single selection on  $s$  using the join condition.

■ If indices are available on join attributes of both  $r$  and  $s$ , use the relation with fewer tuples as the outer relation.

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jan-Apr., 2018  
Database System Concepts - 8<sup>th</sup> Edition  
38.34 ©Silberschatz, Korth and Sudarshan

The third strategy which we will use very often is efficiently applicable if you are if your join is an equijoin or a natural join as we have seen that we often need to do a natural

join. So, there are two attributes between these two relations on which during join the values that they match are retained, the values that they do not match are not retained in the natural join.

So, if we now assume that we have an index available on the inner relation then every time we go with the outer relation will be able to access the inner relation very efficiently because for each tuple in the outer relation the index to look up the tuples in nests will satisfy the condition will be found very efficiently because they are index. So, they will occur if through the index I can find them in terms of the consecutivity.

So, there the cost in that case will turn out to be very simply the cost of the  $b_r$  which is the outer relation the number of blocks in the outer relation the seek and transfer cost of that and then the number of record times, the estimated cost of a single selection using the join condition. So, we often use the nested loop join when we have to do equal join or natural join.

(Refer Slide Time: 33:44)

The slide has a title 'Example of Nested-Loop Join Costs' in red at the top right. To the left of the title is a small icon of a sailboat on water. On the far left edge, there is vertical text that reads 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Date, IIT Kanpur - Jan-Apr- 2018'. The main content area contains a bulleted list of points:

- Compute  $student \bowtie takes$ , with  $student$  as the outer relation.
- Let  $takes$  have a primary B-tree index on the attribute  $ID$ , which contains 20 entries in each index node.
- Since  $takes$  has 10,000 tuples, the height of the tree is 4, and one more access is needed to find the actual data.
- $student$  has 5000 tuples
- Cost of block nested loops join
  - $400 * 100 + 100 = 40,100$  block transfers +  $2 * 100 = 200$  seeks
    - ▶ assuming worst case memory
    - ▶ may be significantly less with more memory
- Cost of indexed nested loops join
  - $100 + 5000 * 5 = 25,100$  block transfers and seeks.
  - CPU cost likely to be less than that for block nested loops join

At the bottom of the slide, there is footer text: 'Database System Concepts - 8<sup>th</sup> Edition', '38.35', and '©Silberschatz, Korth and Sudarshan'. There is also a decorative horizontal bar with various icons.

So, here is an example with the same students and takes example. So, it shows that the cost of block nested join if you work out for the block nested join then you have so, many 40,100 block transfers and 200 seek. Whereas, if you do index to one on the assuming that the smaller relation the inner relation has a index then you have 25,000 block transfer and seek. So, this will turn out to be a naturally more efficient way of implementing the join.

So, there are several other strategies particularly hashing based strategies, merging based strategies which we are not discussing here, but there are different strategies through which you can do join in more and more efficient manner.

(Refer Slide Time: 34:36)

The screenshot shows a presentation slide with the following elements:

- Section Title:** Other Operations
- List:** A bulleted list of six operations:
  - Duplicate elimination
  - Projection
  - Aggregation
  - Set Operations
  - Outer Join
- Navigation:** A small video player interface at the bottom left shows a video thumbnail of a man speaking, the text "Database System Concepts - 6<sup>th</sup> Edition", the number "38.37", and a progress bar.
- Decorations:** A small sailboat icon is in the top left corner, and a decorative footer bar with various icons is at the bottom right.

Couple of other operations which are often required is duplicate elimination because if they we know that there are duplicate records cannot be kept, duplicate in the sense the records which match in the key field and the duplicate will often happen in terms of the result, they will happen in terms of when we do projection, we will need to do aggregation set operations outer join and so, on. So, the first three we will quickly outline.

(Refer Slide Time: 35:05)

The slide has a header 'Other Operations' with a sailboat icon. The content includes a list of operations:

- **Duplicate elimination** can be implemented via hashing or sorting
  - On sorting duplicates will come adjacent to each other, and all but one set of duplicates can be deleted
  - *Optimization*: duplicates can be deleted during run generation as well as at intermediate merge steps in external sort-merge
  - Hashing is similar – duplicates will come into the same bucket
- **Projection:**
  - perform projection on each tuple
  - followed by duplicate elimination

At the bottom, there is a video player showing a speaker, the title 'Database System Concepts - 6<sup>th</sup> Edition', the time '38.38', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, duplicate naturally can be very easily eliminated through sorting, they can be done through hashing also because if we sort they will come on side by side will come consecutively after the sort, if we hash then they will necessarily hash to the same value which becomes easier to check whether they are identical or not. If whenever we are doing projection we can project on each tuple and then you can perform a duplicate elimination to actually get to the final result. Aggregation group that is whatever you do group by kind of.

(Refer Slide Time: 35:37)

The slide has a header 'Other Operations : Aggregation' with a sailboat icon. The content includes a list of aggregation operations:

- **Aggregation** can be implemented in a manner similar to duplicate elimination
  - Sorting or hashing can be used to bring tuples in the same group together, and then the aggregate functions can be applied on each group
  - *Optimization*: combine tuples in the same group during run generation and intermediate merges, by computing partial aggregate values
    - For count, min, max, sum: keep aggregate values on tuples found so far in the group
      - When combining partial aggregate for count, add up the aggregates
    - For avg, keep sum and count, and divide sum by count at the end

At the bottom, there is a video player showing a speaker, the title 'Database System Concepts - 6<sup>th</sup> Edition', the time '38.39', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, aggregation is certainly will be efficiently done if you have again done sorting because if you are grouping by something then if you have sorted on that those elements will come together and or if you are hashing then they will also come together. So, you can easily do the computation on that.

And what you can do is instead of for example, you are doing a count or you are doing a minimum, maximum, sum this kind of all of these are associative operations. So, you can do it in parts that if you have done in this sorted order, in the hashed order if you have done the sum of 10 records then you can actually do not need these 10 records when you do the sum for the next 10 records and so, on. So, in this manner the aggregation can be efficiently implemented. So, we can for average keep sum and count and divide the sum by count at the end and so, on.

(Refer Slide Time: 36:39)

**Module Summary**

- Understood the overall flow for Query Processing and defined the Measures of Query Cost
- Studied the algorithms for processing Selection Operations, Sorting, Join Operations and a few Other Operations

SWAYAM NPTEL-NOCO MOOCs Instructor: Prof. P. P. Deshpande, IIT Kharagpur - Jam-Apr-2018

Database System Concepts - 8<sup>th</sup> Edition

38.40 ©Silberschatz, Korth and Sudarshan

So, we have in this module we have just given a very brief outline of what are the steps involved in query processing and what are the measures that define a query cost typically and we have been talked about some of the simple algorithms for selection, sorting, join and aggregation operations.

In the next module we will talk about elementary optimization strategies for processing of queries.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 39**  
**Query Processing and Optimization/2: Optimization**

Welcome to module 39 of database management systems, we have been discussing about query processing and optimization, in the last module.

(Refer Slide Time: 00:25)

The slide has a header 'Module Recap' in red. On the left, there is a small image of a sailboat on water. A vertical sidebar on the left contains the text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018'. The main content area lists the following topics:

- Overview of Query Processing
- Measures of Query Cost
- Selection Operation
- Sorting
- Join Operation
- Other Operations

At the bottom, there is footer text: 'Database System Concepts - 8<sup>th</sup> Edition', '39.2', and '©Silberschatz, Korth and Sudarshan'. There is also a decorative toolbar icon at the bottom right.

We talked about the basic issues of query processing, what is the overview and the measures of query cost that would be used in terms of disk seek time and disk access time the read write time and we agreed we assume that we will ignore the CPU and other time for the time being. And then we took a look into how the certain SQL queries like the selection, the sorting which is required for other operations, different join operations and other aggregation and those kind of operations can be processed in a structured way.

(Refer Slide Time: 01:10)

Module Objectives

- To understand the basic issues for optimizing queries
- To understand how transformation of Relational Expressions can create alternates for optimization

PPD

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur, Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

39.3

©Silberschatz, Korth and Sudarshan

In view of this in this module we would like to understand the basic issues of optimizing queries. So, that the same query as we have seen little bit can be performed executed in multiple ways and we would like to choose the one which is the least cost possibly estimated cost should be the least.

So, we would need to understand two aspects, one is given a query how do I generate alternate queries that is in terms of the relational expression how a particular expression can be transformed into equivalent expressions and then we choose from these equivalence expressions for optimization. So, these are the two topics to discuss.

(Refer Slide Time: 01:57)

**Introduction**

- Alternative ways of evaluating a given query
  - Equivalent expressions
  - Different algorithms for each operation

$\text{course}(\underline{\text{course id}}, \text{title}, \text{dept name}, \text{credits})$   
 $\text{instructor}(\underline{\text{ID}}, \text{name}, \text{dept name}, \text{salary})$   
 $\text{teaches}(\underline{\text{ID}}, \underline{\text{course id}}, \underline{\text{sec id}}, \text{semester}, \text{year})$

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018  
 Database System Concepts - 8<sup>th</sup> Edition      39.6      ©Silberschatz, Korth and Sudarshan

So, to introduce on the query optimization let us take a simple example. So, I am referring to the university database that we have discussed earlier, it has three relations as listed above and using that we want to do the perform the query where we join **instructor**  $\bowtie$  **teaches**  $\bowtie$  **course** and do a  $\sigma$  on that based on department name. So, this will give us naturally the instructors who are in teaching some course in the instructor is in music department and is teaching some course there and we want the name and title of these. So, we want the name of the instruction instructor and the title of the course that the person is teaching.

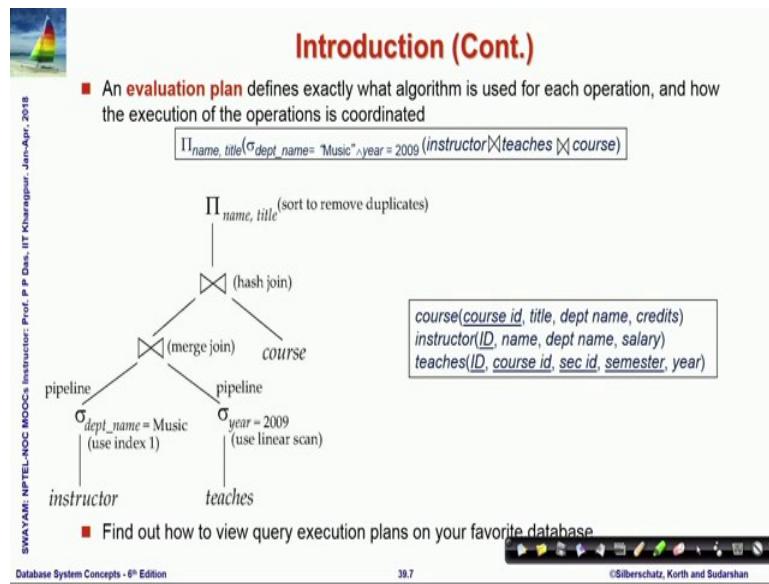
So, if we write the query in equivalent relational form this is what we will get to see and this is basically is a first join happening here then the next join happening, find next the  $\sigma$  and finally, the  $\Pi_{\text{name}, \text{title}}$  happening here which will give us the result of this query. Now what we observe is it is possible that if you look at carefully the department name should be music.

So, if we look into these relations then we can easily figure out that instructor has the department name attribute. So, in this after this joint if a tuple has to qualify through this selection then the instructors in the instructor relation the department name of that join people must be music otherwise it will not get selected.

So, what if instead of doing the join and then doing the selection as we are doing here if we first do a selection on the instructor relation itself and then join it with the join of

teachers and courses and finally, do the projection we should actually get the same result. So, these are what is it is a simple example of what are equivalent relational expressions that we can make use of in terms of our query processing.

(Refer Slide Time: 04:20)



So, what given that this is another example we were showing. So, here the query is to find the teacher and instructor and the course name for back the instructor is from department of the music and the course he taught is in the year 2009. So, we want these and corresponding to the equivalent at SQL if we write the relational expression then the in relational algebra this is what it looks like. So, what we can eventually observe from this that again as we observed last time department name is an is an instructor.

So, if we are doing a final  $\sigma_{dept\_name = music}$  then it is possible that I can first filter the instructor relation with the department name being music that should not affect the result. At the same time  $\sigma_{year=2009}$ , which happens only in the teacher's relation.

So, instead of actually filtering it after the join  $(\sigma_{dept\_name = music} \bowtie \sigma_{year=2009}) \bowtie course$  do  $\Pi_{name,title} ((\sigma_{dept\_name = music} \bowtie \sigma_{year=2009}) \bowtie course)$ . So, here what we show that along with this tree the parse tree of the query in relational algebra we are also trying to put some annotations to say how this particular query will be processed.

So, if we want to find out the tuples where the year is 2009 and there is no specific index on that or anything to for doing this selection as we have seen earlier the best way would

be to use a linear scan. Whereas, if I am trying to do a selection with department name is equal to is music there will be a could be an index one the secondary index based on the department name. So, I will be able to use that index to find this.

So, these when we are doing this putting and that here we will use this index, here we will use linear scan these are what is called annotations which tell the query processing engine that how the query should actually be executed. Then they will be pipelined in the sense that this will be put one after the other actually these two can happen in parallel and then we will use a merge join to join these two then this merge that is joined result would be joined with course based on now the course is already indexed in course id and this joined relation of instructor and teachers will have id and course id on which they will have index.

So, we can use a hash join on that we did not discuss about merge join, hash join as processing steps in detail, but right now just assume that these are different ways of doing join which can make things efficient and these are the annotations which are generated in the process. And such a query tree such a query parse tree with the annotation is called an execution plan so, that or the evaluation plan which the query processing engine would be able to use to actually execute and find the results.

(Refer Slide Time: 07:58)

The slide has a header 'Introduction (Cont.)' with a sailboat icon. The main content is a bulleted list:

- Cost difference between evaluation plans for a query can be enormous
  - E.g. seconds vs. days in some cases
- Steps in **cost-based query optimization**
  1. Generate logically equivalent expressions using **equivalence rules**
  2. Annotate resultant expressions to get alternative query plans
  3. Choose the cheapest plan based on **estimated cost**
- Estimation of plan cost based on:
  - Statistical information about relations.
    - ▶ Examples: number of tuples, number of distinct values for an attribute
    - Statistics estimation for intermediate results
      - ▶ to compute cost of complex expressions
    - Cost formulae for algorithms, computed using statistics

Navigation icons are at the bottom right, and footer text includes 'SWAYAM: NPTEL-MOOCs Instructor: Prof. P. Desai, IIT Kharagpur - Jan-Apr. 2018', 'Database System Concepts - 6<sup>th</sup> Edition', '39.8', and '©Silberschatz, Korth and Sudarshan'.

So, this is the basic approach so, for optimization what we need to do we need to find out that particular evaluation plan, that particular order of we the evaluation and the use of

algorithms, the use of indexes which will make the query possibly most efficient. And that cost difference could be really really huge in a real database it could vary between seconds in one way of doing the query or in terms of number of days if we do it in a non optimal way in a non optimized way.

So, typical steps in this kind of query based optimization would be to first generate the candidates I am sorry first generate the candidates that is generate the equivalent expressions that is given a query I have one relational expression and we would like to generate equivalent expression using a set of equivalence rules we will see what these equivalence rules are.

So, that this equivalent queries expressions can any one of them can be actually executed and we then annotate them to with the result the resultant expression we annotate to get different query plans evaluation plans. And then we put the cost estimates based on the cost structure that say we had used in the other in the earlier module and from these alternate evaluation plans we will choose the one that will have the least estimated cost.

So, the cost can be based on as we had seen earlier it could be based on number of tuples, number of distinct values frequently used attributes and so, on and we will use statistics for also intermediate results that if there are intermediate results to be stored we will make estimates of what would be the size of that result because it needs to fit into memory for optimal execution, the cost formula for different algorithms will also be used through statistics.

So, based on all these estimation which will have an estimated cost and based on that we will choose the particular evaluation plan which looks to be the best and that is the crux of the query optimization strategy. So, as we have seen the first step is to be able to generate alternate expressions equivalent expressions through transformations. So, we look through the relational algebra operators again.

(Refer Slide Time: 10:27)

**Transformation of Relational Expressions**

- Two relational algebra expressions are said to be **equivalent** if the two expressions generate the same set of tuples on every *legal* database instance
  - Note: order of tuples is irrelevant
  - We do not care if they generate different results on databases that violate integrity constraints
- In SQL, inputs and outputs are multisets of tuples
  - Two expressions in the multiset version of the relational algebra are said to be equivalent if the two expressions generate the same multiset of tuples on every legal database instance.
- An **equivalence rule** says that expressions of two forms are equivalent
  - Can replace expression of first form by second, or vice versa

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

39.10

©Silberschatz, Korth and Sudarshan

And check what are what is meant by equivalence of two relational expressions. So, two relational expressions are equivalent if they generate the same set of tuples for any instance of the database, it is not enough to just show that for one instance it gives the same result.

So, two expressions are equivalent for in if I take any legal instance of the database then it must be equivalent and in this process we can note that the order of tuples are really relevant that we have told repeatedly and also we would make sure that the results are same provided the database provided the relation satisfy all the integrity constraints. If they violate integrity constraints then it is a problem of the user then we the database really does not care if the two expressions will give equivalent or equal results. We also note that in SQL input output could be multisets so, the same thing has to be satisfied in terms of multisets. So, based on this we define an equivalence a set of equivalence rule that say that two expressions are equivalent so, you can use that rule to transform one expression by the other and vice versa.

(Refer Slide Time: 11:47)

The slide has a header 'Equivalence Rules' in red. On the left, there is a small sailboat icon and some vertical text: 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018'. Below the header is a numbered list of four equivalence rules:

1. Conjunctive selection operations can be deconstructed into a sequence of individual selections  
$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$
2. Selection operations are commutative  
$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$
3. Only the last in a sequence of projection operations is needed, the others can be omitted  
$$\Pi_{L_1}(\Pi_{L_2}(\dots(\Pi_{L_n}(E))\dots)) = \Pi_{L_1}(E)$$
4. Selections can be combined with Cartesian products and theta joins
  - a.  $\sigma_{\theta_1}(E_1 \times E_2) = E_1 \bowtie_{\theta_1} E_2$
  - b.  $\sigma_{\theta_1}(\sigma_{\theta_2}(E_1 \times E_2)) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

At the bottom, there is a video player showing a person speaking, with the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr., 2018' and a progress bar indicating '38 / 51'.

So, let us take a look into this expression so, most of these expressions are relatively easy to understand. They can be formally proved using the corresponding set theoretic condition of the relational expressions. So, for every relational expression we had a set theoretic condition that we had studied so, using those you can prove that we will not do the proof of these equivalence relations here, but it you should be able to do that very easily.

So, he said if we have a conjunctive selection of a relation based on two conditions  $\Theta_1$  and  $\Theta_2$ , then we should be able to apply the selection first based  $\Theta_2$ ,  $\sigma_{\Theta_2}$  and then based on  $\Theta_1$ ,  $\sigma_{\Theta_1}(\sigma_{\Theta_2})$  or actually  $\sigma_{\Theta_2}(\sigma_{\Theta_1})$  vice versa because conjunction is commutative so, the selection operation is also commutative. So, this gives us two transformation rules and we might want to use any so, these all will give equivalent form.

So, applying these two rules we get three relational expressions which are all equivalent this, this and this are all equivalent, but if you actually think in terms of processing the query and the cost involved you will be able to figure out that naturally the cost of doing this may be relatively more involved because you have the relation and then on the whole relation you are applying the conditions, but here if you do for example, if you first do say first apply  $\Theta_2$  certainly by that selection the relation would become much smaller. So, applying  $\Theta_1$  on that would be easier or vice versa depending on whichever is

a smaller set there could be other for example, if you have a sequence of projections then naturally in that sequence the last projection that you have done is what is retained.

So, if we have a sequence of projections that is simply doing the last projection all other projections can actually be ignored. The selection can be combined with Cartesian product and  $\Theta$  join also that is taking a Cartesian product and then doing a selection is equivalent to doing a  $\Theta$  join this of course, is almost the definition. And if I have a join  $\sigma_{\Theta_1}(E_1 \bowtie_{\Theta_2} E_2)$  it is equivalent to doing a  $E_1 \bowtie_{\Theta_1 \wedge \Theta_2} E_2$

So, these are all equivalent forms so, these are equivalent in the sense that left hand side and right hand side are interchangeable. So, it does not mean that the left hand side implies the right hand side or vice versa it means that either of them implies the other, they are really equivalent in that sense.

(Refer Slide Time: 14:33)

The slide has a title 'Equivalence Rules (Cont.)' in red at the top right. On the left is a small sailboat icon. The main content area contains the following text:

5. Theta-join operations (and natural joins) are commutative  
 $E_1 \bowtie_{\Theta_1} E_2 = E_2 \bowtie_{\Theta_1} E_1$

6. (a) Natural join operations are associative:  
 $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$

(b) Theta joins are associative in the following manner:  
 $(E_1 \bowtie_{\Theta_1} E_2) \bowtie_{\Theta_2 \wedge \Theta_3} E_3 = E_1 \bowtie_{\Theta_1 \wedge \Theta_3} (E_2 \bowtie_{\Theta_2} E_3)$   
where  $\Theta_2$  involves attributes from only  $E_2$  and  $E_3$ .

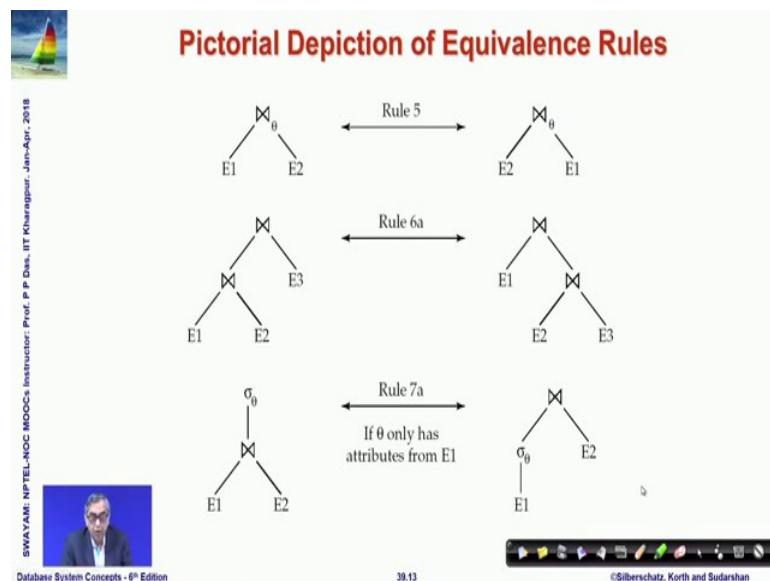
At the bottom left is a video thumbnail showing a professor. The footer includes the text 'SWAYAM: NPTEL-NOC's Instructional Platform', 'Database System Concepts - 8<sup>th</sup> Edition', '39.12', and '©Silberschatz, Korth and Sudarshan'.

Then  $\Theta$  join operation are natural or natural join operations are commutative, we can change interchange their relations.

So, this might impact for example, you have seen the algorithms of nested join and block join block nested join algorithms; obviously, depending on the size of the relations the cost of doing  $E_1 \bowtie_{\Theta} E_2$  or  $E_1 \bowtie E_2$  and  $E_2 \bowtie_{\Theta} E_1$  may be different and we would choose the one which has a lesser cost.

Next set of transformation rules tell us that the join operation is associative which is obvious  $\Theta$  joins are associative in the in this specific manner also, that is if I do a  $(E_1 \bowtie_{\Theta_1} E_2) \bowtie_{\Theta_2} E_3$  then we may be able to take out the condition  $\Theta_3$  outside provided  $\Theta_2$  the condition  $\Theta_2$  uses only the attributes of  $\Theta$ ,  $E_1$  and  $E_2$  so,  $E_1 \bowtie_{\Theta_1} E_2 \bowtie_{\Theta_2} E_3$  it should be possible to do that. Look here that on the left hand side that restriction is not there on the condition  $\Theta_2$  it could also use attributes of  $E_1$ , but if it does not then it is possible to simplify it in this manner and these are the equivalent rules that we have.

(Refer Slide Time: 16:01)



So, often we will draw the rules in this form so, just to explain you one so, this shows the associativity of join.

So, here what you are saying is first you do  $E_1, E_2$  and with the result you join  $E_3$ ,  $(E_1 \bowtie_{\Theta} E_2) \bowtie_{\Theta_3} E_3$  here you are saying first you do  $E_2, E_3$  and then you join with  $E_1$ .  $(E_2 \bowtie_{\Theta} E_3) \bowtie_{\Theta_1} E_1$  So, this is basically associativity this basically is commutativity of  $\Theta$  join. So, we will often draw them in such forms of parse trees and show the equivalence that becomes easy to understand for example, in this case you are doing a join and then doing the selection and here you are doing it select early you are doing a select early on this relation  $E_1$  of course, you will be able to do this provided  $\Theta$  contains only attribute from  $E_1$ , if  $\Theta$  contains attributes from both  $E_1$  and  $E_2$  this transformation will not be applicable this transformation will not be possible.

(Refer Slide Time: 17:00)

The slide has a header 'Equivalence Rules (Cont.)' in red. On the left, there is a small sailboat icon and a vertical footer column with text: 'SWAYAM: NPTEL-NOC MOOCs', 'Instructor: Prof. P. P. Deshpande', 'IIT Kharagpur - Jan-Apr. 2018', and 'Database System Concepts - 8th Edition'. The main content area contains a numbered list and two sub-lists under point 7. It also includes two mathematical equations. At the bottom right, there is a navigation bar with icons and the text 'Silberschatz, Korth and Sudarshan'.

7. The selection operation distributes over the theta join operation under the following two conditions:

- When all the attributes in  $\Theta_0$  involve only the attributes of one of the expressions ( $E_1$ ) being joined
$$\sigma_{\Theta_0}(E_1 \bowtie_{\Theta} E_2) = (\sigma_{\Theta_0}(E_1)) \bowtie_{\Theta} E_2$$
- When  $\Theta_1$  involves only the attributes of  $E_1$  and  $\Theta_2$  involves only the attributes of  $E_2$ .
$$\sigma_{\Theta_1 \wedge \Theta_2}(E_1 \bowtie_{\Theta} E_2) = (\sigma_{\Theta_1}(E_1)) \bowtie_{\Theta} (\sigma_{\Theta_2}(E_2))$$

So, these are some more of the transformation rules the selection operation distributes over  $\Theta$  join operation. So, I can you can I can see here that there is a  $\Theta$  join and then I am doing a selection. So, I can first do the selection and then do the  $\Theta$  join of course, for the selection it must involve only the attributes of  $E_1$ , otherwise this will not be valid.

And similarly, another distribution rule is shown here where you are doing a selection on conjunction on  $\sigma_{\Theta_1 \wedge \Theta_2}(E_1 \bowtie_{\Theta} E_2)$  then you should be able to actually distribute based on the condition  $\Theta_1$  and  $\Theta_2$ , if  $\Theta_1$  involves only the attributes of  $E_1$  and  $\Theta_2$  involves only the attributes of  $E_2$ ,  $\sigma_{\Theta_1}(E_1) \bowtie_{\Theta} \sigma_{\Theta_2}(E_2)$ . These are these are pretty straightforward rules if you think about the corresponding set theoretic reason.

(Refer Slide Time: 17:58)

The slide has a header 'Equivalence Rules (Cont.)' in red. On the left, there is a small image of a sailboat on water. The main content is as follows:

8. The projection operation distributes over the theta join operation as follows:

(a) if  $\Theta$  involves only attributes from  $L_1 \cup L_2$ :

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\Theta E_2) = (\Pi_{L_1}(E_1)) \bowtie_\Theta (\Pi_{L_2}(E_2))$$

(b) Consider a join  $E_1 \bowtie_\Theta E_2$ .

- Let  $L_1$  and  $L_2$  be sets of attributes from  $E_1$  and  $E_2$ , respectively
- Let  $L_3$  be attributes of  $E_1$  that are involved in join condition  $\Theta$ , but are not in  $L_1 \cup L_2$ , and
- Let  $L_4$  be attributes of  $E_2$  that are involved in join condition  $\Theta$ , but are not in  $L_1 \cup L_2$ .

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\Theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\Theta (\Pi_{L_2 \cup L_4}(E_2)))$$

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Deshpande, IIT Kanpur Date: Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

39.15

©Silberschatz, Korth and Sudarshan

So, the other rules will be in terms of projection so, you can have if you have union projection like this then you would be able to break it down over to do separate projections and their  $\Theta$  join and in case you have a  $\Theta$  join of  $E_1, E_2$  based on  $\Theta$  and if  $L_1$  and  $L_2$  are the attributes of these two relations.

So, if  $L_3$  be the attributes of  $E_1$  that are involved in the join condition  $\Theta$  and  $L_4$  are what are. So, you have the join condition  $\Theta$ . So, what you are saying that the attributes  $L_3$  of you are not involved here and attributes  $L_4$  of  $E_2$  are involved here, but they are not so, in terms of  $L_1 \cup L_2$  so, then this kind of I should be able to distribute the projection in steps and make the results smaller.

(Refer Slide Time: 19:04)



## Equivalence Rules (Cont.)

9. The set operations union and intersection are commutative

$$E_1 \cup E_2 = E_2 \cup E_1$$
$$E_1 \cap E_2 = E_2 \cap E_1$$

■ (set difference is not commutative).

10. Set union and intersection are associative.

$$(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$$
$$(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)$$

11. The selection operation distributes over  $\cup$ ,  $\cap$  and  $-$ .

$$\sigma_0(E_1 - E_2) = \sigma_0(E_1) - \sigma_0(E_2)$$

and similarly for  $\cup$  and  $\cap$  in place of  $-$

Also:  $\sigma_0(E_1 - E_2) = \sigma_0(E_1) - E_2$   
and similarly for  $\cap$  in place of  $-$ , but not for  $\cup$

12. The projection operation distributes over union

$$\Pi_L(E_1 \cup E_2) = (\Pi_L(E_1)) \cup (\Pi_L(E_2))$$


SWAYAM NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
39.16 ©Silberschatz, Korth and Sudarshan

So, this is another possible transformation that now finally, the set theoretic operations have their normal set rules so, union and intersection are commutative, naturally set difference is not commutative, union intersection are associative as well. The selection operation distributes over union, intersection and set difference and the projection also distributes over union. So, you can see the exceptions here you can reason that out.

(Refer Slide Time: 19:35)



## Exercise

■ Create equivalence rules involving

- The group by/aggregation operation
- Left outer join operation



SWAYAM NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
39.17 ©Silberschatz, Korth and Sudarshan

So, we have in short we have presented a set of different transformation rules by each which you can make equivalent expressions and between two equivalent expressions or

more equivalent expressions our objective will always be to choose the one whose evaluation plan will have a lesser cost. So, as an exercise I have left that you can create equivalence rules that involve group by or aggregation operations or different kinds of outer join, left outer join, right outer join and so, on so, please work those out.

(Refer Slide Time: 20:05)

**Transformation Example: Pushing Selections**

- Query: Find the names of all instructors in the Music department, along with the titles of the courses that they teach
  - $\Pi_{name, title}(\sigma_{dept\_name = "Music"}(instructor) \bowtie (\text{teaches} \bowtie \Pi_{course\_id, title}(course)))$
- Transformation using rule 7a
  - $\Pi_{name, title}((\sigma_{dept\_name = "Music"}(instructor)) \bowtie (\text{teaches} \bowtie \Pi_{course\_id, title}(course)))$
- Performing the selection as early as possible reduces the size of the relation to be joined

Diagram illustrating Rule 7a:

```

    graph LR
      E1 --> E2
      E1 --> S
      S --> E2
      S --> O
      O --> E2
      style S fill:none,stroke:none
      style O fill:none,stroke:none
  
```

The diagram shows two relations, E1 and E2, connected by a join symbol. A third node, S, is connected to both E1 and E2. A curved arrow labeled "Rule 7a" points from the original query structure to this diagram, indicating that applying the rule results in a smaller intermediate relation S.

Let us move on to a couple of examples so, here is a query here the relations from the university database, here is a query find the names of all instructors in the music department along with the titles of the course they teach. So, a while ago we saw this so, this is what the query is in the relational algebra form and if you use the transformation rule of select early the rule 7a, whose transformation I have just shown here then you should be able to hear what you are doing is you are first doing a join of teaches and the projection of course. And then joining instructor with that and finally, doing the selection, but you should you would be able to do this select early because it involves the attribute department name which is the attribute of instructor alone here.

So, you should be able to first do this selection, mind you here courses also show that there is an attribute department name, but actually the courses is being used after projection. So, after projection in the projected relation there is no department name. So, department name is involved only the instructor.

So, I can first I can take this do early, I can do this selection early and take this out and then do the final join operation. So, in that process possibly the this will reduce the size

of the relation to be joined significantly because you do not naturally expect to be to have too many instructors in the music department. So, the instructor relation after the selection would become much smaller.

(Refer Slide Time: 21:52)

**Example with Multiple Transformations**

■ Query: Find the names of all instructors in the Music department who have taught a course in 2009, along with the titles of the courses that they taught

- $\Pi_{name, title}(\sigma_{dept\_name = "Music"} \wedge year = 2009 (instructor \bowtie (teaches \bowtie \Pi_{course\_id, title} (course))))$

■ Transformation using join associatively (Rule 6a):

- $\Pi_{name, title}(\sigma_{dept\_name = "Music"} \wedge year = 2009 ((instructor \bowtie teaches) \bowtie \Pi_{course\_id, title} (course)))$

■ Second form provides an opportunity to apply the “perform selections early” rule, resulting in the subexpression

$$\sigma_{dept\_name = "Music"} (instructor) \bowtie \sigma_{year = 2009} (teaches)$$

course(course\_id, title, dept\_name, credits)  
 instructor(ID, name, dept\_name, salary)  
 teaches(ID, course\_id, sec\_id, semester, year)

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018  
 Database System Concepts - 8<sup>th</sup> Edition  
 39.19  
 ©Silberschatz, Korth and Sudarshan

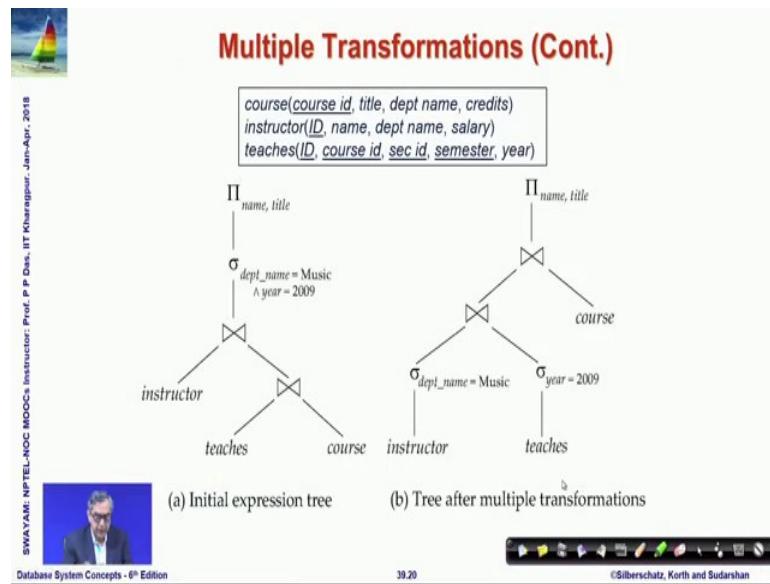
So, this is one example, this is another example query we are taking find the names of all instructors in the music department, we have taught a course in 2009 along with the titles of the course they taught. So, here is a the corresponding here is the corresponding relational query which you can convince yourself is indeed the same.

And then we can transform using the rule 6a which I have shown here which is basically the associativity of joint because there are two join relations and we are using the associativity of join to first join the here then second and third relations are joined first and then the first relation is joined with that. Here we are using associativity of joining the first and second and then using the third, now if you join first and second and then it is possible that the department name actually the department name is an instructor and the other condition is here which is in the teaches.

So, this department name is in the instruction here is in the teaches. So, I should be able to do select early, I should be able to select on the instructor based only on the department name being music and also select early on the teachers by using a year as 2009 and then do their joint.

So, naturally each one of these will become much smaller corresponding to the whole instructor or teaches relation therefore, their join will also be smaller. So, which means eventually this part this part of the query will become much smaller in size in the result and the consequent second join would be much smaller to perform. So, this will naturally give it whereas, if we had done all of these join earlier we would have used the whole of the teaches and the instructor relations and that would have been quite a lot of tuples would have been there.

(Refer Slide Time: 23:57)



So, these are different example so, this is the same example being shown in terms of the tree parts tree structure so, we can convince yourself by using the transformation rules that they are actually equivalent.

(Refer Slide Time: 24:12)

**Transformation Example: Pushing Projections**

- Consider:  $\Pi_{name, title}(\sigma_{dept\_name = 'Music'}(instructor) \bowtie \text{teaches}) \bowtie \Pi_{course\_id, title}(course)$
- When we compute  
 $(\sigma_{dept\_name = 'Music'}(instructor) \bowtie \text{teaches})$   
we obtain a relation whose schema is:  
 $(ID, name, dept\_name, salary, course\_id, sec\_id, semester, year)$
- Push projections using equivalence rules 8a and 8b; eliminate unneeded attributes from intermediate results to get:  
 $\Pi_{name, title}(\Pi_{name, course\_id}(\sigma_{dept\_name = 'Music'}(instructor) \bowtie \text{teaches})) \bowtie \Pi_{course\_id, title}(course)$
- Performing the projection as early as possible reduces the size of the relation to be joined

$$\begin{array}{l} \text{course(course\_id, title, dept\_name, credits)} \\ \text{instructor(ID, name, dept\_name, salary)} \\ \text{teaches(ID, course\_id, sec\_id, semester, year)} \end{array}$$

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie E_2) = (\Pi_{L_1}(E_1)) \bowtie (\Pi_{L_2}(E_2))$$

8a

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_2}(E_1)) \bowtie (\Pi_{L_1 \cup L_2}(E_2)))$$

8b

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

SWAYAM

Bibliothek, Kalkutta

And then certainly they give you a significant advantage and now this is an example which show that you can push the projection.

So, here is what you wanted to do and while we compute this we will get a relation of this form and we can push, we can use these rules rule 8a and 8b, this is rule 8a, this is rule 8b by this we can push the projections inside and make the relations smaller because now if you push the projection then you are actually cutting down on the on the number of columns. So, your relation becomes smaller that will make the with the projection this will reduce and when you project also there will be duplicates which will get removed so, in every way your relation becomes smaller in size and your subsequent join operations would be more efficient.

(Refer Slide Time: 25:02)

The slide features a sailboat icon in the top left corner. The title 'Join Ordering Example' is at the top right. A vertical sidebar on the left contains the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018'. The main content area contains two bullet points:

- For all relations  $r_1$ ,  $r_2$  and  $r_3$ ,  
$$(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$$

(Join Associativity)
- If  $r_2 \bowtie r_3$  is quite large and  $r_1 \bowtie r_2$  is small, we choose  
$$(r_1 \bowtie r_2) \bowtie r_3$$

so that we compute and store a smaller temporary relation

A video player interface is visible at the bottom, showing a thumbnail of a person speaking, the title 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018', the time '39:22', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

You can also see these are examples where you can take care of the fact that you can reorder joining to get better result for example, the you can if I have to do join three relations like this, I could do either this first or this first. Now if I do this first ( $r_1 \bowtie (r_2 \bowtie r_3)$ ) then this is a temporary relation which I will need to maintain in memory and then join with  $r_1$ . If I do this first then this will be a temporary relation and then I will join it with  $r_3 ((r_1 \bowtie r_2) \bowtie r_3)$ . So, if this is large enough then compared to this then I will be better off by doing this and will be able to have a more optimized execution of the query. So, here is an example of that again two joints.

(Refer Slide Time: 25:46)

The slide features a sailboat icon in the top left corner. The title 'Join Ordering Example (Cont.)' is at the top right. A vertical sidebar on the left contains the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018'. The main content area contains several bullet points:

- Consider the expression  
$$\Pi_{name, title}(\sigma_{dept\_name = "Music"}(instructor) \bowtie teaches) \bowtie \Pi_{course_id, title}(course))$$
- Could compute  $teaches \bowtie \Pi_{course_id, title}(course)$  first, and join result with  
$$\sigma_{dept\_name = "Music"}(instructor)$$
- but the result of the first join is likely to be a large relation
- Only a small fraction of the university's instructors are likely to be from the Music department
  - it is better to compute  
$$\sigma_{dept\_name = "Music"}(instructor) \bowtie teaches$$
 first

At the bottom, there is a box containing the following table definitions:

course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
teaches(ID, course_id, sec_id, semester, year)

A video player interface is visible at the bottom, showing a thumbnail of a person speaking, the title 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018', the time '39:23', and the copyright notice '©Silberschatz, Korth and Sudarshan'.

So, what we are trying to show is when we are having this instead of actually are actually trying to compute this join first because we expect that this set to be much smaller, if this set is much smaller then naturally this joint would be much smaller and it is more likely to fit into the memory then if we had just done teaches and course id's which I expected to be large relations.

(Refer Slide Time: 26:20)

**Enumeration of Equivalent Expressions**

- Query optimizers use equivalence rules to **systematically** generate expressions equivalent to the given expression
- Can generate all equivalent expressions as follows:
  - Repeat
    - apply all applicable equivalence rules on every subexpression of every equivalent expression found so far
    - add newly generated expressions to the set of equivalent expressions
  - Until no new equivalent expressions are generated above
- The above approach is very expensive in space and time
  - Two approaches
    - Optimized plan generation based on transformation rules
    - Special case approach for queries with only selections, projections and joins

SWAYAM-NPTEL-NOC Instructor: Prof. P P Desai, IIT Kharagpur - Jan-Apr, 2018  
Database System Concepts - 8<sup>th</sup> Edition  
39.24 ©Silberschatz, Korth and Sudarshan

So, basically therefore, the strategy turns out that given a query you will have to generate it whole lot of equivalent expressions. So, the number of equivalent expressions could be really really large. So, we will have to systematically generate all these alternate equivalent expressions and apply by applying these transformation rules that we have just seen and we will have to continue till no new expression can be generated and then we have to evaluate each one of them based on their evaluation plan.

So, this could be very expensive because the number of alternates could be really really large. So, the optimize plan generation is also based on the transformation rules and we may have only different special kits approaches to take care of the common optimization plans that we might have.

(Refer Slide Time: 27:16)

**Implementing Transformation Based Optimization**

■ Space requirements reduced by sharing common sub-expressions:

- when E1 is generated from E2 by an equivalence rule, usually only the top level of the two are different, subtrees below are the same and can be shared using pointers
  - E.g. when applying join commutativity

■ Same sub-expression may get generated multiple times

- Detect duplicate sub-expressions and share one copy

■ Time requirements are reduced by not generating all expressions

- Dynamic programming

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018  
Database System Concepts - 8<sup>th</sup> Edition  
39.25  
©Silberschatz, Korth and Sudarshan

Now, one way to do that is for example, if we are looking at say the join of two relations E1 and E2 here then; obviously, if we do certain transformations with this join that does not change the way E1 and E2 are actually evaluated.

So, if that be the case then in terms of generating the alternate we do not really need to keep or evaluate all of the expressions the whole of the expression in every alternate. We could actually do something like sorry we could actually do something like we could have a join and then instead of really replicating the whole of the evaluation of E1 and evaluation of E2, we can simply make pointers to the same sub trees which are called basically sub expression optimization.

If you have studied the expression optimization in compiler at any point of time you will understand this very well, this is the same strategy which is used here. So, if you can detect duplicate sub expressions then you can have only one copy and you can make the things more efficient to run.

So, the general strategy for doing this is a dynamic programming we would not be able to cover that in the current course, but just know that all these explosion of generating alternate and choosing the best one is usually handled in terms of dynamic programming which is a strategy to make sure that if we have solved a sub earlier then I do not need to solve that sub problem again we can just reuse that same earlier result and go ahead with that.

So, by this way we can common sub expressions we may only find the plan for a best plan for a common sub expression only once and then use it subsequently again.

(Refer Slide Time: 29:24)

The slide is titled "Module Summary" in red at the top right. On the left, there is a small image of a sailboat on water. The main content area contains a bulleted list of three items:

- Understood the basic issues for optimizing queries
- For every relational expression, usually there are a number of equivalent expressions that can be created by simple transformations
- Final execution plan can be created by choose the estimated least cost expression from the alternates

On the left margin, vertical text reads: "SWAYAM: NITTEL-NOCOCS Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018". At the bottom left, there is a small video thumbnail showing a person speaking. The bottom right corner features the copyright notice "©Silberschatz, Korth and Sudarshan". The footer also includes the course title "Database System Concepts - 8<sup>th</sup> Edition" and the slide number "39.26".

So, in this module covered starting from the notions of query processing in the earlier module, we have discussed the basic issues of optimizing queries and we have shown that using a set of simple transformational rules you can convert a relational expression into a number of equivalent relational expressions. And then you can evaluate them based on the estimated cost that the model that you are using and choose the best one and dynamic programming is usually a good way of doing that.

So, in through the previous module and this one we have taken a very elementary look I should say, but this will give you some idea of how actually an SQL query is transformed into relational algebra parsed and translated into relational algebra and how equivalent expressions for that relational algebra expression is generated and evaluated.

And finally, an evaluation plan is made where specific choice is made for the different algorithms for doing different operations and that final plan which is which has the best cost is passed on to the query processing engine to query evaluation engine. And that then goes forward and does the B tree and disk operations in according to the plan and produces the best result in possibly the best shot is possible time period.

**Database Management System**  
**Prof. Partha Pratim Das**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 40**  
**Course Summarization**

---

Welcome to module 40 of Database Management Systems. This is for course summarization this is the last module of the course.

(Refer Slide Time: 00:23)

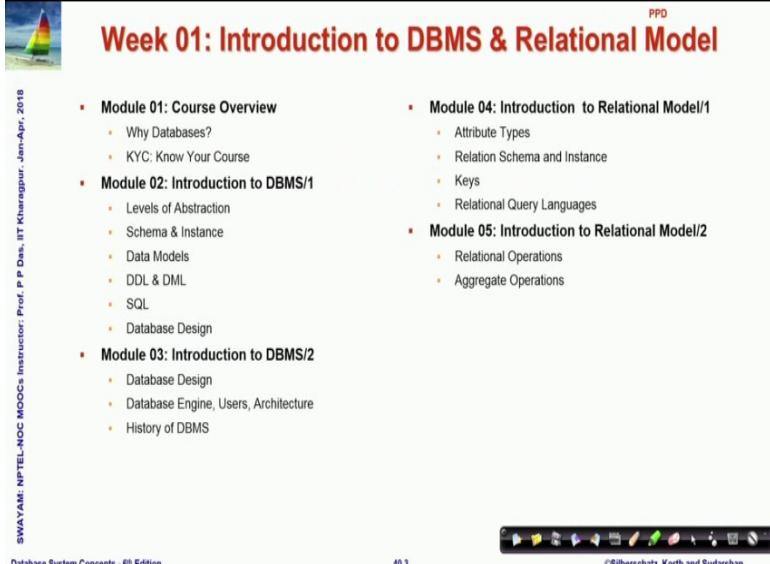
The slide is titled "Course Recap" in red text at the top right. In the top left corner, there is a small image of a sailboat on water. The footer contains several lines of text and icons:

- SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr, 2018
- Database System Concepts - 8<sup>th</sup> Edition
- 40.2
- ©Silberschatz, Korth and Sudarshan

At the bottom, there is a horizontal bar with various presentation control icons (e.g., back, forward, search).

So, I would just start with doing a quick recap of what all did we cover and what we expectedly learnt.

(Refer Slide Time: 00:32)



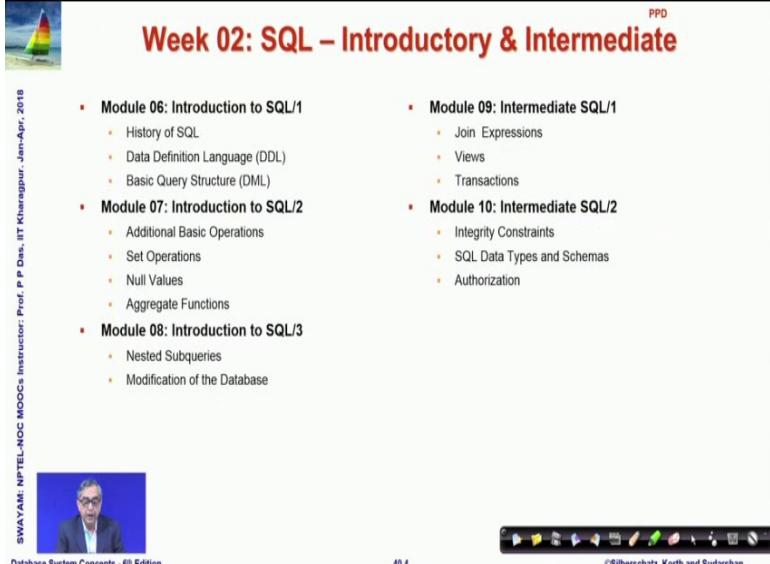
The slide is titled "Week 01: Introduction to DBMS & Relational Model". It features a sailboat icon in the top left corner and a decorative border at the bottom. The content is organized into two columns of bullet points:

Module	Topics
Module 01: Course Overview	Why Databases? KYC: Know Your Course
Module 02: Introduction to DBMS/1	Levels of Abstraction Schema & Instance Data Models DDL & DML SQL Database Design
Module 03: Introduction to DBMS/2	Database Design Database Engine, Users, Architecture History of DBMS
Module 04: Introduction to Relational Model/1	Attribute Types Relation Schema and Instance Keys Relational Query Languages
Module 05: Introduction to Relational Model/2	Relational Operations Aggregate Operations

Small text on the left edge reads: SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018. The bottom right corner shows the copyright notice: ©Silberschatz, Korth and Sudarshan.

In the week 1, we talked primarily of Introduction to Database Management System and the Relational Model which is the foundation of a database system.

(Refer Slide Time: 00:42)



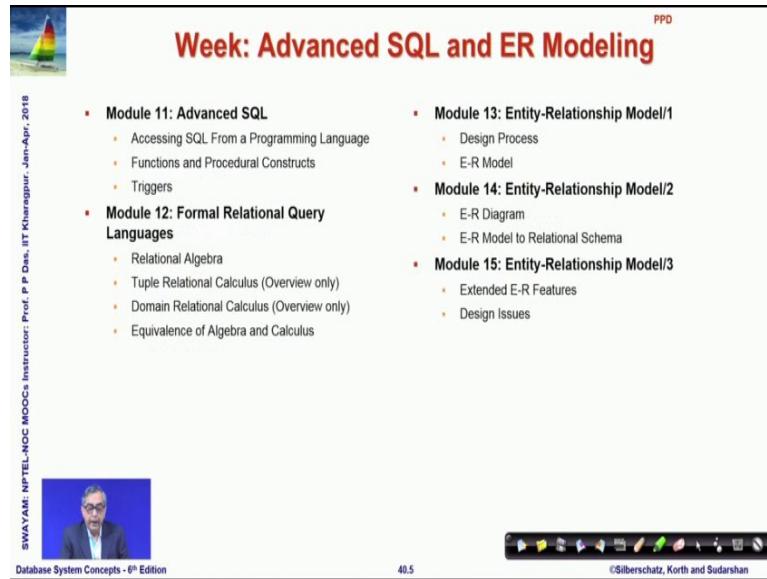
The slide is titled "Week 02: SQL – Introductory & Intermediate". It features a sailboat icon in the top left corner and a video thumbnail of a professor in the bottom left corner. The content is organized into two columns of bullet points:

Module	Topics
Module 06: Introduction to SQL/1	History of SQL Data Definition Language (DDL) Basic Query Structure (DML)
Module 07: Introduction to SQL/2	Additional Basic Operations Set Operations Null Values Aggregate Functions
Module 08: Introduction to SQL/3	Nested Subqueries Modification of the Database
Module 09: Intermediate SQL/1	Join Expressions Views Transactions
Module 10: Intermediate SQL/2	Integrity Constraints SQL Data Types and Schemas Authorization

Small text on the left edge reads: SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018. The bottom right corner shows the copyright notice: ©Silberschatz, Korth and Sudarshan.

In week 2, we started off with query language SQL at an Introductory level and then at an Intermediate level which are really really the first major aspect of a database particularly relational database that a student must master.

(Refer Slide Time: 01:01)



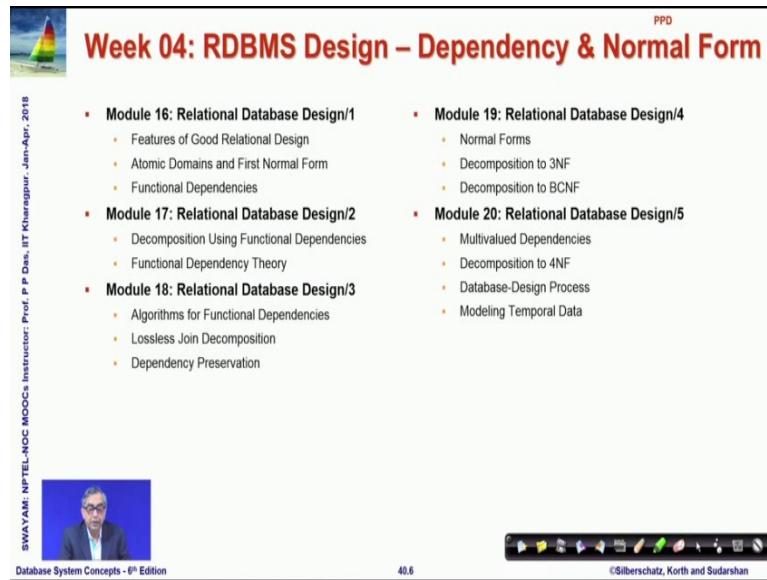
The slide title is "Week: Advanced SQL and ER Modeling". It features a small sailboat icon in the top left corner and a decorative bar at the bottom. The content is organized into two columns of bullet points:

<ul style="list-style-type: none"><li>▪ <b>Module 11: Advanced SQL</b><ul style="list-style-type: none"><li>◦ Accessing SQL From a Programming Language</li><li>◦ Functions and Procedural Constructs</li><li>◦ Triggers</li></ul></li><li>▪ <b>Module 12: Formal Relational Query Languages</b><ul style="list-style-type: none"><li>◦ Relational Algebra</li><li>◦ Tuple Relational Calculus (Overview only)</li><li>◦ Domain Relational Calculus (Overview only)</li><li>◦ Equivalence of Algebra and Calculus</li></ul></li></ul>	<ul style="list-style-type: none"><li>▪ <b>Module 13: Entity-Relationship Model/1</b><ul style="list-style-type: none"><li>◦ Design Process</li><li>◦ E-R Model</li></ul></li><li>▪ <b>Module 14: Entity-Relationship Model/2</b><ul style="list-style-type: none"><li>◦ E-R Diagram</li><li>◦ E-R Model to Relational Schema</li></ul></li><li>▪ <b>Module 15: Entity-Relationship Model/3</b><ul style="list-style-type: none"><li>◦ Extended E-R Features</li><li>◦ Design Issues</li></ul></li></ul>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

On the left side, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018". At the bottom left is a video thumbnail of the instructor, and at the bottom right are navigation icons and the text "Database System Concepts - 8<sup>th</sup> Edition" and "40.5". The bottom right also includes the copyright notice "©Silberschatz, Korth and Sudarshan".

In week 3, we continued with the advanced SQL and did the aspects of modeling from specification in terms of Entity Relationship Model.

(Refer Slide Time: 01:12)



The slide title is "Week 04: RDBMS Design – Dependency & Normal Form". It features a small sailboat icon in the top left corner and a decorative bar at the bottom. The content is organized into two columns of bullet points:

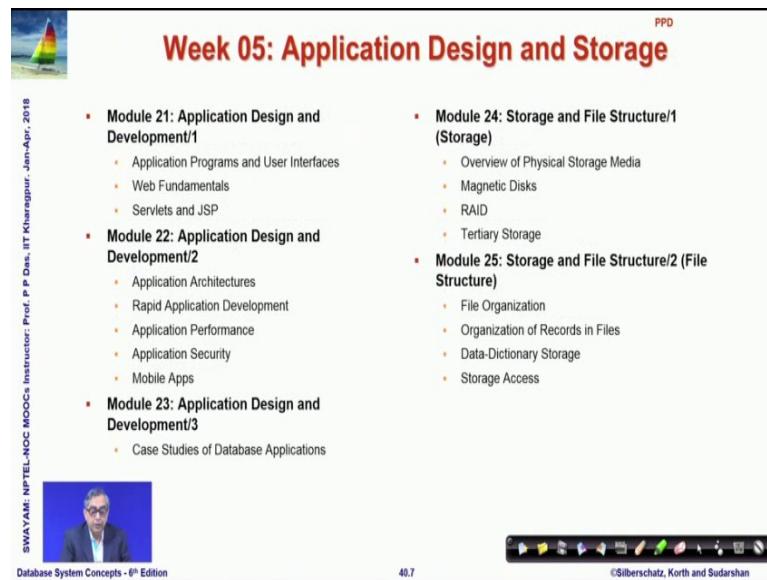
<ul style="list-style-type: none"><li>▪ <b>Module 16: Relational Database Design/1</b><ul style="list-style-type: none"><li>◦ Features of Good Relational Design</li><li>◦ Atomic Domains and First Normal Form</li><li>◦ Functional Dependencies</li></ul></li><li>▪ <b>Module 17: Relational Database Design/2</b><ul style="list-style-type: none"><li>◦ Decomposition Using Functional Dependencies</li><li>◦ Functional Dependency Theory</li></ul></li><li>▪ <b>Module 18: Relational Database Design/3</b><ul style="list-style-type: none"><li>◦ Algorithms for Functional Dependencies</li><li>◦ Lossless Join Decomposition</li><li>◦ Dependency Preservation</li></ul></li></ul>	<ul style="list-style-type: none"><li>▪ <b>Module 19: Relational Database Design/4</b><ul style="list-style-type: none"><li>◦ Normal Forms</li><li>◦ Decomposition to 3NF</li><li>◦ Decomposition to BCNF</li></ul></li><li>▪ <b>Module 20: Relational Database Design/5</b><ul style="list-style-type: none"><li>◦ Multivalued Dependencies</li><li>◦ Decomposition to 4NF</li><li>◦ Database-Design Process</li><li>◦ Modeling Temporal Data</li></ul></li></ul>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

On the left side, there is vertical text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018". At the bottom left is a video thumbnail of the instructor, and at the bottom right are navigation icons and the text "Database System Concepts - 8<sup>th</sup> Edition" and "40.6". The bottom right also includes the copyright notice "©Silberschatz, Korth and Sudarshan".

In week the next week, we did the design issues which was really the involved part and possibly the most important aspect of the relational database design beyond query coding query being able to write queries.

So, this is based on dependency and different normal forms and I am sure you have spent a good time on mastering these.

(Refer Slide Time: 01:40)



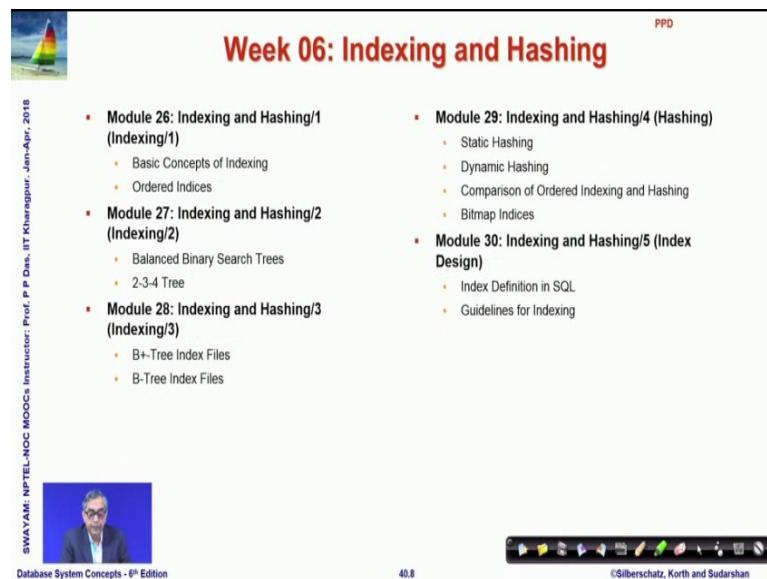
The slide is titled "Week 05: Application Design and Storage". It features a sailboat icon in the top left corner and a "PPD" logo in the top right. The main content is organized into four sections:

- Module 21: Application Design and Development/1**
  - Application Programs and User Interfaces
  - Web Fundamentals
  - Servlets and JSP
- Module 22: Application Design and Development/2**
  - Application Architectures
  - Rapid Application Development
  - Application Performance
  - Application Security
  - Mobile Apps
- Module 23: Application Design and Development/3**
  - Case Studies of Database Applications
- Module 24: Storage and File Structure/1 (Storage)**
  - Overview of Physical Storage Media
  - Magnetic Disks
  - RAID
  - Tertiary Storage
- Module 25: Storage and File Structure/2 (File Structure)**
  - File Organization
  - Organization of Records in Files
  - Data-Dictionary Storage
  - Storage Access

On the left side, there is vertical text: "SWAYAM NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018". In the center, there is a small video thumbnail of a man speaking. At the bottom, it says "Database System Concepts - 6<sup>th</sup> Edition" and "40.7". On the right, it says "©Silberschatz, Korth and Sudarshan".

We followed up in week 5 with application design and discussing aspects of storage structure, how will actually the items, data items be stored in the different memory and disk structure.

(Refer Slide Time: 01:56)



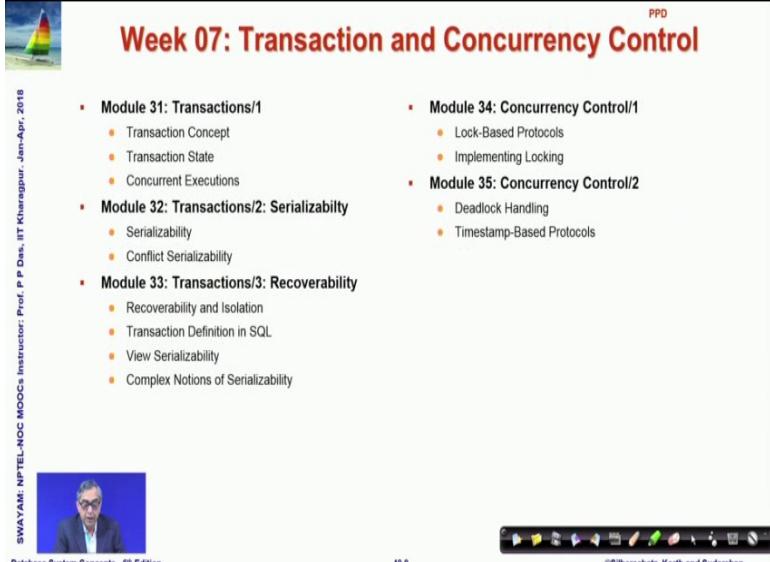
The slide is titled "Week 06: Indexing and Hashing". It features a sailboat icon in the top left corner and a "PPD" logo in the top right. The main content is organized into five sections:

- Module 26: Indexing and Hashing/1 (Indexing/1)**
  - Basic Concepts of Indexing
  - Ordered Indices
- Module 27: Indexing and Hashing/2 (Indexing/2)**
  - Balanced Binary Search Trees
  - 2-3-4 Tree
- Module 28: Indexing and Hashing/3 (Indexing/3)**
  - B+-Tree Index Files
  - B-Tree Index Files
- Module 29: Indexing and Hashing/4 (Hashing)**
  - Static Hashing
  - Dynamic Hashing
  - Comparison of Ordered Indexing and Hashing
  - Bitmap Indices
- Module 30: Indexing and Hashing/5 (Index Design)**
  - Index Definition in SQL
  - Guidelines for Indexing

On the left side, there is vertical text: "SWAYAM NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018". In the center, there is a small video thumbnail of a man speaking. At the bottom, it says "Database System Concepts - 6<sup>th</sup> Edition" and "40.8". On the right, it says "©Silberschatz, Korth and Sudarshan".

In the following week, in week 6, we discussed about indexing and hashing to for making the accesses really efficient.

(Refer Slide Time: 02:05)



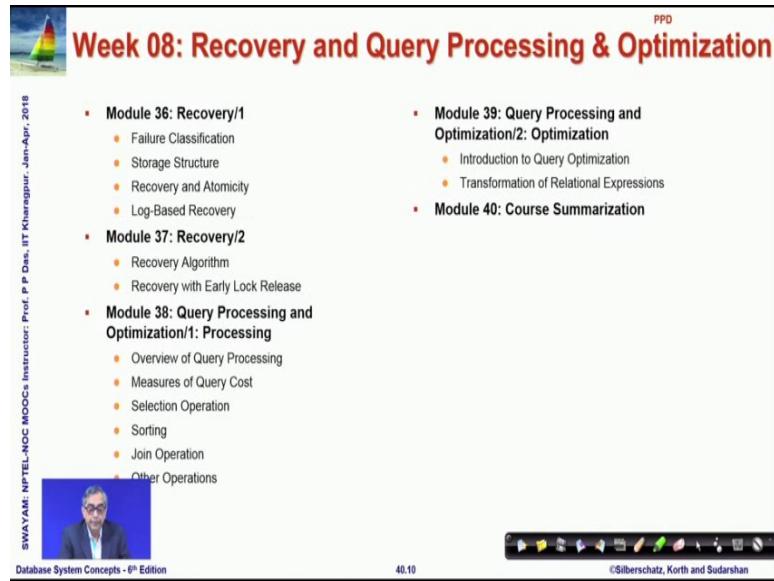
The slide is titled "Week 07: Transaction and Concurrency Control" in red at the top right. It features a small sailboat icon in the top left corner. The content is organized into two columns of bullet points:

<ul style="list-style-type: none"><li>▪ <b>Module 31: Transactions/1</b><ul style="list-style-type: none"><li>○ Transaction Concept</li><li>○ Transaction State</li><li>○ Concurrent Executions</li></ul></li><li>▪ <b>Module 32: Transactions/2: Serializability</b><ul style="list-style-type: none"><li>○ Serializability</li><li>○ Conflict Serializability</li></ul></li><li>▪ <b>Module 33: Transactions/3: Recoverability</b><ul style="list-style-type: none"><li>○ Recoverability and Isolation</li><li>○ Transaction Definition in SQL</li><li>○ View Serializability</li><li>○ Complex Notions of Serializability</li></ul></li></ul>	<ul style="list-style-type: none"><li>▪ <b>Module 34: Concurrency Control/1</b><ul style="list-style-type: none"><li>○ Lock-Based Protocols</li><li>○ Implementing Locking</li></ul></li><li>▪ <b>Module 35: Concurrency Control/2</b><ul style="list-style-type: none"><li>○ Deadlock Handling</li><li>○ Timestamp-Based Protocols</li></ul></li></ul>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

At the bottom left, there is a small video thumbnail showing a man speaking. The footer contains the text "SWAYAM, NPTEL-NOC, MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018", "Database System Concepts - 8<sup>th</sup> Edition", "40.9", and "©Silberschatz, Korth and Sudarshan".

In week 7, we did another critical aspect of database systems that is how to make transactions work concurrently. So, we defined transactions and define what is Concurrency and then we took in two different critical aspects of Serializability that it is possible that we can execute transactions in a manner so that their instructions are intermixed, but then even in that case, they actually produce a result which is as if these transactions could have been executed in the serial order and we talked about the issues of recoverability in this respect and we specifically looked at different protocols, particularly two phase locking protocol for managing this kind of concurrency and the evils of deadlock that may happen when you do it concurrency and how simple protocols like time based protocol can handle that.

(Refer Slide Time: 03:03)



The slide is titled "Week 08: Recovery and Query Processing & Optimization". It features a sailboat icon in the top left corner and the letters "PPD" in the top right corner. The main content is organized into two columns of bullet points:

- Module 36: Recovery/1
  - Failure Classification
  - Storage Structure
  - Recovery and Atomicity
  - Log-Based Recovery
- Module 37: Recovery/2
  - Recovery Algorithm
  - Recovery with Early Lock Release
- Module 38: Query Processing and Optimization/1: Processing
  - Overview of Query Processing
  - Measures of Query Cost
  - Selection Operation
  - Sorting
  - Join Operation
  - Other Operations
- Module 39: Query Processing and Optimization/2: Optimization
  - Introduction to Query Optimization
  - Transformation of Relational Expressions
- Module 40: Course Summarization

At the bottom left, there is a small video thumbnail showing a man speaking, with the text "SWAYAM NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kharagpur - Jan-Apr. 2018". At the bottom center, it says "Database System Concepts - 8<sup>th</sup> Edition" and "40.10". At the bottom right, it says "©Silberschatz, Korth and Sudarshan".

And in the current week, we have dwelt with different strategies of recovery, particularly log based recovery and we have touched upon query processing and optimization.

So, this is you have got a very first level overview of this course. This is in a limited time and with limited number of assignments. So, you will just get a first level idea, this is not to make you really an expert of database systems, but this will certainly get you started well in terms of the database management programs, in terms of you are taking up advanced courses later on or in terms of actually taking up a job in different database area.

(Refer Slide Time: 03:48)

The slide is titled "Module Objectives" in red at the top right. It features a small sailboat icon in the top left corner. On the left edge, there is vertical text: "SWAYAM, NPTEL-NOC MOOCs", "Instructor: Prof. P.P. Desai, IIT Kanpur", and "Jan-Apr., 2018". The main content area contains a bulleted list of objectives:

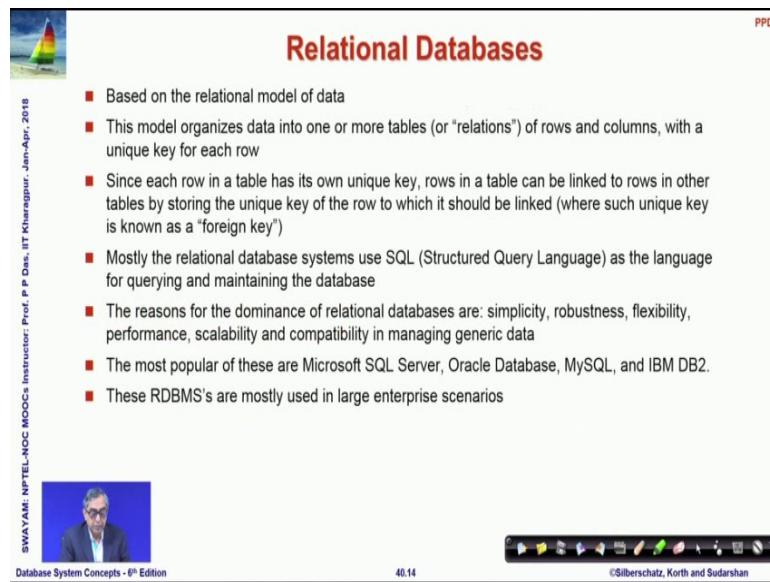
- The space of RDBMSs is crowded. We take a look into common RDBMS systems
- Non-Relational database systems are starting to dominate emerging applications. We present a brief overview
- What is the road forward? We outline likely job profiles in terms of a skills-profiles matrix and the companies to work for

At the bottom left is a small video thumbnail showing a person speaking. The bottom right shows the slide number "40.11" and copyright information "©Silberschatz, Korth and Sudarshan". A decorative toolbar is visible at the very bottom.

So, given that in this current module, I would discuss about few things beyond the textbook actually, the space of databases RDBMS bases are quite crowded, the lot of RDBMS bases you will see. So, I will just take a quick look in terms of the common RDBMS systems and there had been a number of queries on the forum and in the live session about that. I will also touch upon what we would like to discuss about non relational database systems which was not a part of the curriculum that we did here, but will just present a brief overview.

And then finally, I would like to conclude with what should be the road forward from you, presenting you a kind of a skill profile matrix so that what skills you must pick up to actually get a job off certain profile and what are the companies that you might look at working for.

(Refer Slide Time: 04:55)



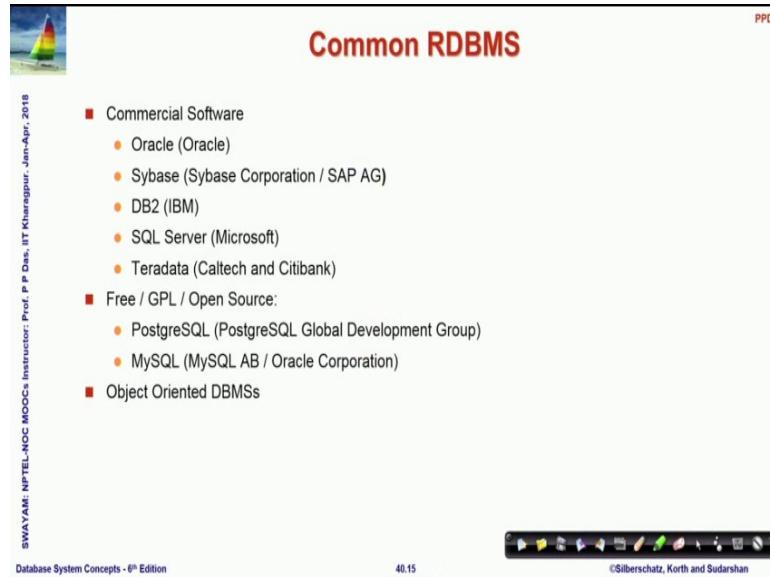
The slide is titled "Relational Databases" in red at the top right. It features a small sailboat icon in the top left corner. A vertical sidebar on the left contains the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018". The main content area lists several bullet points about relational databases:

- Based on the relational model of data
- This model organizes data into one or more tables (or "relations") of rows and columns, with a unique key for each row
- Since each row in a table has its own unique key, rows in a table can be linked to rows in other tables by storing the unique key of the row to which it should be linked (where such unique key is known as a "foreign key")
- Mostly the relational database systems use SQL (Structured Query Language) as the language for querying and maintaining the database
- The reasons for the dominance of relational databases are: simplicity, robustness, flexibility, performance, scalability and compatibility in managing generic data
- The most popular of these are Microsoft SQL Server, Oracle Database, MySQL, and IBM DB2.
- These RDBMS's are mostly used in large enterprise scenarios

At the bottom left is a small portrait of a man, and the bottom right shows a navigation bar with icons and the text "Database System Concepts - 8<sup>th</sup> Edition", "40.14", and "©Silberschatz, Korth and Sudarshan".

So, starting with the common databases, there are several relational database systems. So, in this slide you summarize basically what are the different aspects of relational database systems which you have been discussing so far. So, this is just a summary of that.

(Refer Slide Time: 05:07)



The slide is titled "Common RDBMS" in red at the top right. It features a small sailboat icon in the top left corner. A vertical sidebar on the left contains the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018". The main content area lists categories of database systems:

- Commercial Software
  - Oracle (Oracle)
  - Sybase (Sybase Corporation / SAP AG)
  - DB2 (IBM)
  - SQL Server (Microsoft)
  - Teradata (Caltech and Citibank)
- Free / GPL / Open Source:
  - PostgreSQL (PostgreSQL Global Development Group)
  - MySQL (MySQL AB / Oracle Corporation)
- Object Oriented DBMSs

At the bottom left is a small portrait of a man, and the bottom right shows a navigation bar with icons and the text "Database System Concepts - 8<sup>th</sup> Edition", "40.15", and "©Silberschatz, Korth and Sudarshan".

Now, these are the common database systems. So, I have chosen the ones which are most widely used, most easily accessible and kind of large companies use them, large databases exist on them. So, there are primarily 2 classifications; one is a set of database

systems are commercial Oracle from the Oracle corporation, Sybase from Sybase corporation which is now SAP AG, DB2 from IBM, SQL Server from Microsoft and the recent entrant to that who is making a regular ripples is Teradata which is a you know joint database systems from Caltech and certain group of Citibank. So, if you are working for a company who subscribes to any of these database software, then you should be able to use them and understand what all you can do, but if you are working with smaller companies or you are working as a student, then you will need to use some of the database systems which are free or are on the GPL licensing or open source.

So, most prominent amongst them is PostgreSQL which is from a Postgres global development group. So, these are non commercial the software in the sense that you do not need to pay for them and they are on the GPL and some of some part of that would be open source as well and a very commonly used is MySQL which was originally from a Swedish company called MySQL AB, but now it is acquired by Oracle corporation, but it still does not you do not need to pay for that. So, these are the databases and systems to primarily look for and besides that there are some other database systems which use certain object oriented features on top of the relational features.

So, if you look through these, then in most cases you will find in terms of the gross functionality of the kind of SQL that you can write, a large subset of the SQL that you can write on databases maintained through these database systems will be same. So, what you have learnt here would be applicable irrespective of which database which of these database systems you are using, but of course there are specifics which would be different amongst them. So, in the next couple of slides, I have for on each one slide, I have given a brief background about the particular database system.

(Refer Slide Time: 07:48)

The slide has a header 'Oracle' and a small sailboat icon. The content lists features of Oracle, including its history, latest version, languages, and tools. The footer includes the source 'Database System Concepts - 8<sup>th</sup> Edition', page number 40.16, and copyright information.

■ Multi-model commercial database management system produced and marketed by **Oracle Corporation**.  
■ Larry Ellison, Bob Miner and Ed Oates started a consultancy called Software Development Laboratories (SDL) in 1977, and developed the original version of Oracle.  
■ Latest Version: **Oracle Database 12c Release 2: 12.2.0.1 (patchset as of March 2017)**  
■ Used for running online transaction processing (OLTP), data warehousing (DW) and mixed (OLTP & DW) database workloads  
■ Languages: Structured Query language (SQL), Procedural SQL (PL- SQL)  
■ Tools/ Editions: Oracle SQL Developer, Oracle Forms, Oracle JDeveloper, Oracle Reports for development of applications, Oracle Live SQL for test environment  
■ Oracle can be accessed from Java through JDBC, Microsoft.NET through ODP.NET, C, C++ through OCI, ODBC, ODPI-C, Python through cx\_Oracle

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018

PPD

Database System Concepts - 8<sup>th</sup> Edition 40.16 ©Silberschatz, Korth and Sudarshan

So that you know you know how stable, how old or you know what are the basic nuances of that database system for example, Oracle started in 77. So, you can say, it is a it is a 40 year old database system. The latest version is 12 C and these are the different supports that it has.

(Refer Slide Time: 08:13)

The slide has a header 'Sybase' and a small sailboat icon. The content lists features of Sybase, including its history, latest version, languages, and tools. The footer includes the source 'Database System Concepts - 8<sup>th</sup> Edition', page number 40.17, and copyright information.

■ Relational model database server product for businesses developed by **Sybase Corporation** which became part of SAP AG.  
■ Originally for unix platforms in 1987, Sybase Corporation's primary DBMS product was initially marketed under the name Sybase SQL Server.  
■ Latest Version: **Sybase 16, released on 2014**  
■ Languages: Sybase IQ, Transact-SQL  
■ Tools/ Editions: Sybase SQL server for development of applications. Has a developer and express edition.  
■ Sybase can be accessed from C, C++ through SQLAPI++, Java through JDBC

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018

PPD

Database System Concepts - 8<sup>th</sup> Edition 40.17 ©Silberschatz, Korth and Sudarshan

Sybase also started in 1987. So, that is almost 30 years, but it is less you know less vibrant right now, the last stable released happen in 2014 about nearly more than 3 and a

half or 4 years ago and, but Sybase is a has been a very good database systems for programming through API's and it has really good support for that.

(Refer Slide Time: 08:42)

DB2

■ Db2 contains database-server products developed by **IBM**. Mostly relational models, but now includes object relational models  
■ In 1970, Edgar F.Codd, researcher in IBM published the model for data manipulation.  
■ Latest Version: **DB2 LUW 11.1 released on 2016**  
■ Used for running online transaction processing (OLTP), data warehousing (DW) and mixed (OLTP & DW) database workloads  
■ Languages: Structured Query language (SQL), XML Query  
■ Tools/ Editions: Advanced Enterprise Server Edition, Enterprise Server Edition, Advanced Workgroup Server Edition, Workgroup Server Edition, Direct and Developer Editions and Express-C.  
■ Db2 can be accessed from C, C++, Java, Ruby, Perl through a package of DB2 API's

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr, 2018

Database System Concepts - 8<sup>th</sup> Edition

40.18

©Silberschatz, Korth and Sudarshan

DB 2 is also a very old possibly the oldest surviving database systems which started in 1970. So, when almost the E. F Codd of the Boyce Codd normal form published the data manipulation schemes from IBM. So, this is also a widely used, last release 2016.

(Refer Slide Time: 09:07)

SQL Server

■ Relational database management system developed by **Microsoft**.  
■ SQL Server 1.0, a 16-bit server for the OS/2 operating system in 1989  
■ Latest Version: **SQL Server 2017**  
■ Used for running online transaction processing (OLTP) and online analytical processing (OLAP)  
■ Languages: Transact SQL  
■ Tools/ Editions: Enterprise, Standard, Web, Business Intelligence, WorkGroup, Express  
■ SQL server can be accessed from Java through JDBC, C, C++ through ODBC

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr, 2018

Database System Concepts - 8<sup>th</sup> Edition

40.19

©Silberschatz, Korth and Sudarshan

Microsoft started database systems in 1989, last release happened last year and that is very widely used if you are particularly on windows system, it is one of the most popular one in terms of the windows operating system.

(Refer Slide Time: 09:24)

■ Relational database management system developed by Caltech and Citibank's advanced technology group  
■ In 1984, the first version of Teradata was released  
■ Latest Version: **Teradata 15**  
■ Used for running online transaction processing (OLTP), data warehousing (DW) and mixed (OLTP & DW) database workloads  
■ Languages: BTEQ(Basic Teradata query)  
■ Tools/ Editions: Developer Edition, Express Edition  
■ SQL server can be accessed from Java through JDBC, C, C++ through ODBC

Teradata is relatively new. It was released in 1984 and it is, but it is one where lot of new developments are still happening and new experiments keep on happening and the current version is a Teradata 15.

(Refer Slide Time: 09:43)

■ Open source relational database management system produced by PostgreSQL Global Development Group, a diverse group of many companies and individual contributors.  
■ First version in 1986 by researchers of POSTGRES project  
■ Latest Version: **PostgreSQL 10.3 released on 2018**  
■ Used for running online transaction processing (OLTP), data warehousing (DW) and mixed (OLTP & DW) database workloads, Supports big data analytics  
■ Languages: Structured Query language (SQL), Procedural SQL (PL- SQL)  
■ Oracle can be accessed from Java through JDBC, Microsoft.NET through npgsql, C, C++ through libpq,

And in terms of the ma free or open source GPL databases Postgres started in 1988 about 30 years back and as I said, this is this is has a release even half of this year.

(Refer Slide Time: 10:04)

**MySQL**

- Open source relational database management system produced by Swedish company MySQL AB, now owned by Oracle Corporation
- First internal release on 23 May 1995
- Latest Version: MySQL 8.0.4 released on 2018
- Used for running online transaction processing (OLTP), data warehousing (DW) and mixed (OLTP & DW) database workloads
- Languages: Structured Query language (SQL), Procedural SQL (PL-SQL)
- Oracle can be accessed from Java through JDBC, Microsoft.NET through ADO.NET, C, C++ through ODBC

SWAYAM-NIETL-NOC-MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr, 2018

Database System Concepts - 8<sup>th</sup> Edition

40.22

©Silberschatz, Korth and Sudarshan

So, it is a very vibrant system. MySQL probably most widely used amongst the free community among the open source community also where the first internal release happened in 1995. The recent releases happened this year. So, these are the common database systems that you will come across. So, I mean given the organization that you are working with, first find out which database system it uses and then look into the specific manual for that and specific features.

(Refer Slide Time: 10:34)

**Object-oriented DBMS (OODBMSs)**

- Combines database capabilities with object oriented programming language capabilities
- OODBMSs allow object-oriented programmers to develop the product, store them as objects, and replicate or modify existing objects to make new objects within the OODBMS
- Objects have a many to many relationship and are accessed by the use of pointers.
- Access to data can be faster because an object can be retrieved directly without a search, by following pointers.
- Most object databases also offer some kind of query language, allowing objects to be found using a declarative programming approach.
- Examples:
  - Objectivity/DB,
  - O2,
  - Object Store

Ref: [https://en.wikipedia.org/wiki/Object\\_database](https://en.wikipedia.org/wiki/Object_database)

SWAYAM, NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

40.23 ©Silberschatz, Korth and Sudarshan

Beyond these a Relational Database Systems also, there are certain database systems which use Object oriented notions in that. So, if you are familiar with object orientation then you would have understood that the relational approach does not make keep things object oriented because you are always flattening out in terms of attributes and you are trying to look at the attributes, but it went for example, when you want to model the same thing in terms of a C++ or java program, you would like to look at a course as an as an object, as a class you would like to look at instructor as a class, you would like to look at teaches as a as a kind of class and their instances. So, there has been attempts to make give a object orientation kind of layer on top of relational databases or define things in that way. Objectivity DBO 2 objects store are some of the examples, but unfortunately this is have not been as popular as a regular relational databases.

So, if you happen to use any one of them, then you should be cause a cautious that you know you really know why you are using it and you would be able to go a long way in terms of that.

(Refer Slide Time: 11:53)

The slide is titled "Parameters" in red at the top right. On the left, there is a small image of a sailboat on water. The background is white. At the bottom, there is a decorative footer bar with various icons.

**Parameters**

- We compare the RDBMSs based on the following parameters:
  - OS support
  - Fundamental features
  - Limits
  - Tables and views
  - Indexes
  - Database capabilities
  - Data types
  - Other objects
  - Partitioning
  - Access control
  - Programming Language Support

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Desai, IIT Kanpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

40.24

©Silberschatz, Korth and Sudarshan

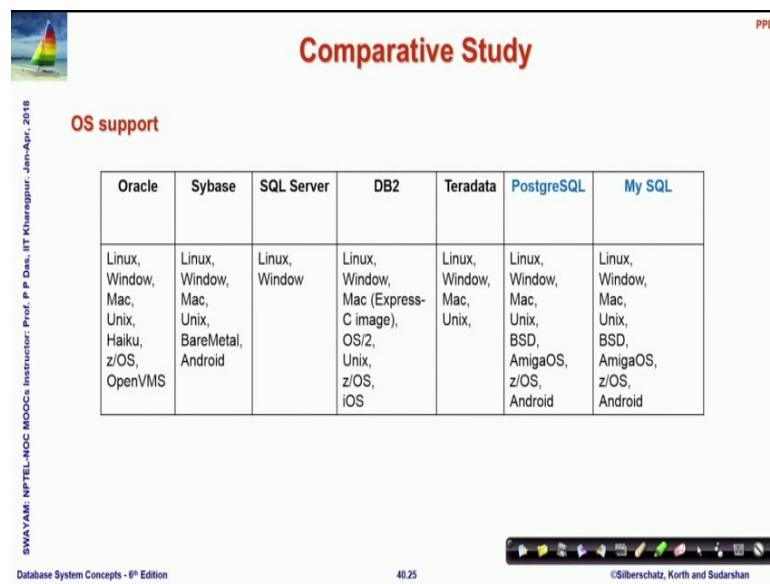
Now, when you come across a particular system that your company or your university needs to use and you would like to choose, then it will be good to look at the different aspects of that system. These are here are some of the parameters on which these database systems vary in a minimal to a very large extent, in terms of what operating systems it supports, what are the fundamental features, what are the limits for example, every database sets a number of limits in terms of the index size, in terms of the table size and whole lot of that.

How are the tables and views created what the kind of restrictions you have that, what kind of indexes has support the capabilities the data types, the different databases support. We have talked about a very limited set of data types in terms of SQL, but in an actual commercial or even you know open source database, the data types could be wider than that what kind of other objects partitioning access control mechanism. Access control is very important for ensuring security and finally, what kind of programming language support do you have.

Because as we have seen in the application development module, that it is not enough to just have a you know the database firing SQL queries, no application user will actually fire SQL queries. The application user needs and in GUI possibly or a text interface through which it will put queries in a different form and that needs to be processed by taking it to the database engine. So, you need possibly an interface which is in terms of

C, C++, Java, Python, this kind of programming language. So, how do you connect to or embed such embed your relational query into different languages that differ between different database systems. So, these are the parameters that you must look at. In the next series of slides, which I will not you know discuss really because the these are more like data.

(Refer Slide Time: 14:03)



**Comparative Study**

**OS support**

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	MySQL
Linux, Window, Mac, Unix, Haiku, z/OS, OpenVMS	Linux, Window, Mac, Unix, BareMetal, Android	Linux, Window	Linux, Window, Mac (Express-C image), OS/2, Unix, z/OS, iOS	Linux, Window, Mac, Unix,	Linux, Window, Mac, Unix, BSD, AmigaOS, z/OS, Android	Linux, Window, Mac, Unix, BSD, AmigaOS, z/OS, Android

SWAYAM: NPTEL-NOCCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

40.25

©Silberschatz, Korth and Sudarshan

But here after given a compilation of different you know on different aspects, how do these common database RDMS systems agree or differ. So, this is like a slide which shows what are the operating system support for different databases. So, if you are for example, working on android, then you can easily make out that you do not have a choice to use SQL server or to use Oracle, but you can use Sybase.

But you can use Postgres and MySQL actually, if you look into these two columns, right most columns which are for the open source databases, you will find that they have the widest choice in terms of operating system. In many aspects you will find that these free database systems have a better you know options for you; obviously, when it comes to you know really the core, core of database systems in terms of really really supporting very large databases, really really supporting very fast operations, really really supporting very secure applications, you might need to only work with commercial software because they offer that, but otherwise for a large number of common you know

medium scale or low cost applications the free database systems, Postgres and MySQL are really good options there are different such features.

(Refer Slide Time: 15:23)



### Comparative Study

PPD

**Basic Features**

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	MySQL
Supports ACID properties for transactions, implicit commit for DDL, referential integrity, row level locking for fine grained locking, Concurrency control	Supports ACID properties for transactions, referential integrity, Concurrency control	Supports ACID properties for transactions, referential integrity, row level locking for fine grained locking, Concurrency control	Supports ACID properties for transactions, referential integrity, row level locking for fine grained locking, Concurrency control	Supports ACID properties for transactions, referential integrity, hash and partition for fine grained locking, Concurrency control	Supports ACID properties for transactions, referential integrity, row level locking for fine grained locking, Concurrency control	Supports ACID properties for transactions, referential integrity, row level locking for fine grained locking, Concurrency control

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

40.26

©Silberschatz, Korth and Sudarshan

The basic features are compared to here.

(Refer Slide Time: 15:27)



### Comparative Study

PPD

**Limits**

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	MySQL
Max DB Size: 2PB	Max DB Size: 104TB	Max DB Size: 524,272 TB	Max DB Size: Unlimited	Max DB Size: Unlimited	Max DB Size: Unlimited	Max DB Size: Unlimited
Max Table Size: 4GB * block size	Max Table Size: File size	Max Table Size: 524,272 TB	Max Table Size: 2 ZB	Max Table Size: Unlimited	Max Table Size: 32 TB	Max Table Size: 256 TB
Max Row Size: 8KB	Max Row Size: File size	Max Row Size: 2TB	Max Row Size: 32,677 B	Max Row Size: 64 GB	Max Row Size: 1.6TB	Max Row Size: 64KB
Max Column per Row: 1,000	Max Column per Row: 45,000	Max Column per Row: 1,024	Max Column per Row: 1,012	Max Column per Row: 2048	Max Column per Row: 1600	Max Column per Row: 4096
Max CHAR size: 32,767 B	Max CHAR size: 2GB	Max CHAR size: 2GB	Max CHAR size: 32 KB	Max CHAR size: 64,000 bits	Max CHAR size: 1GB	Max CHAR size: 64 KB
Max Number size: 126 bits	Max Number size: 64 bits	Max Number size: 126 bits	Max Number size: 64 bits	Max Number size: 38 bits	Max Number size: Unlimited	Max Number size: 64 bits
Max Column Name size: 128		Max Column Name size: 128	Max Column Name size: 128	Max Column Name size: 128	Max Column Name size: 63	Max Column Name size: 64

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition

40.27

©Silberschatz, Korth and Sudarshan

At compile different limits of different database systems here. So, you can see in terms of maximum row size, columns per row and so on and so forth, how do they differ. So, if you are making a choice in terms of what database I will use.

(Refer Slide Time: 15:42)

The slide title is "Comparative Study" with a subtitle "Tables and Views". It contains two tables comparing Oracle, Sybase, SQL Server, DB2, Teradata, PostgreSQL, and MySQL.

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	MySQL
Supports Temporary tables and Materialised views (apart from basic)	Supports Temporary tables and Materialised views (apart from basic)	Supports Temporary tables and Materialised views (apart from basic)	Supports Temporary tables and Materialised views (apart from basic)	Supports Temporary tables and Materialised views (apart from basic)	Supports Temporary tables and Materialised views (apart from basic)	Supports Temporary tables (apart from basic)

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	MySQL
Static+Dynamic	Static	Static	Static+Dynamic	Static	Static	Static

SWAYAM-NPTEL-NOC-MOCs Instructor: Prof. P. P. Das, IIT Kharagpur Date: Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition 40.28 ©Silberschatz, Korth and Sudarshan

You can use these information. This talks about tables we use the type systems, what kind of typing is used.

(Refer Slide Time: 15:48)

The slide title is "Comparative Study" with a subtitle "Data Types". It contains two tables comparing Oracle, Sybase, SQL Server, DB2, Teradata, PostgreSQL, and MySQL.

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	MySQL
Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; And other miscellaneous types like Spacial, Image, Audio, Dicom, Video	Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; And other miscellaneous types like Money	Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; Bit	Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; Bit	Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; Date/Time; Period, Interval, Geometry, xml, json	Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; Boolean	Supports various variants of Integer; Floating Point; Decimal; String; Binary; Date/Time; Bit

SWAYAM-NPTEL-NOC-MOCs Instructor: Prof. P. P. Das, IIT Kharagpur Date: Jan-Apr., 2018

Database System Concepts - 8<sup>th</sup> Edition 40.29 ©Silberschatz, Korth and Sudarshan

The different data types that are used are given here.

(Refer Slide Time: 15:52)



### Comparative Study

PPD

**Indexes**

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	My SQL
Supports	Supports	Supports	Supports	Supports	Supports	Supports
R/R++, Hash, Partial, Bitmap, Reverse		R/R++, Hash, Partial, Bitmap, Reverse	R/R++, Hash, Partial, Bitmap, Reverse	Hash, Partial, Bitmap,	R/R++, Hash, Partial, Bitmap, Reverse	R/R++, Hash,

Apart from Basic B/B++ indexes

only Basic B/B++ indexes

Apart from Basic B/B++ indexes

SWAYAM: NPTEL-NOC MOOCs; Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

40.30

©Silberschatz, Korth and Sudarshan

The index mechanisms are discussed here.

(Refer Slide Time: 15:57)



### Comparative Study

PPD

**Database Capabilities**

Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	My SQL
Supports	Supports	Supports	Supports	Supports	Supports	Supports
Union, Intersect, Inner Joins, Outer Joins, Except, Inner Selects, Merger Joins, Blobs and Clob, Common Table Expressions, Windowing Functions, Parallel Query	Union, Intersect, Inner Joins, Outer Joins, Except, Inner Selects, Merger Joins, Blobs and Clob, Common Table Expressions, Windowing Functions, Parallel Query	Union, Intersect, Inner Joins, Outer Joins, Except, Inner Selects, Merger Joins, Blobs and Clob, Common Table Expressions, Windowing Functions, Parallel Query	Union, Intersect, Inner Joins, Outer Joins, Except, Inner Selects, Merger Joins, Blobs and Clob, Common Table Expressions, Windowing Functions, Parallel Query	Union, Intersect, Inner Joins, Outer Joins, Except, Inner Selects, Merger Joins, Blobs and Clob, Common Table Expressions, Windowing Functions, Parallel Query	Union, Intersect, Inner Joins, Outer Joins, Except, Blobs and Clob	Union, Outer Joins, Except, Inner Selects, Blobs and Clob, Common Table Expressions

SWAYAM: NPTEL-NOC MOOCs; Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition

40.31

©Silberschatz, Korth and Sudarshan

The capabilities of the database, what it can do overall.

(Refer Slide Time: 16:01)



### Comparative Study

PPD

Other Objects						
Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	My SQL
Supports	Supports	Supports	Supports	Supports	Supports	Supports
Data Domain, Cursor, Trigger, Function, Procedure, External Routine	Data Domain, Cursor, Trigger, Function, Procedure, External Routine	Data Domain, Cursor, Trigger, Function, Procedure, External Routine	Data Domain, Cursor, Trigger, Function, Procedure, External Routine	Cursor, Trigger, Function, Procedure, External Routine	Data Domain, Cursor, Trigger, Function, Procedure, External Routine	Cursor, Trigger, Function, Procedure, External Routine

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition      40.32      ©Silberschatz, Korth and Sudarshan



Other kinds of objects that it supports.

(Refer Slide Time: 16:05)



### Comparative Study

PPD

Partitioning						
Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	My SQL
Supports	Supports	Supports	Supports	Supports	Supports	Supports
Range, Hash, Composite, List	none	Range, Hash, Composite, List	Range, Hash, Composite, List	Range, Hash, Composite, List	Range, Hash, Composite, List	Range, Hash, Composite, List

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition      40.33      ©Silberschatz, Korth and Sudarshan



The different partitioning mechanism.

(Refer Slide Time: 16:08)



## Comparative Study

PPD

Access Control						
Oracle	Sybase	SQL Server	DB2	Teradata	PostgreSQL	MySQL
Supports	Supports	Supports	Supports	Supports	Supports	Supports
Native network encryption, Separation of Duties, Password Complexity Rules, Enterprise Directory compatibility, Audit, Resource Limit,	Native network encryption, Separation of Duties, Password Complexity Rules, Enterprise Directory compatibility, Audit, Resource Limit,	Native network encryption, Separation of Duties, Password Complexity Rules, Enterprise Directory compatibility, Audit, Resource Limit,	Native network encryption, Separation of Duties, Password Complexity Rules, Enterprise Directory compatibility, Audit, Resource Limit,	Native network encryption, Separation of Duties, Password Complexity Rules, Enterprise Directory compatibility, Audit, Resource Limit,	Native network encryption, Separation of Duties, Password Complexity Rules, Enterprise Directory compatibility, Audit, Resource Limit,	Native network encryption, Enterprise Directory compatibility, Patch Access
						

SWAYAM, NPTEL-NOC, Instructor: Prof. P. P. Das, IIT Kharagpur, 2018

Database System Concepts - 8<sup>th</sup> Edition

40.34

©Silberschatz, Korth and Sudarshan

The access control which is critical for ensuring good security and good management of that database are given here.

So, these are kind of the different across different features, this is parameters, this is how they compare. So, well this is for your you know reference only, this is this is not for your assignment or examination, but I just wanted to have a view that with all the theory and you know a small hands on that you have seen, when you go to the real life what are the expected things what are the expected system this will have to work with. Now let us just move on and let us let me just briefly talk about the a group of database systems which are non relational and of course I would warn you that the basis for these DBMS is are not covered in this course, is beyond the course, but just for your information and to keep in keep you in tune with what is happening frequently around the industry today.

(Refer Slide Time: 17:16)

**What is Big Data?**

- Big data is data sets that are so voluminous and complex that traditional data-processing application software are inadequate to deal with them
- Big data challenges include capturing data, data storage, data analysis, search, sharing, transfer, visualization, querying, updating, information privacy and data source
- **5V's (characteristics) of big data:**
  - **Volume:** The quantity of generated and stored data. The size of the data determines the value and potential insight, and whether it can be considered big data or not.
  - **Variety:** The type and nature of the data. This helps people who analyze it to effectively use the resulting insight. Big data draws from text, images, audio, video; plus it completes missing pieces through data fusion.
  - **Velocity:** In this context, the speed at which the data is generated and processed to meet the demands and challenges that lie in the path of growth and development. Big data is often available in real-time.
  - **Variability:** Inconsistency of the data set can hamper processes to handle and manage it.
  - **Veracity:** The data quality of captured data can vary greatly, affecting the accurate analysis

So, the non-relational database systems have arisen from what they all have must have heard of is the whole aspect of Big Data. Big Data as a name suggests is certainly voluminous data, complex data and now the question that you might have is if I have done a good relational design, if I have a good RDMS, can I not handle big data?. The question here really is, big data is not only about volume, the volume is only one aspect which is really large. So, big data typically are characterized by certain Vs.

So, these are not any very standardized characterization, but these are more commonly accepted once. So, one is volume, that the quantity of data when a for a big data situation has to be very very large. Now again what is a very very large is again a subjective question, some you can say that a million record is large, someone else would say no million record is small, it is actually 10 million is large; some might say that it is a database need to be petabytes to be large and so on.

But these are all subjectivity, but it is large has a voluminous existent in certain sense, there has to be different variety different types of data. So, all that we have seen in the relational database is basically your you know strings and numbers if you look at it in different ways we are seeing, whatever we have dealt with in all these through all this 39 modules so far, they are primarily about strings and numbers, but nothing else we have not, but big data can be about free text, it could be about natural language comments your regularly writing comments on your Facebook.

So, those Facebook comments are phrases and if I want to make certain query based on that, if I want to make a query that, how many Facebook users has commented on the success of Virat Kohli as a captain of India if I want to make such a query, then the question is how do I do that? Because it is not something where you have a at the information in a very structured way this is no there is no relational schema which says that well the values are put in terms of Virat Kohli having done very good, moderately or marginally or you know the captain Indian captains are successful, not successful and so on.

These are this happen in terms of various texts, phrases, clauses that we write. So, variety is a major issue then, it could have word your images video. So, big data includes all of that. The third V is about velocity that the processing speed may need to be really really fast, often in big data often we say that the processing has to be real time which means what is real time. Real time is basically that from the time I fire the query and to the time I get the result, there is a fixed time limit within which it has to happen.

So, if I if I if I really want to do a railway reservation that also is a kind of real time, but that is not very critical because it is if I get the reservation done in one minute, it is also ok, if it takes 5 minutes, it is good if we can happen in 10 seconds, but I do not ever need it in say 20 millisecond. So, but in when you talk about real time, it could really be about getting all these processing done in millisecond, microsecond, nanosecond and so on. And those kind of real time systems with a large volume of varied data, it is a big challenge.

So, those are the challenges of big data, then there could be variability inconsistency of the data that you are because maintaining integrity is a big problem; there could be issues in terms of quality of the data that is called veracity. So, actually these things characteristics that I have put, there are lot of debates in terms of that or many people take these 3 and say that there these are the 3 V s of big data. There is a 3 main characteristics, but off let more and more people are also considering variability and veracity are as the characteristics of big data. Now as it happens, is if you look into these requirements and what you have you have a fairly good idea of relational databases now what they can do, how to design them, how to query them, how to implement them, you will understand that it is not easy to meet these requirements using the relational model.

(Refer Slide Time: 22:07)

Why do we need non-relational databases?

To effectively support Big Data

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition 40.37 ©Silberschatz, Korth and Sudarshan

So, we need their non-relational databases to effectively support big data. That is that is a one major reason that you need big data.

(Refer Slide Time: 22:16)

Non-Relational Databases

PPD

- Does not follow the relational model
- Offer flexible schema design
- Handle unstructured data that does not fit neatly into rows and columns
- Typically Open Source
- Scalable using cheap commodity servers
- The popular ones are:
  - MongoDB, DocumentDB, Cassandra, Couchbase, HBase, Redis, and Neo4j
- These databases are usually grouped into four categories:
  - Key-value stores
  - Graph stores
  - Column stores, and
  - Document stores

Non-Relational Databases are also known as NoSQL Databases

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr. 2018

Database System Concepts - 8<sup>th</sup> Edition 40.38 ©Silberschatz, Korth and Sudarshan

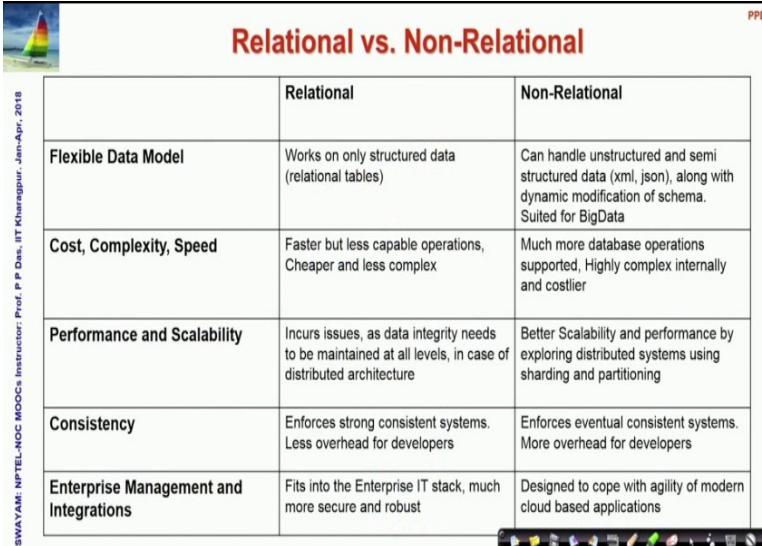
So, that in non-relational databases certainly as the name suggests, do not follow relational model, they offer flexible schema design. The schema may itself change while the database is evolving which is not the case in the relational schema is fixed, only the data can change, but here the schema itself can change. It may be able to handle unstructured data, make natural language comments like images like audio coming in

which do not fit nearly into your you know table structure, some of the other feature sites they are typically open source because you know still in an experimental stage and needs to be scalable and some of the popular ones are, these are the names you must have heard about at least some of them like MongoDB like Cassandra like HBase and so on.

Now again in terms of the non relational database, it does differ in terms of all non relational database error are not of the same type, there are there have been 4 different styles or strategies to actually generate realize these non relational databases, they are called key value store graph, store columns stores and document store. They are also known as no SQL databases. I, I personally find the name no SQL a little misnomer, it no SQL does not mean that you are strictly prohibited from not using SQL in these databases.

But I would rather like to read it more as no SQL means that it is not only SQL like in a relational database, you can use only SQL and solve problems; here you need to do lot of other things beyond that.

(Refer Slide Time: 24:00)



The slide features a title 'Relational vs. Non-Relational' in red font, a small sailboat icon in the top left, and a 'PPD' watermark in the top right. The main content is a comparison table with five rows. The first row is a header with two columns: 'Relational' and 'Non-Relational'. The remaining four rows compare specific characteristics:

	Relational	Non-Relational
<b>Flexible Data Model</b>	Works on only structured data (relational tables)	Can handle unstructured and semi structured data (xml, json), along with dynamic modification of schema. Suited for BigData
<b>Cost, Complexity, Speed</b>	Faster but less capable operations, Cheaper and less complex	Much more database operations supported, Highly complex internally and costlier
<b>Performance and Scalability</b>	Incurs issues, as data integrity needs to be maintained at all levels, in case of distributed architecture	Better Scalability and performance by exploring distributed systems using sharding and partitioning
<b>Consistency</b>	Enforces strong consistent systems. Less overhead for developers	Enforces eventual consistent systems. More overhead for developers
<b>Enterprise Management and Integrations</b>	Fits into the Enterprise IT stack, much more secure and robust	Designed to cope with agility of modern cloud based applications

Navigation icons are located at the bottom right of the slide.

So, here is a quick comparison between the relational and the non-relational, in terms of the flexibility of the data model, relational is very structured when on relational has to have handle unstructured data, semi structured data and therefore it has to be flexible in terms of the data model, cost complexity and speed faster less capable, but cheaper and less complex, but in non-relational, you are talking about much more database operations

highly complex in internal structure usually costlier, performance and scalability certainly non-relational ones need to be better scalable, consistency have a very strict consistency rules in relation, but in non-relational you use some kind of you know eventual consistent system. So, maybe not always not everything is consistent in that way, enterprise management and integration, relational fits very well into because it is been around for as you have seen the little bit of history of all these common databases, it is more than 40 years that they have been around.

So, they easily fit into the IT stack whereas, non-relational is still on the in the agile form of development that is becoming more and more common it fits into the cloud based development and so on. So, these are some of the distinctions that exist.

(Refer Slide Time: 25:27)

**Types of NoSQL Databases**

1. **Key-value stores:**
  - These type of databases work by matching keys with values, similar to a dictionary. There is no structure nor relation. **Example:** Redis, MemcacheDB
2. **Graph stores:**
  - These use tree-like structures (i.e. graphs) with nodes and edges connecting each other through relations. **Example:** OrientDB, Neo4J
3. **Column stores:**
  - Column-based NoSQL databases are two dimensional arrays whereby each key (i.e. row / record) has one or more key / value pairs attached to it and these management systems allow very large and unstructured data to be kept and used (e.g. a record with tons of information). **Example:** Cassandra, Hbase
4. **Document stores:**
  - These DBMS work in a similar fashion to column-based ones; however, they allow much deeper nesting and complex structures to be achieved (e.g. a document, within a document, within a document).  
Documents overcome the constraints of one or two level of key / value nesting of columnar databases. **Example:** MongoDB, Couchbase

And these are the different types of no SQL databases, there is a key value store strategy Redis and MemcacheDB follow this strategy, graph store is used by orient DB and Neo4J; column store is used by Cassandra and HBase document store, MongoDB, Couchbase, they use document store. So, these are I am in this is not just about going a deeper into what they are or how they are distinguished, I just want you to have an idea that well. These are different from the relational databases they can do lot of structured, handling of unstructured data they can actually use a scalability of volume a variety which is relationships cannot do and, but they have there are different principles for actually implementing them and there is a deeper.

So, if you are interested, you can take specific courses which deal with the big data and prepare yourself for the bigger challenges ahead.

(Refer Slide Time: 26:29)



### Comparative Study

PPD

**When to Use**

Redis	MemcacheDB	OrientDB	Neo4J	Cassandra	Hbase	MongoDB	Couchbase
Caching, Queuing frequent information, Keeping Live information, Supports lists, sets, queues and more	Caching, Queuing frequent information, Keeping Live information,	Handling complex relational information, Modelling and handling classifications	Handling complex relational information, Modelling and handling classifications	Keeping unstructured non-volatile information, useful for content management	Keeping unstructured non-volatile information, useful for content management	Works with deeply nested and complex data structures, JavaScript friendly, useful for content management	Works with deeply nested and complex data structures, JavaScript friendly, useful for content management

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr, 2018

Database System Concepts - 8<sup>th</sup> Edition      40.41      ©Silberschatz, Korth and Sudarshan

I have also done similar to the relational database, I have presented here a tentative comparative study between these different non no SQL databases in terms of what is the context in which you use them.

(Refer Slide Time: 26:46)



### Comparative Study

PPD

**Handling Relational Data**

Redis	Memcache DB	OrientDB	Neo4J	Cassandra	Hbase	MongoDB	Couchbase
Supports ACID, query only on key	Supports ACID, query only on key	Supports ACID and joins	Supports ACID and joins	Supports ACID, Multiple Queries	Supports ACID, Multiple Queries	Supports ACID, Multiple Queries, Nesting Data	Supports ACID, Multiple Queries, Nesting Data

SWAYAM-NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr, 2018

Database System Concepts - 8<sup>th</sup> Edition      40.42      ©Silberschatz, Korth and Sudarshan

Or now while you are doing this unstructured data handling, there will be lot of data which is also structured.

So, how do you, along with this know SQL, how do you handle the relational data with these?

(Refer Slide Time: 26:59)



### Comparative Study

PPD

Performance							
Redis	Memcache DB	OrientDB	Neo4J	Cassandra	Hbase	MongoDB	Couchbase
Highly Scalable, Highly Flexible, not complex in terms of representation and use	Highly Scalable, Highly Flexible, not complex in terms of representation and use	Variable Scalability, Highly Flexible, Highly complex in terms of representation and use	Variable Scalability, Highly Flexible, Highly complex in terms of representation and use	Highly Scalable, Moderately Flexible, Moderately complex in terms of representation and use	Highly Scalable, Moderately Flexible, Moderately complex in terms of representation and use	Highly Scalable, Highly Flexible, Moderately complex in terms of representation and use	Highly Scalable, Highly Flexible, Moderately complex in terms of representation and use
SWAYAM-NPTEL-NOCS Instructor: Prof. P P Das, IIT Kharagpur - Jan-Apr, 2018							



Database System Concepts - 8<sup>th</sup> Edition      40.43      ©Silberschatz, Korth and Sudarshan

Databases what how do how do the performance compare between these different no SQL databases and based on all that you can make some judgment and it is it is very important to in today's time naturally knowing relational databases the foundational ones are very important, but it is always good to look forward be with the time and I will urge that if you have started growing interest in handling of data do take specific courses on big data and no SQL databases. I will end this discussion with a very simple skill job profile matrix which Will give you some idea in terms of, it will you can use it for a certain kind of self-assessment as well.

(Refer Slide Time: 27:37)

Skills	Years of Experience	Under-standing Specs and Schema	Coding Query In SQL (DML)	Analyzing Specification, Schema Design and Normalization (DDL)	Application / Database Architecture, Indexing, Performance Optimization	Database Administration	Databases, Algorithms, Architecture, Compilers, ...	Data Mining, Machine Learning, Big Data, Programming
Job Profiles								
Application Programmer	0-4	X X						
Senior Application Programmer	2-6	X X X X	X					
Database Analyst / Architect	4-8	X X X X	X					
Database Administrator	8-10	X X X X X	X		X			
Database Engineer (multiple grades)	2-10	X X X X X	X		X X			
Big Data Programmer, Analyst	0-6	X X X X X					X	

So, let me just explain the structure of this matrix, what I have tried to do is here I am sorry. Here on the left, I have given different typical job profiles this is if you look into LinkedIn, Naukri and all that you will find these kind of profiles being. So, then at the lowest level there are application programmers for which typically 0 to 4 years of experience are asked for.

Then the next level, this is so, this is your kind of your career progression also. If you if you choose to take up databases as your primary job profession, this next level is a senior application programmer which requires 2 to 6 years of experience depending on the organization and depending on your skills. Then you move on to database analyst or architect which you happen in 4 to 8 years of time and on a little different track because these are these are primarily in terms of application development and hierarchy on that and the other is an administrator track who actually administers the database in an organization, controls all the all that is happening in different database applications, typically 8 to 10 year's experience is required.

And some of that, so this these are the about actually in terms of you know profiles that are related to applications and this is a profile which is related to, if you really want to become a database engineer in a sense that you want to you know make changes in Oracle, you want to make changes in say MySQL, you want to make changes in say MongoDB or say that relation will say the Sybase.

So, if you want to become a database engineer who changes the database system itself or develops the database system itself, then this is the kind of background you will need. There is a kind of number of years, you would need and of course it is not a single grid there are multiple grades, you know junior and mid levels in here and those kind and last which have shown in different color are the whole set of profiles which relate to programming the big data, analyzing the big data and so on.

We are at present, it is companies are typically asking for 0 to 6 years of experience depending on actual skills that you have. On this side, I have shown a whole grouping of skills, this is the first basic level that you must understand specs and schema, without that you cannot do any of this and you must have a skill for coding in inquire in SQL, the DML part significantly, without that as you can see you cannot pick up any of these profiles whereas, if you go a little if you go little senior you know gaining experience and you should be able to analyze specs that is, you should be able to design schema do normalization and get into this.

So, at a very initial level, you may not be expected to do all of that schema design and normalization by yourself, but it would be good to be able to do that, but well there will be seniors to help you, but if you once you become a senior application programmer that becomes onward that becomes a critical skill to have. Then the next level would be in terms of application or database architecture management deciding on how to index, performance optimization.

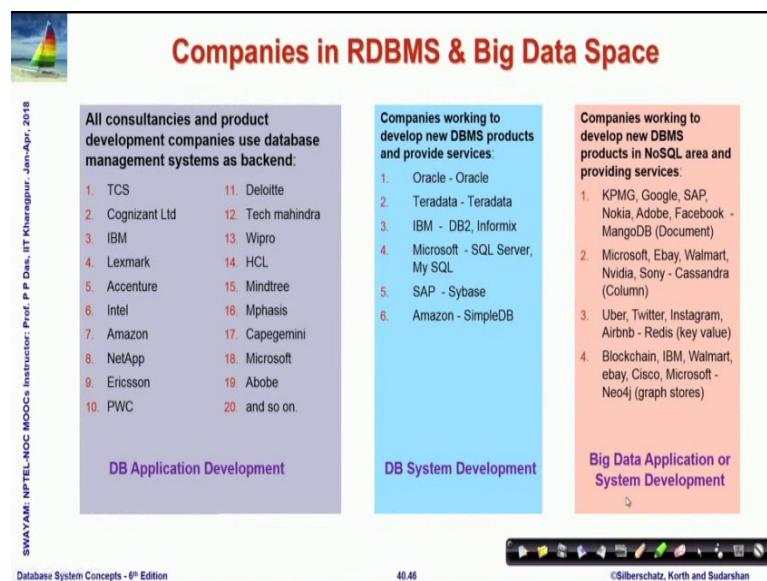
So, between these two, there will be certain overlaps a senior application programmer in addition to doing this might do some of these optimization techniques depending on how competent he or she is. Or some database architect may focus only on this, but these are the typical skills that you need. But to be a database administrator, you need all of these skills, but you are specifically administering a certain organizations enterprises whole database system. So, it is just not one database application, but a whole lot of databases and whole lot of user groups, security, network connectivity and all that.

So, that needs certainly bigger experience it can, you can see that experience level is much higher and the skill sets. If you want to become a database engineer, that is not focus only on the application side, but also have some more understanding in terms of actually doing working in the internals of the database systems, then you need whole lot

of additional skills like good knowledge and algorithms, in architecture, in compiler all of that; only then and coupled with coupled with all the database knowledge, then you will be able to work as a database system engineer. And in the emerging areas of what is big data where, you need to have now of course, the I am saying this is 0 to 6, it could be 0 to 8 kind of, not more than that because it did not exist quite a long time ago, but you need to have a basic level of at least this much of the relational database understanding and knowledge, but what is critical is a whole set of other skills like, you must be aware with big data the data mining, warehousing strategies machine learning or is often very useful in this kind of big data applications, python programming, tensor flow all these become critical. You have to be a good programmer in any case I mean not only just an SQL program and you might have to be a good program and in C or C plus plus or python of these, but that is it that is a very very emerging area.

So, if you can acquire a little bit of besides database you know he said that the basics of the database along with that if you pick up few basics or from here, you will be able to enter into the space and that will give you a very very bright future in my view otherwise you can focus on the application programming stat as I have mentioned. So, this is the basically skill profile matrix that you have mapping that you have that you can focus on.

(Refer Slide Time: 33:49)



So, finally, before I close here a glimpses of companies that are in the very active in the RDMS space really really any big organization you talk about, they have consultancy

projects, product development different database management back end services and so on. So, DB application development, I have listed some around 20 companies, but there are really 100s of them almost. Any big organization in any area you think of, they require databases. So, in terms of beta based application programmer and senior programmer and to some extent architect, you have a wide range of jobs available which you may just grab; if you have been able to study write the basics of the database. The second group of companies which I show here, these are system development companies who are actually working on the new DBMS products and services around that.

So, these are companies like Oracle, Teradata or Microsoft and so on, naturally these are big companies and you need more lot of more skills besides the database like I said algorithms programming and all that to crack a job here and here are some of some companies which I have mentioned, but there are many others who are focusing on the big data space.

So, I have tried to you know these may not be absolutely accurate because you know these are all collected from different sources, but these are the different companies and the kind of non relational database that they are focusing with working with. So, if you pick up certain skills in those in a certain non-relational no SQL database, then you can target the corresponding companies better or other companies and you can see that all. You know new generation companies, the companies were working for products for the next 10, 10, 15 years are in this space.

So, there are whole lot of opportunities for you all if you if you prepare a little hard, then you will I mean job will run after, you will not have to run after the job.

(Refer Slide Time: 35:56)

The screenshot shows a presentation slide with the title "Final Words" in red at the top right. On the left, there is a small sailboat icon. The main content is a bulleted list of five items:

- Read the DBMS Text book thoroughly and solve exercises
- Practice query coding
- Practice database design from specs
- Besides DBMS, develop good knowledge in programming, data structure, algorithms and discrete structures
- Seek help, if you need to – mail us

On the left margin, vertical text reads: "SWAYAM, NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur - Jan-Apr. 2018". At the bottom left is a small video thumbnail of a man speaking. The bottom right corner shows "40.47" and "©Silberschatz, Korth and Sudarshan".

So, with that I conclude this course a couple of final words the hygiene words. Read the DBMS textbook thoroughly and solve exercises. There is no shortcut to that, there is no other way to master the horse other than this you must practice query coding as much as you can, practice database design from specification. We are releasing a tutorial on this where for a hospital management system we are showing from the specification how you can do the initial schema and the refinements and finally, how can you implement it using my SQL.

So, do similar practices very heavily. Keep in mind the database the knowledge of database system alone will not be good enough to get a good job, get a good placement. So, develop good knowledge in programming data structure, algorithms and discrete structures; these are the minimum required around the database systems which will really make you powerful and if you need we are there to help you.

As long as the course is on, the forum would be on. You can post in the forum beyond that also if you need help, please ask for it mail us and wish you all the very best with your course in your examination and the future course of your profession in life, all the very best.

**THIS BOOK  
IS NOT FOR  
SALE  
NOR COMMERCIAL USE**



(044) 2257 5905/08



nptel.ac.in



swayam.gov.in