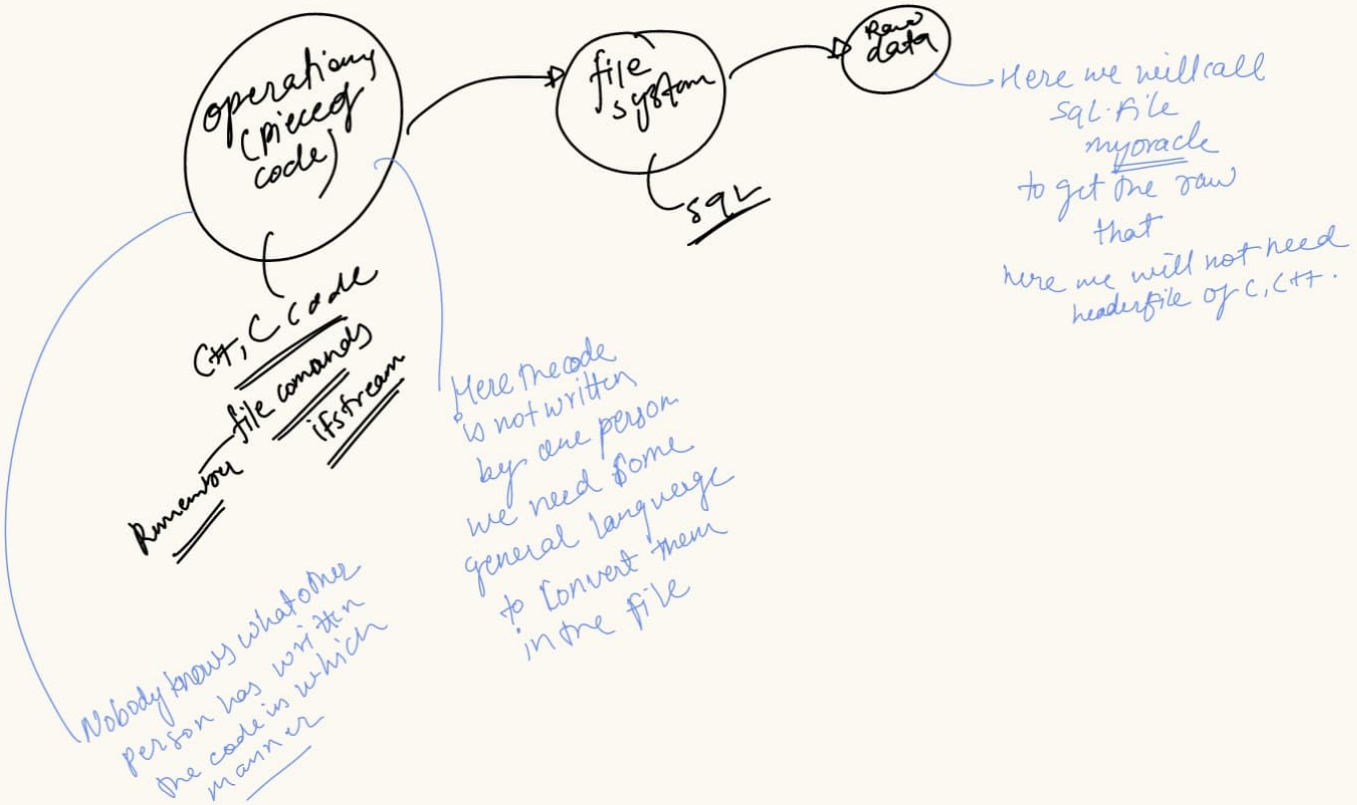
8/1/2025

Rollno	Name	DOB	Branch	CGPA	Employ	Name2	Dept	HOD

What are the issues for DBs ?

* for 2 or more name employs would be same
* and if HOD or Employ change is there to delete HOD may delete all the data.
* CGPA is not for employ & HOD

where do you find DBS's ! → hospitals, Ecommerce., shopping inventory management.



* Redundancy & Inconsistency

Recovery datas for some emercency purpose

like for lope

like array of line

```
for line in file:
    record = PARSE (line)
    if "split" = record[0]
        print (string (record[3])
```

in for LOOP

* if coding is in C then DBMS is also in C language ....
(Same for other languages)

7/1/25

DBMS

deals with the Real value data we can't say they words a string any no. as int.

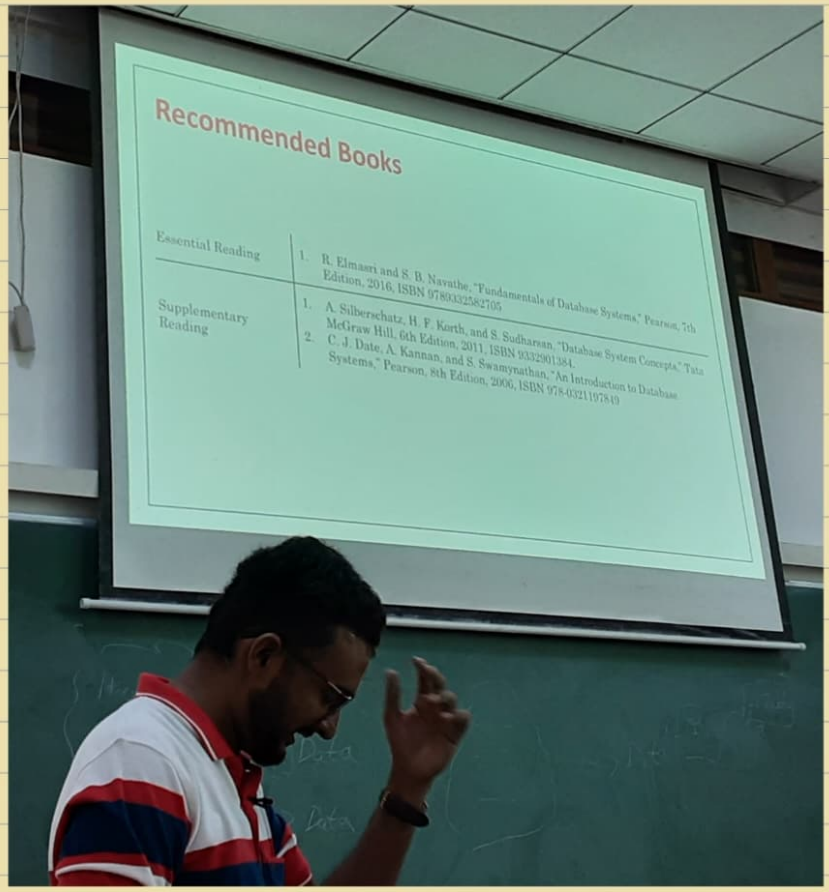
it will have a meaning like name or salary

9/1

Hostel Management → Data

Library Management → Data

Admin Management → Data



Redundance causes Inconsistency. → Makes it difficult to query.

Standard → SQL

Issues:

Concurrency → Multithreading

Availability & Recovery → Backup

Security → Public, Protected & Private

Integrity → Accuracy, Consistency & Completeness
(Edge cases)

[Schema → How the table is supposed to be.]

Meta Data helps seperate the codes and the data

10/1

Meta Data : Data about Data

↳ Kindof Mapping

Meta Data: Allows applications to interact with data without needing to understand its physical storage details

What does the layer of DBMS Provide ?

Separation of Data & Metadata:

Meta data sits in-between the DBMS S/W and the actual Data, provides flexibility.
Helps to achieve Program-Data Independence.
The notion of "**Metadata**" is introduced by DBMS.

↓

It describes how (i.e., its structure & organization) the raw data is stored.

Remember:

Structure of the Data ← Hardcoded into → App Programs

Metadata helps to remove this dependency by storing the structure information separately.

Meta Data - Catalogue
Data Dictionary
Schema
} Other names

What does the layer of DBMS Provide ?

A Standard way to organize the data:

Most of the existing DBMS systems follows "SQL" standard to organize the data.
This means uniformity and easy to interface with different applications.

Interoperability

By standardizing **data access**, DBMS allow different application and systems to interact With the same DB seamlessly.

One single DB can serve multiple applications.

What does the layer of DBMS Provide ?

Programmatic Convenience:

- Less effort for system development.
- Concentrate only on logical design [No physical level implementation worries].
- Concurrency control, recovery, access/authoraziation control, security integrity check.

And many more things will be taken by the DBMS S/W. You need not worry about it.

DBMS → Generic
Data & Schema are stored seperately
↳ Organized & Structured

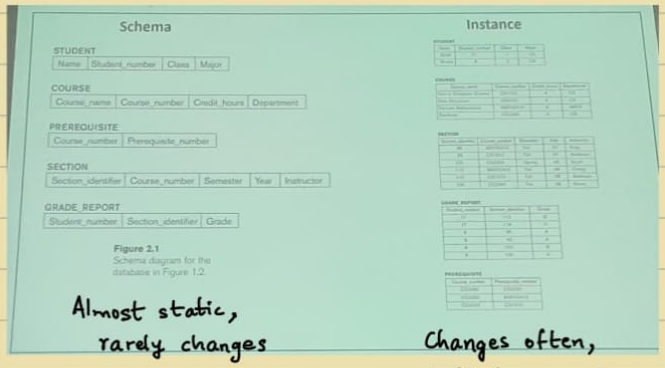
SQL: Structural Query Language

What Makes DBMS Generic ?

We know that in DBMS Systems, Data is organized and structured in a **standard** way.

i.e. Data & Schema are stored separately

Schema — Description	Instance
This defines the structure in which data Is to be stored.	It is the actual data stored in a format as prescribed by the schema.
It sets up the skeleton structure of the DB.	
int a ;	a = 10;



Almost static, rarely changes

Changes often, Reflects current situation

Summary:

- Life Before DBMS – Early Information Systems.
- Issues in EIS – Redundancy, Inconsistency, Other Issues.
- Issues in EIS – Rootcause.
- How DBMS Solves these issues.
  - Metadata
  - Schema vs Instance

→ DB : DataBase [Raw Data]

DBMS : Database Management System

DBMS helps to define, create, access, manage, query, update and administer a data.

Data : Known facts that can be recorded & have implicit meaning → Particular context specificity

By itself, data is just raw facts or figures, but its implicit meaning comes from its relationship to the real-world entities, events or concepts it represent.

Ex.

For example:

1.Raw Data:

1. "100" - Without context, this is just a number.

2.Implicit Meaning with Context:

1. "100" as **temperature in Fahrenheit** suggests it's a hot day.
2. "100" as **marks in an exam** indicates a perfect score.

Operational data : Data which we are going to work with

DataBase (DB)

It is an *organized* collection of Inter-Related data that model some aspects of the real world.

Real World: DBMS is typically concerned with data associated with some company/organization/enterprise or business.

Given the data of an Enterprise or a Company:
The company may have current needs and requirements with it. (Different needs may also arise in the future)

↓

It is necessary to organize data in some fashion, so that it is easier to serve these needs.
While data should be organized robustly, such that future needs may also be accommodated.

DB : Reflects the current state of affairs of the Enterprice / Company.
The organizational heirarchy within the company is often reflected in the DB.

Mini-World: Some part of the real world about which data is stored in a database.
For example, student grades and transcripts at a university

16/1

The data you saw in DS Vs The data you are going to see in DBMS

- Main Memory
- Program Specific
- Array, Tree, HashTable, Dict, Heap...
- Easy to Maintain (within the app Code)
- Has meaning in the context of the program

- Disk Resident
- Enterprise Specific.
- Tables, Metadata...
- Requires DB's and DBMS
- Often has real world meaning

The data you saw in DS Vs The data you are going to see in DBMS

In traditional programming, the data and the code are tightly coupled.
If you change the type of the variable, or declare a new one, the entire code needs to be recompiled.

In DBMS the application code and the data are **loosely coupled**.
DBMS sits in b/w the app code and the data, eliminating any direct dependency.
The DBMS S/W may rearrange the data underneath, while the Application may completely be unaware of it.

What is Loose Coupling and Tight Coupling ?

Loose coupling refers to a design principle where components or modules in a system are minimally dependent on each other. They interact through well-defined interfaces, allowing changes in one module without significantly impacting others.

•Easier to replace or update components.
•High modularity and flexibility.

Example:
In a web application, the front end (UI) communicates with the back end through APIs. The back end can be updated or replaced without impacting the front end, as long as the API remains consistent.

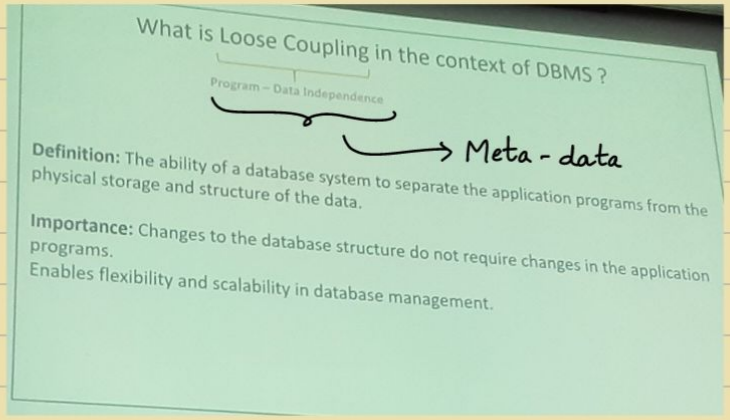
What is Loose Coupling and Tight Coupling ?

Tight coupling occurs when components or modules in a system are highly dependent on each other, often sharing internal details or assumptions. Changes in one module often require changes in the others.

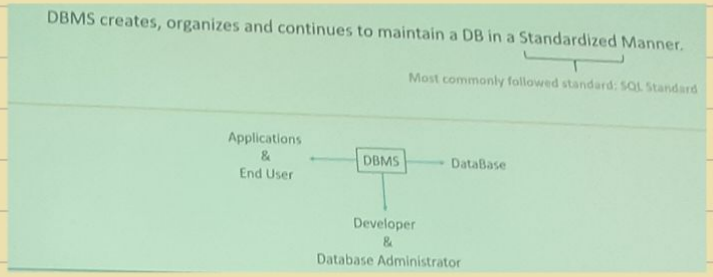
•Low modularity and flexibility.
•Difficult to modify or replace components.

Example:
The washing machine's firmware is tailored to work only with a specific hardware configuration (e.g., a particular microcontroller or motor type).

•Hardware upgrades or changes necessitate rewriting significant portions of the firmware.



DBMS : General Purpose Software / Package / Tool to facilitate the creation and maintanance of a computerized database.



Application → Website, App

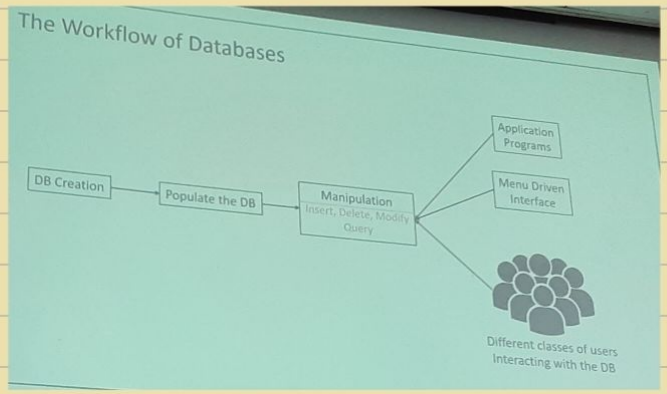
A single DBMS S/W may manage multiple DBs at a time.

Developer → developers the program, UI/UX, etc
DBA → deals with the DB, encrypting, etc

Database refers to DB and DBMS.

→ Workflow of Database :

Design the DB schema
DB creation → Create Schema
Populate the DB → Enter entries
Manipulate the DB → Insert, Delete, Modify, Query



The Workflow of Databases

- ***Define*** a particular database in terms of its data types, structures, and constraints
- *Construct* or **Load** the initial database contents on a secondary storage medium
- ***Manipulating*** the database:
  - Retrieval: Querying, generating reports
  - Modification: Insertions, deletions and updates to its content
  - Accessing: the database through Web applications
- ***Sharing*** *of data* by a set of concurrent users and application programs – yet, keeping all data valid and consistent.

- Example :

College Database Example

What are some things or entities in the college about which we would like to store information ?

- STUDENTs
- COURSEs
- SECTIONs (of COURSEs)
- DEPARTMENTs
- INSTRUCTORs

Relationships between the Entities

- SECTIONs *are of specific* COURSEs
- STUDENTs *take* SECTIONs
- COURSEs *have prerequisite* COURSEs
- INSTRUCTORs *teach* SECTIONs
- COURSEs *are offered by* DEPARTMENTs
- STUDENTs *major in* DEPARTMENTs

Meta - Data :

RELATIONS

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5
GRADE_REPORT	3
PREREQUISITE	2

COLUMNS

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
...	...	...
...	...	...
...	...	...
Prerequisite_number	XXXXNNNN	PREREQUISITE

Figure 1.3
An example of a database catalog for the database in Figure 1.2.

Role of Metadata in Achieving PDI

- **Self-describing nature of a database system:**
  - A DBMS **catalog** stores the description of a particular database (e.g. data structures, types, and constraints)
  - The description is called **meta-data**.
  - This allows the DBMS software to work with different database applications.
- **Insulation between programs and data:**
  - Called **program-data independence**.
  - Allows changing data structures and storage organization without having to change the DBMS access programs.

Abstracting Physical Details
•Metadata provides a layer of abstraction between application programs and physical data storage.
•Applications interact with the metadata rather than directly accessing raw data.

Managing Schema Evolution
•When the database schema changes (e.g., adding a new column), the metadata is updated.
•Applications continue to function without modification because they query the schema via metadata.

Simplifying Queries
•Metadata helps applications understand the structure of the database (e.g., which tables and columns to access).
•SQL queries rely on metadata to execute commands like SELECT, INSERT, etc.

Supporting Interoperability
•Different applications can interact with the same database without knowledge of its physical storage, thanks to metadata.

→ Users of the Db Systems :

Users of the Database Systems

Users may be divided into

- Those who actually use and control the database content, and those who design, develop and maintain database applications (called "**Actors on the Scene**")
- Those who design and develop the DBMS software and related tools, and the computer systems operators (called "**Workers Behind the Scene**").

End Users

End-users: They use the data for queries, reports and some of them update the database content. End-users can be categorized into:

- **Casual**: access database occasionally when needed
- **Naïve** or Parametric: they make up a large section of the end-user population.

When Not To Use DBs

Overhead Costs of DBMS:

- High initial investment in hardware, software, and training
- The generality that a DBMS provides for defining and processing data
- Overhead for providing security, concurrency control, recovery, and integrity functions

Though these features are desirable, they incur significant cost to achieve.
Applications which do not demand these features, it is better not to use DBMS systems

When Not To Use DBs

Examples:

- Simple, well-defined database applications that are not expected to change at all
- Stringent, real-time requirements for some application programs that may not be met because of DBMS overhead
- Embedded systems with limited storage capacity, where a general-purpose DBMS would not fit

Niche applications like: Medical systems, Avionics, Stock Trading systems, Simulations

Use optimized in-memory databases or direct data access through lightweight data structures.

A Quick Recap:

4 Characteristics of Database Approach

- Self-describing nature of a database system
- Insulation between programs and data, and data abstraction
- Support of multiple views of the data
- Sharing of data and multiuser transaction processing

Section 1.3, Chapter 1, Navathe

16/1

→ Designing a DBMS Application:

The Workflow of Databases

- ***Define*** a particular database in terms of its data types, structures, and constraints
- *Construct* or **Load** the initial database contents on a secondary storage medium
- ***Manipulating*** the database:
  - Retrieval: Querying, generating reports
  - Modification: Insertions, deletions and updates to its content
  - Accessing: the database through Web applications

The (So Called) Right Approach to Coding

- Flowchart
- Algorithms

Tools to organize out thoughts / soln approach systematically

Flowchart → Algorithm → Code

What is the rationale behind this approach ?

Flowchart → Algorithm → Code

Flowchart	Algorithm	Code
It is a visual approach.	Step-by-step textual representation of logic	Ready to execute
Easier to interpret.	Easier to read, compared to code.	Difficult to read & interpret
Can be used to explain the logic even to non-technical users.	Used among the developers to discuss logic	

Diagram → Step-by-step logic → Actual Implementation

DBMS Design – Step 1

First, we should imagine and design or plot a conceptual layout of the entire DB.

How do you 'Express' or want the data to be organized ?

Note that there could be multiple ways/approaches to organize the data
- Just like multiple ways exists to sort an array.

Substeps in step 1

Requirement Collection:

What are the various pieces of data that needs to be stored ?
What are the interrelationships between those data ?

E/R Model is used for this purpose.

Substeps in step 1

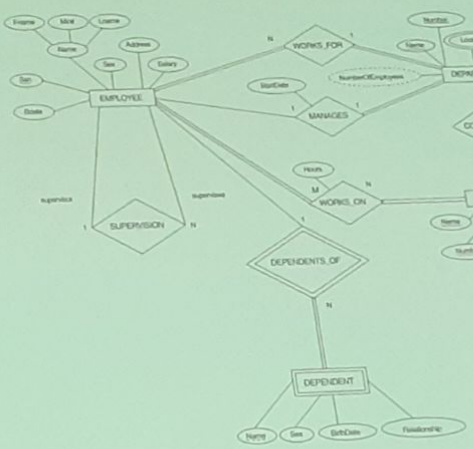
Functional Requirements

What operation do you want to perform upon the DBs ?

Func Requirements ends up becoming the Appcn Progrm.

What do you want to Store ?
What do you want to do with the stored data ?

This is how stage 1 Output looks like:



DBMS Design – Step 2

Write the **Schema**

Textual (Table kind of) expression of how the DB will be organized.

Schema is the logical layout of the DB.

This is how you imagine the information to reside in the Hard Disk

Note that there wont be any involvement of the client/customer at this stage, in general.

This is how stage 2 Output looks like:

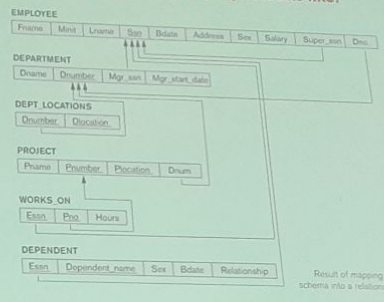


Figure 7.2
Result of mapping the COMPANY ER schema into a relational database schema.

Slix

DBMS Design – Step 3

Implementation

The DBMS software takes care of implementing & maintaining the DB at the physical level

As a programmer you take care of converting functional requirements to Application programs.

Tools for Step 1 & 2

Data Model:

It is a conceptual representation of how an enterprise's data is stored and organized in a DB. It provides a structured way to represent data, its relationships, and the rules governing it.

DM's helps to construct a comprehensive picture of an enterprise / company's data, which in turn can be used to create the actual database.

Data Model = Data Description (its like a blueprint to what you're going to implement later)

Note :

Data Model – Its Importance

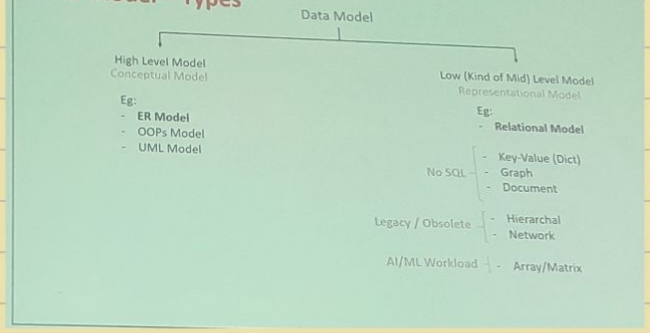
Provides a Blueprint: Helps in database design by visualizing the structure and relationships.

Ensures Data Integrity: Enforces rules to maintain consistency and accuracy.

Facilitates Communication: Acts as a bridge between developers, DBAs, and stakeholders.

Supports Query Optimization: Helps in structuring data for efficient access and manipulation.

Data Model – Types



Flow Chart ⟶ Algorithms

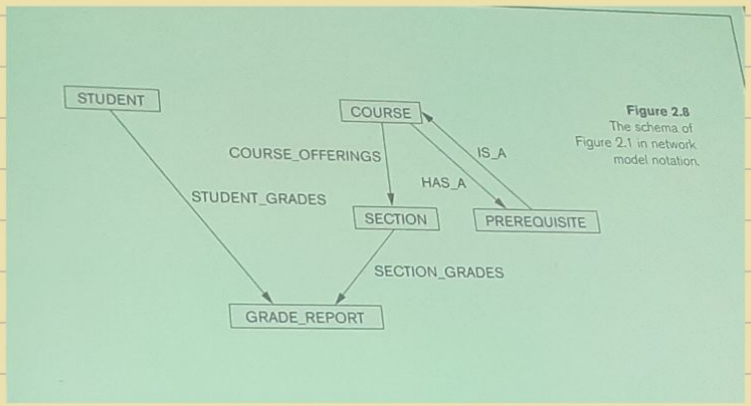
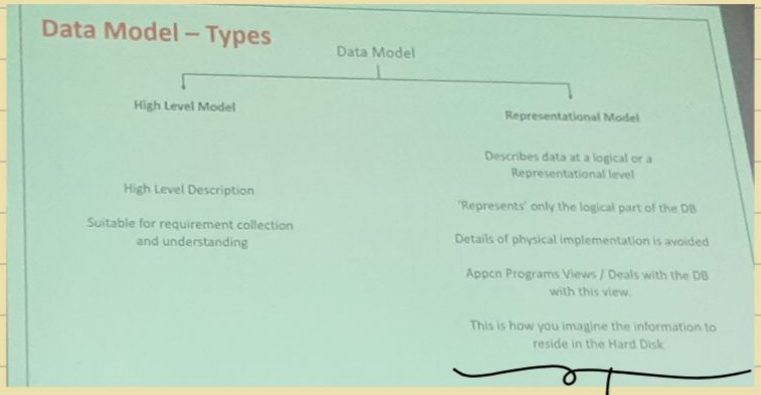


Figure 2.8
The schema of Figure 2.1 in network model notation.

Data Model – Types



Concerned with Design

Also known as 'Record Based Model'

Physical Data Model (Storage Model)

Describes how data is actually stored in the storage medium.

- Deals with record, files, and its formats.
- Exact data types, Length, default values are specified.
- Exact location, file size, indexing, sorting are taken care at this level.
- Data partioninig, buffer management.

Concerned with implementation

Summary:

- The role of Flowchart & Algorithms in problem solving as an Analogy to DBMS design
- Steps involved in DBMS Design.
- Data Models.
- Types of Data Models

21/1

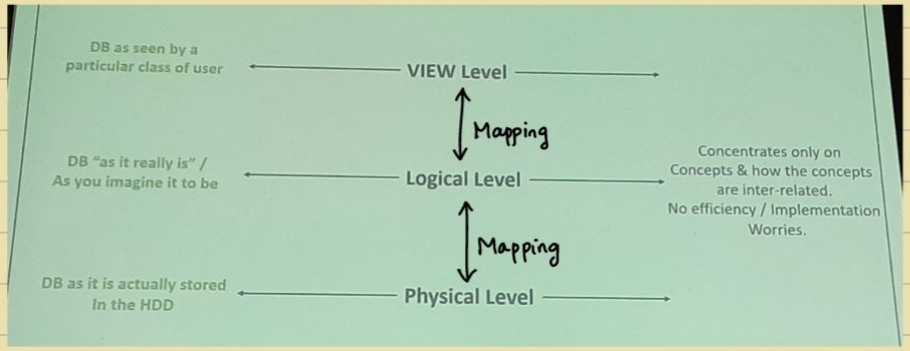
→ DBMS Architechture :

Chapter 1 & 2 of book

Three Schema Architecture	Concerned with Modelling and Data Independence
Client Server Architecture (Three – Tier Architecture)	Concerned with Application development
DBMS's Architecture	Concerned with internal Modules of DBMS itself

(1)

3 Levels of Abstraction :



Mapping → To achieve program-data independence

↳ Achieved through metadata

Remember, we start with Conceptual/logical Model, and then move on to other levels

External Schema ← **Conceptual Schema** → Internal / Physical Schema

Data Independence (DI)

The capacity to change the schema at one level of a database system without having to change the schema at the next higher level.

VIEW Level

Logical DI: is the capacity to change the conceptual schema without having to change external schemas or application programs

Logical/ Conceptual Level

Physical DI: is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well

Physical Level

The **mapping** between the levels helps to achieve DI

(ii) Three tier Architechture of DBMS Application :

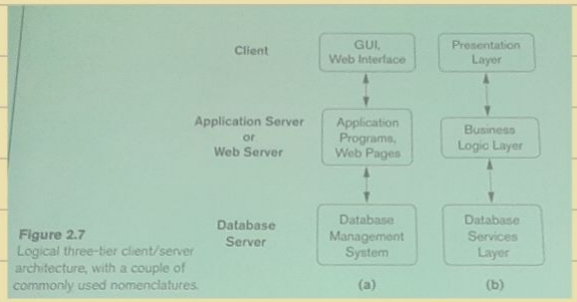
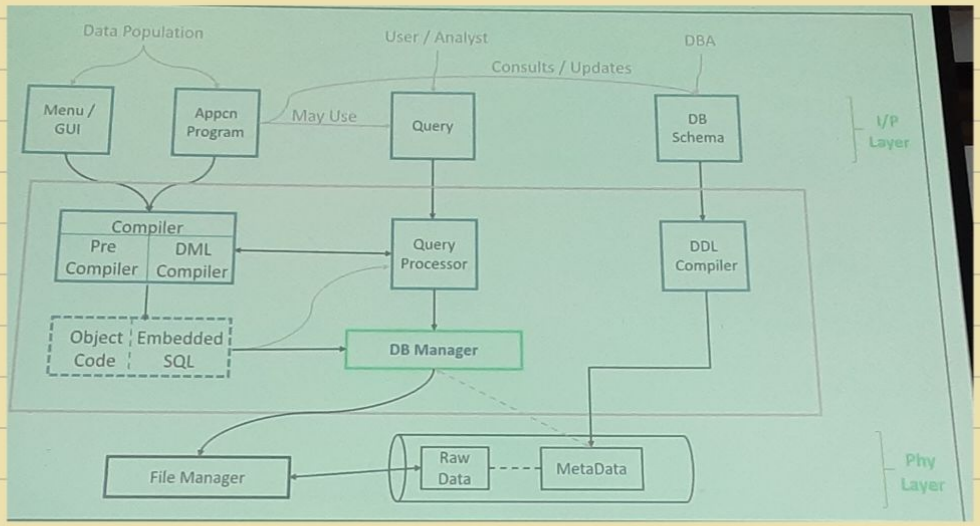


Figure 2.7
Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.

→ DBMS Architechture :



→ Major components of a DB system :

- DDL - Data definition Language
  Defines / Creates a schema

- DML - Data Manipulation Language
  (Retrieve , Insert , Update , Delete)

- VDL - View Definition Language

- DB Manager :

  All logics inside a package

Functionality	Details
Query Processing	Interprets, optimizes, and executes queries efficiently.
Storage Management	Manages physical data storage and buffer memory.
Transaction Management	Ensures ACID properties and handles concurrency.
Integrity Enforcement	Enforces data validity through constraints.
Security and Authorization	Controls user access and ensures data security.
Recovery Management	Restores database consistency after failures.

- DBA (DataBase Administrator) :

Typically its not one single person, it would be a team of people.

DBA's job is to decide exactly, what information is to be held in the DB (so that client's need can be served).

After identifying this he/she is the one who writes the schema with DDL.

DBA's in general have a comprehensive view of the entire DB and appcn programs.

Various Routines / Utilities that a DBA carries out or Supervise

Loading Routine	– To create the initial version of the DB.
Reorganization Routine	- Rearrange to reclaim space, avoid fragmentation, delete obsolete data.
Journaling Routine	- Log of each operations, with users & time-stamps.
Recovery Routine	- To restore during failures, backups.
Analysis Routine	- To monitor performance & improve Efficiency.

Read Section: 2.4.2 Database System Utilities – Chapter 2, Navathe

→ DBMS Interfaces :

Menu Based Interfaces

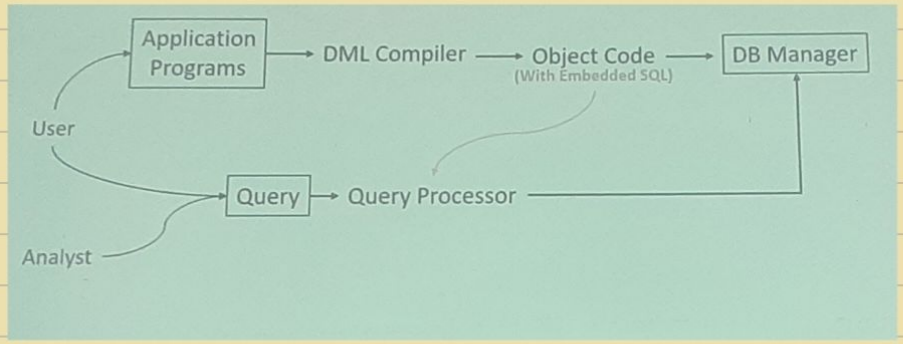
Form Based Interfaces

GUI Based Interfaces

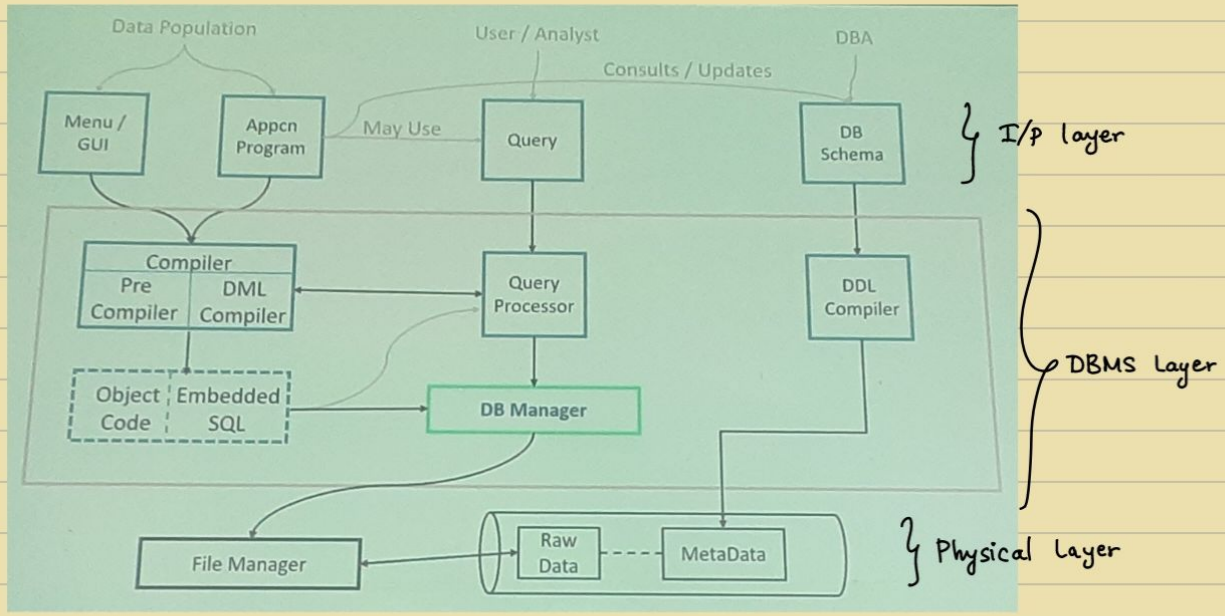
→ Major workflows in a DB :

① DBA → DB Schema → DDL Compiler → MetaData (Object code of Schema)

(2)



- Building everything, we get :



→ Next Chapter :

1. **Conceptual Modelling**
   - ER Modelling
2. **Relational Model**
   - Tables, Attributes, Schema...
   - Queries
     - Relational Algebra & Calculus
     - SQL
3. **RDBMS Design**
   - Functional Dependency
   - Normalization / Decomposition
4. **Transaction Management**
   - ACID Property
   - Concurrency...
5. **File Storage**
   - Indexing
   - B Trees...

21/1

→ ER Model :

↳ Not flowchart

But pictorial representation

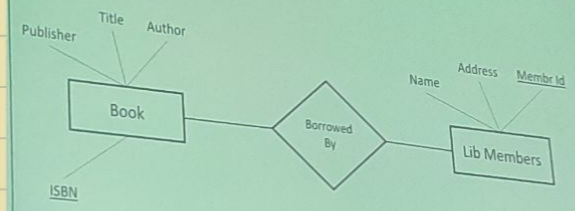
What is an ER Model ?

- It is a conceptual model/tool with which you conceive the broad structure of the DB. You identify the components of the system, and their relationship between them. Identify various constraints that the system should satisfy.
- Proposed by Peter P Chen in the 1970s

It is a pictorial description of the database.

- Simple enough for clients, yet Rigorous enough for system building.

How does an ER model Look:



Always Remember: Everything is ultimately a Table

BOOK

ISBN	Name	Auth

Borrowed By

Id	ISBN	DOI

Lib Members

Id	Name	Dept

→ Entity :

Thing with independent and distinct existence.

↳ Anything about which you would like to store information

What is an Entity ?

My Definition: 'Anything' about which you would like to store information.

Should have either Physical or Conceptual Existence

Tangible objects

Virtual or Logical

A Set of **distinguishable** items/objects

Examples of Entities

Physical Entities	Conceptual/Logical Entities
Student	Course
Teacher/Faculty	Department
Book	Subject
Vehicle	Payment
Product	Order
Building	Reservation
Machine	Project
Laboratory	Feedback
Office	Policy
Employee	Role
Computer	License
Equipment	Subscription
Patient	Appointment
Customer	Invoice
Animal	Membership
Factory	Work Shift
Library	Fine
Warehouse	Inventory
Room	Event
Store	Transaction

Entity vs Entity Set

Book

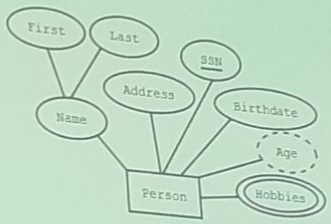
The Alchemist, Paulo Coelho...
Dune, Frank Herbert...
Sherlock Holmes, Sir Arthur Conan Doyle...
Feluda, Satyajit Ray...

Each row is a 'Book' with same set of attributes, (i.e. Book Name, author,...) but with different values.

Entity Set :

Collection of entities of the same type

Attributes



What do you Store ?

Entity: A thing about which you would like to '**store**' information.

What is an Attribute ?

↳ Name of the column

These are **Features/Properties** which helps to characterize or distinguish an Entity

It is a piece of Information which distinguishes **each row** in a table.

Domain of an Attribute

The set of possible values an attribute of an Entity can take.

Phone_Num – Any 10 Digit number.
Name – String of max length 250
Email – String matching the format: *<username>@<domain>.<extension>*

Gender – Enum: {'Male', 'Female', 'Other'}
Date_of_Birth – Date in the format *YYYY-MM-DD*
Salary – Decimal value with up to 2 decimal places, greater than 0.

What is an Attribute ? - Continued

All entities in an Entity Set have the same set of Attributes

Formally,
Attribute is a mapping from the Entity Set to a domain of values

Book —Title→ String
Book —ISBN→ 12 Digit Num
Book —Year→ 4 Digit Num

Types of Attributes

- **Simple Attributes –** Atomic, Indivisible
- **Composite Attributes –** Has Multiple components
- **Derived Attributes -** Depends on other attributes
- **Multi Valued Attributes**
- **Complex Attributes –** Composite + Multivalued.

PRIME Attributes (KEYS in DBMS)

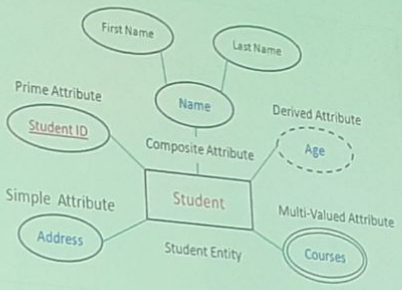
These are special attributes which helps to uniquely identify a row in a table.

Examples:
Aadhar Card,
Roll Num,
ISBN

What about (Aadhar + Roll Num) ? → SuperKey

Prime Attributes need not be always a single attribute.

Types of Attributes



PRIME Attributes

- Candidate Key
- Super Key
- Primary Key

Candidate Key

StudID	Roll No.	First Name	Last Name	Email
1	21	Tom	Cox	abc@gfg.org
2	22	John	Butler	xyz@gfg.org
3	23	Alice	Peterson	mno@gfg.org

Primary Key

Super key vs Candidate Key

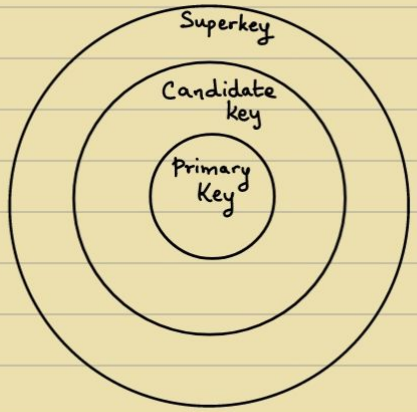
Superkey: A set of one or more attributes that can uniquely identify a row in a table.

Candidate Key: A minimal superkey, (it has no redundant attributes)

EXAMPLE:
Students(Roll_no, Name, Email)

Superkeys: {Roll_no},
{Email}
{Roll_no, Name},
{Roll_no, Email},
{Email, Name},
{Roll_no, Name, Email}

Candidatekeys: {Roll_no},
{Email},



Why do we need a Key

StudID	Roll No.	First Name	Last Name	Email
1	21	Tom	Cox	abc@gfg.org
2	22	John	Butler	xyz@gfg.org
3	23	Alice	Peterson	mno@gfg.org

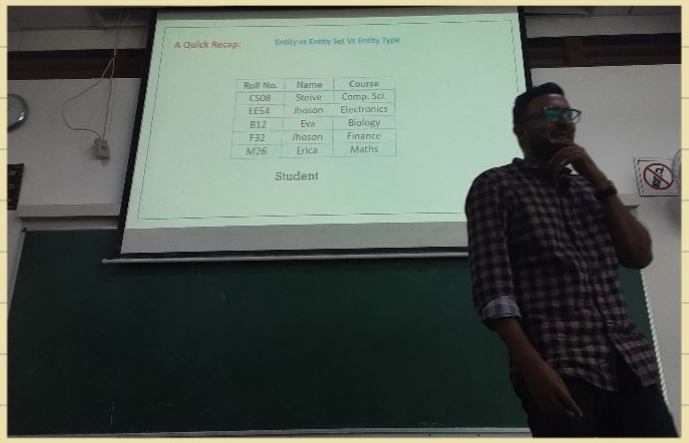
Say you want to edit this Last Name
How do you even access it at the first place ?

Key is just a Name for the rows

Keys are designed by the designers not from the instance.

Upcoming : Relationships ⟶ Cardinality ⟶ Special Cases

23/1



A Quick Recap: Entity vs Entity Set Vs Entity Type

	Roll No.	Name	Course	
Entity Set	CS08	Steive	Comp. Sci.	Entity
	EE54	Jhoson	Electronics	Entity
	B12	Eva	Biology	Entity
	F32	Jhoson	Finance	Entity
	M26	Erica	Maths	Entity

Student

You can build several Superkeys (by adding attributes)

↑

You find out the candidate keys

↓

You choose one out of it to be the Primary Key

Students(Roll_no, Name, Email)

{Roll_no},
{Email}
{Roll_no, Name},
{Roll_no, Email},
{Email, Name},
{Roll_no, Name, Email}

↑

{Roll_no}, {Email},

↓

I choose, Roll_no to be the Primary Key

What is a Relationship ?

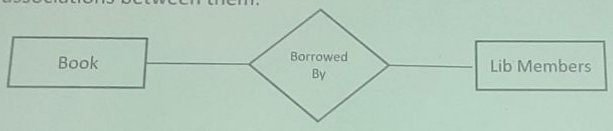
How normal people see relationships

How database engineers see relationships

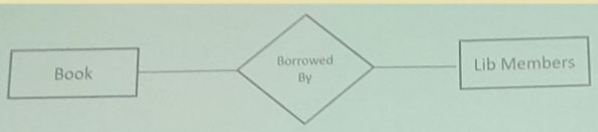
What is a Relationship ?

It is an association / Mapping between the entities

Entities by themselves requires information to be stored on their behalf. Additionally oftentimes, there is also a need to maintain information about the associations between them.



Borrowed by, is a relationship between the participating entities Book and Members



Formally,

Borrowed by $\subset$ Book X Members

The relationship *Borrowed_By* is a subset of the cartesian product between the entities Book and Members

What is a Cartesian Product:

A:

a1	a2
1	2
3	4
5	6

B:

b1
7
8

A × B ⇨

a1	a2	b1
1	2	7
1	2	8
3	4	7
3	4	8
5	6	7
5	6	8

Person:

firstname	*gender*
Fred	M
Sue	F

Person × Person ⇨

firstname	*gender*	*firstname*	*gender*
Fred	M	Fred	M
Fred	M	Sue	F
Sue	F	Fred	M
Sue	F	Sue	F

↳ Suesue

ISBN	Name	Auth
1234	Promised Neverland	Kaiu Shirai
5678	Ponniyin Selvan	Kalki

Id	Name	Dept
s1	Ram	CS
s2	Shyam	AI

Cross Product – Shows all possible associations / combinations

Cross Product – Shows all possible associations / combinations

ISBN	Name	Auth	Id	Name	Dept
1234	Promised Neverland	Kaiu Shirai	s1	Ram	CS
1234	Promised Neverland	Kaiu Shirai	s2	Shyam	AI
5678	Ponniyin Selvan	Kalki	s1	Ram	CS
5678	Ponniyin Selvan	Kalki	s2	Shyam	AI

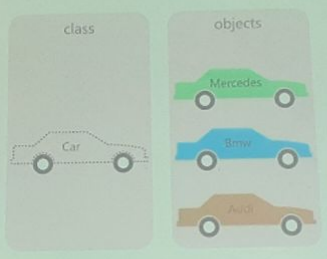
Note: Relationship is a **SUBSET** of this cross product → All rows need not be necessarily va

ISBN	Name	Auth	Id	Name	Dept	DOI
1234	Promised Neverland	Kaiu Shirai	s1	Ram	CS	
1234	Promised Neverland	Kaiu Shirai	s2	Shyam	AI	
5678	Ponniyin Selvan	Kalki	s1	Ram	CS	
5678	Ponniyin Selvan	Kalki	s2	Shyam	AI	

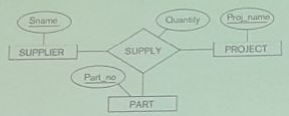
↳ Row itself ↳ Name of the table

Relationship vs Relationship Type / Set

Same Story

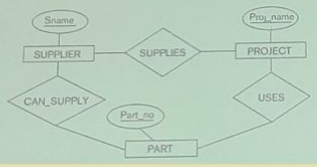


Degree of a Relationship



Ternary Relationship

Not equivalent

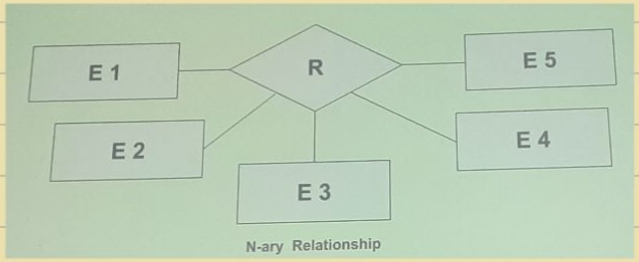


Binary Relationships

- **Ternary Relationship**:
  A supplier provides a specific part for a specific project.
  Example: Supplier "A" provides "Bolt" for Project "X".

- **Three Binary Relationships**:
  - Supplier "A" is supplying to Project "X".
  - Supplier "A" supplies "Bolt".
  - Project "X" uses "Bolt".

  Problem: No way to specify that "A" supplies "Bolt" **only** for "X".



N-ary Relationship

- Cardinality Ratio :

Cardinality : no. of elements in a set (in Math)

Cardinality Ratio

தமிழ் இல் In English

In database management, cardinality **represents the number of times an entity of an entity set participates in a relationship set.** Or we can say that the cardinality of a relationship is the number of tuples (rows) in a relationship. 30 Sept 2024

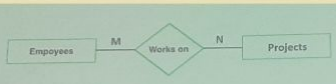
GeeksforGeeks
https://www.geeksforgeeks.org

Cardinality in DBMS - GeeksforGeeks

Often used to describe "Constraints" on a Binary Relationship

Consider ***borrowed_by*** relationship:
A book can be borrowed only by one person at a time.
But one person can borrow multiple books at a time.

C.Ratio helps to capture such kind of constraints



The maximum number of entities (Rows) from E1 that can be possible associated to E2 through R

Dictated by the usecase or the Policy of the organization

Cardinality Ratio

- One to One
- One to Many
- Many to One
- Many to Many

Constraint ⟶ Consistency

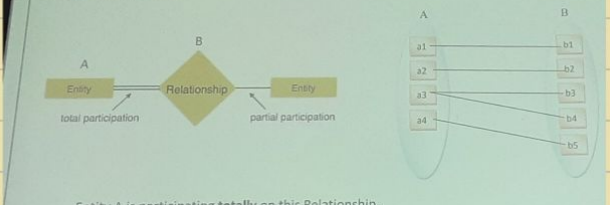
Participation Constraints

Participation constraint in an Entity-Relationship (ER) model defines the extent to which entities in an entity set are required to participate in a relationship set.

Simple Definition: It specifies whether all or only some of the rows in a Table/entity are involved in the relationship.

Participation Constraint == Minimum Cardinality Constraint

Total Participation



*Entity A is participating **totally** on this Relationship.*
It means that, all rows of A should appear / take part in the relationship.

Structural Constraints:

Cardinality Ratio + Participation Constraints = Structural Constraints

It reflects / captures the policy of the Organization, and Needs to be enforced by the DBMS Software

Min – Max Notation

Cardinality Ratio - Bothers only about Maximum

Participation Constraints – Bothers only about Minimum

Go for **Min – Max Notation**

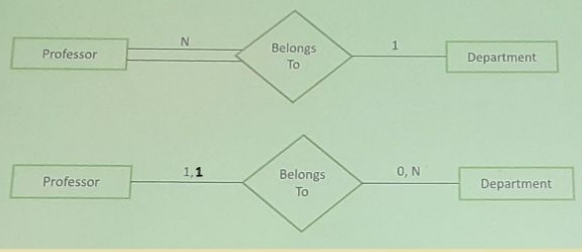
- Very Precise & Explicit
- Exact numbers can be used.

M : N

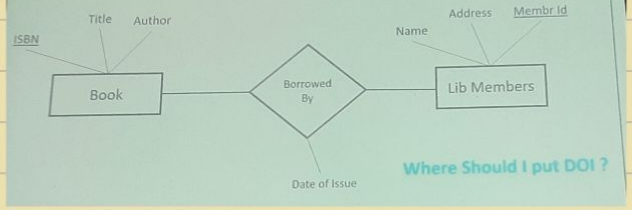
Min Time — Max Time

0 → Partial Participation
>1 → Total Participation

Min – Max Notation



Attributes of Relationships - Revisited

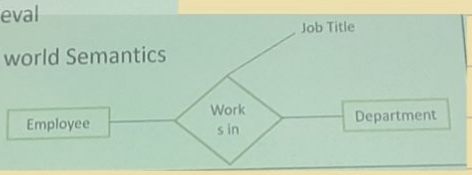


Where Should I put DOI ?

The cardinality Ratio gives a guideline to shift the Attributes of relationships

Why should I shift it at the first place ?

- Simplicity / Clarity
- Efficient query Retrieval
- Alignment with real world Semantics



Attributes Shifting - Guidelines

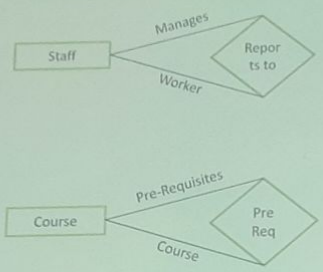
If it is a **1 to Many** Relationship - Shift to the Many side.

If it is a **1 to 1** Relationship - Shift to any side. (Total Participating side preferred)

If it is **Many to Many** Relationship - Cant be shifted.

M N

Recursive Relationships



Weak Entity

An entity set whose members "**Owe**" their existence to some other Entity.

Strong / Owner Entity

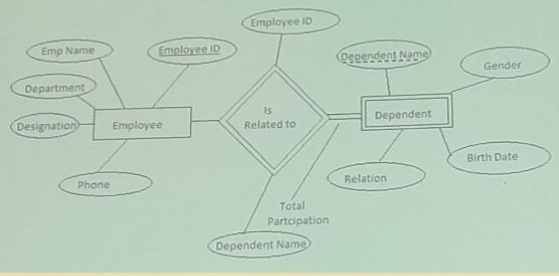
They cannot exist Independently. (Not enough attributes to distinguish them **Uniquely**)

No key, Only Partial Key

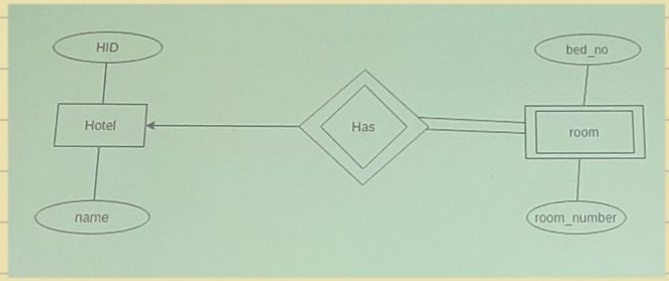
They are **Existential Dependent** – To exist they must depend on some other entity

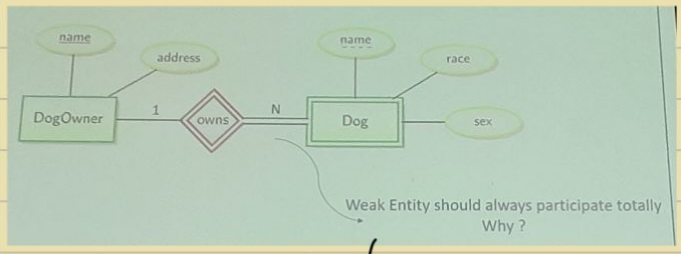
[Dependence of an attribute]

Weak Entity - Examples



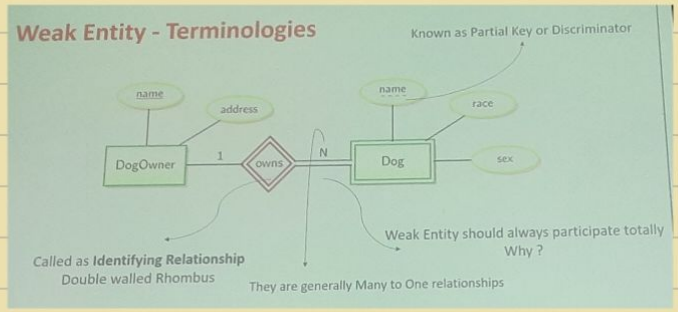
Partial key:
Unique in local scope



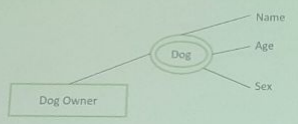


Must be associated with an attribute

Weak Entity - Terminologies



Weak Entity – As Composite Multi Valued Attribute



This can be done, when you are sure that the 'Dog' entity is not going to take part in a relationship with any other entity. In that case it is simple enough to be modelled as an attribute.

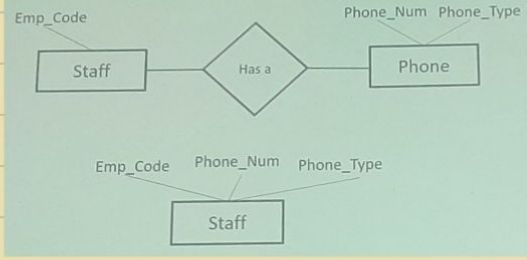
If a thing, even though not of independent existence, but participates on other relationships on its own
→ Best captured as a weak entity

- ## Entity Vs Relationships

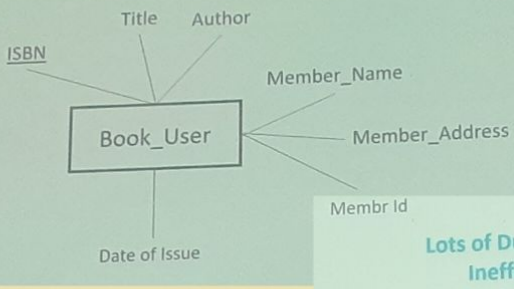
In some sense both of them are very similar.

- Both Entity and Relationships are ultimately converted into Tables
- Both of them have attributes

Example



Example



Lots of Duplication
Inefficient
Missing Values for Some Attributes

Good Approach: If all staff has a phone, and all phone is associated with a staff.
Bad Approach: Say if an office phone is general & No staff is associated to it → Go for Relationships

28/1

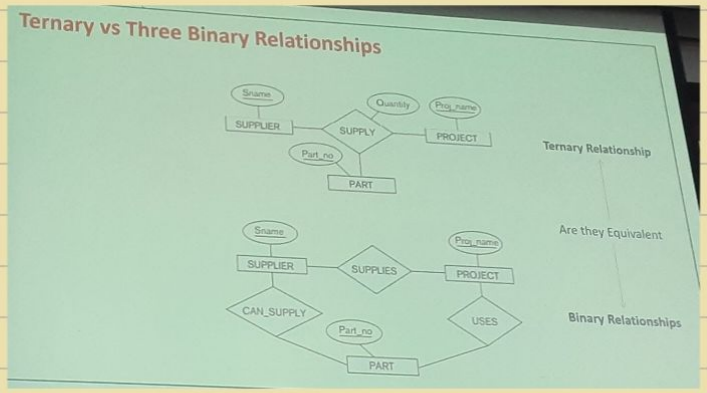
- Recap :

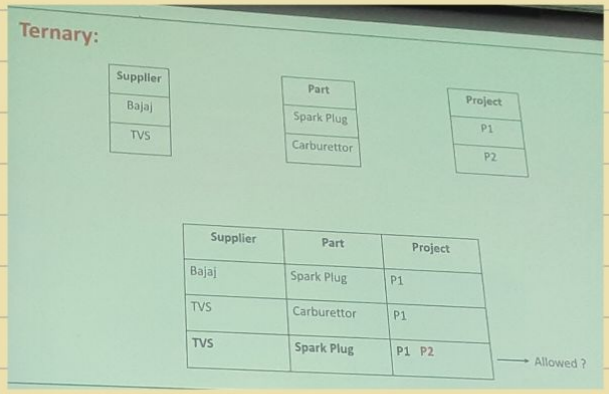
Relationships

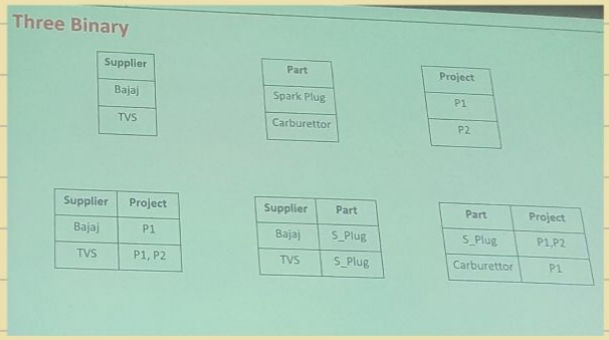
A relationship is a subset of a cartesian product

Cartesian Product: Lists out all possible associations / combinations

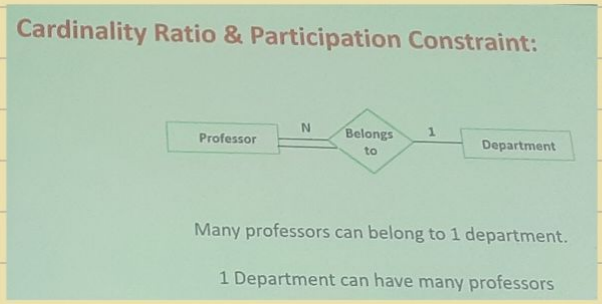
But only a subset of them will be true in reality.

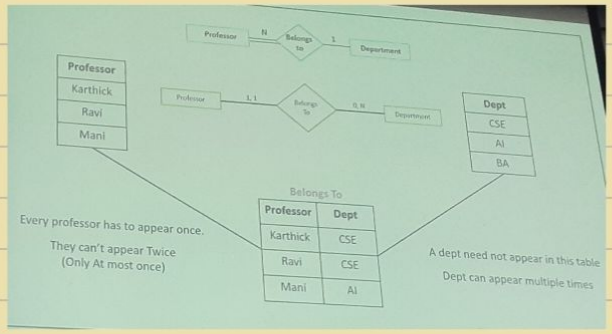


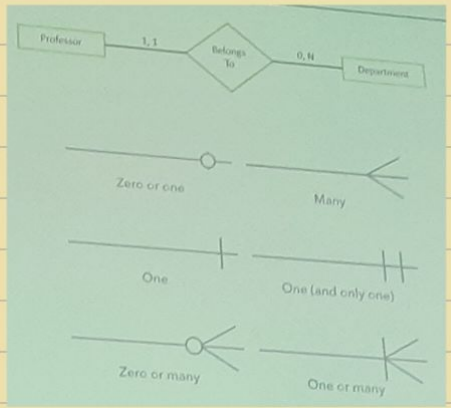




- **Cardinality Ratio & Participation Constraint:**

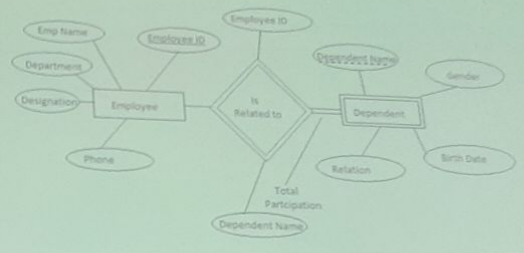




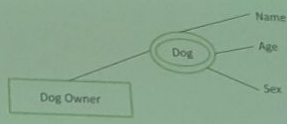


Software : Lucid Chart

Weak Entity - Examples



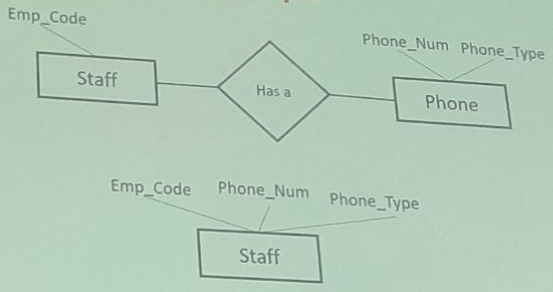
Weak Entity – As Composite Multi Valued Attribute



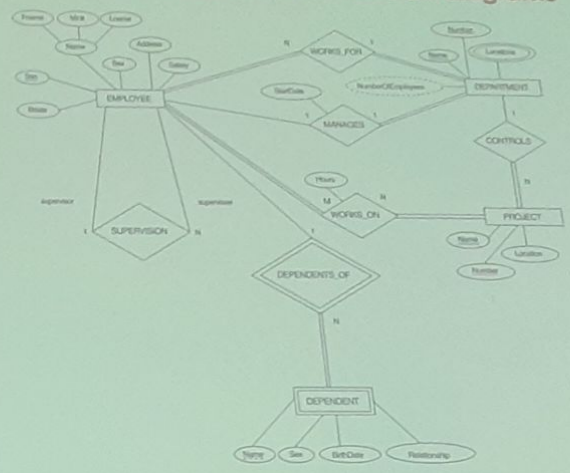
This can be done, when you are sure that the 'Dog' entity is not going to take part in a relationship with any other entity. In that case it is simple enough to be modelled as an attribute.

If a thing, even though not of independent existence, but participates on other relationships on its own
→ Best captured as a weak entity

Example



You should be able to build such ER diagrams



Relational Model - Next

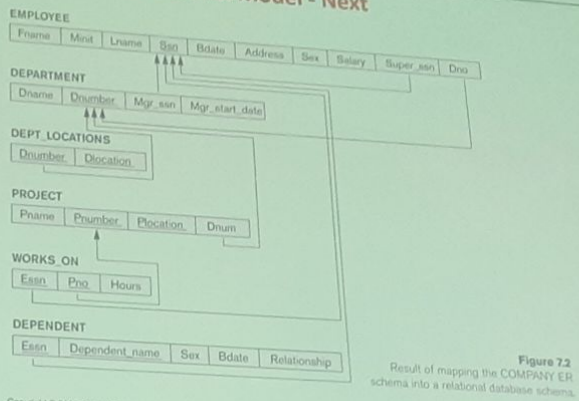


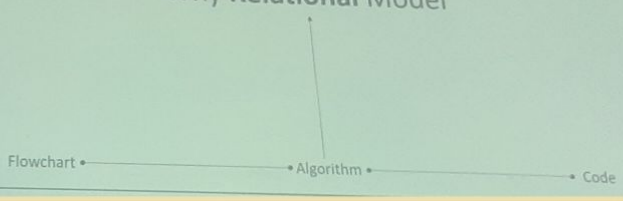
Figure 7.2
Result of mapping the COMPANY ER schema into a relational database schema.

Slide

Relational Model

From the Realm of Rectangles and Rhombuses to the World of Rows and Columns

Why **Relational** Model



Chapter organisation :

- Intro & Terminologies
- RM formally defined (Set Theory basis)
- Components of Relational Model
- Constraints on RM
  - Basic Constraints
  - Key
  - Foreign Key

What is a Relational Model ?

- It is a conceptual model/tool which uses collection of "Tables" to represent data & its Inter-relationships.
- Proposed by Edgar F. Codd (Ted Codd) in the 1970s

RM is based upon Set Theory and First Order Predicate Logic

Relational Model, allows data to be organized in such a way that SQL queries can be easily executed on them.

Table is a set

SQL : Sofisticated set operations

Ordering is not important.

RM, In simple terms :

Everything is a Table here

The entire DB is a Collection of Tables

In RM, everything is a SET, → Set operations can be easily performed upon them.

Relation ⟷ Table

Informally A Relation is a Table of Values (Rows)

Relation Name → STUDENT

Attributes

Tuples

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

Figure 5.1
The attributes and tuples of a relation STUDENT.

What is a Tuple ?

Simple Definition: Tuples are Just "**Rows**" in a table

Formal Definition: Tuple is an ordered sequence of values $<v_1, v_2, .. v_i, .. v_n>$

$t[a_i]$ refers to the value of v_i corresponding to attribute a_i

Tuple ⟷ Row

Example:

STUDENT

	Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
u	Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
t	Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25
v	Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
.	Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
.	Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21

u[Ssn] = 422-11-2320

t[Age,Gpa] = 19, 3.25

Example:

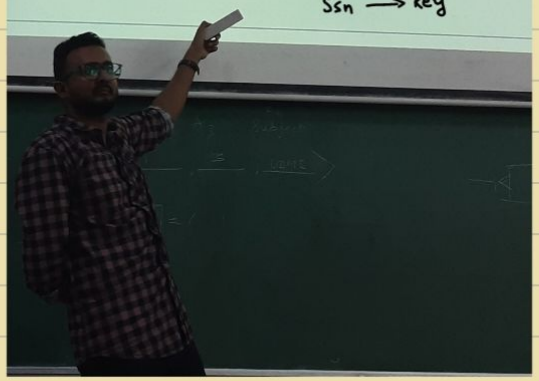
STUDENT

	Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
t_1	Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
t_2	Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25
t_3	Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
.	Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
.	Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21

t_1[Ssn] = 422-11-2320

$t_i[Ssn] \neq t_j[Ssn]$ for all i and j; i≠j

Ssn ⟶ key



What is an Attribute

Its just the name of the column, which helps to record information

Every attribute has a "**Domain**", which is the set of possible values that attribute can take.

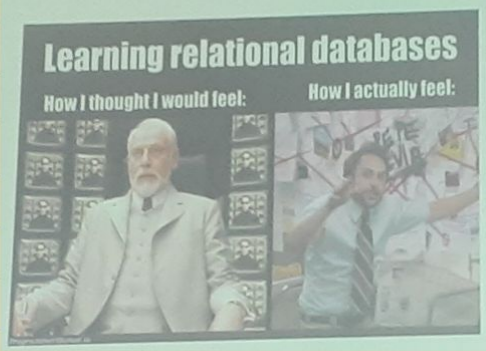
As per RM, Domain should be a set of Atomic Values only

Attributes ⟷ Columns

[No composite or Multivalued attributes allowed]

29/1

Relational Model



Tuple Notation:

STUDENT

	Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
t_1	Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
t_2	Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25
t_3	Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
.	Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
.	Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21

$t_1[Ssn] = 422\text{-}11\text{-}2320$

Domain in ER Model Vs Relational Model

Phone: {1234}
{4321 , 2122}
{4412 , 2314 , 5564}

Name: {Levi | Null | Ackerman }
{George | W | Bush }
{Kalki | Null | Null }

Phone: {1234}
{4321}
{4412}

Name: {Levi Ackerman }
{George W Bush }
{Kalki}

Relational Database (**RDB**) – A DB that is organized based on the RM.

ER vs Relational Model

ER Model	Relational Model
High Level Conceptual / Pictorial Description of the DB	Logical description of the DB
For Users / Common People	For Programmers / DBA
Doesn't worry about retrieval / No SQL	Organized for Efficient SQL Execution
Primary focus is on Ideation & design	Focus is on storage, query execution, and manipulation
Abstract Representation of Data	Concrete, Implementable representation of Data

RM: In Formal Terms

RM Represents data as a collection of Relations

What is a Relation ??. Lets discuss that later.
As of now Relation is just a 'table' of values to us

Rows / Tuples represents facts related to real world entity.

The table name and the Column names (Attributes) helps to interpret the meaning of each value in a row

Advantages of Adopting Relational Model:

It is Simple and easy to understand.
In fact in some cases people skip, ER modelling and go straight to the RM.

Visualising data as a table is both intuitive to the developer and as well is concrete enough to be implemented.

Relational Model led to the development of Data Dependencies & Normalization.

Huge Improvement in bringing down Redundancy & Inconsistency

Relation : Subset of Cartesian Product

Example: Less Than (<) Relation

Let's define two sets:
- **Set A** = {1, 2, 3}
- **Set B** = {2, 3, 4}

The **Cartesian product** A×B contains all possible pairs:
A×B={(1,2),(1,3),(1,4),(2,2),(2,3),(2,4),(3,2),(3,3),(3,4)}

Now, let's define a **relation** R:
"An element from A is less than an element from B."

So, R will be:
R={(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)}

→ 1 and 2 are **related** by the property: 'lesser than'

The "Relation" in Relational Model:

Consider an Entity with few Attributes and its Corresponding Domains

Attribute_1 → D1
Attribute_2 → D2
Attribute_3 → D3

Relation $\subset$ D1 X D2 X D3

Relationship: It is a subset of the cross product between the participating entities.

Relation: It is a subset of cross product between the domain of its attributes.

The earlier definition of a relation can be *restated* more formally using set theory concepts as follows. A relation (or relation state) $r(R)$ is a **mathematical relation** of degree n on the domains $\text{dom}(A_1)$, $\text{dom}(A_2)$, ..., $\text{dom}(A_n)$, which is a **subset** of the **Cartesian product** (denoted by $\times$) of the domains that define R:

$$r(R) \subseteq (\text{dom}(A_1) \times \text{dom}(A_2) \times \ldots \times \text{dom}(A_n))$$

The Cartesian product specifies all possible combinations of values from the underlying domains. Hence, if we denote the total number of values, or **cardinality,** in a domain D by $|D|$ (assuming that all domains are finite), the total number of tuples in the Cartesian product is

$$|\text{dom}(A_1)| \times |\text{dom}(A_2)| \times \ldots \times |\text{dom}(A_n)|$$

This product of cardinalities of all domains represents the total number of possible instances or tuples that can ever exist in any relation state $r(R)$. Of all these possible combinations, a relation state at a given time—the **current relation state**—reflects only the valid tuples that represent a particular state of the real world. In general, as the state of the real world changes, so does the relation state, by being transformed into another relation state. However, the schema R is relatively static and changes *very* infrequently—for example, as a result of adding an attribute to represent new information that was not originally stored in the relation. Chapter 3, Section 3.1, Page 63,Navathe

So, What is a Relation In DBMS

It is an unordered set that contains the relationship of Attributes which represents the entities.

BOOK

ISBN	Name	Auth
1234	PS-1	Kalki
5678	Dune	Frank Herbert

This table shows the valid "relation" that exists between the author, book name and ISBN, for the book entity

Relational Model – What it is made up of ?

It Consists of:

- Relational Schema
- Relational Algebra
- Relational Calculus

1. What is a Relational Schema

A **relational schema** is the blueprint or structure that defines the organization of a relational database.

It specifies the **name of the relation (table)**, its **attributes (columns)**, and the **domain (type of values)** that each attribute can hold.

It may also include constraints, such as primary keys, foreign keys...

Simply: It's a description of the Table, without the data or rows

Relational Schema - Example

Student (Student_ID: INT, Name: CHAR(50), Age: INT, Department: CHAR(50))

This is the "Schema" of the Student table.

Information Conveyed by The Schema:

Relation Name: Student (i.e., table name)
Attributes: Student_ID, Name, Age, Dept

Domain of each attributes:
Student_Id: Integer
Name: It's a string of 50 Characters

Relational Schema - Example

Student (Student_ID: INT, Name: CHAR(50), Age: INT, Department: CHAR(50))

The Data which is going to populate this Student table with this particular schema, should conform to this specified format.

(501, Norman, 18, CSE)

(CS20D1009, Light Yagami, 19, AI) ← Invalid

Schema Vs Type Declaration

Student (Student_ID: INT, Name: CHAR(50), Age: INT, Department: CHAR(50))

```
struct Student {
    int Student_ID;
    char Name[50];
    int Age;
    char Department [50]
};
```

Both are Similar Isn't it

Relational Schema Vs DB Schema

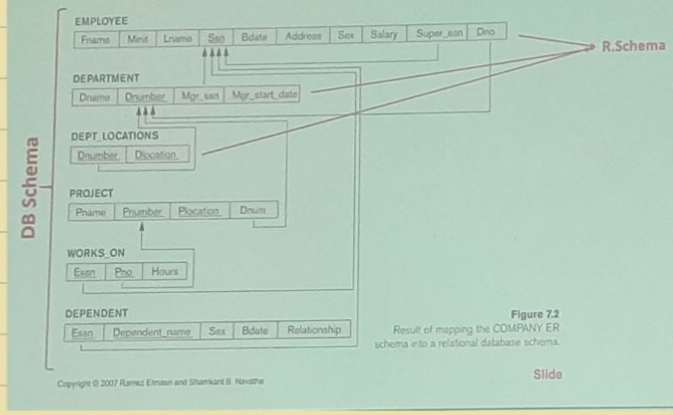


Figure 7.2
Result of mapping the COMPANY ER schema into a relational database schema.

Slide

Relational Schema

R.Schma is denoted as $R(A_1,...,A_i,...,A_N)$ N → Degree of the Relation

where, For all i, $A_i \in D_i$

r(R)

↳ Relation r follows the Schema R

This is R.Schema

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21

One Possible Relation State

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

Another Possible Relation State

Relation

Denoted by 'r(R)' ——— 'r' is of the schema $R(A_1,...,A_i,...,A_N)$

A Relation should always follow or Correspond to a R.Schema

Class → Object

R.Schema (Intension)

Book_Scheme(ISBN, Title, Author,...)

Often stays static.
Rarely changes.

Relation (Extension)

Book_in_First_floor(Book_Scheme)
Book_in_Second_floor(Book_Scheme)

It may change often (i.e. contents may change)
It reflects real world state.

This is a DB Schema

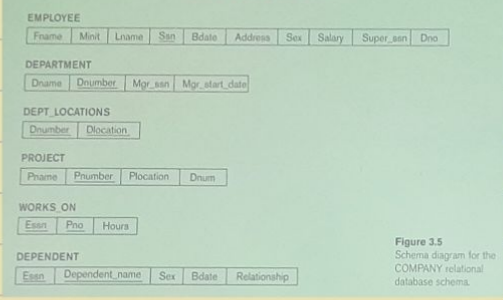
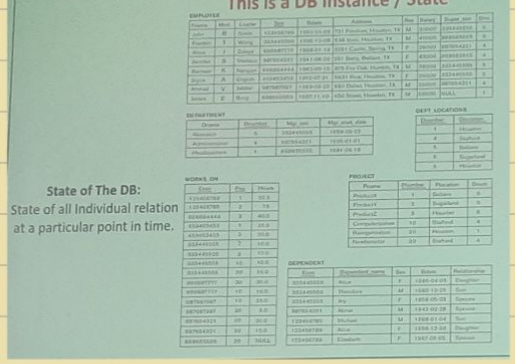


Figure 3.5
Schema diagram for the COMPANY relational database schema.

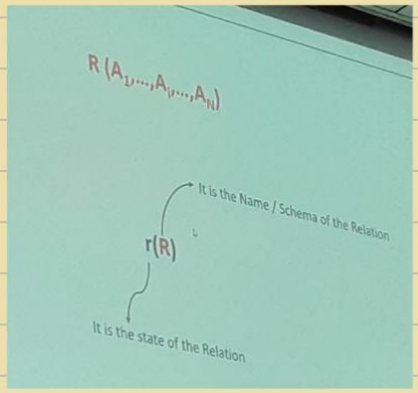
This is a DB Instance / State

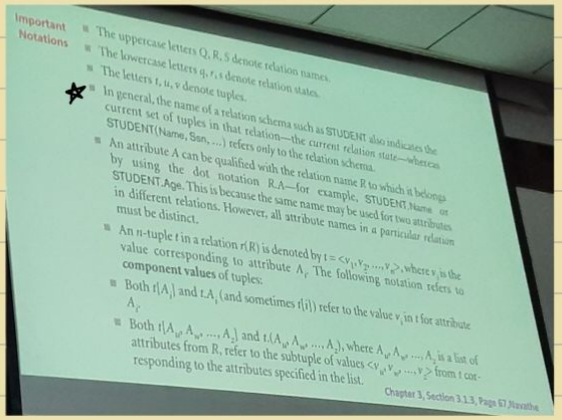


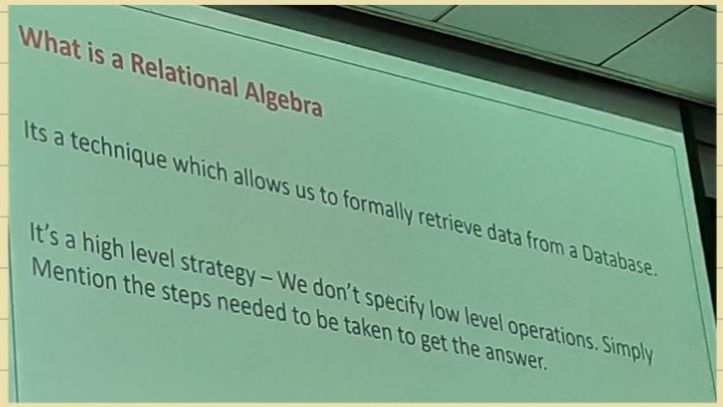
State of The DB:
State of all Individual relation at a particular point in time.

29/1 Lab : Use Internet/ ChatGPT/ whatever & Use LucidChart → Library Management system

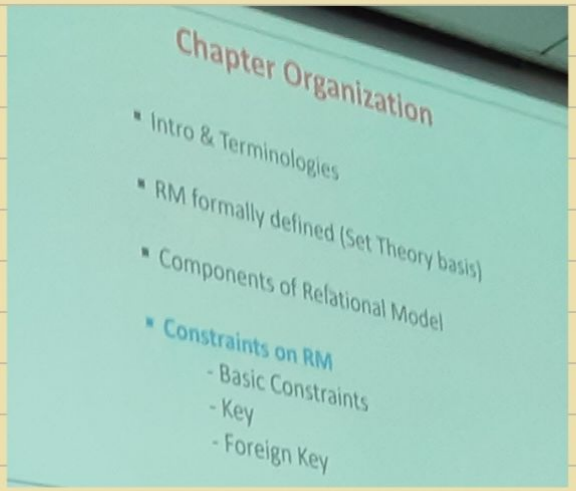
30/1

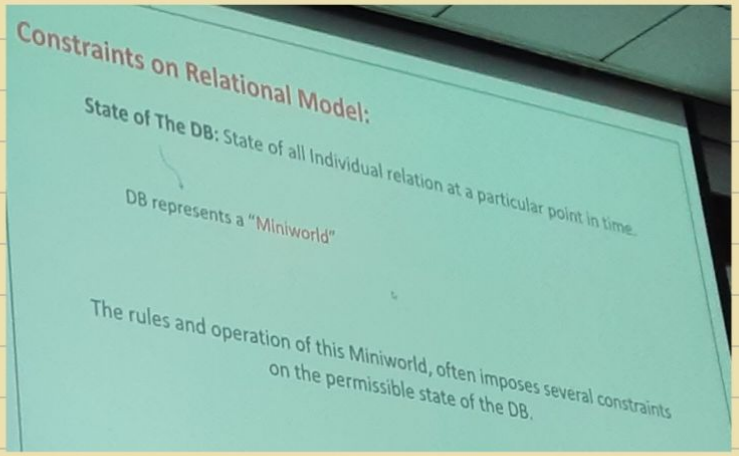


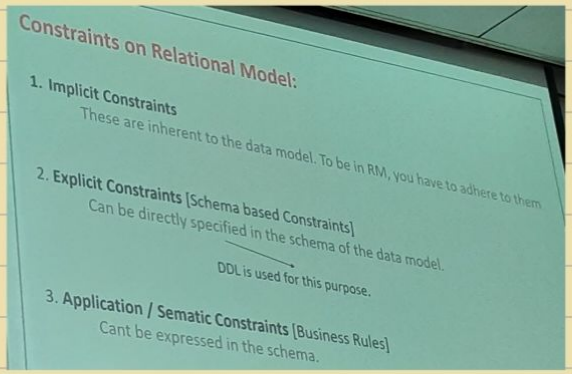


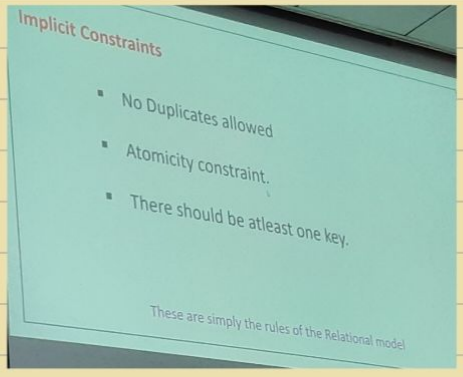


$$\pi_{<Name>}\left(\sigma_{<CGPA>9 \wedge GENDER=M>}^{(Students)}\right)$$









Explicit Constraints

1. Domain Constraint
2. Key Constraint
3. Entity Integrity Constraint
4. Referential Integrity

2. Key Constraint

By Definition: Relation is a **set** of tuples

No Duplicates

→ No two rows can have the same combination of values for all its attributes.

NOTE: A "subset of Attributes" may also hold this property.

→ super key $(t_1[SK] \neq t_2[SK])$

If two or more attributes (columns) are involved in a key, then it is called as **Composite Key**

Customer Id	Product Id	Quantity
121	9034	50
333	9440	67
121	3009	30
205	9446	67
225	9033	85

Super keys are in general composite keys

Super Key

The entire schema (i.e. all attributes together) is always a superkey.

Superkey can have **redundant** attributes.

Go for Minimal Superkeys (**Candidate Keys**)

Remove any of its attributes and it's no more a Valid key

Of all the candidate keys, one could be chosen as a **Primary Key** by the DBA.

KEY

- It is derived from the meaning of the attribute.
- It is time invariant.
- Key is a property of Relation Schema.

A key should hold on every valid relation state of the schema

3. Entity Integrity Constraint

- No primary key value can be NULL
- Also, no part of the primary value can be NULL

All the constraints we discussed till now are worried only about Single relation

4. Referential Integrity Constraint

- Specified between two relations
- Helps to maintain consistency between tables

Refers to

E_Name	Emp_Id	Dept No
Eren	E_180	12
Mikasa	E_180	15

Dept_Name	Dept_Id	Start Year
Scout Regiment	12	1450
Military Corps	13	1300

Invalid Tuple

↳ foreign key

Foreign Key

Dept No. refers Dept Id

E_Name	Emp_Id	Dept No
Eren	E_180	12
Mikasa	E_181	12

Referencing Relation

Dept_Name	Dept_Id	Start Year
Scout Regimen	12	1450
Military Corps	13	1300

Referred Relation

Invalid Tuple

A tuple in one relation that refers to another relation must refer to an "existing tuple" in that other relation

Foreign Key

Dept No. refers Dept Id

E_Name	Emp_Id	Dept No
Eren	E_180	12
Mikasa	E_181	12

Referencing Relation

Dept_Name	Dept_Id	Start Year
Scout Regimen	12	1450
Military Corps	13	1300

Referred Relation

Note: Dept_Id is a Primary Key

A F.Key should always refer to an attribute, which is a P.Key in that referred table.

Referred Relation

A F.Key should always refer to an attribute, which is a P.Key in that referred table.

Domain (F.Key) == Domain(P.Key(R_2))

Foreign Key (Table 1) $\subset$ Primary Key (Table 2)

You cant refer to a Department that doesn't exist.

F.Key helps to eliminate many inconsistencies.

E_Name	Emp_Id	Dept No
Eren	E_180	12
Mikasa	E_181	12

Related Tuples

Dept_Name	Dept_Id	Start Year
Scout Regimen	12	1450
Military Corps	13	1300

Tuple <Eren,E_180,12> and <Mikasa,E_181,12> refers to one "Scout_regimen" tuple.

F.Key helps to combine information in a meaningful way.

F.Key helps to answer queries involving two tables

In which year was the department where Eren works established?

To define referential integrity more formally, first we define the concept of a foreign key. The conditions for a foreign key, given below, specify a referential integrity constraint between the two relation schemas R_1 and R_2. A set of attributes FK in relation schema R_1 is a **foreign key** of R_1 that **references** relation R_2 if it satisfies the following rules:

1. The attributes in FK have the same domain(s) as the primary key attributes PK of R_2; the attributes FK are said to **reference** or **refer to** the relation R_2.
2. A value of FK in a tuple t_1 of the current state $r_1(R_1)$ either occurs as a value of PK for some tuple t_2 in the current state $r_2(R_2)$ or is NULL. In the former case, we have $t_1[FK] = t_2[PK]$, and we say that the tuple t_1 **references** or **refers to** the tuple t_2.

In this definition, R_1 is called the **referencing relation** and R_2 is the **referenced relation**. If these two conditions hold, a **referential integrity constraint** from R_1 to R_2 is said to hold. In a database of many relations, there are usually many referential integrity constraints.

EMPLOYEE: Fname, Minit, Lname, Ssn, Bdate, Address, Sex, Salary, Super_ssn, Dno

DEPARTMENT: Dname, Dnumber, Mgr_ssn, Mgr_start_date

DEPT_LOCATIONS: Dnumber, Dlocation

PROJECT: Pname, Pnumber, Plocation, Dnum

WORKS_ON: Essn, Pno, Hours

DEPENDENT: Essn, Dependent_name, Sex, Bdate, Relationship

F.Key can be within the same table

Figure 3.7
Referential integrity constraints displayed on the COMPANY relational database schema.

Constraint Violation

There are operations which can violate constraints.

Two types of operations

- Retrieval
- Update

Set of Relations (Tables) → Retrieval Operations → New Relation - Result

They do not affect any kind of Constraints

Update Operations

They can potentially violate Integrity Constraints (IC)

- Insert
- Delete
- Update (or Modify)

ICs should not fail when performed

Delete Operation

It can violate only Referential Integrity

Counter Measures

- Reject Deletion
- Cascade (Propagate deletion to referring table)
- Set Null (Problem if part of P.Key)

Update Operation

No problem if it is a Non-Key attribute.
Problem only if it is a key attribute or F.Key

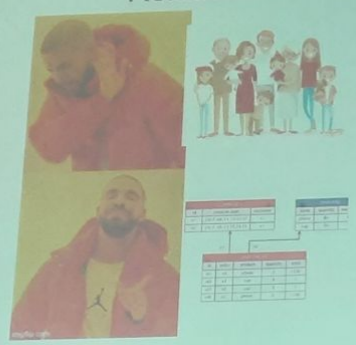
Counter Measures

Update == Delete + Insert

In DB development stage, when F.Key is specified, the DBA also specifies which strategy to follow if RIC is violated.

4/1

Relations



A Quick Recap

A Chronological Recap

We Started with **Introduction To Database Systems:**

- ❑ Problems with Early Information System (EIS)
- ❑ DBMS as a solution.
- ❑ Metadata and Schema for PDI.
- ❑ DBMS Architechture (3 versions)

A Chronological Recap

Designing a DBMS System:

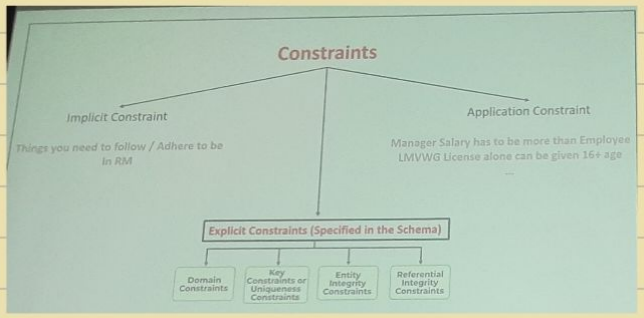
1. Visual / Conceptual Modelling
2. Mid-Level Model (Relational Model)
3. Physical Model (Storage)

Relational Model

It Consists of:

- Relational Schema
- Relational Algebra
- Relational Calculus

To be seen in the Upcoming Chapter



Constraints

Application Constraint

Is Total Participation an Application Constraint ?

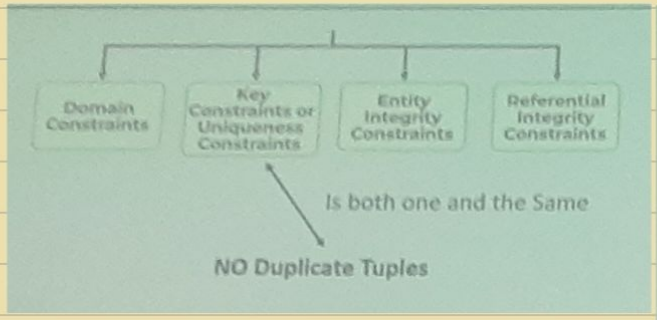
Yes, Total participation is an application constraint.
RM doesn't have any provision to directly enforce Total participation constraints.

Implicit Constraints

Implicit Constraint	Description
No Duplicate Tuples	Each tuple must be unique in a relation.
Atomicity of Attributes (1NF)	Each attribute must store atomic values (no multi-valued or composite attributes).
Tuples are Unordered	The order of tuples in a relation does not matter.
Attributes are Unordered	The order of attributes in a relation does not matter.
Unique Attribute Names	No two attributes in the same relation can have the same name.

These are simply the rules of the Relational model

Partial Participation ⟶ Not a constraint at all



NO Duplicate Tuples ⟷ Key Constraint

Definition: *The key constraint explicitly enforces that every relation must have a superkey (a set of attributes that uniquely identify each tuple).*

- *This means there must be at least one unique identifier (e.g., Primary Key).*
- *While the no duplicates rule ensures that the entire row is unique, the key constraint ensures that a specific attribute (or set of attributes) uniquely determines each row.*

What set of Attributes is enough to Make a row unique

NO Duplicate Tuples ⟷ Key Constraint

Though rows are guaranteed to be unique,
Key constraint is asking you to Explicitly declare a subset of Attributes as the Primary Key.

Though you know, rows are unique, its your job to choose and declare some attributes as the Primary Key.

2. Key Constraint

By Definition: Relation is a **set** *of tuples*

No Duplicates

➔ *No two rows can have the same combination of values for all its attributes.*

$$t_1[SK] \neq t_2[SK]$$

KEY

- *It is derived from the meaning of the attribute.*
- *It is time invariant.*
- *Key is a property of Relation Schema.*

A key should hold on every valid relation state of the schema

Foreign Key

Dept No, refers Dept Id

Note: Dept_Id is a Primary Key

E_Name	Emp_Id	Dept No
Eren	E_180	12
Mikasa	E_181	12

Referencing Relation

Dept_Name	Dept_Id	Start Year
Scout Regimen	12	1450
Military Corps	13	1300

Referred Relation

Foreign Key

Dept No, refers Dept Id

Note: Dept_Id is a Primary Key

E_Name	Emp_Id	Dept No
Eren	E_180	12
Mikasa	E_181	12

Referencing Relation

Dept_Name	Dept_Id	Start Year
Scout Regimen	12	1450
Military Corps	13	1300

Referred Relation

A F.Key should always refer to an attribute, which is a P.Key in that referred table.

Domain (F.Key) == Domain(P.Key(R_2))

Foreign Key (Table 1) ⊆ Primary Key (Table 2)

You cant refer to a Department that doesn't exist.
F.Key helps to eliminate many inconsistencies.

E_Name	Emp_id	Dept No
Eren	E_180	12
Mikasa	E_181	12

Related Tuples

Dept_Name	Dept_id	Start Year
Scout Regiment	12	1450
Military Corps	13	1300

Tuple <Eren,E_180,12> and <Mikasa,E_181,12> refers to one "Scout_regiment" tuple.

F.Key helps to combine information in a meaningful way.

F.Key helps to answer queries involving two tables
In which year was the department where Eren works established?

Foreign Key in SQL

```
CREATE TABLE Department(
D_No INT PRIMARY KEY ,
D_Name VARCHAR (100)
);
```

Foreign Key in SQL

```
CREATE TABLE Department(
D_No INT PRIMARY KEY ,
D_Name VARCHAR (100)
);

CREATE TABLE Employee(
Emp_ID INT PRIMARY KEY,
Emp_Name VARCHAR (100),
Dept_ID INT,
FOREIGN KEY (Dept_ID) REFERENCES Department (D_No)
);
```

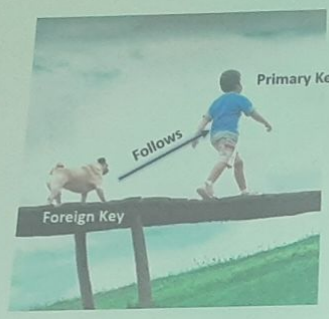
Foreign Key in SQL

```
CREATE TABLE Department(
D_No INT PRIMARY KEY ,
D_Name VARCHAR (100)
);

CREATE TABLE Employee(
Emp_ID INT PRIMARY KEY,
Emp_Name VARCHAR (100),
Dept_ID INT,
FOREIGN KEY (Dept_ID) REFERENCES Department (D_No)
);
```

In this side, you declare that I'm pointing to an attribute Of the other table.

What is special about this Line ?
It is more like a pointer in C

In F.Key why the arrow points towards the referred attribute ?



Foreign Key should always follow the Rules of P.Key to which it is referring to

Can a Foreign Key be NULL?

In an Employee table, a column Manager_ID (foreign key referencing Employee_ID) can be NULL for employees who do not have a manager.

If a record does not currently have a valid reference, the foreign key can be NULL to indicate the absence of a relationship.

Constraint Violation

There are operations which can violate constraints.

Two types of operations

- *Retrieval*
- *Update*

Set of Relations (Tables) → *Retrieval Operations* → *New Relation - Result*

They do not affect any kind of Constraints

Response / Countermeasure

Reject the insertion.
- *Provide reason for rejection.*
- *Raise an prompt, to direct the user to modify their input.*

If Referential Integrity is violated one option is to:
- *Set it to NULL*

Delete Operation

It can violate only Referential Integrity

Counter Measures

- Reject Deletion
- Cascade (Propagate deletion to referring table)
- Set Null (Problem if part of P.Key)

What if I Delete an Entire Column ?

If the Column is a Primary Key

If the deleted column is part of a Primary Key, removing it would violate the uniqueness and non-nullability constraints.

Handling:

- If it is a **single-column** primary key, either:
  - Assign a new primary key.
  - Avoid deletion unless a proper key replacement exists.
- If it is part of a **composite primary key**, ensure that another column can still uniquely identify records.

What if I Delete an Entire Column ?

If the Column is a Foreign Key in the Master Table

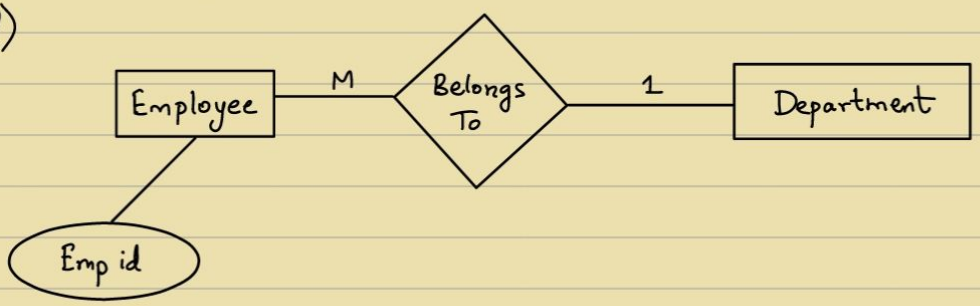
If the deleted column is referenced as a foreign key in another table, the referential integrity will be broken.

Handling:
First, remove the foreign key constraint before deleting the column.

Update dependent tables by either:
Dropping the foreign key.
Modifying the referencing table to use a different column.

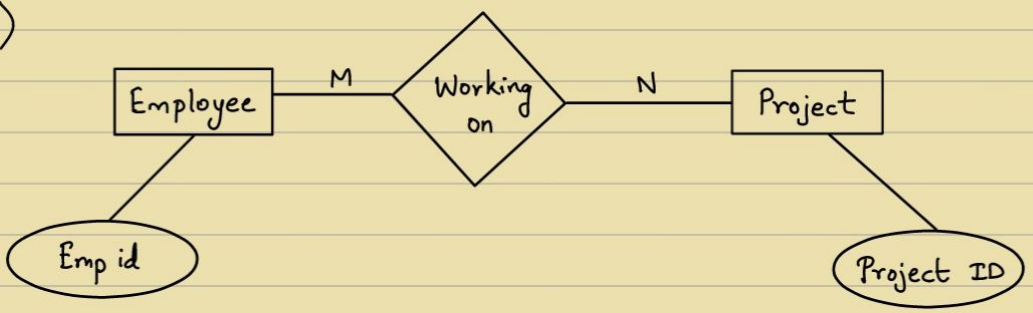
- ER Model to Relation Model (Schema):

Q)

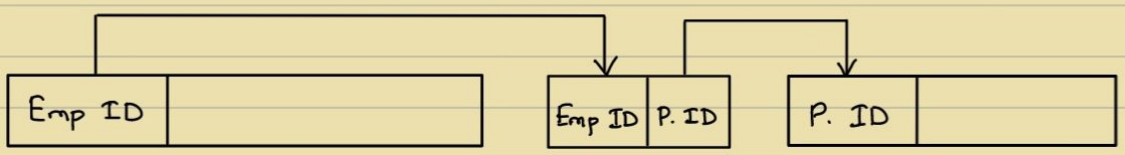


Sol

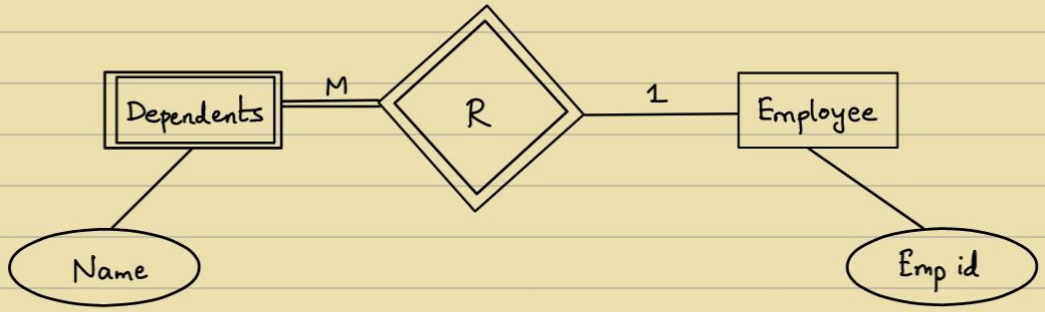
Q)



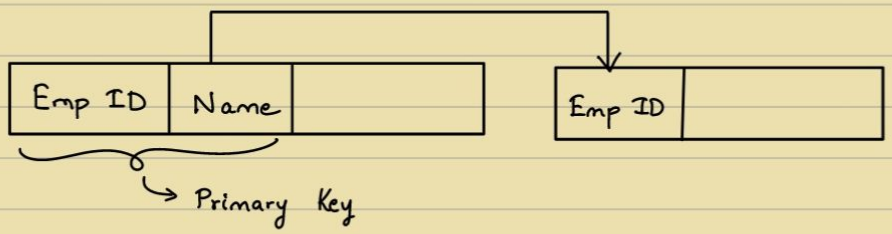
Sol



Q)



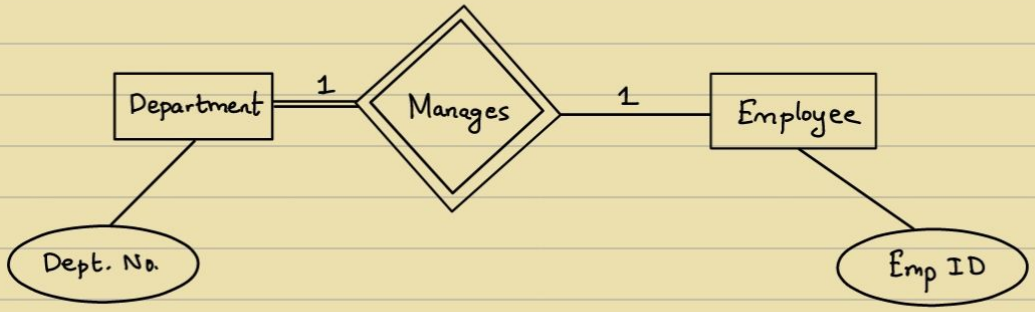
Sol



" Wife.. only 1 I can Have! "

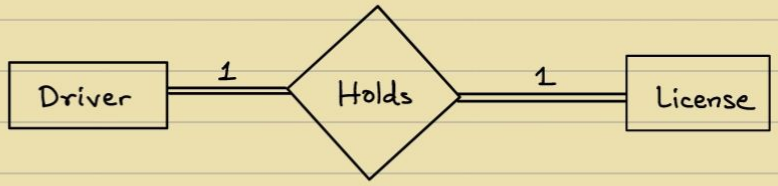
↳ ~ Shahul Hameed Sir

Q)



Sol

Q)

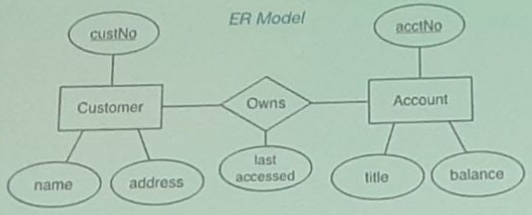


Sol

∵ One-one mapping

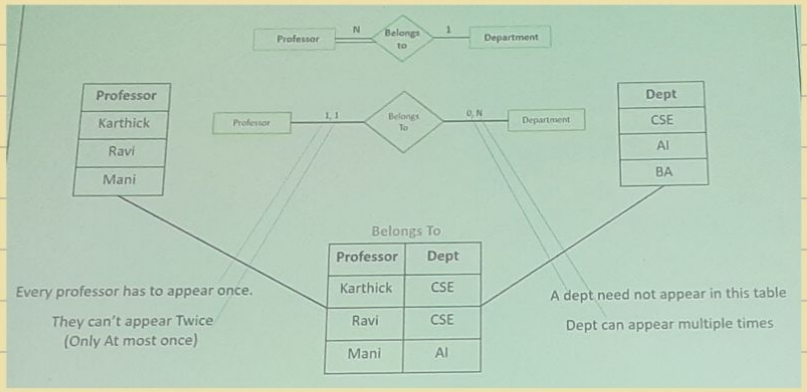
5/1

ER to Relational Model Conversion



Relational Version

Customer	custNo	name	address
Account	acctNo	title	balance
Owns	acctNo	custNo	lastAccessed



- Relational Algebra :

Relational Algebra & Calculus

A very General Introduction – What are they, How they Look ?

Constructing an House – 3 Possible Way

You hire an Civil
Contractor to build
The house for you

You Hire masons
& you plan and build the
House through them

Let me build my home
By myself.

I have modelled everything as Relations
Now what do I, Do with that ?

You need a set of Operators to perform some actions upon your model.

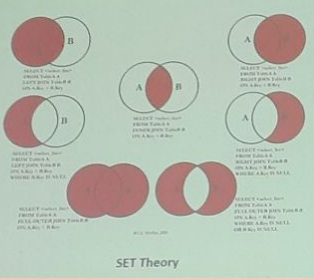
There are two formal languages for Relational Model:

Relational Algebra

Relational Calculus

RA & RC were developed before SQL.
SQL is based mostly upon RC, but also uses RA in some parts

RA & RC are Based upon



SET Theory

∧	*and*	[conjunction]
∨	*or*	[disjunction]
⇒	*implies*	[implication]
¬	*not*	[negation]
∀	*For all*	
∃	*There exists*	

First Order Predicate Logic

What is Relational Algebra ? (Briefly)

→ Perform operations on relations to get another relation

What is Algebra ?

It is a framework for representing problems and solving them systematically using operations like addition, multiplication, and subtraction

What is Relational Algebra ?

Relational Algebra is a formal system for manipulating relational data, based on mathematical algebra principles. It defines a set of operations (like selection, projection, union, and join) to query and manipulate relations (tables) in a database

RA is a set of operations that can be performed upon the Relational Model

Example

Aggregate operator

$$\pi_{Name}(\sigma_{Dept='CSE' \wedge Grade>80 \wedge Age<(\gamma_{AVG(Age)}(Student))}(Student))$$

→ What we retrieve → condition → From which table

Example

$$\pi_{Name}(\sigma_{Dept='CSE' \wedge Grade>80 \wedge Age<(\gamma_{AVG(Age)}(Student))}(Student))$$

1. $\gamma_{AVG(Age)}(Student)$ → Computes the **average age** of all students.
2. $\sigma_{Dept='CSE' \wedge Grade>80 \wedge Age<(\gamma_{AVG(Age)}(Student))}(Student)$ → Filters students who:
   - Belong to the **CSE department**.
   - Have a **Grade greater than 80**.
   - Have an **Age less than the computed average age**.
3. $\pi_{Name}(\ldots)$ → Extracts only the **Name** of such students.

```
SELECT Name
FROM Student
WHERE Dept = 'CSE'
AND Grade > 80
AND Age < (SELECT AVG(Age) FROM Student);
```

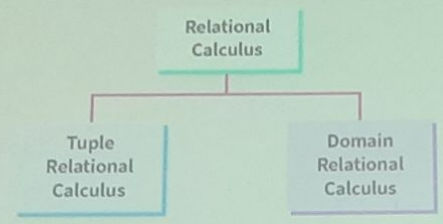
What is Relational Calculus ?

It is an Higher Level declarative language for specifying queries.

Specify only WHAT do you need. Doesn't specify HOW it is to be done.

Relational calculus doesn't prescribe any particular sequence of operations to be performed Which will ultimately yield the requested information.

Types of Relational Calculus



How do they Look ?

TRC vs DRC: Examples

Retrieve students with grades > 80

{t | t ∈ Students ∧ t.Grade > 80}

→ Tuple RC → The set of all tuples **t** such that **t** belongs to the **Students** relation and the **Grade** attribute of **t** is greater than 80.

{(Name) | ∃ Grade (Name, Grade ∈ Students ∧ Grade> 80)}

→ Domain RC → *The set of all* **Name** *values such that there exists a* **Grade,** *where* **(Name, Grade)** *belongs to the* **Students** *relation and,* **Grade** *is greater than 80.*

BASIS FOR COMPARISON	RELATIONAL ALGEBRA	RELATIONAL CALCULUS
Basic	Relational Algebra is a Procedural language.	Relational Claculus is Declarative language.
States	Relational Algebra states how to obtain the result.	Relational Calculus states what result we have to obtain.
Order	Relational Algebra describes the order in which operations have to be performed.	Relational Calculus does not specify the order of operations.
Domain	Relational Algebra is not domain dependent.	Relation Claculus can be domain dependent.
Related	It is close to a programming language.	It is close to the natural language.

Thus, **relational algebra** focuses on **sets of tuples**, while **relational calculus** can operate at the **attribute level**, making it domain-dependent.

RA Vs RC

•Relational algebra operates procedurally using a set of operations (like selection, projection, join, etc.) on entire relations (tables).

•It does not rely on the domains (data types) of attributes explicitly; rather, it **manipulates tuples as a whole**.

σ Grade > 80 (Students) → This retrieves all tuples where the Grade is greater than 80, without explicitly focusing on the domain of Grade.

•Domain Relational Calculus (DRC) specifically **works on individual attribute values**, making it domain-dependent.

{ Name | ∃ Grade (Name, Grade ∈ Students ∧ Grade > 80) } → This explicitly refers to **Name** and **Grade** attributes, relying on their domains for evaluation

Constructing an House VS Relational Algebra & Calculus

Relational Calculus

Relational Algebra

Writing raw code for Query.

6/2

9.1 Relational Database Design Using ER-to-Relational Mapping 293

Table 9.1 Correspondence between ER and Relational Models

ER MODEL	RELATIONAL MODEL
Entity type	?
1:1 or 1:N relationship type	?
M:N relationship type	?
n-ary relationship type	?
Simple attribute	?
Composite attribute	?
Multivalued attribute	?

Solution:

9.1 Relational Database Design Using ER-to-Relational Mapping 293

Table 9.1 Correspondence between ER and Relational Models

ER MODEL	RELATIONAL MODEL
Entity type	*Entity* relation
1:1 or 1:N relationship type	Foreign key (or *relationship* relation)
M:N relationship type	*Relationship* relation and *two* foreign keys
n-ary relationship type	*Relationship* relation and *n* foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key

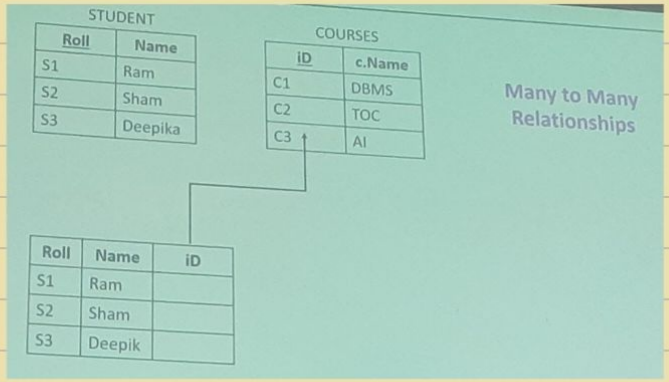
STUDENT

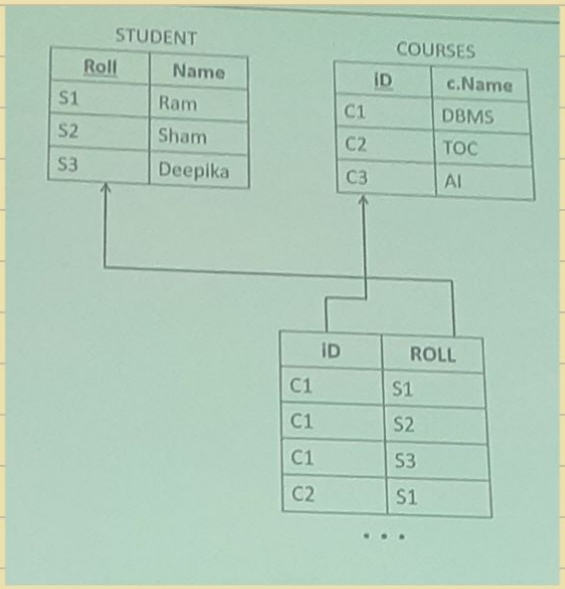
Roll	Name
S1	Ram
S2	Sham
S3	Deepika

COURSES

iD	c.Name
C1	DBMS
C2	TOC
C3	AI

Many to Many Relationships





Consider a Student Table with lots of Attributes: Stud_Id, Name, Email, Age, Gender, DOJ, Height, Weight.....

Query: Retrieve All student_ID whose age is between 20 and 25

RA : $\Pi_{S\_id}(\sigma_{Age>20 \wedge Age<25}(Students))$

$\Pi_{S\_id}(\sigma_{Age\ b/w\ 20\ \&\ 25}(\Pi_{S\_id,\ Age}(Students)))$ → More efficient

[OR]

RC : $\{t \mid t \in Students \wedge t.Age > 20 \wedge t.Age < 25\}$

Different Higher Level Strategies can be deployed

A single RA expression can be typically written in multiple ways.

One way can be *better* than the other, in terms of efficiency

A RC Expression can *typically* be specified in only one way.

You just declare what you want, You don't specify the sequence of operations.

- $\{x \mid x > 5\}$: Set of all x S.T. $x > 5$

  Similarly,

{t | condition}

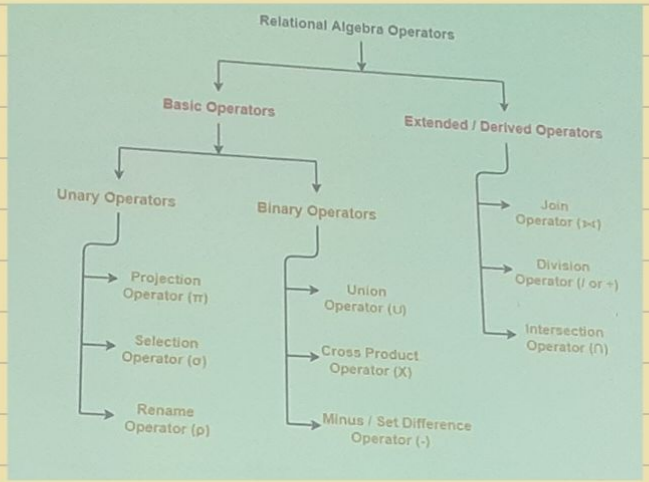
The set of all tuples t that satisfy the given condition.

The Braces helps to explicitly declare that, the collection of tuples, satisfying the condition will be a SET.

Then Why no Braces for RA Expressions ?

Remember RA is just a set of operators, which takes relations as inputs and gives another relation as Output.

A relation is a SET by definition, so it is redundant to explicitly wrap them with Braces.



Select Operator (σ)

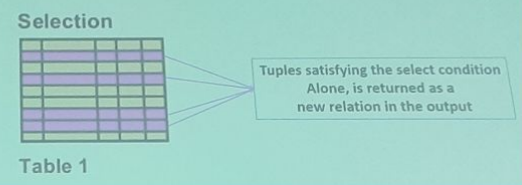
$$\sigma_{<\text{selection condition}>}(R)$$

$$\sigma_{(Department='CSE' \wedge Grade \geq 60)}(Students)$$ Say 60 is the pass mark

SQL Query :

```
SELECT *
FROM Students
WHERE Department = 'CSE' AND Grade >= 60;
```

Select operator, **Horizontally partitions** the Table.



Different Operators that can be used in the Select Condition

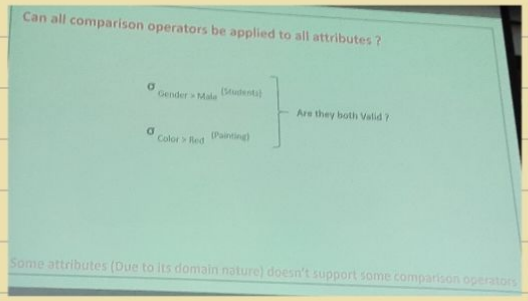
- = (Equal to)
- ≠ (Not equal to)
- < (Less than)
- > (Greater than)
- ≤ (Less than or equal to)
- ≥ (Greater than or equal to)

Comparison Operator

- ∧ (AND)
- ∨ (OR)
- ¬ (NOT)

Logical Operator

$\sigma_{Gender > Female}(Student)$ } X
Name > Krishna



If the domains are "**Ordered Values**" then all the comparison operator's are applicable.

Integer, Float, Date, Grades are of ordered domains

Set of all colors, Name, Address, these things have no meaningful order.
Bob > Alice → doesn't make any sense

{=, ≠} alone is applicable for unordered domains

How an Select expression is processed ?

The select condition is applied independently to each individual tuple *t* in R.

All tuple *t*, which satisfies / passes the condition appear in the result.

Select is Unary ➔ Applied to only one single relation.
It is applied to each tuple individually.

- ## Selectivity:

The fraction of tuples selected by a selection condition is referred to as the selectivity of the Condition.

Consider a Students table with 10,000 records.
σ Age > 18 (Students) •If 9,000 students are older than 18, the selectivity = 9000 / 10000 = 0.9 (90%) •This means the condition is not very selective (i.e., it retrieves most of the records).
σ Age = 21 (Students) •If only 500 students are exactly 21, the selectivity = 500 / 10000 = 0.05 (5%) •This means the condition is highly selective (i.e., it filters out most records)

Is Selection Operator Commutative ?

$$\sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C2}(\sigma_{C1}(R))$$ Are they both same ?

Does applying condition **C1** first and then **C2** gives the same result as applying **C2** first and then **C1**_?

$\sigma_{Age>20}(\sigma_{Grade='A'}(Students))$

$\sigma_{Grade='A'}(\sigma_{Age>20}(Students))$

Selection **only filters tuples** and does not change their structure. Since both C1 and C2 remove unwanted tuples, applying them in any order produces the same final set of tuples

- ## Project Operator (π)

$$\pi_{<\text{attribute list}>}(R)$$

$$\pi_{\text{Student\_ID, Name}}(Students)$$

```
SELECT Student_ID, Name FROM
Students;
```

$\pi_{\text{Student\_ID, Name}}(Students)$

Are the both Same ?

$\pi_{\text{Name, Student\_ID}}(Students)$

- ## Vertical Partitioning of the Table

Projection

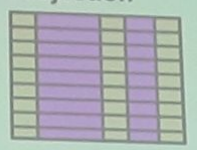


Table 1

Project Operator (π)

$$\pi_{\text{Name, Age}}(\sigma_{\text{Grade}>80}(Students))$$

Display the Desired Columns — Select the Desired Tuples

```
SELECT Name, Age
FROM Students
WHERE Grade > 80;
```

- ## Is Project Operator Commutative ?

Students (Student_ID, Name, Age, Grade)

$\pi_{\text{Student\_ID, Name}}(\pi_{\text{Student\_ID, Name, Age}}(Students))$

$\pi_{\text{Student\_ID, Name, Age}}(\pi_{\text{Student\_ID, Name}}(Students))$

SQL Counterpart

$\pi_{\text{Student\_ID, Name}}(\pi_{\text{Student\_ID, Name, Age}}(Students))$

```
SELECT Student_ID, Name FROM (
    SELECT Student_ID, Name, Age FROM Students
)
```

Nested Sub Queries

What about Duplicates ?

Characters

Name	Gender	Age
Senku Ishigami	Male	15
Hermione	Female	14
Saitama	Male	26
Paul Atreides	Male	15

$\pi_{\text{Gender}}(\text{Characters})$

Gender
Male
Female

$\pi_{\text{Gender, Age}}(\text{Characters})$

Gender	Age
Male	15
Female	14
Male	26

→ This row Appear Only once

Project Operator (π)

$$\pi_{\text{Name, Age}}(\sigma_{\text{Grade}>80}(Students))$$

→ You have to explicitly instruct in SQL To remove duplicates

```
SELECT DISTINCT Name, Age
FROM Students
WHERE Grade > 80;
```

Select Operator and Duplicates

Does Select Operator remove duplicates ?

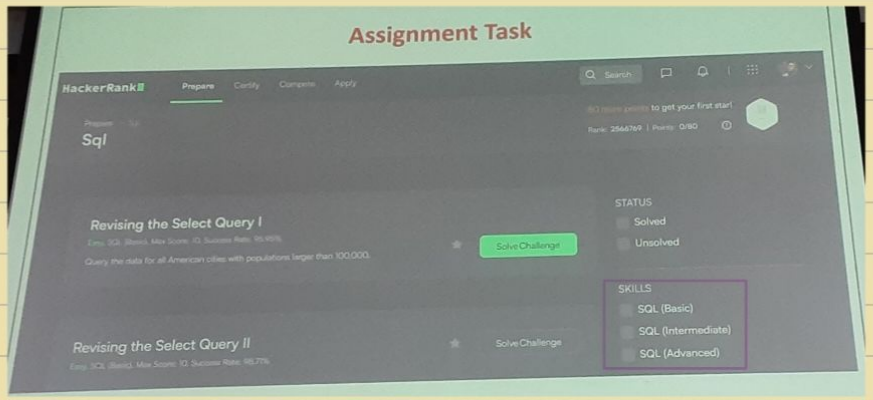
Select Operator and Duplicates

The SELECT operation (σ) in relational algebra does not remove duplicates because it only filters tuples based on a condition.

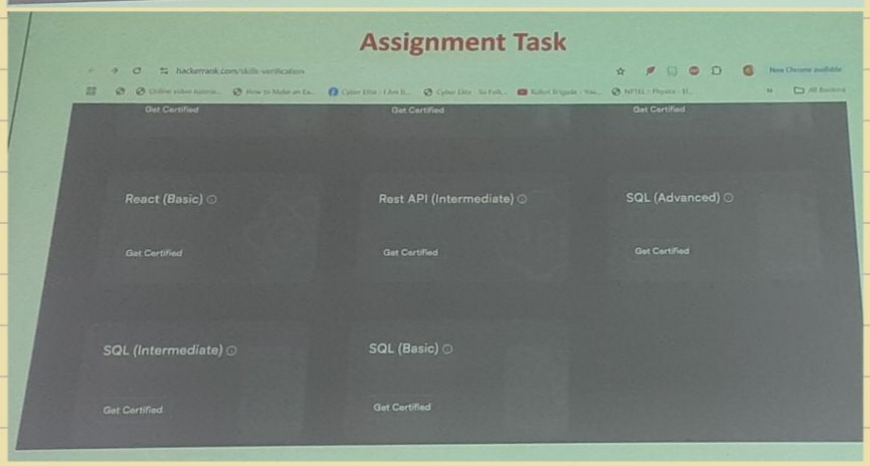
The result contains all tuples that satisfy the given condition, including duplicates if they exist in the original relation.

7/1

Assignment Task



Assignment Task



Quick Review

In RA in goes a Relation and outcomes another Relation.

➔ No need of explicit braces

RC on the other hand need explicit usage of braces to indicate set of tuples

One Query – Multiple Strategies / Approaches

$$\pi_{Student\_ID}(\sigma_{Age>20 \wedge Age<25}(Students))$$

$$\pi_{Student\_ID}(\sigma_{Age>20 \wedge Age<25}(\pi_{Student\_ID,Age}(Students)))$$

Project

Allows you to choose
Columns you are interested in

Select

Allows you to choose
Rows you are interested in

First Name	Last Name	Address	City	Age
Mickey	House	123 Fantasy Way	Anaheim	73
Bat	Man	321 Cavern Ave	Gotham	54
Wonder	Woman	987 Truth Way	Paradise	39
Donald	Duck	555 Quack Street	Mallard	65
Bugs	Bunny	567 Carrot Street	Rascal	50
Wiley	Coyote	999 Acme Way	Canyon	61
Cat	Woman	234 Purrfect Street	Hairball	32
Tweety	Bird	543	Itotlaw	28

Together, you get complete Two-Dimensional control over the Relation.
You can choose any cell of interest.

Some Comparison operators cannot be used with some attributes.

Vijay > Ajith
Messi > Ronaldo
Sachin > Kohli

→ Does it make any Sense ?

SQL Correspondence

$$\pi_{\text{Name, Age}}(\sigma_{\text{Grade}>80}(Students))$$

Display the Desired Columns — **Select the Desired Tuples**

```
SELECT DISTINCT Name, Age
FROM Students
WHERE Grade > 80;
```

Characters

Name	Gender	Age
Senku Ishigami	Male	15
Hermione	Female	14
Saitama	Male	26
Paul Atreides	Male	15

π_{Gender} (Characters)

Gender
Male
Female

$\pi_{\text{Gender, Age}}$ (Characters)

Gender	Age
Male	15
Female	14
Male	26

→ This row Appear Only once

How do you deal with lengthy RA Expressions

$$\pi_{Student\_ID,Name}(\sigma_{Age>20}(\pi_{Student\_ID,Name,Age}(Students)))$$

How do you deal with lengthy RA Expressions

$$\pi_{Student\_ID,Name}(\sigma_{Age>20}(\pi_{Student\_ID,Name,Age}(Students)))$$

Store this as an intermediate relation
In a relation named as "**Temp**"

$$\rho_{Temp}(\pi_{Student\_ID,Name,Age}(Students))$$

$$\pi_{Student\_ID,Name}(\sigma_{Age>20}(Temp))$$

Rename Operator (ρ)

$$\rho_{NewName}(Expression)$$

The result of the expression will be renamed as "NewName"

$$\rho_{NewName(A1,A2,\ldots,An)}(Expression)$$

You can also explicitly rename each of the attributes, if you wish

Rename Operator (ρ)

```
SELECT Temp.Student_ID, Temp.Name

FROM (
      SELECT Student_ID, Name, Age
      FROM Students
      WHERE Age > 18
) AS Temp

WHERE Temp.Student_ID LIKE 'S%';
```

```
SELECT Student_ID, Name, Age
FROM Students
WHERE Age > 18 AS Temp
```

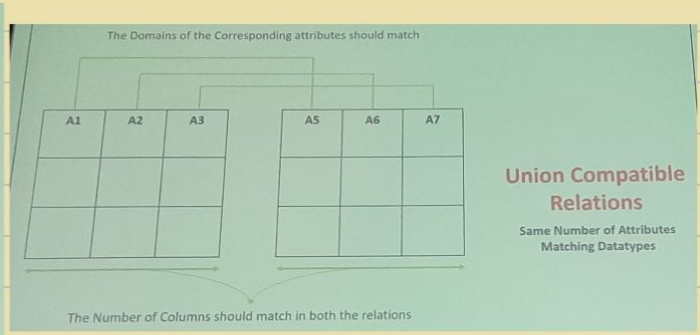
Filters students who are above 18 & Store it as Temp

Union Operator (U)

$$R_1 \cup R_2$$

SELECT column1, column2, ...FROM table1
UNION
SELECT column1, column2, ...FROM table2;

Can you apply Union operator between any two relations ?



Union Compatible Relations

Same Number of Attributes
Matching Datatypes

$$\pi_{Student\_ID,Name,Year}(Students\_AI) \cup \pi_{Student\_ID,Name,Year}(Students\_DD)$$

Student_ID	Name	Year
101	Aakash	2023
102	Bhavna	2023
103	Chaitanya	2024
104	Divya	2024
105	Eshwar	2024

$$\pi_{Student\_ID,Name,Year}(Students\_AI) \cup \pi_{Student\_ID,Name,Year}(Students\_DD)$$

Here Both Students_AI and Students_DD table are Union Compatible

$$\pi_{Student\_ID,Name,Year}(\text{Students}) \cup \pi_{Course\_ID,Course\_Name}(\text{Courses})$$

Students and Courses are not Union Compatible

Other Set Operators

- INTERSECTION: The result of this operation, denoted by $R \cap S$, is a relation that includes all tuples that are in both R and S.
- SET DIFFERENCE (or MINUS): The result of this operation, denoted by $R - S$, is a relation that includes all tuples that are in R but not in S.

Course_1

C_id	C_name
11	Foundation C
21	C++
31	JAVA

Course_2

C_id	C_name
12	Python
21	C++

Course_1 ∩ Course_2

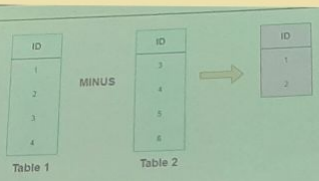
C_id	C_name
21	C++

Note: Both the Tables should be "Union Compatible" for Intersect to work

SQL Counterpart

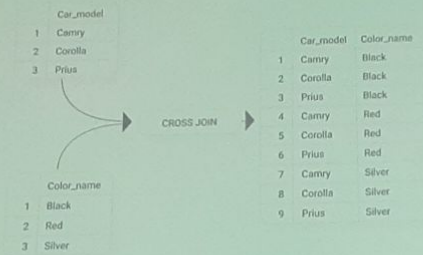
```
SELECT column1, column2, ... FROM TableA
INTERSECT
SELECT column1, column2, ... FROM TableB;
```

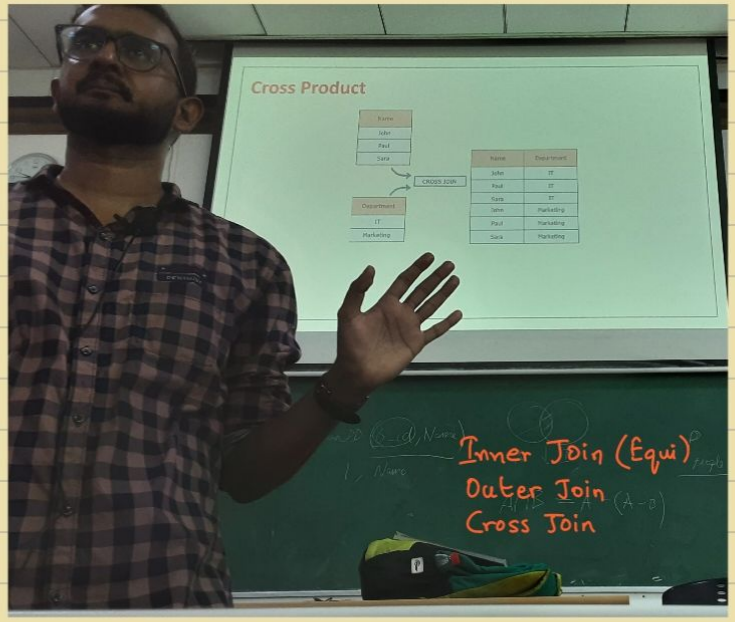
```
SELECT Student_ID, Name, Age
FROM Students_AI
WHERE Age > 21

INTERSECT

SELECT Student_ID, Name, Age
FROM Students_DD
WHERE Age > 21;
```



Note: Both the Tables should be "Union Compatible" for Minus to work

Cross Product





Remember the case, where we wanted To combine related tuples of two Different tables

E_Name	Emp_id	Dept No
Eren	E_180	12
Mikasa	E_181	12

Related Tuples

Dept_Name	Dept_id	Start Year
Scout Regimen	12	1450
Military Corps	13	1300

In which year was the department where Eren works established?

How do you combine tuples belonging two tables But are related ?

You can take a cross product between both the relations.

This will create a very big table, where most of the rows are meaningless.

And then use a Select condition to filter out only the meaningful rows.

Employee

E_Name	Emp_id	Dept No
Eren	E_180	12
Mikasa	E_181	12

Department

Dept_Name	Dept_id	Start Year
Scout Regimen	12	1450
Military Corps	13	1300

Big Table = Employee X Department

E_Name	Emp_id	Dept No	Dept_Name	Dept_Id	Start Year
Eren	E_180	12	Scout Regimen	12	1450
Eren	E_180	12	Military Corps	13	1300
Mikasa	E_181	12	Scout Regimen	12	1450
Mikasa	E_181	12	Military Corps	13	1300

These two rows Are Meaningless.

Big Table

E_Name	Emp_id	Dept No	Dept_Name	Dept_Id	Start Year
Eren	E_180	12	Scout Regimen	12	1450
Eren	E_180	12	Military Corps	13	1300
Mikasa	E_181	12	Scout Regimen	12	1450
Mikasa	E_181	12	Military Corps	13	1300

$\sigma_{\text{<Dept No = Dept_Id>}}$ (Big Table)

Actual_Emp_Dept__info

E_Name	Emp_id	Dept No	Dept_Name	Dept_Id	Start Year
Eren	E_180	12	Scout Regimen	12	1450
Mikasa	E_181	12	Scout Regimen	12	1450

A Cartesian Product is almost always followed by a Select Condition To make the result meaningful.

SINCE, A Cartesian Product followed by a Select Condition is often used,

A dedicated operator was created for this purpose.

$$R \bowtie_{\text{<join condition>}} S$$

$\sigma_{\text{<A = D>}}$ (R X S) ⟶ $R \bowtie_{\text{<A=D>}} S$

Helps to combine related tuples into "Longer Tuples", so that interesting information could be retrieved across tables

Big Table ← Employee X Department

Actual_Emp_Dept__info ← $\sigma_{\text{<Dept No = Dept_Id>}}$ (Big Table)

A Cartesian Product Followed by a Select condition

Join operator combines these two lines into one operation.

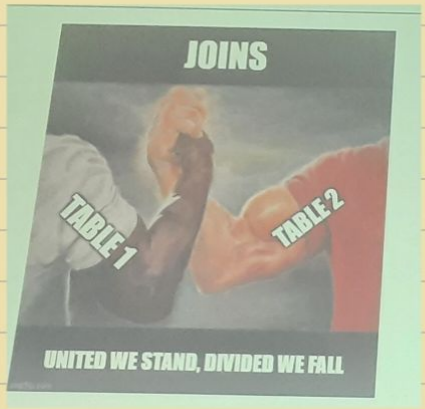
Actual_Emp_Dept__info ← Employee $\bowtie_{\text{<Dept No=Dept_id>}}$ Department

$$R \bowtie_{\text{<join condition>}} S$$

$$R \bowtie_{\text{<A=D>}} S$$

In join we almost always use, "=", so this is also called as **Equi Join**

11/2



Project

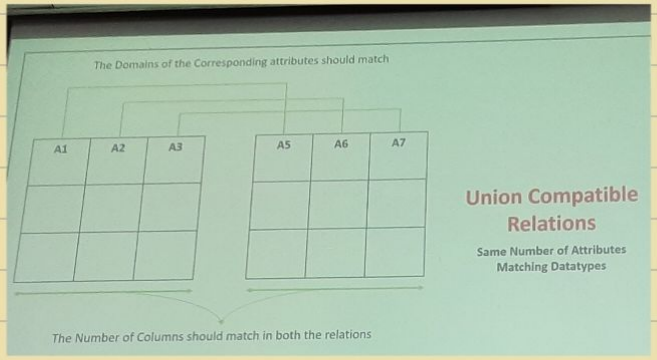
Allows you to choose
Columns you are interested in

Select

Allows you to choose
Rows you are interested in

First Name	Last Name	Address	City	Age
Hickey	House	123 Fantasy Way	Anaheim	73
Bat	Man	321 Cavern Ave	Gotham	54
Wonder	Woman	987 Truth Way	Paradise	39
Donald	Duck	555 Quack Street	Mallard	65
Bugs	Bunny	567 Carrot Street	Rascal	58
Wiley	Coyote	999 Acme Way	Canyon	61
Cat	Woman	234 Purrfect Street	Hairball	32
Tweety	Bird	543	Itotitaw	28

Together, you get complete Two-Dimensional control over the Relation. You can choose any cell of interest.



Union Compatible Relations

Same Number of Attributes
Matching Datatypes

Union Appends Table Vertically

Stud_AI

S_id	Name
1	Ram
2	Shyam

Stud_DD

S_id	Name
1	Ravi
2	Mary

Take Union of the Two

S_id	Name
1	Ram
2	Shyam
1	Ravi
2	Mary

Not a primary Key anymore.

Rename Operator (ρ)

$$\rho_{NewName}(Expression)$$

The result of the expression will be renamed as "NewName"

$$\rho_{NewName(A1,A2,\ldots,An)}(Expression)$$

You can also explicitly rename each of the attributes, if you wish

Rename Operator (ρ) – What gets the Renamed ??

The rename operator (ρ) in Relational Algebra does not rename the original table permanently. Instead, it creates a temporary table with the new name, for the duration of the query.

The original relation remains unchanged.

Rename Operator (ρ) – What gets the Renamed ??

```
SELECT Student_ID, Name AS Full_Name, Age
FROM Students AS Temp_Students;
```

This is how You rename The attributes in SQL

```
ALTER TABLE Students RENAME TO StudentRecords;
ALTER TABLE Students RENAME COLUMN Name TO Full_Name;
ALTER TABLE Students MODIFY COLUMN Age VARCHAR(10);
```

```
SELECT Student_ID, Name AS Full_Name, Age
FROM Students AS Temp_Students;
```

This is just a Query. You rename things temporarily for your convenience

```
ALTER TABLE Students RENAME TO StudentRecords;
```

Here you actually modify the table

Course_1

C_id	C_name
11	Foundation C
21	C++
31	JAVA

Course_2

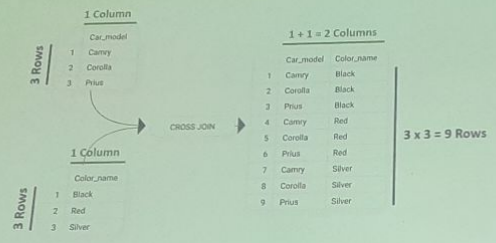
C_id	C_name
12	Python
21	C++

Course_1 ∩ Course_2

C_id	C_name
21	C++

Note: Both the Tables should be "Union Compatible" for Intersect to work

Cross Product



A Cartesian product blindly combines all rows in all different ways possible, Most of them may be meaningless

(Rows) Cardinality : Multiplied
(Columns) Degree : Added

Remember the case, where we wanted To combine related tuples of two Different tables

E_Name	Emp_id	Dept No
Eren	E_180	12
Mikasa	E_181	12

Dept_Name	Dept_Id	Start Year
Scout Regimen	12	1450
Military Corps	13	1300

Related Tuples

In which year was the department where Eren works established?

How do you combine tuples belonging two tables But are related ?

You can take a cross product between both the relations.

This will create a very big table, where most of the rows are meaningless.

And then use a Select condition to filter out only the meaningful rows.

A Cartesian Product is almost always followed by a Select Condition To make the result meaningful.

This is Cartesian Product

```
SELECT D.Start_Year
FROM Employees AS E, Departments AS D
WHERE E.Dept_No = D.Dept_id AND E.E_Name = 'Eren';
```

SELECT 2 tables
↳ Automatically does cross product

SINCE, A Cartesian Product followed by a Select Condition is often used,

A dedicated operator was created for this purpose.

$$R \bowtie_{\text{<join condition>}} S$$

$$\sigma_{<A = D>}(R \times S) \longrightarrow R \bowtie_{<A=D>} S$$

Helps to combine related tuples into "Longer Tuples", so that interesting information could be retrieved across tables.

$$R \bowtie_{\text{<join condition>}} S$$

$$R \bowtie_{<A=D>} S$$

In join we almost always use, "=", so this is also called as **Equi Join**

```
SELECT D.Start_Year
FROM Employees E JOIN Departments D
ON E.Dept_No = D.Dept_id
WHERE E.E_Name = 'Eren';
```

Join Condition

It's a notational convenience

$\sigma_{<Dept\ No\ =\ Dept\_Id>}$ (Big Table)

Actual_Emp_Dept__Info

E_Name	Emp_id	Dept No	Dept_Name	Dept_Id	Start Year
Eren	E_180	12	Scout Regimen	12	1450
Mikasa	E_181	12	Scout Regimen	12	1450

Do you find something Weird in this table ?

Two Columns with same values

This is called as superfluous attribute, or Join Attribute

It would be nice, if we can get rid of that

Natural Join (*)

It is just Equijoin, followed by removal of the superfluous attribute

E_Name	Emp_id	Dept No

Dept_Name	Dept_Id	Start Year

To apply Natural Join the "Join Attribute" should have exactly same name.

You can always Rename the attributes Using the rename operator

Dept_Name	Dept_No	Start Year

Natural Join (*)

E_Name	Emp_id	Dept No

Dept_Name	Dept_No	Start Year

These two tables are natural join Compatible

To perform Natural Join do I need to specify any Join Condition ?

```
SELECT Start_Year
FROM Employees NATURAL JOIN Departments
WHERE E_Name = 'Eren';
```

This work fine If the join attribute Has same name already

```
SELECT Start_Year
FROM (SELECT E_Name, Emp_id, Dept_No AS Dept_id FROM Employees) AS E
NATURAL JOIN Departments AS D
WHERE E_Name = 'Eren';
```

What about other Join Conditions (< , >...) ?

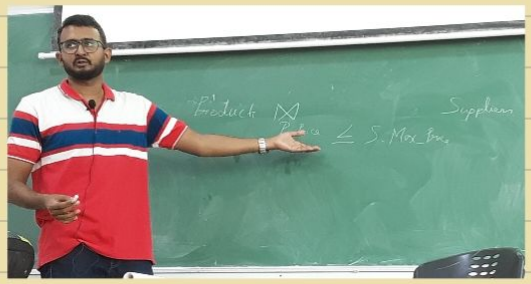
Products (information about products):

Product_ID	Product_Name	Price
101	Laptop	1200
102	Tablet	500
103	Smartphone	800

Suppliers (information about suppliers):

Supplier_ID	Supplier_Name	Max_Price
201	Supplier_A	1000
202	Supplier_B	1500

Find products whose price is less than or equal to the maximum price a supplier can handle.



What about other Join Conditions (< , >...) ?

Find products whose price is less than or equal to the maximum price a supplier can handle.

$$Products \bowtie_{Products.Price \leq Suppliers.Max\_Price} Suppliers$$

Product_Name	Price	Supplier_Name	Max_Price
Laptop	1200	Supplier_B	1500
Tablet	500	Supplier_A	1000
Tablet	500	Supplier_B	1500
Smartphone	800	Supplier_A	1000
Smartphone	800	Supplier_B	1500

Any General Join Condition: **θ Join**

Join with Only Equality: **Equijoin**

Join with implicit join attribute matching: **Natural Join**

θ Join → Equijoin → Natural Join

More General — Specific / Restricted

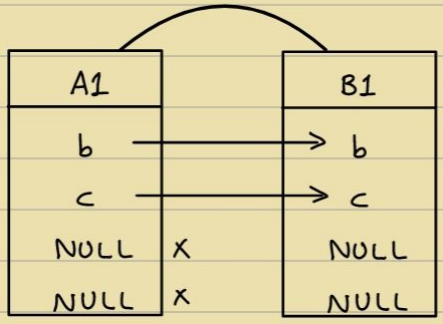
Inner Joins

θ Join → Equijoin → Natural Join

More General — Specific / Restricted

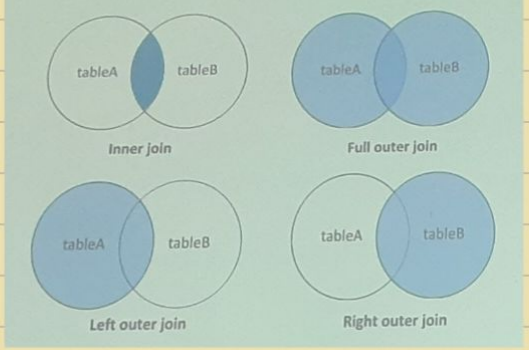
Hierarchy of Relationships

1. **Inner Join** is the broad category.
2. **Theta Join** is a type of Inner Join that uses general comparison conditions.
3. **Equi-Join** is a specific type of Theta Join where the comparison operator is =.
4. **Natural Join** is a specialized form of Equi-Join where matching attributes are automatically identified and duplicate columns are removed.



NULL ≠ NULL

Outer Join



Example

LEFT OUTER JOIN in SQL

Left Table

Food	Price
Pizza	$8
Burger	$6
French Fries	$4

Right Table

Food	Quantity
Pizza	3
Burger	5
Coke	2

Food	Price	Quantity
Pizza	$8	3
Burger	$6	5
French Fries	$4	Null

Note: There is a null in this row

INNER Join + all left out rows from the left table

Right Outer Join Example

Student ID	Name
1001	A
1002	B
1003	C
1004	D

Student ID	Department
1004	Mathematics
1005	Mathematics
1006	History
1007	Physics
1008	Computer Science

Student ID	Name	Department
1004	D	Mathematics
1005	NULL	Mathematics
1006	NULL	History
1007	NULL	Physics
1008	NULL	Computer Science

Right Outer Join Example

Left Table

Date	CountryID	Units
1/1/2020	1	40
1/2/2020	1	25
1/3/2020	3	30
1/4/2020	4	35

Right Table

ID	Country
3	Panama

Merged Table

Date	CountryID	Units	Country
1/3/2020	3	30	Panama

Why only one row in the RO Join ?

Remember: Whatever Matches + Whatever that remains (unmatched) in the Right Table


Left Outer Join


Right Outer Join


Full Outer Join

Division Operator

R

ColA	ColB
F	1
F	2
F	3
E	1
E	3
S	1
S	2

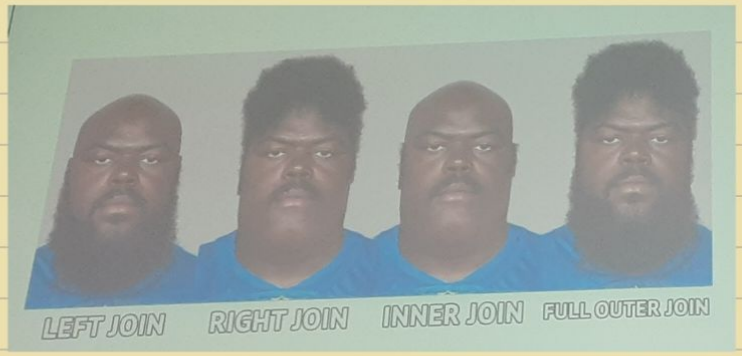
S

ColB
1
2

$R \div S =$

ColA
F
S

12/2



Union Appends Table Vertically

Stud_AI

S_Id	Name
1	Ram
2	Shyam

Course

C_Id	C_Name
1	DBMS
2	AI

S_Id	Name
1	Ram
2	Shyam
1	DBMS
2	AI

This union makes no Logical Sense

Any General Join Condition: Θ Join

Join with Only Equality: Equijoin

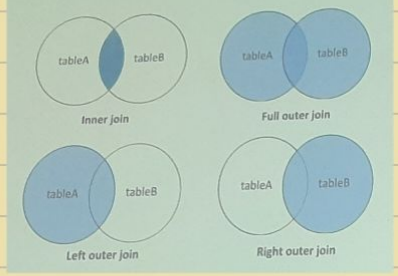
Join with implicit join attribute matching: Natural Join

Inner Joins

Θ Join → Equijoin → Natural Join

More General — Specific / Restricted

Outer Join



Example

LEFT OUTER JOIN in SQL

Left Table

Food	Price
Pizza	$8
Burger	$6
French Fries	$4

Right Table

Food	Quantity
Pizza	3
Burger	5
Cake	2

Food	Price	Quantity
Pizza	$8	3
Burger	$6	5
French Fries	$4	Null

Note: There is a null in this row

INNER Join + all left out rows from the left table

Right Outer Join Example

Student ID	Name
1001	A
1002	B
1003	C
1004	D

Student ID	Department
1004	Mathematics
1005	Mathematics
1006	History
1007	Physics
1008	Computer Science

Student ID	Name	Department
1004	D	Mathematics
1005	NULL	Mathematics
1006	NULL	History
1007	NULL	Physics
1008	NULL	Computer Science

If the Join Attribute name is same, it will be eliminated.

Which is Right ?

Outer Join = Theta Join + Left / Right or Both ✓

Outer Join = Natural Join + Left / Right or Both ✗

Outer Join = Theta Join + Left / Right or Both

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

Division Operator

A

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

B1

pno
p2

A/B1 → S_1, S_2, S_3, S_4

B2

pno
p2
p4

A/B2 → S_1, S_4

B3

pno
p1
p2
p4

A/B3 → S_1

List of all suppliers which supply both P_2 & P_4

Division Operator

Find students who have passed all subjects in their curriculum.

$$\pi_{Student\_ID}(\text{Exam\_Results}) \div \pi_{Subject\_ID}(\text{Curriculum})$$

Find sales representatives who have sold all products in a given region 'r'.

$$\pi_{Rep\_ID}(\text{Sales}) \div \pi_{Product\_ID}(\sigma_{Region=r}(\text{Products}))$$

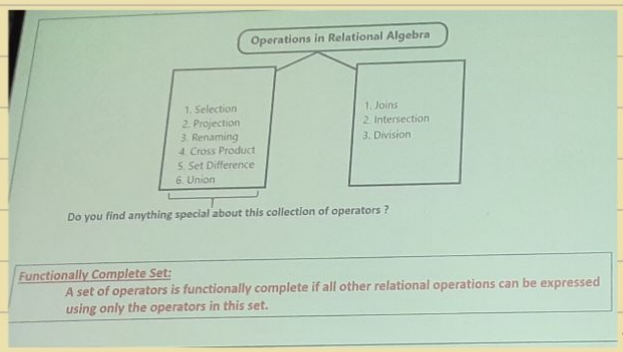
Division Operator

$$\pi_{Scholar\_ID}(\text{Publications}) \div \pi_{Journal\_ID}(\sigma_{Tier=1}(\text{Journals}))$$

Find research scholars who have published in all tier 1 journals.

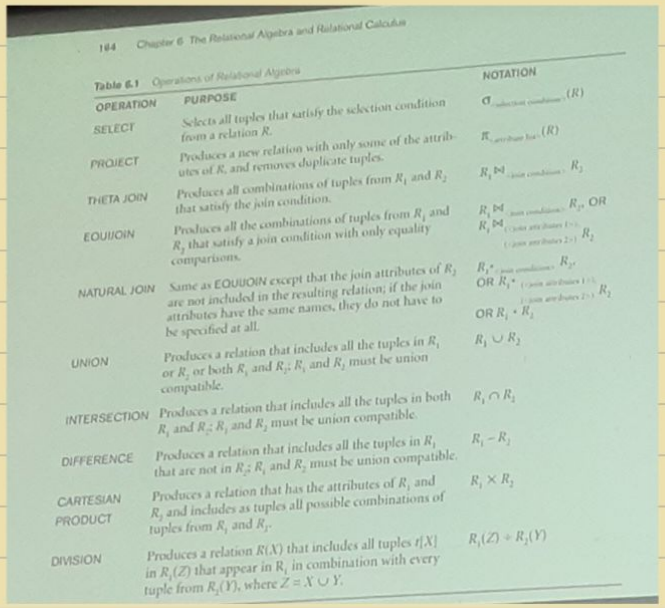
$$\pi_{Student\_ID}(\text{Workshop\_Attendance}) \div \pi_{Workshop\_ID}(\sigma_{Mandatory=True}(\text{Workshops}))$$

Find students who have attended all mandatory workshops for their course.



Functionally Complete Set:
A set of operators is functionally complete if all other relational operations can be expressed using only the operators in this set.

} (Universal Operator)



184 Chapter 6 The Relational Algebra and Relational Calculus

Table 6.1 Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<selection condition>}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<attribute list>}}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\text{<join condition>}} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\text{<join condition>}} R_2$, OR $R_1 \bowtie_{(\text{<join attributes 1>}), (\text{<join attributes 2>})} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 * _{\text{<join condition>}} R_2$, OR $R_1 * _{(\text{<join attributes 1>}), (\text{<join attributes 2>})} R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Does Foreign Key play any role in Joining Tables ?

Typically the join attribute will be a Foreign key.

- Many database systems automatically create an index on foreign key columns, making joins faster.
- Even though the join condition doesn't require a foreign key, having one allows the database engine to optimize join execution.

SQL

SQL – Structured Query Language

SEQUEL – Structured English QUEry Language

Parts of SQL:
DDL
DML
Views
Security & Authorization
Transaction Control

CREATE SCHEMA

CREATE SCHEMA COMPANY **AUTHORIZATION** 'Jsmith';

Authorization Identifier – Who owns the Schema

In general, not all users are authorized to create schemas and schema elements. The privilege to create schemas, tables, and other constructs must be explicitly granted to the relevant user accounts by the system administrator or DBA.

```
CREATE TABLE EMPLOYEE
        ( Fname          VARCHAR(15)          NOT NULL,
          Minit          CHAR,
          Lname          VARCHAR(15)          NOT NULL,
          Ssn            CHAR(9)              NOT NULL,
          Bdate          DATE,
          Address        VARCHAR(30),
          Sex            CHAR,
          Salary         DECIMAL(10,2),
          Super_ssn      CHAR(9),
          Dno            INT                  NOT NULL,
        PRIMARY KEY (Ssn),
        FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber) );
```

Tables are only created within sql query

Do you Find Something Weird In this ?

```
CREATE TABLE EMPLOYEE
        ( Fname          VARCHAR(15)          NOT NULL,
          Minit          CHAR,
          Lname          VARCHAR(15)          NOT NULL,
          Ssn            CHAR(9)              NOT NULL,
          Bdate          DATE,
          Address        VARCHAR(30),
          Sex            CHAR,
          Salary         DECIMAL(10,2),
          Super_ssn      CHAR(9),
          Dno            INT                  NOT NULL,
        PRIMARY KEY (Ssn),
        FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber) );
```

Foreign Keys can be specified later, through ALTER Command

शायद

Won't work ∵ Employee Table hasn't been created yet, ∴ Can't reference to uncreated table

∴ You HAVE to use ALTER command to self-reference

Attributes are considered To be ordered In this order

```
CREATE TABLE EMPLOYEE
        ( Fname          VARCHAR(15)          NOT NULL,
          Minit          CHAR,
          Lname          VARCHAR(15)          NOT NULL,
          Ssn            CHAR(9)              NOT NULL,
          Bdate          DATE,
          Address        VARCHAR(30),
          Sex            CHAR,
          Salary         DECIMAL(10,2),
          Super_ssn      CHAR(9),
          Dno            INT                  NOT NULL,
        PRIMARY KEY (Ssn),
        FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber) );
```

4.1.3 Attribute Data Types and Domains in SQL

Character String

Bit String

Boolean

Date

Timestamp

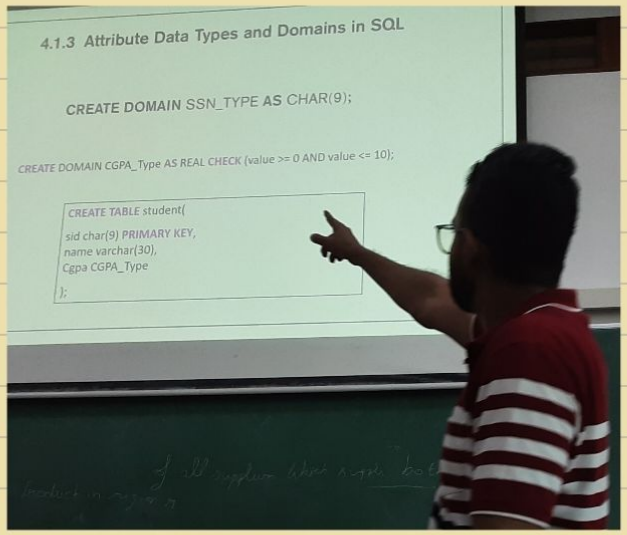
4.1.3 Attribute Data Types and Domains in SQL

- **Numeric** data types include integer numbers of various sizes (INTEGER or INT, and SMALLINT) and floating-point (real) numbers of various precision (FLOAT or REAL, and DOUBLE PRECISION). Formatted numbers can be declared by using DECIMAL(*i*,*j*)—or DEC(*i*,*j*) or NUMERIC(*i*,*j*)—where *i*, the *precision*, is the total number of decimal digits and *j*, the *scale*, is the number of digits after the decimal point. The default for scale is zero, and the default for precision is implementation-defined.

[Datatype : C :: Domain : SQL]

4.1.3 Attribute Data Types and Domains in SQL

```
CREATE DOMAIN SSN_TYPE AS CHAR(9);

CREATE DOMAIN CGPA_Type AS REAL CHECK (value >= 0 AND value <= 10);

CREATE TABLE student(
sid char(9) PRIMARY KEY,
name varchar(30),
Cgpa CGPA_Type
);
```

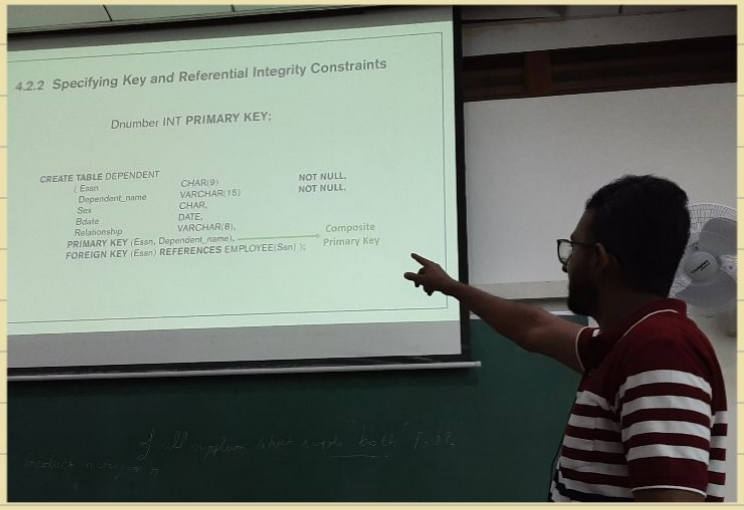


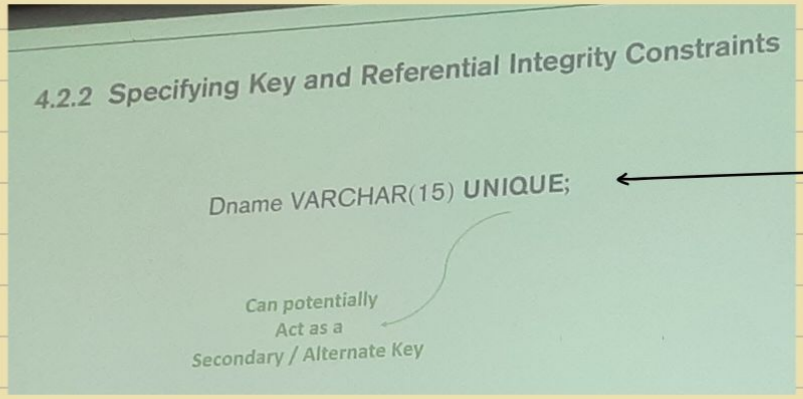
4.2.1 Specifying Attribute Constraints and Attribute Defaults

NOT NULL

DEFAULT

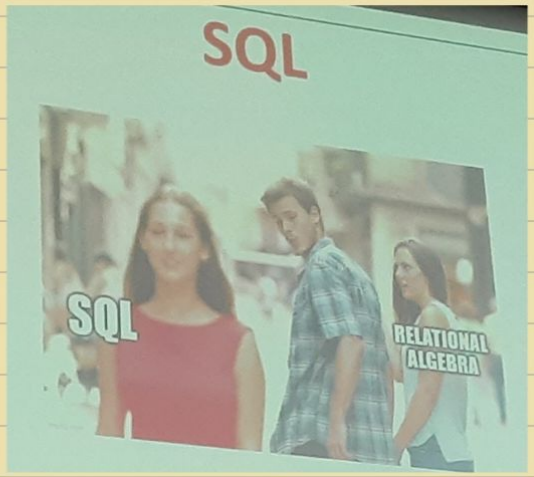
```
Dnumber INT NOT NULL CHECK (Dnumber > 0 AND Dnumber < 21);

CREATE DOMAIN D_NUM AS INTEGER
CHECK (D_NUM > 0 AND D_NUM < 21);
```

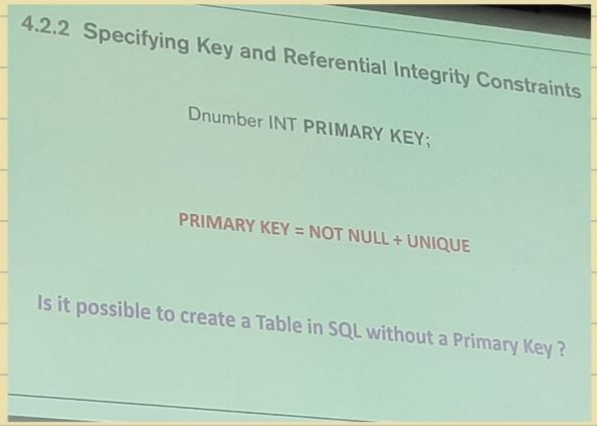


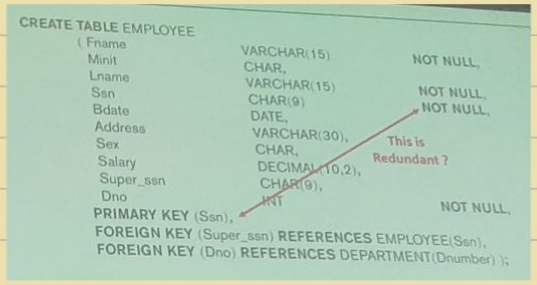


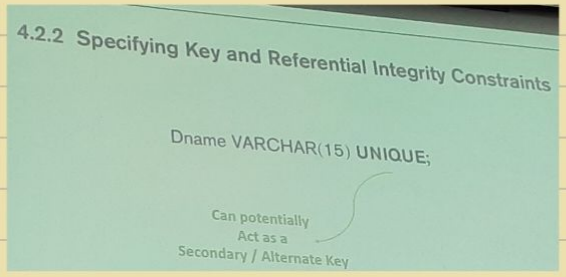
Candidate Key

13/2









Composite Foreign Key

```
CREATE TABLE Course_Registration (
        Student_ID INT,
        Course_ID INT,
        Semester VARCHAR(10),
        PRIMARY KEY (Student_ID, Course_ID, Semester)
);
```

```
CREATE TABLE Grades (
        Student_ID INT,
        Course_ID INT,
        Grade CHAR(2),
        Semester VARCHAR(10),
        PRIMARY KEY (Student_ID, Course_ID, Semester),

        FOREIGN KEY (Student_ID, Course_ID, Semester) REFERENCES
        Course_Registration(Student_ID, Course_ID, Semester)
);
```

Foreign Key

Syntax for F.Key you know already.

What about actions when referential integrity is violated ?

The default action is to "REJECT" the update operation.

Referential Triggered Action

```
CREATE TABLE DEPARTMENT
    ( ...,
      Mgr_ssn     CHAR(9)       NOT NULL          DEFAULT '888665555',
      ...,
    CONSTRAINT DEPTPK
       PRIMARY KEY(Dnumber),
    CONSTRAINT DEPTSK
       UNIQUE (Dname),
    CONSTRAINT DEPTMGRFK
       FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn)
                     ON DELETE SET DEFAULT   ON UPDATE CASCADE);
```

Referential Triggered Action

```
CREATE TABLE DEPT_LOCATIONS
    ( ...,
      PRIMARY KEY (Dnumber, Dlocation),
      FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber)
                     ON DELETE CASCADE                ON UPDATE CASCADE);
                       Trigger          Action
```

4.2.4 Specifying Constraints on Tuples Using CHECK

```
CHECK (Dept_create_date <= Mgr_start_date);
```

5.4 Schema Change Statements in SQL

Called as **Schema Evolution Commands**

```
DROP SCHEMA COMPANY

DROP TABLE DEPENDENT
```

```
ALTER TABLE COMPANY.EMPLOYEE ADD COLUMN Job VARCHAR(12);
```

Question Can you use NOT NULL constraint with this new column ?

Queries

Multisets vs Sets

To constrain a table to be a Set, use Key constraint or DISTINCT Command

```
SELECT    <attribute list>
FROM      <table list>
WHERE     <condition>;
```

where

- <attribute list> is a list of attribute names whose values are to be retrieved by the query.
- <table list> is a list of the relation names required to process the query.
- <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.

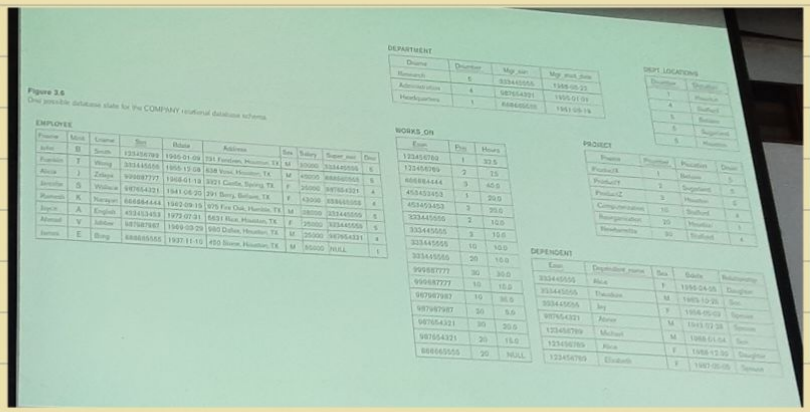


Figure 3.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

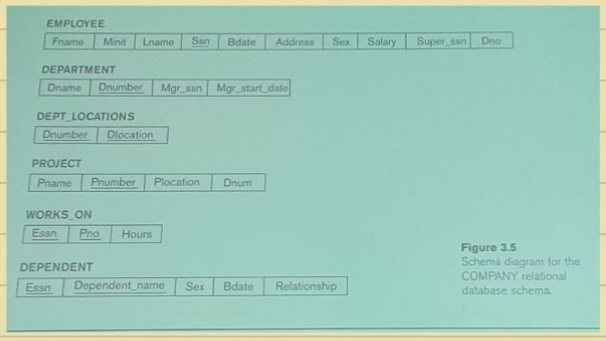


Figure 3.5
Schema diagram for the COMPANY relational database schema.

Query 1. Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:  SELECT   Fname, Lname, Address
     FROM     EMPLOYEE, DEPARTMENT
     WHERE    Dname='Research' AND Dnumber=Dno;
```

Query 2. For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

```
Q2:  SELECT   Pnumber, Dnum, Lname, Address, Bdate
     FROM     PROJECT, DEPARTMENT, EMPLOYEE
     WHERE    Dnum=Dnumber AND Mgr_ssn=Ssn AND
              Plocation='Stafford';
```

(c)

Pnumber	Dnum	Lname	Address	Bdate
10	4	Wallace	291Berry, Bellaire, TX	1941-06-20
30	4	Wallace	291Berry, Bellaire, TX	1941-06-20

The ambiguity of attribute names also arises in the case of queries that refer to the same relation twice, as in the following example.

Query 8. For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

```
Q8:  SELECT   E.Fname, E.Lname, S.Fname, S.Lname
     FROM     EMPLOYEE AS E, EMPLOYEE AS S
     WHERE    E.Super_ssn=S.Ssn;
```

One Level Recursive Query

E → Subordinates
S → Supervisor

```
Q9:  SELECT   Ssn
     FROM     EMPLOYEE;

Q10: SELECT   Ssn, Dname
     FROM     EMPLOYEE, DEPARTMENT;
```

Why SQL Allows Duplicates ?

- Duplicate elimination is an expensive operation. One way to implement it is to sort the tuples first and then eliminate duplicates.
- The user may want to see duplicate tuples in the result of a query.
- When an aggregate function (see Section 5.1.7) is applied to tuples, in most cases we do not want to eliminate duplicates.

What if I want to eliminate Duplicates ?

```
Q11A: SELECT   DISTINCT Salary
      FROM     EMPLOYEE;
```

Set Operations in SQL

No Duplicates	Duplicates Allowed
Union	Union ALL
Intersect ...	Intersect ALL ...

Pattern Matching in SQL

Query 12. Retrieve all employees whose address is in Houston, Texas.

```
Q12: SELECT   Fname, Lname
     FROM     EMPLOYEE
     WHERE    Address LIKE '%Houston,TX%';
```

Query 12A. Find all employees who were born during the 1950s.

```
Q12: SELECT   Fname, Lname
     FROM     EMPLOYEE
     WHERE    Bdate LIKE '_ _ 5 _ _ _ _ _ _ _';
```

What if I want to search for '%' or '_' itself

```
'AB\_CD\%EF' ESCAPE '\'
```

Query 14. Retrieve all employees in department 5 whose salary is between $30,000 and $40,000.

```
Q14: SELECT   *
     FROM     EMPLOYEE
     WHERE    (Salary BETWEEN 30000 AND 40000) AND Dno = 5;
```

Order By

Query 15. Retrieve a list of employees and the projects they are working on, ordered by department and, within each department, ordered alphabetically by last name, then first name.

```
Q15: SELECT   D.Dname, E.Lname, E.Fname, P.Pname
     FROM     DEPARTMENT D, EMPLOYEE E, WORKS_ON W,
              PROJECT P
     WHERE    D.Dnumber= E.Dno AND E.Ssn= W.Essn AND
              W.Pno= P.Pnumber
     ORDER BY D.Dname, E.Lname, E.Fname;
```

Structure of a Simple Query

Mandatory Clauses: SELECT <attribute list>
FROM <table list>
Optional Clauses: [WHERE <condition>]
[ORDER BY <attribute list>];

Delete Command

U4A: DELETE FROM EMPLOYEE
WHERE Lname='Brown';

U4B: DELETE FROM EMPLOYEE
WHERE Ssn='123456789';

U4C: DELETE FROM EMPLOYEE
WHERE Dno=5;

U4D: DELETE FROM EMPLOYEE;

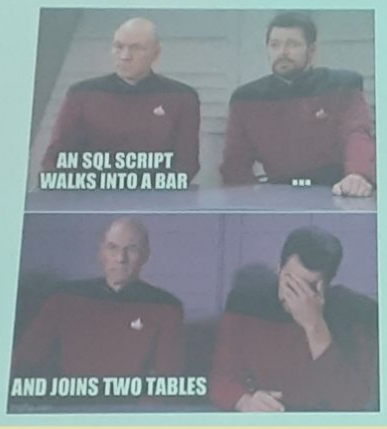
Update Command

UPDATE PROJECT
SET Plocation = 'Bellaire', Dnum = 5
WHERE Pnumber=10;

U6: UPDATE EMPLOYEE
SET Salary = Salary * 1.1
WHERE Dno = 5;

18/2

SQL



MultiSets in SQL

First Name	Last Name	Address	City	Age
Hickey	House	123 Fantasy Way	Anaheim	73
Bat	Han	321 Cavern Ave	Gotham	54
Wonder	Woman	987 Truth Way	Paradise	39
Donald	Duck	555 Quack Street	Mallard	65
Bugs	Bunny	567 Carrot Street	Rascal	50
Wiley	Coyote	999 Acme Way	Canyon	61
Cat	Woman	234 Purrfect Street	Hairball	32
Tweety	Bird	543	Itotitaw	28

A TABLE in SQL **may** have duplicates, if it has no Primary Key defined.

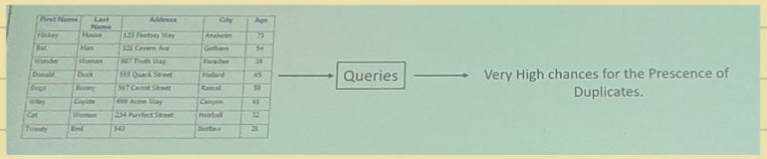
A TABLE in SQL **can** have duplicates, **If and only if** it has no Primary Key defined.

When storing application logs, Caches, or system events, duplicate entries are normal and may even be necessary

```
CREATE TABLE SystemLogs (

LogID INT AUTO_INCREMENT,
EventTime TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
EventType VARCHAR(50),

);
```

```
CREATE TABLE API_Cache (

APIEndpoint VARCHAR(255),
ResponseData TEXT,
Timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP

);
```

Highly rare cases



What if I want to eliminate Duplicates ?

Q11A: SELECT DISTINCT Salary
FROM EMPLOYEE;

Set Operations in SQL

No Duplicates	Duplicates Allowed
Union	Union ALL
Intersect ...	Intersect ALL ...

Referential Triggered Action

```
CREATE TABLE DEPT_LOCATIONS
    ( ...,
    PRIMARY KEY (Dnumber, Dlocation),
    FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber)
                ON DELETE CASCADE          ON UPDATE CASCADE);
```

Trigger — Action

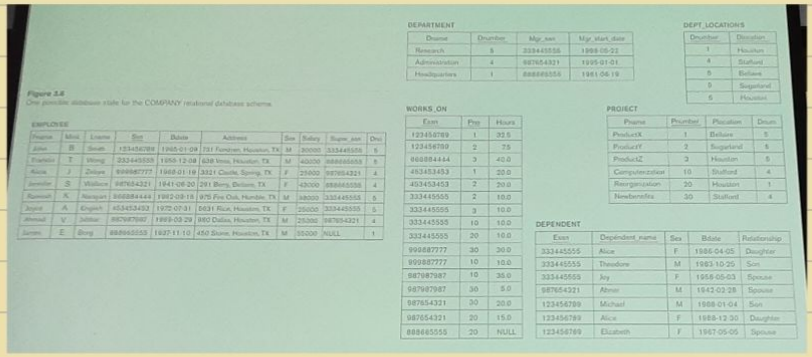


Figure 3.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

State / Instance of Schema

The ambiguity of attribute names also arises in the case of queries that refer to the same relation twice, as in the following example.

Query 8. For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

```
Q8:   SELECT   E.Fname, E.Lname, S.Fname, S.Lname
      FROM     EMPLOYEE AS E, EMPLOYEE AS S
      WHERE    E.Super_ssn=S.Ssn;
```

One Level Recursive Query

E → Subordinates
S → Supervisor

Tuple Variables / Aliases / Copies

Usage of Arithmetic operators within Queries

Query 13. Show the resulting salaries if every employee working on the 'ProductX' project is given a 10 percent raise.

```
Q13:  SELECT   E.Fname, E.Lname, 1.1 * E.Salary AS Increased_sal
      FROM     EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P
      WHERE    E.Ssn=W.Essn AND W.Pno=P.Pnumber AND
               P.Pname='ProductX';
```

Order By

Query 15. Retrieve a list of employees and the projects they are working on, ordered by department and, within each department, ordered alphabetically by last name, then first name.

```
Q15:  SELECT   D.Dname, E.Lname, E.Fname, P.Pname
      FROM     DEPARTMENT D, EMPLOYEE E, WORKS_ON W,
               PROJECT P
      WHERE    D.Dnumber= E.Dno AND E.Ssn= W.Essn AND
               W.Pno= P.Pnumber
      ORDER BY D.Dname, E.Lname, E.Fname;
```

Structure of a Simple Query

```
Mandatory Clauses:  SELECT   <attribute list>
                    FROM     <table list>
Optional Clauses:   [ WHERE    <condition> ]
                    [ ORDER BY <attribute list> ];
```

Delete Command

```
U4A:  DELETE FROM   EMPLOYEE
      WHERE         Lname='Brown';

U4B:  DELETE FROM   EMPLOYEE
      WHERE         Ssn='123456789';

U4C:  DELETE FROM   EMPLOYEE
      WHERE         Dno=5;

U4D:  DELETE FROM   EMPLOYEE;
```

NULL In SQL

- Unknown → Value exists but doesn't know
- Not Available / Unavailable → Information exists, but hidden
- Not Applicable → Value Does not exist at all

NULL ⇝ UK

1. **Unknown value.** A person's date of birth is not known, so it is represented by NULL in the database.
2. **Unavailable or withheld value.** A person has a home phone but does not want it to be listed, so it is withheld and represented as NULL in the database.
3. **Not applicable attribute.** An attribute LastCollegeDegree would be NULL for a person who has no college degrees because it does not apply to that person.

It is often not possible to determine which of the meanings is intended; for example, a NULL for the home phone of a person can have any of the three meanings. Hence, SQL does not distinguish between the different meanings of NULL.

A	B	A ∩ B	A ∪ B
T	UK	UK	T
F	UK	F	UK
UK	UK	UK	UK

$\sim(UK) = UK$

NULL + 1 = NULL

Three Valued Logic in SQL

Table 5.1 Logical Connectives in Three-Valued Logic

(a)

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

(b)

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(c)

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

Query 18. Retrieve the names of all employees who do not have supervisors.

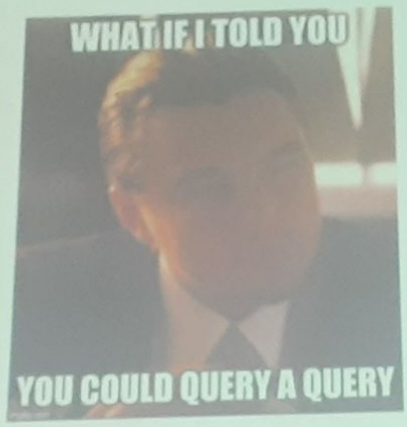
```
SELECT  names
FROM    EMPLOYEE
WHERE   Super_ssn = NULL
```

✗

```
Q18:  SELECT  Fname, Lname
      FROM    EMPLOYEE
      WHERE   Super_ssn IS NULL;
```

✓

Nested Queries



Some queries require that existing values in the database be fetched and then used in a comparison condition. Such queries can be conveniently formulated by using **nested queries**, which are complete select-from-where blocks within the WHERE clause of another query. That other query is called the **outer query**.

Nested Queries

Query 4. Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

```
Q4A:  ( SELECT    DISTINCT Pnumber
        FROM      PROJECT, DEPARTMENT, EMPLOYEE
        WHERE     Dnum=Dnumber AND Mgr_ssn=Ssn
                  AND Lname='Smith' )

      UNION

      ( SELECT    DISTINCT Pnumber
        FROM      PROJECT, WORKS_ON, EMPLOYEE
        WHERE     Pnumber=Pno AND Essn=Ssn
                  AND Lname='Smith' );
```

Nested Queries

Query 4. Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

```
Q4A:  SELECT    DISTINCT Pnumber
      FROM      PROJECT
      WHERE     Pnumber IN
                ( SELECT    Pnumber
                  FROM      PROJECT, DEPARTMENT, EMPLOYEE
                  WHERE     Dnum=Dnumber AND
                            Mgr_ssn=Ssn AND Lname='Smith' )
                OR
                Pnumber IN
                ( SELECT    Pno
                  FROM      WORKS_ON, EMPLOYEE
                  WHERE     Essn=Ssn AND Lname='Smith' );
```

If a nested query returns a single attribute *and* a single tuple, the query result will be a single (scalar) value. In such cases, it is permissible to use = instead of IN for the comparison operator. In general, the nested query will return a **table** (relation), which is a set or multiset of tuples.

Nested Queries

Query 4. Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

```
Q4A:  SELECT    DISTINCT Pnumber
      FROM      PROJECT
      WHERE     Pnumber IN
                ( SELECT    Pnumber
                  FROM      PROJECT, DEPARTMENT, EMPLOYEE
                  WHERE     Dnum=Dnumber AND
                            Mgr_ssn=Ssn AND Lname='Smith' )
                OR
                Pnumber IN
                ( SELECT    Pno
                  FROM      WORKS_ON, EMPLOYEE
                  WHERE     Essn=Ssn AND Lname='Smith' );
```

```
SELECT    DISTINCT Essn
FROM      WORKS_ON
WHERE     (Pno, Hours) IN ( SELECT    Pno, Hours
                            FROM      WORKS_ON
                            WHERE     Essn='123456789' );
```

Notice the paranthesis

Comparing Single Value to a Set of Values

In addition to the IN operator, a number of other comparison operators can be used to compare a single value *v* (typically an attribute name) to a set or multiset *v* (typically a nested query). The = ANY (or = SOME) operator returns TRUE if the value *v* is equal to *some value* in the set *V* and is hence equivalent to IN. The two keywords ANY and SOME have the same effect. Other operators that can be combined with ANY (or SOME) include >, >=, <, <=, and <>. The keyword ALL can also be combined with each of these operators. For example, the comparison condition (*v* > ALL *V*) returns TRUE if the value *v* is greater than *all* the values in the set (or multiset) *V*.

Comparison operators in Nested Queries

```
SELECT    Lname, Fname
FROM      EMPLOYEE
WHERE     Salary > ALL ( SELECT    Salary
                         FROM      EMPLOYEE
                         WHERE     Dno=5 );
```

Query 16. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

Book
- Intro - 1, 2
- ER
- RM
- RA
- SQL - 4, 5

19/2:

Comparison operators in Nested Queries

```
SELECT    Lname, Fname
FROM      EMPLOYEE
WHERE     Salary > ALL ( SELECT    Salary
                         FROM      EMPLOYEE
                         WHERE     Dno=5 );
```

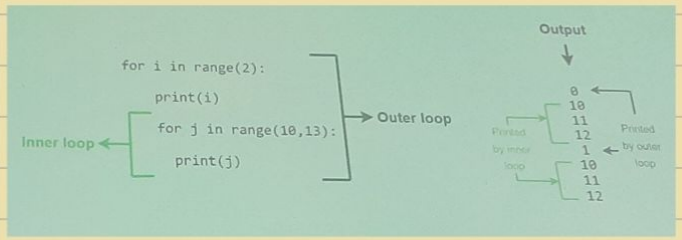
Query 16. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

```
SELECT    E.Fname, E.Lname
FROM      EMPLOYEE AS E
WHERE     E.Ssn IN ( SELECT    Essn
                     FROM      DEPENDENT AS D
                     WHERE     E.Fname=D.Dependent_name
                               AND E.Sex=D.Sex );
```

5.1.3 Correlated Nested Queries

Whenever a condition in the WHERE clause of a nested query references some attribute of a relation declared in the outer query, the two queries are said to be **correlated**.

The Inner Query is evaluated once for each row of the outer query




```
SELECT    E.Fname, E.Lname
FROM      EMPLOYEE AS E
WHERE     E.Ssn IN ( SELECT    Essn
                     FROM      DEPENDENT AS D
                     WHERE     E.Fname=D.Dependent_name
                               AND E.Sex=D.Sex );
```

For each EMPLOYEE tuple:

Retrieve the Essn values for all DEPENDENT tuples with the same sex and name as that EMPLOYEE tuple;

If the Ssn value of the EMPLOYEE tuple matches with the any of the Essn Value, then return the F.name and L.Name

```
SELECT    E.Fname, E.Lname
FROM      EMPLOYEE AS E, DEPENDENT AS D
WHERE     E.Ssn=D.Essn AND E.Sex=D.Sex
          AND E.Fname=D.Dependent_name;
```

In general, a query written with nested select-from-where blocks and using the = or IN comparison operators can *always* be expressed as a single block query. For exam-

EXISTS Keyword in Nested Queries

The EXISTS function in SQL is used to check whether the result of a correlated nested query is empty (contains no tuples) or not.

```
SELECT    E.Fname, E.Lname
FROM      EMPLOYEE AS E
WHERE     EXISTS ( SELECT    *
                   FROM      DEPENDENT AS D
                   WHERE     E.Ssn=D.Essn AND E.Sex=D.Sex
                             AND E.Fname=D.Dependent_name);
```

For each Employee:
If there exist atleast one dependent with Same sex and same name, then Return that employee

If at least one tuple EXISTS in the result of the nested query, then select that EMPLOYEE tuple

```
SELECT    Fname, Lname
FROM      EMPLOYEE
WHERE     NOT EXISTS ( SELECT    *
                       FROM      DEPENDENT
                       WHERE     Ssn=Essn );
```

Query 6. Retrieve the names of employees who have no dependents.

Query 3. Find the names of employees who work on *all* the projects controlled by department number 5.

```
SELECT    Fname, Lname
FROM      EMPLOYEE
WHERE     NOT EXISTS ( ( SELECT    Pnumber
                         FROM      PROJECT
                         WHERE     Dnum=5)
                         EXCEPT    ( SELECT    Pno
                                     FROM      WORKS_ON
                                     WHERE     Ssn=Essn) );
```

Not Correlated with the Outer Query

Selects all projects that the particular employee being considered works on

For each employee:
IF (First Subquery – Second Subquery) = Empty
It means that the employee works on all the projects and is therefore selected

→ Aggregate Functions :

Function Name	Meaning	Example
SUM(column name)	Total sum of the values in a numeric column	SUM(salary)
AVG(column name)	Average of the values in a column	AVG(salary)
MAX(column name)	Largest value in a column	MAX(salary)
MIN(column name)	Smallest value in a column	MIN(salary)
COUNT(*)	Count of the number of rows selected	COUNT(*)

Query 20. Find the sum of the salaries of all employees of the 'Research' department, as well as the maximum salary, the minimum salary, and the average salary in this department.

```
SELECT    SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
FROM      (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
WHERE     Dname='Research';
```

Count the number of distinct salary values in the database.

```
SELECT    COUNT (DISTINCT Salary)
FROM      EMPLOYEE;
```

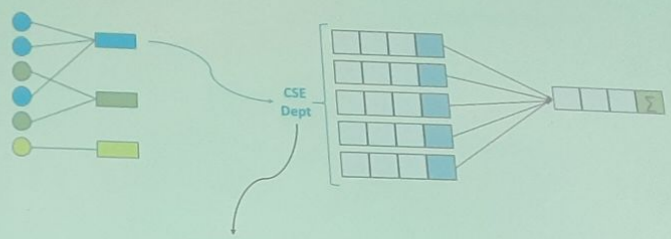
What if someone's salary is NULL, will it be counted ?

In **general**, NULL values are discarded when aggregate functions are applied to a particular column (attribute).

```
SELECT    Lname, Fname
FROM      EMPLOYEE
WHERE     ( SELECT    COUNT (*)
            FROM  DEPENDENT
            WHERE     Ssn=Essn ) >= 2;
```

Cluster related tuples into a group

Apply the aggregate function For the tuples of each group



Grouping Attribute: Has same value for a particular group

Query 24. For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
Q24:   SELECT    Dno, COUNT (*), AVG (Salary)
       FROM      EMPLOYEE
       GROUP BY  Dno;
```

Note that: Select clause includes only the Grouping Attribute and the Aggregate functions to be applied on each Group of tuples

```
Q24:   SELECT    Dno, COUNT (*), AVG (Salary)
       FROM      EMPLOYEE
       GROUP BY  Dno;
```

(a)

Fname	Minit	Lname	Ssn	...
John	B	Smith	123456789	
Franklin	T	Wong	333445555	
Ramesh	K	Narayan	666884444	
Joyce	A	English	453453453	
Alicia	J	Zelaya	999887777	
Jennifer	S	Wallace	987654321	
Ahmad	V	Jabbar	987987987	
James	E	Bong	888665555	

Grouping EMPLOYEE tuples by the value of Dno

Salary	Super_ssn	Dno
30000	333445555	5
40000	888665555	5
38000	333445555	5
25000	333445555	5
25000	987654321	4
43000	888665555	4
25000	987654321	4
55000	NULL	1

Dno	Count (*)	Avg (Salary)
5	4	33250
4	3	31000
1	1	55000

Result of Q24

What if there were some employees whose Dno = NULL

Query 25. For each project, retrieve the project number, the project name, and the number of employees who work on that project.

```
Q25:   SELECT    Pnumber, Pname, COUNT (*)
       FROM      PROJECT, WORKS_ON
       WHERE     Pnumber=Pno
       GROUP BY  Pnumber, Pname;
```

```
Q26:   SELECT    Pnumber, Pname, COUNT (*)
       FROM      PROJECT, WORKS_ON
       WHERE     Pnumber=Pno
       GROUP BY  Pnumber, Pname
       HAVING    COUNT (*) > 2;
```

Query 25. For each project, retrieve the project number, the project name, and the number of employees who work on that project.

Say I'm adding one more condition to this query
I want only projects with more than two employees to appear in the result

Query 26. For each project *on which more than two employees work*, retrieve the project number, the project name, and the number of employees who work on the project.

WHERE Clause	HAVING Clause
Applicable without GROUP BY clause	Cannot be used without GROUP BY Clause
Can be used with SELECT, UPDATE, DELETE statement.	Can only be used with SELECT statement
Cannot contain aggregate function	Can contain aggregate function
Used to filter the records from the table based on the specified condition.	Used to filter record from the groups based on the specified condition.

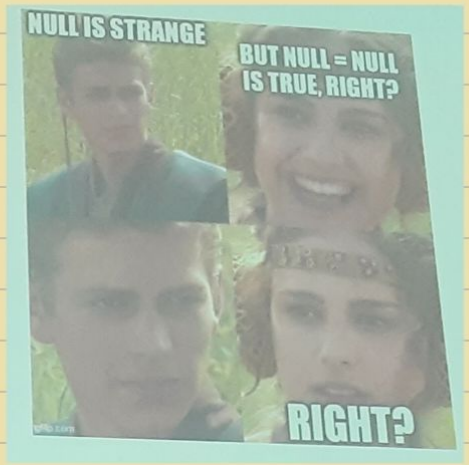
```
Q26:   SELECT    Pnumber, Pname, COUNT (*)
       FROM      PROJECT, WORKS_ON
       WHERE     Pnumber=Pno
       GROUP BY  Pnumber, Pname
       HAVING    COUNT (*) > 2;
```

WHERE helps to choose tuples to which the aggregate functions are applied.

HAVING clause serves to choose whole groups.

```
SELECT <attribute and function list>
FROM <table list>
[ WHERE <condition> ]
[ GROUP BY <grouping attribute(s)> ]
[ HAVING <group condition> ]
[ ORDER BY <attribute list> ];
```

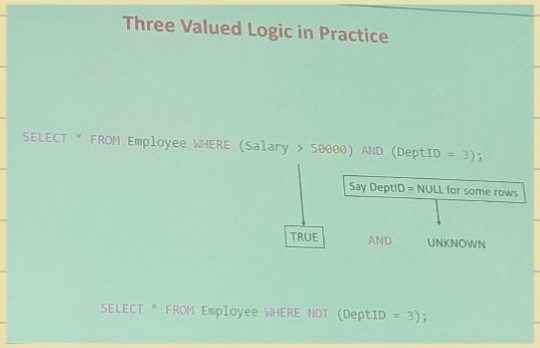
20/2

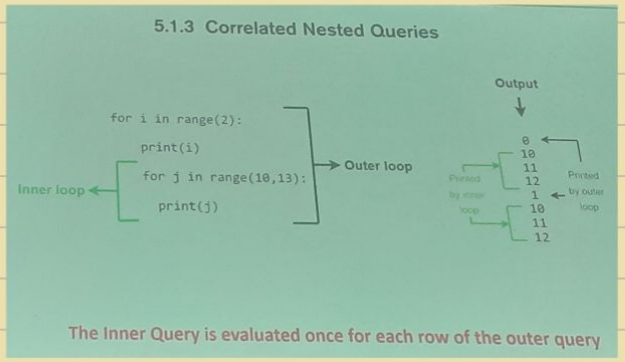


A	B
1	NULL
2	NULL
NULL	NULL
3	NULL

COUNT(*) = 4
COUNT(A) = 3
COUNT(B) = 0
AVG(A) = 2
AVG(B) = NULL

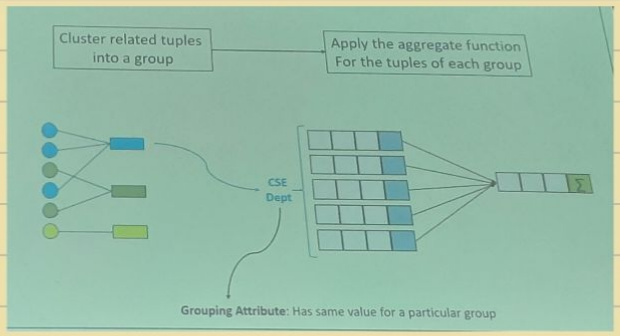
→ NULL vs Aggregates :





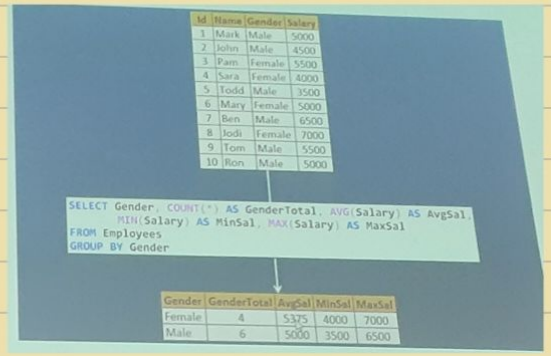
- **Nested** subquery runs only once for the entire nesting (outer) query. It does not contain any reference to the outer query row.
- **Correlated** subquery runs once for each row selected by the outer query. It contains a reference to a value from the row selected by the outer query.

#	Correlated Subquery	Non-Correlated Subquery
Execution	Triggered for each row in the outer query	Executed only once, regardless of outer query rows
Dependency	Relies on values from the outer query	Independent of the outer query
Efficiency	Less efficient for large datasets	More efficient for large datasets



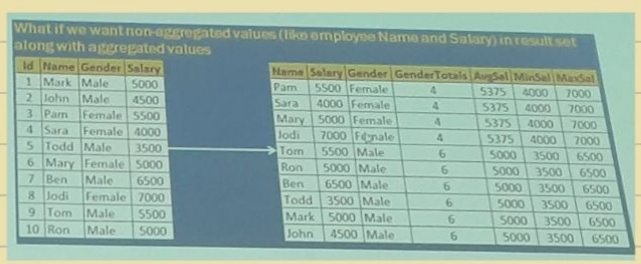
General Structure of an SQL Query :

SELECT <attribute and function list>
FROM <table list>
[**WHERE** <condition>]
[**GROUP BY** <grouping attribute(s)>]
[**HAVING** <group condition>]
[**ORDER BY** <attribute list>];



Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

Gender	GenderTotal	AvgSal	MinSal	MaxSal
Female	4	5375	4000	7000
Male	6	5000	3500	6500



Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

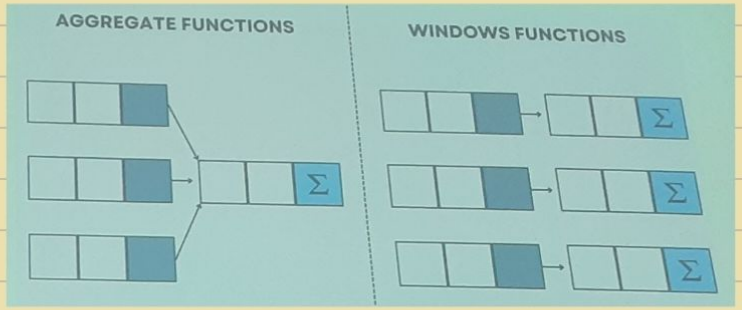
Name	Salary	Gender	GenderTotals	AvgSal	MinSal	MaxSal
Pam	5500	Female	4	5375	4000	7000
Sara	4000	Female	4	5375	4000	7000
Mary	5000	Female	4	5375	4000	7000
Jodi	7000	Female	4	5375	4000	7000
Tom	5500	Male	6	5000	3500	6500
Ron	5000	Male	6	5000	3500	6500
Ben	6500	Male	6	5000	3500	6500
Todd	3500	Male	6	5000	3500	6500
Mark	5000	Male	6	5000	3500	6500
John	4500	Male	6	5000	3500	6500

What if we want non-aggregated values (like employee Name and Salary) in result set along with aggregated values

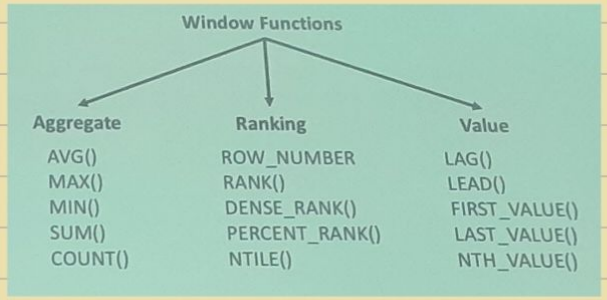
Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

Name	Salary	Gender	GenderTotals	AvgSal	MinSal	MaxSal
Pam	5500	Female	4	5375	4000	7000
Sara	4000	Female	4	5375	4000	7000
Mary	5000	Female	4	5375	4000	7000
Jodi	7000	Female	4	5375	4000	7000
Tom	5500	Male	6	5000	3500	6500
Ron	5000	Male	6	5000	3500	6500
Ben	6500	Male	6	5000	3500	6500
Todd	3500	Male	6	5000	3500	6500
Mark	5000	Male	6	5000	3500	6500
John	4500	Male	6	5000	3500	6500

```
SELECT Name, Salary, Gender,
       COUNT(Gender) OVER(PARTITION BY Gender) AS GenderTotals,
       AVG(Salary) OVER(PARTITION BY Gender) AS AvgSal,
       MIN(Salary) OVER(PARTITION BY Gender) AS MinSal,
       MAX(Salary) OVER(PARTITION BY Gender) AS MaxSal
FROM Employees
```



SQL window functions are essential for advanced data analysis and database management. They enable calculations across a specific set of rows, known as a "window," while retaining the individual rows in the dataset. Unlike traditional aggregate functions that summarize data for the entire group, window functions allow detailed calculations for specific partitions or subsets of data.



Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

Name	Gender	Salary	RowNumber
Pam	Female	5500	1
Sara	Female	4000	2
Mary	Female	5000	3
Jodi	Female	7000	4
Tom	Male	5500	5
Ron	Male	5000	6
Ben	Male	6500	7
Todd	Male	3500	8
Mark	Male	5000	9
John	Male	4500	10

```
SELECT Name, Gender, Salary,
       ROW_NUMBER() OVER (ORDER BY Gender) AS RowNumber
FROM   Employees
```

Row_Number function with PARTITION BY : In this example, data is partitioned by Gender, so ROW_NUMBER will provide a consecutive numbering only for the rows with in a partition. When the partition changes the row number is reset to 1.

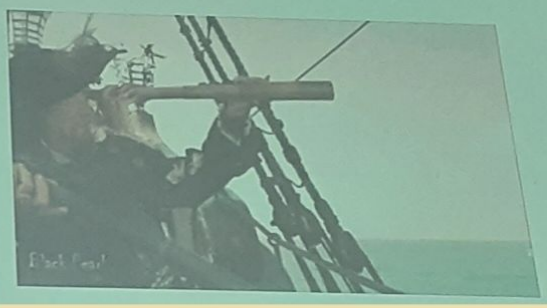
Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

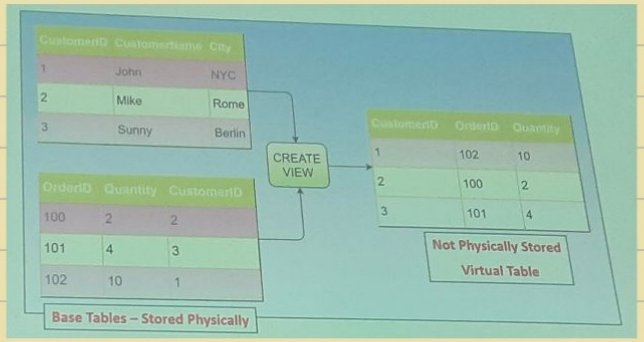
Name	Gender	Salary	RowNumber
Pam	Female	5500	1
Sara	Female	4000	2
Mary	Female	5000	3
Jodi	Female	7000	4
Tom	Male	5500	1
Ron	Male	5000	2
Ben	Male	6500	3
Todd	Male	3500	4
Mark	Male	5000	5
John	Male	4500	6

```
SELECT Name, Gender, Salary,
       ROW_NUMBER() OVER (PARTITION BY Gender ORDER BY Gender) AS RowNumber
FROM   Employees
```

Views

↳ Virtual Table





V1:

```
CREATE VIEW   WORKS_ON1
AS SELECT     Fname, Lname, Pname, Hours
   FROM       EMPLOYEE, PROJECT, WORKS_ON
   WHERE      Ssn=Essn AND Pno=Pnumber;
```

VIEWS are not generally subjected to Updation.

VIEWS can be queried just like any other table.

```
SELECT   Fname, Lname
FROM     WORKS_ON1
WHERE    Pname='ProductX';
```

V2:

```
CREATE VIEW   DEPT_INFO(Dept_name, No_of_emps, Total_sal)
AS SELECT     Dname, COUNT (*), SUM (Salary)
   FROM       DEPARTMENT, EMPLOYEE
   WHERE      Dnumber=Dno
   GROUP BY   Dname;
```

DEPT_INFO

Dept_name	No_of_emps	Total_sal

Querying the VIEWS

QV1:

```
SELECT   Fname, Lname
FROM     WORKS_ON1
WHERE    Pname='ProductX';
```

→

```
SELECT   Fname, Lname
FROM     EMPLOYEE, PROJECT, WORKS_ON
WHERE    Ssn=Essn AND Pno=Pnumber
         AND Pname='ProductX';
```

Approach 1

Approach 2 : VIEW Materialization ⟹ Physical Table (Temporarily created)
Instead of Virtual Table

Updating the VIEWS

- A view with a single defining table is updatable if the view attributes contain the primary key of the base relation, as well as all attributes with the NOT NULL constraint *that do not have* default values specified.
- Views defined on multiple tables using joins are generally not updatable.
- Views defined using grouping and aggregate functions are not updatable.

V1A: DROP VIEW WORKS_ON1;

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK ( NOT EXISTS  ( SELECT    *
                      FROM      EMPLOYEE E, EMPLOYEE M,
                                DEPARTMENT D
                      WHERE     E.Salary>M.Salary
                                AND E.Dno=D.Dnumber
                                AND D.Mgr_ssn=M.Ssn ) );
```

The salary of an employee must not be greater than the salary of the manager of the department that the employee works for.

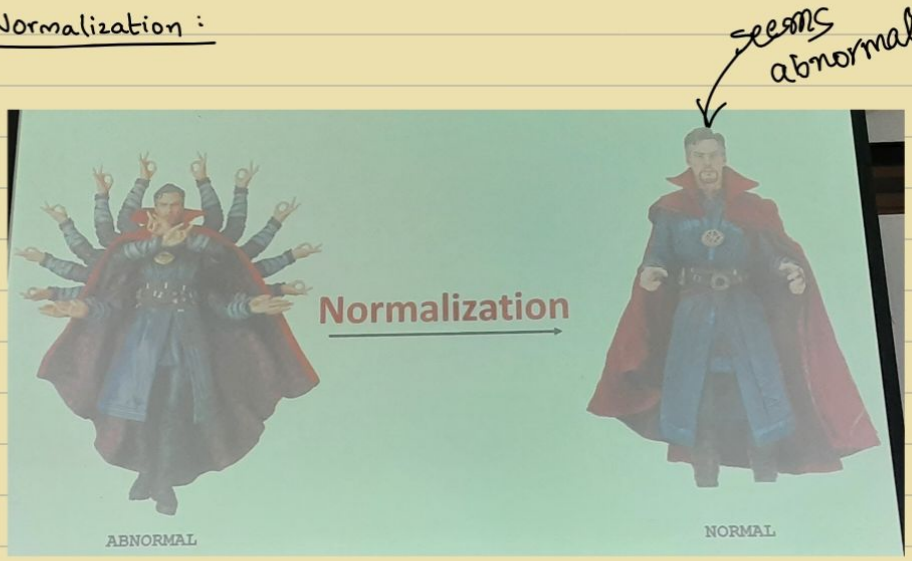
The basic technique for writing such assertions is to specify a query that selects any tuples that violate the desired condition.

By including this query inside a NOT EXISTS clause, the assertion will specify that the result of this query must be empty so that the condition will always be TRUE

Thus, the assertion is violated if the result of the query is not empty

——— X ———

→ Normalization :



Is Duplicates So bad ?
(In Real World)

Duplicates are very rare in real world.

→ Very Problematic (Ambiguous, Extra space)

Relation : Grouping of Attributes

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

PROJECT

Pname	Pnumber	Plocation	Dnum

} Better Schema design

VS

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation

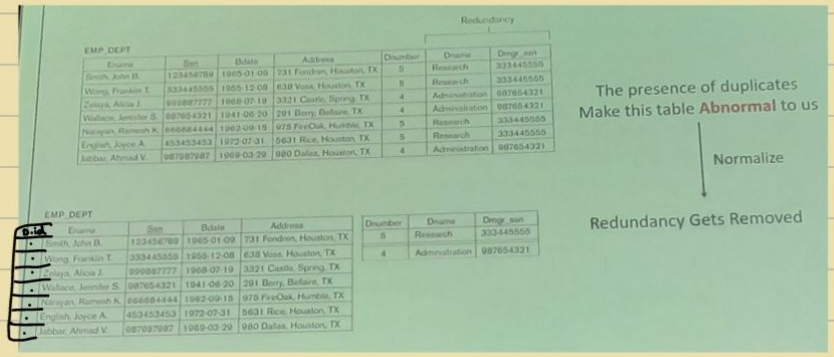
Metric : Redundancy (Space)

- Relational Schema ⟶ Test ⟶ 2 NF
  Relational Schema 2 ⟶ Test ⟶ 3 NF —Procedure→ 4 NF

Higher the NF ⇒ Better the Schema

EMP_DEPT (Redundancy)

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 FireOak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555



But how do you Identify / Detect redundancy '*within*' a relation ? ⟶ Functional Dependency

⟶ FD:

A ⟶ B : A functionally determines B
A is functionally dependendent on B
Given A , B can be determined

Bajaj ⟶ Pulsar ✗
Pulsar ⟶ Bajaj ✓

Ex Name ⟶ DOB
DOB ⟶ Name

Ex. Name ⟶ Roll.No ✗
Roll.No ⟶ Name ✓

Ex. Roll.no ⟶ Email ✓
Email ⟶ Roll No. ✓
} Candidate Keys

Ex. Age ⟶ DOB ✓
DOB ⟶ Age ✓
} Derived Attribute

i.e. Derived Attribute ⟶ FD ✓

Q) Roll.no., Name ⟶ Address ?
TRUE
i.e. More than 1 attribute is possible.

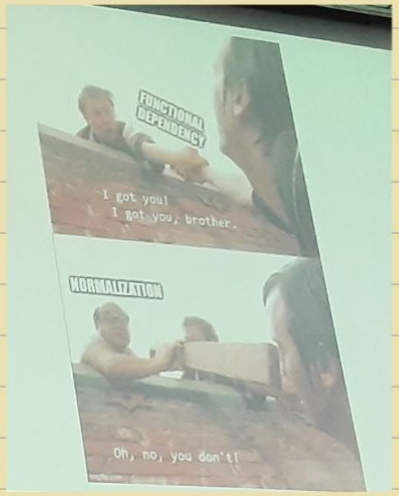
- References : NetNinja (YT)

Front-End Roadmap :
HTML & CSS ⟶ JS ⟶ TypeScript
Back-End Roadmap :
NodeJS ⟶ React
} Pre-requisites : 1 week

DB :
SQLite, Firebase

Ideas for Project :
Relational Algebra Compiler
Joins Simulator for understanding using animations
Photograph of Table & Query ⇒ Updated in App
⇒ Modification in Query should reflect
Something for the Institute (Mess menu app / Attendance tracker)

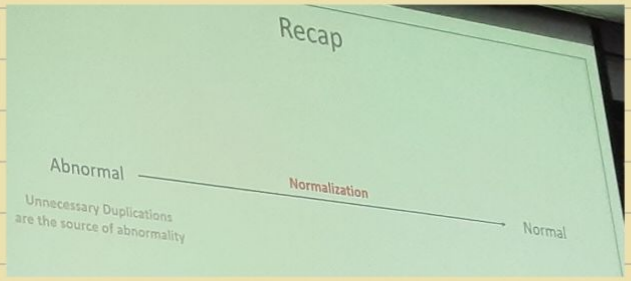
Outcome : Creativity ⇒ Put in CV
Project deadline : April 12 (Tentatively)

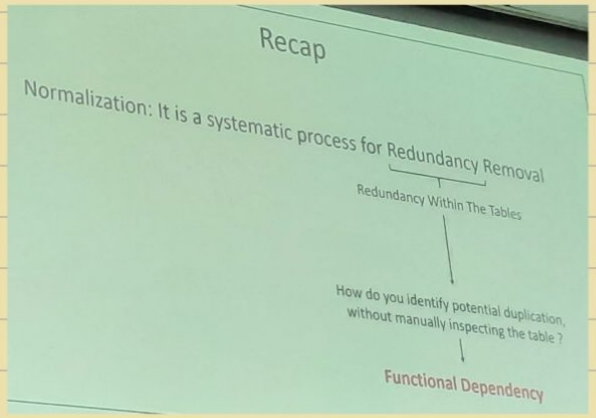


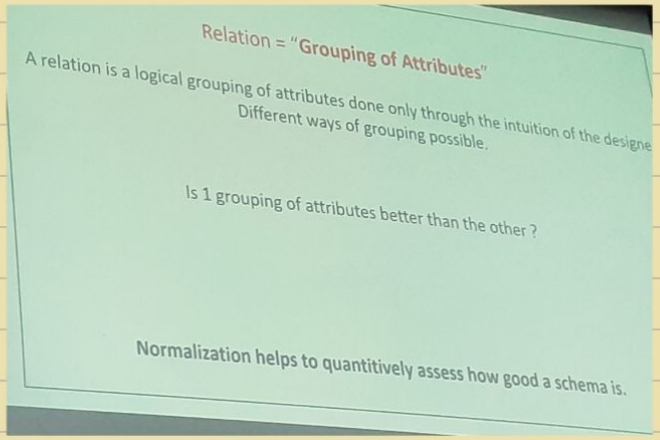
COAP ⟶ Website

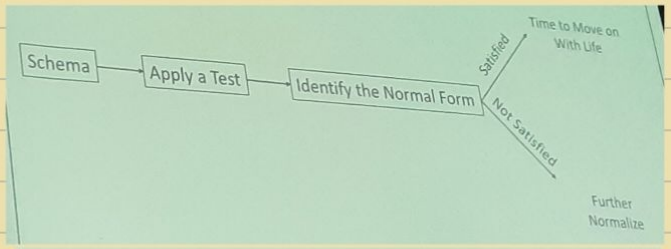
Issues:

(1) Insertion
(2) Update Anamoly
(3) Deletion Anamoly
(4) Space Wastage



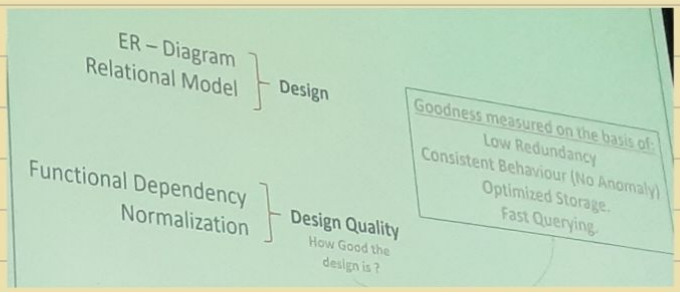


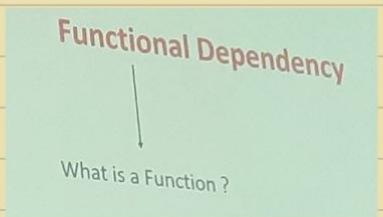


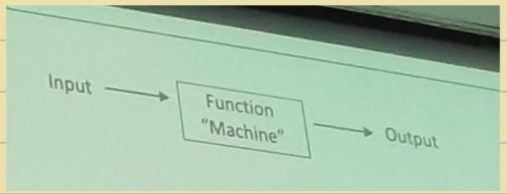


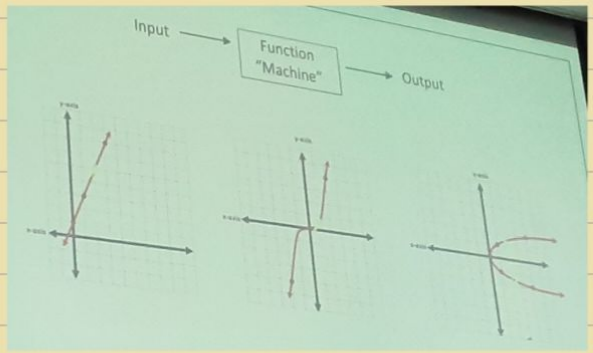
Normalization allows to 'evaluate' Schema's

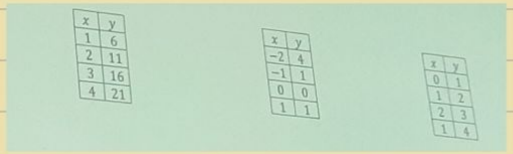
Allows us to use the best Schema



- 



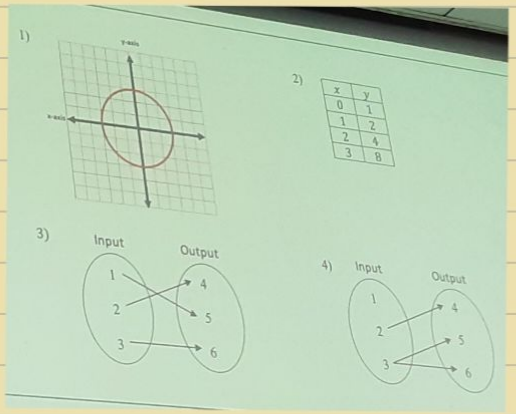




x	y
1	6
2	11
3	16
4	21

x	y
−2	4
−1	1
0	0
1	1

x	y
0	1
1	2
2	3
1	4



Functional Dependency

A → B

Determinant

Cause

A & B need not be single attribute, They can be a set of attributes

Dependent

Effect

A → B

FD is a constraint between two set of attributes, that must always hold true on a schema.

FDs are identified / derived from the meaning of the attribute, not from the instances.

FDs are the property of a Relational Schema

Functional Dependency

X → Y

Whenever two tuples agree on the X value, then they necessarily have to agree on the Y value also

X Functionally (or Uniquely) determines Y

Y is functionally dependent on X

If you Know X, you can Find Y

FDs should not be inferred from the instances...

But given an instance of a relation, what information you can deduce about the FDs

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d2
a3	b3	c4	d3

X → Y

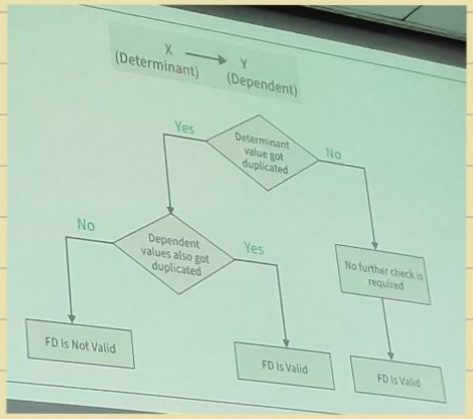
My Definition: Whenever X repeats Y also should repeat.

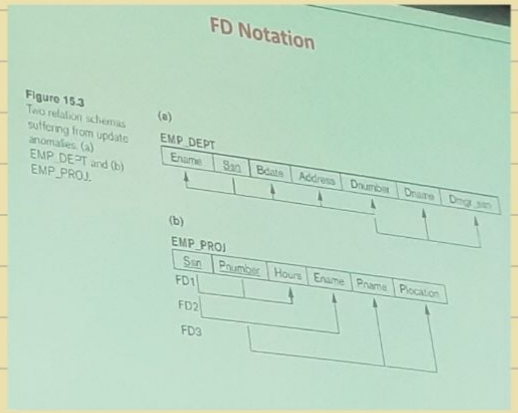
FDs helps to identify "**Potential**" duplicates.

FDs helps to locate areas (within the table) where repetition can occur potentially.

A	B	C
a	1	d
a	1	e
a	1	f
b	2	g

A → B





Project : No Duplicates

Group → Max. 5 students

Web Dev → Not Mandatory, Suggestion only.

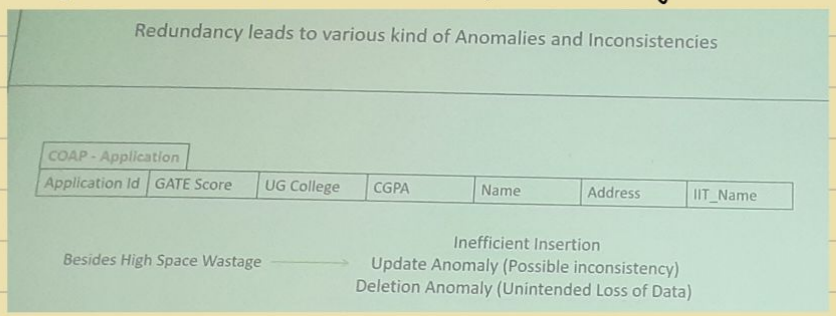
↳ Python / C++ can also be used.

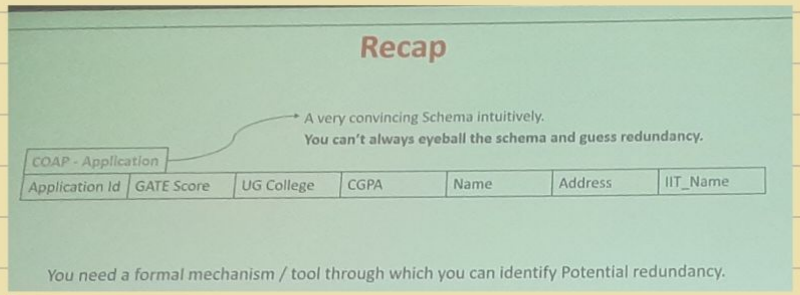
"Do something fun, Enjoy the process"

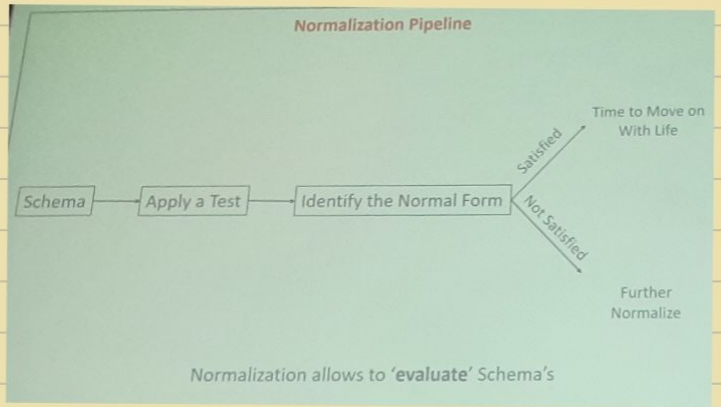
– Shahul Hameed Sir, 2025

18/3 Recap :

Redundancy is not just about space wastage.







ER – Diagram
Relational Model
} Development

Functional Dependency
Normalization
} Testing

How Good the design is ?

Functional Dependency

A → B

A & B need not be single attribute, They can be a set of attributes

Determinant

Dependent

X → Y

Whenever two tuples agree on the X value, then they necessarily have to agree on the Y value also

If you Know X, you can Find Y

My Definition: Whenever X repeats Y also should repeat.

X → Y

My Definition: Whenever X repeats Y also should repeat.

If for 2 different tuples the X Attribute values are same. If, $t_i[X] = t_j[X]$

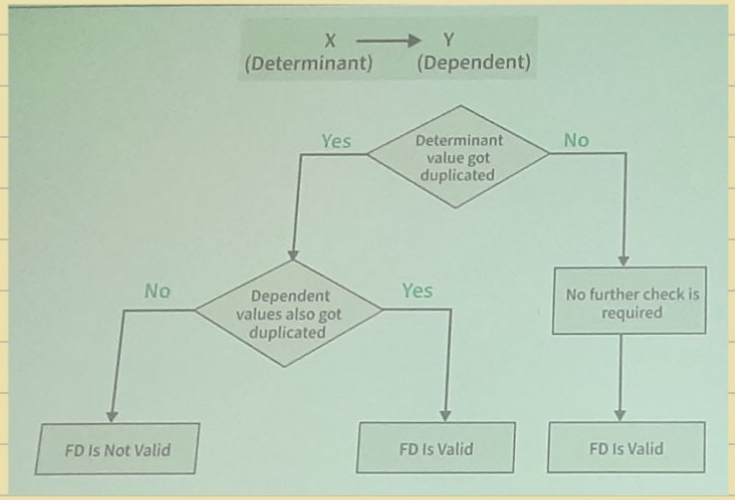
for any i and j;

Then, the Y attribute values Also should be same. Then, $t_i[Y] = t_j[Y]$

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

Given an Instance:

You can rule out FDs



Derived Attributes vs FDs

DOB —Today Date - DOB→ AGE

Height, Weight —Weight / Height²→ BMI

Derived Attributes vs FDs

Explicit Function Definition exists. Formula Exists

DOB —Today Date - DOB→ AGE

Height, Weight —Weight / Height²→ BMI

Derived Attributes vs FDs

The right side is "Computed" From the left side

DOB —(Today Date - DOB)→ AGE

Height, Weight —(Weight / Height²)→ BMI

Since a derived attribute is computed using other attributes, its value is always determined by them. → FD Always

Is the Converse True (FD means Derived Attribute) ?

$A \rightarrow B$

It means that for each unique value of A, there is a unique corresponding value of B.

However, this does not necessarily mean B is *computed or derived* from A.

Is the Converse True (FD means Derived Attribute) ?

$A \rightarrow B$

It means that for each unique value of A, there is a unique corresponding value of B.

However, this does not necessarily mean B is *computed or derived* from A.

Student_ID → Name
A student's name is functionally dependent on their ID, but the name is not derived from the ID.
It's just an *associated* value.

Country → Capital
The capital city is functionally dependent on the country, but it is not *calculated* from the country name

All derived attributes are functionally dependent,

But not all functionally dependent attributes are derived

Functional Dependencies

- **Trivial Functional Dependencies**
- **Non-Trivial Functional Dependencies**

$(AB) \rightarrow A$; $A \rightarrow A$; $Roll\_No \rightarrow Name$

$Roll, email \rightarrow e\_mail$

$X \rightarrow Y$

$X \supseteq Y \rightarrow$ Trivial

$X \cap Y = \phi$ Non-Trivial

Total Number of FDs Possible ?

A	B	C	D

X → Y

2^4-1 = 15 2^4-1 = 15

$$\text{Total number of FDs} = 15 \times 15 = 225$$

Total Number of FDs Possible ?

For a schema with **N attributes**, the total number of possible FDs is:

$$(2^N - 1) \times (2^N - 1)$$

Where:

- $2^N - 1$ is the number of non-empty subsets that can be used as determinants.
- $2^N - 1$ is the number of non-empty subsets that can be used as dependents.

How FDs are identified practically...

FD identification is done *'majorly'* during the design phase.

Recap

A very convincing Schema intuitively.
You can't always eyeball the schema and guess redundancy.

COAP - Application

Application Id	GATE Score	UG College	CGPA	Name	Address	IIT_Name

You need a formal mechanism / tool through which you can identify Potential redundancy.

Functional Dependency

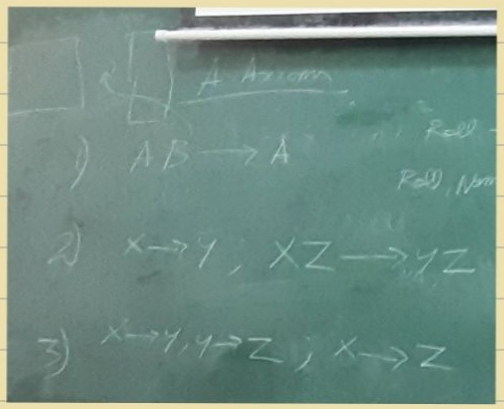
How FDs are identified practically...

FD identification is done *'majorly'* during the design phase.

1. FDs are manually identified from the real world meaning of the Attributes.
2. New FDs can be deduced from the existed FD set by applying Armstrong Axioms.

Additionally,
Some FDs can be empirically ruled out by looking at the instances.

You can also find the "Minimal Cover" to remove the redundant FDs



Armstrong Axioms

if Y⊂X then X→Y

reflexivity

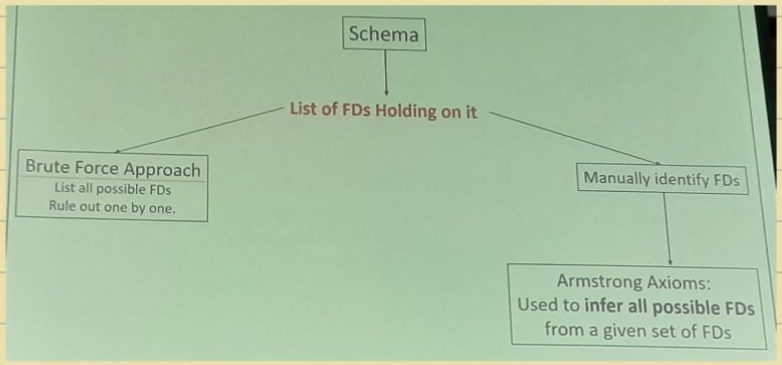
if X→Y then XZ→YZ

augmentation

if X→Y and Y→Z then X→Z

transitivity

19/3



Manually identify FDs

Database designers analyze business rules and domain knowledge to determine the initial set of FDs

This is often done intuitively by discussing with stakeholders, understanding constraints, and examining sample data.

Armstrong Axioms:
Used to infer all possible (implied) FDs
from a given set of FDs

This is not done *exhaustively* in practice.
(Finding all possible FDs is tedious and useless mostly)

Is there any other use of Armstrong Axioms ?

ISBN → Title, Publisher

ISBN,DOP → Author

Publisher → PublisherURL

You are sure that, these three FDs hold in your schema

ISBN → PublisherURL

You want to check whether this FD holds in your Schema

They are also used to find the Minimal Cover

Armstrong Axioms are more like a guideline which we use to manipulate the FDs Whenever the need arise.

The closure of an attribute set A, denoted as A^+,
is the set of all attributes that can be functionally determined by X using the given set of functional dependencies

R(A, B, C)

A → B

B → C

$A^+ = ABC$

$B^+ = BC$

$C^+ = C$

Closure of an Attribute

R (A,B,C,D,E,F,G)

A → B

BC → DE

AEG → G

$AC^+ =$

$\{AC\}^+ = ACB$

$\{ACB\}^+ = ACBDE$

$\therefore \{AC\}^+ = ACBDE$

Closure of an Attribute

R (A,B,C,D,E)

A → BC

CD → E

B → D

E → A

B^+ =

$B^+ = \{BD\}$

$E^+ = \{EABCD\}$

$E \longrightarrow A$

$A \longrightarrow BC$

$B \longrightarrow D$

Q) R(A, B, C, D, E)

$A \longrightarrow B$

$B \longrightarrow D$

$C \longrightarrow DE$

$CD \longrightarrow AB$

$E \longrightarrow A$

$A^+ = ?$

$A^+ = \{ABD\}$

$B^+ = \{BD\}$

$C^+ = \{CDEAB\}$

$D^+ = \{D\}$

$E^+ = \{EABD\}$

$CD^+ = \{CDABE\}$

$AC^+ = \{CDABE\}$

C : Candidate Key

Superset of C → Superkey

Q) R(A, B, C, D)

$A \longrightarrow B$

$B \longrightarrow C$

$C \longrightarrow D$

$D \longrightarrow A$

Candidate Keys :

A, B, C, D,

AB, BC, CD, DA, BD, AC

ABC, BCD, ACD, ABD

ABCD

} 15

$A^+ = \{ABCD\}$

$B^+ = \{BCDA\}$

$C^+ = \{CDAB\}$

$D^+ = \{DABC\}$

Q) R(A, B, C, D, E, F)

$C \longrightarrow F$

$E \longrightarrow A$

$EC \longrightarrow D$

$A \longrightarrow B$

$(?)^+ = \{ABCDEF\}$

$(CE)^+ = \{ABCDEF\}$

Essential Attributes

$CE^+ = \{CEFADB\}$

$CE \longrightarrow$ Candidate Key

Q) R(A, B, C, D, E, H)

$A \longrightarrow B$

$BC \longrightarrow D$

$E \longrightarrow C$

$D \longrightarrow A$

Essential Attributes ?

$(?)^+ = \{ABCDEH\}$

$EH^+ = \{EHC\}$ ×

ACEH has AEH

BCEH has BEH

DCEH has DEH

$AEH^+ = \{AEHBCD\}$ ✓

$EHB^+ = \{EHCBDA\}$ ✓

$CEH^+ = \{CEH\}$ ×

$DEH^+ = \{DEHABC\}$ ✓

∴ No superset from CEH.

∴ Essential Attributes : EH

Candidate Keys : AEH, BEH, DEH

21/3

RECAP

Armstrong Axioms

Technically these are rules to manipulate the FDs

Practically these are like a Intuitive guidelines which we follow to manipulate FDs

Things To Remember:

(1) Transitivity :

$A \longrightarrow B \;\&\; B \longrightarrow C$

$\therefore A \longrightarrow C$

(2) Decomposition :

$AB \longrightarrow DEF$

$\therefore AB \longrightarrow D \;\&$

$AB \longrightarrow E \;\&$

$AB \longrightarrow F$

Closure of an Attribute - Recap

Epsilon Closure of a State:
Set of all state that are reachable through the epsilon transition

Epsilon Closure of an Attribute:
Set of all attributes that can be derived from the given attribute, using the given FDs

Closure of an Attribute

A^+

Given A, what other Information (attributes) I can uniquely determine

Special Case

A^+

Given A, if I can determine "**all**" the attributes, then A is a Candidate Key

Finding Candidate Keys – No. of Possibilities

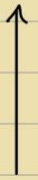
R (A,B,C,D)

2^4-1
Candidate keys possible

The Right Way to Check

R (A,B,C,D)

ABCD
ABC, ABD, ACD, BCD
AB, AC, AD, BC, CD
A, B, C, D



The Right Way to Check

R (A,B,C,D)

ABCD

ABC, ABD, ACD, BCD

AB, AC, AD, BC, CD

A, B, C, D

Super Key

Candidate Key

Shortcut to Check for Candidate Keys

R (A,B,C,D,E,F,G)

A → B
BC → DE
AEG → G

Observe the Right Side of the FDs

Check for attributes that doesn't appear

( )⁺ = A,B,C,D,E,F,G

A, C & F are essential attributes

Essential Attributes

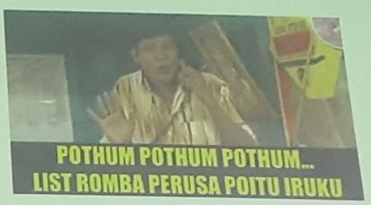
An **essential attribute** is an attribute that **must** be included in at least one candidate key because it **cannot** be derived (functionally determined) from any other attribute or set of attributes in the relation

Attributes that appear only on the left-hand side of the given FDs, but never on the right-hand side are essential.

These attributes are not functionally dependent on any other attributes and must be included in any candidate key to ensure uniqueness

Finding the Key

1. List out all the FDs
2. Identify the Essential Attributes.
3. Check if it is a key.
4. If not check each of its super set.



What happened to the idea of Finding the candidate key with Real world meaning and intuition ?

COAP - Application

Application Id	GATE Score	UG College	CGPA	Name	Address	IIT_Name

Application_Id → GATE, UG College, CGPA, Name, Address

What are the essential attributes here ?

(Application_ID , IIT_Name)

Name → Address ×

Equivalence of Two FD Sets

F: {A → C, AC → D, E → AD, E → H}

G: {A → CD, E → AH}

Step 2: Check if "G covers F"
If every FD in F is implied / Derivable from G

A^+ =
AC^+ = Find the Closure using G
E^+ =

F:
$A \rightarrow C$
$AC \rightarrow D$
$E \rightarrow AD$
$E \rightarrow H$

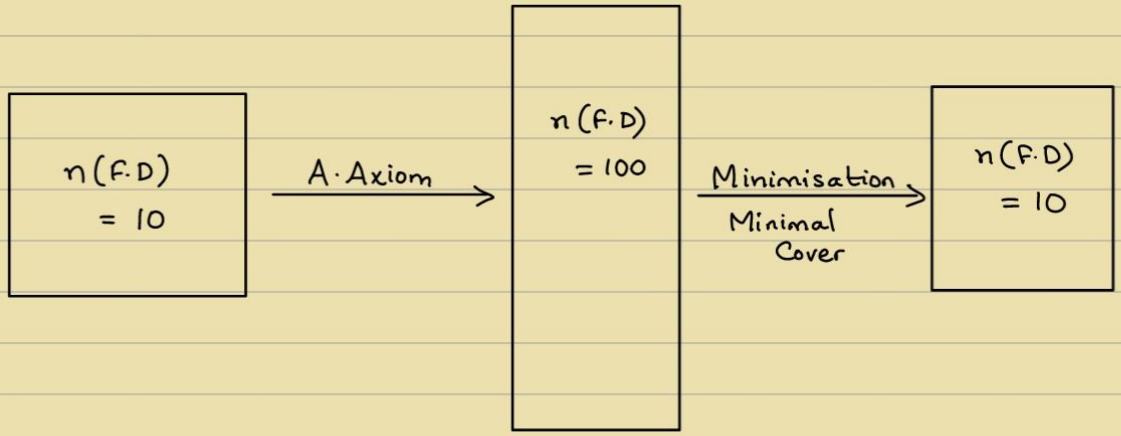
G:
$A \rightarrow CD$
$E \rightarrow AH$

$A^+ = \{ACD\}$
$(AC)^+ = \{ACD\}$
$E^+ = \{EAHCD\}$

G covers F

$A^+ = \{A, C, D\}$
$E^+ = \{A, D, E, H, C\}$

F covers G



Procedure to Find Minimal Cover

1. Decompose the FDs such that RHS contain Single attribute
   $A \rightarrow BC$; $A \rightarrow B$, $A \rightarrow C$

2. Find the redundant FDs and delete from the Set
   $A \rightarrow B$; $B \rightarrow C$;
   $A \rightarrow C$

3. Find the redundant attributes on LHS and delete them
   $AB \rightarrow C$;
   A Can be deleted if B^+ contains A

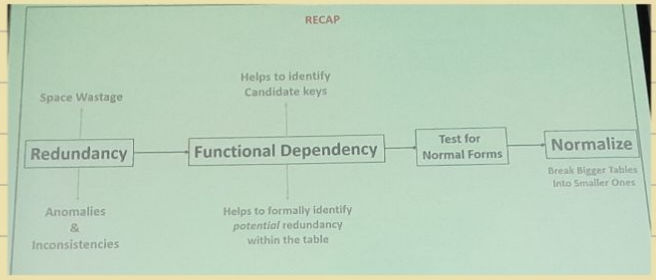
Q) $A \rightarrow C$, $AC \rightarrow D$, $E \rightarrow AD$, $E \rightarrow H$

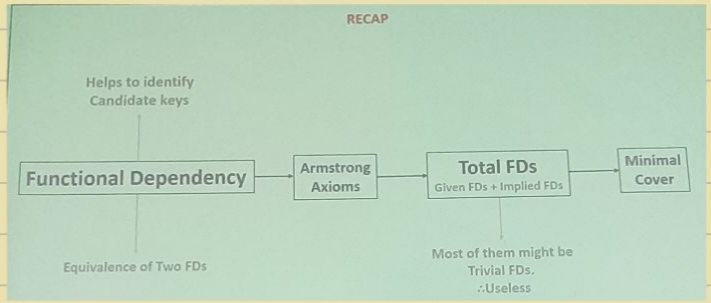
Step 1:
$A \rightarrow C$
$AC \rightarrow D$
$E \rightarrow A$
$E \rightarrow D$ ← X
$E \rightarrow H$

Step 2:
$A^+ = \{A\}$
$AC^+ = \{AC\}$
$E^+ = \{EDH\}$
$E^+ = \{EAHCD\}$
$E^+ = \{DAC\}$
↳ Closure without the corresponding

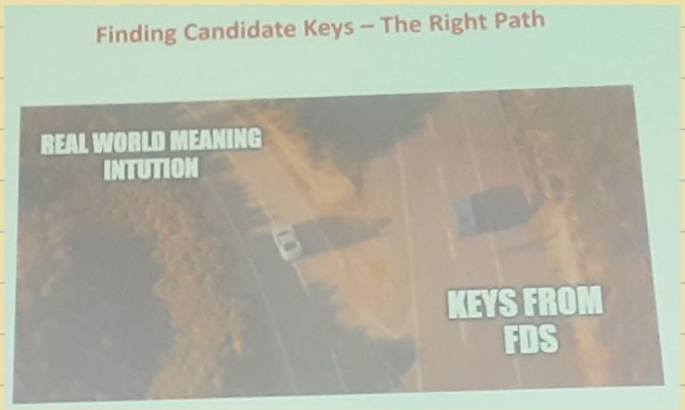
Step 3:
$AC \rightarrow D$
$A^+ = \{AC\}$ $\quad A \rightarrow D$
$C^+ = \{C$
∴ C is not required

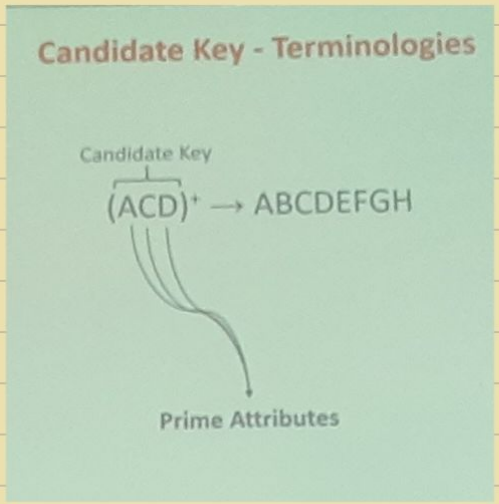
25/3





Closure - Essential attributes



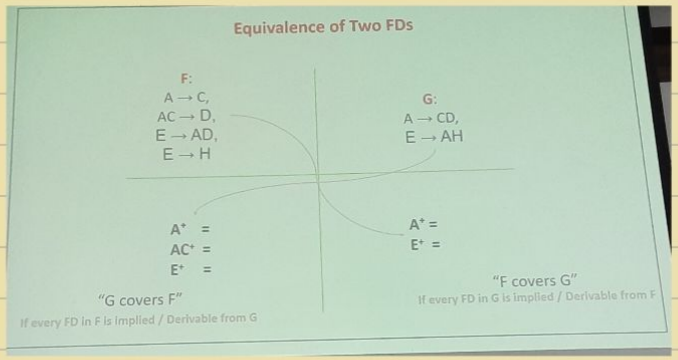


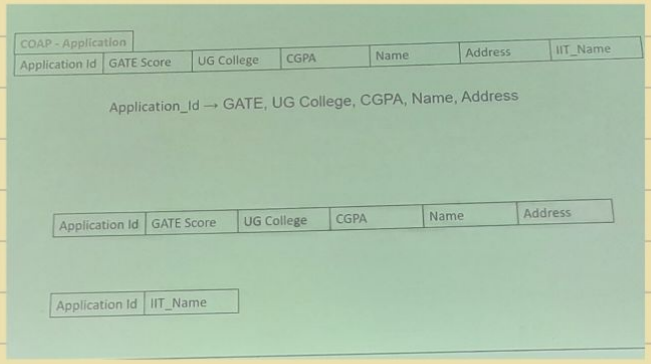
Procedure to Find Minimal Cover

1. Decompose the FDs such that RHS contain Single attribute
   $A \rightarrow BC$; $A \rightarrow B$, $A \rightarrow C$

2. Find the redundant FDs and delete from the Set
   $A \rightarrow B$; $B \rightarrow C$;
   $A \rightarrow C$

3. Find the redundant attributes on LHS and delete them
   $AB \rightarrow C$;
   A Can be deleted if B^+ contains A



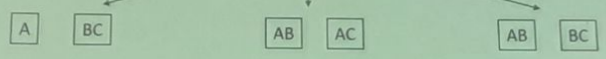


Anatomy of Decomposition

A	B	C
A1	B1	C1
A2	B1	C1
A1	B2	C2

Anatomy of Decomposition

A	B	C
A1	B1	C1
A2	B1	C1
A1	B2	C2



Lossy Decomposition

Anatomy of Decomposition

A	B	C
A1	B1	C1
A2	B1	C1
A1	B2	C2

Ensure there is a common attribute (Other wise only Cross Product Possible)

The common attribute should be a Key in any one of the Table.

Anatomy of Decomposition

R(A1, A2, ...AN)

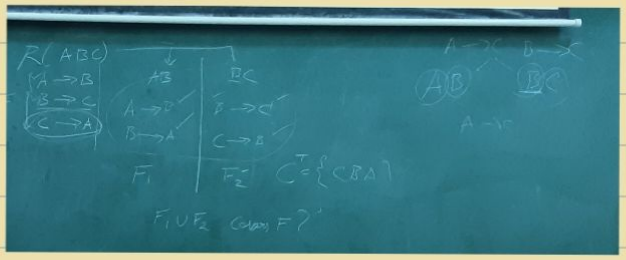
FDs Holding on R (Say F)

R1 (.....) $F_1 \subseteq F$

R2 (.....) $F_2 \subseteq F$

$F_1 \cup F_2 = F$ → FD Preserving Decomposition

Desirable, but a Necessity



First Normal Form

A relation is in first normal form if and only if no attribute domain has relations as elements.

Or more informally, that no column of a table, can have tables as values

Customer	Customer ID	Transactions		
		Transaction ID	Date	Amount
Abraham	1	12890	2003-10-14	-87
		12904	2003-10-15	-50
Isaac	2	Transaction ID	Date	Amount
		12898	2003-10-14	-21
Jacob	3	Transaction ID	Date	Amount
		12907	2003-10-15	-18
		14920	2003-11-20	-70
		15003	2003-11-27	-60

https://en.wikipedia.or

First Normal Form

Only Atomic Values are allowed.

No Multivalued, or Composite attributes allowed.

Customer	Customer ID
Abraham	1
Isaac	2
Jacob	3

Customer ID	Transaction ID	Date	Amount
1	12890	2003-10-14	-87
1	12904	2003-10-15	-50
2	12898	2003-10-14	-21
3	12907	2003-10-15	-18
3	14920	2003-11-20	-70
3	15003	2003-11-27	-60

First Normal Form

Only Atomic Values are allowed.

No **Multivalued**, or **Composite attributes allowed**.

The very definition of Relational Model ensures that, a relation is by default is in 1NF

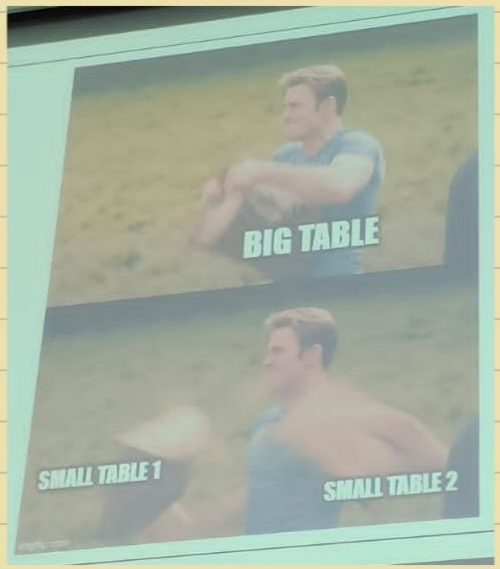
WHY 1NF ?

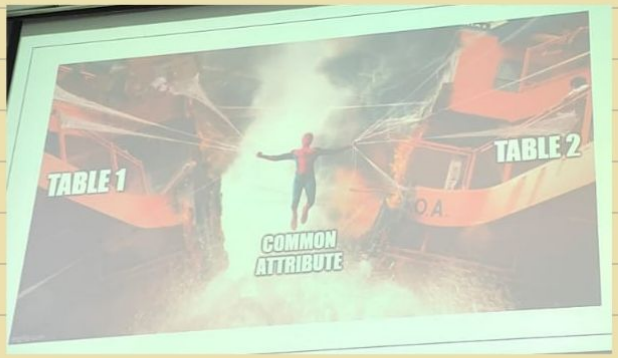
Allows presenting, storing and interchanging relational data in the form of regular two-dimensional arrays.
(Supporting nested relations would require more complex data structures.)

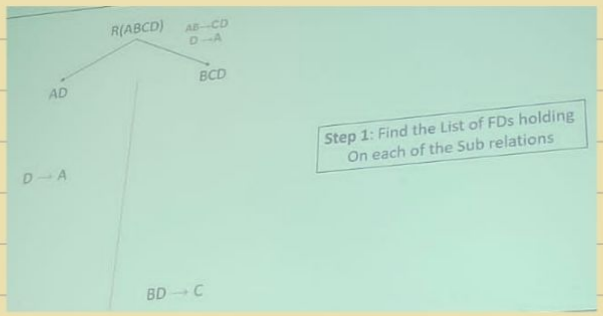
Makes further normalization levels possible which eliminate data redundancy and anomalies.

1st NF : Table is flat (2D)

27/4







First Normal Form

1NF, ensures that, its just plain simple table, that you are working with

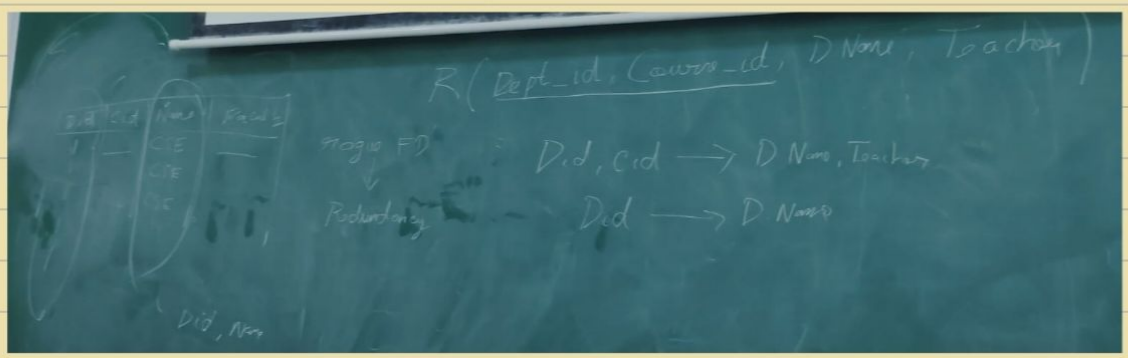
Second Normal Form

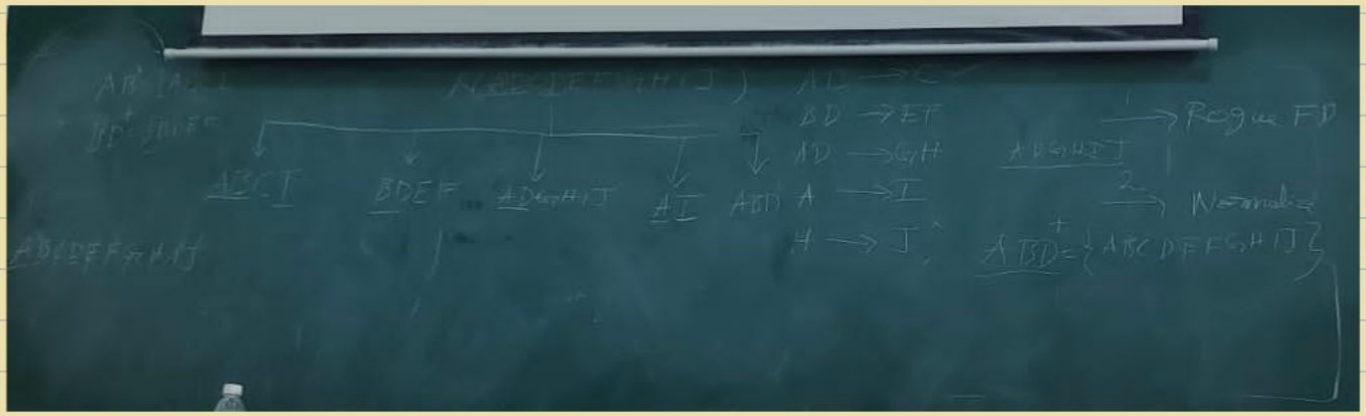
Check Whether it is in 2NF

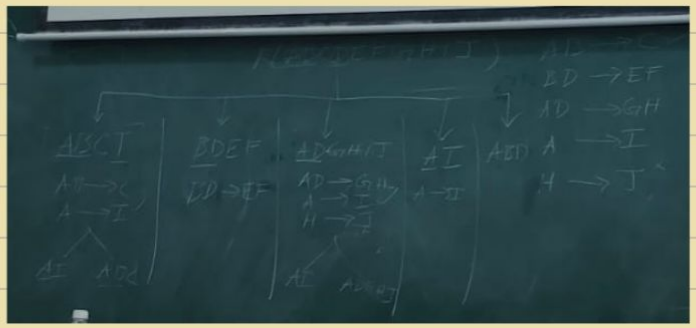
If not Normalize it to 2NF

How Do you Check ?

You look out for certain kind of FDs.
If They are present, then the table is not in 2NF







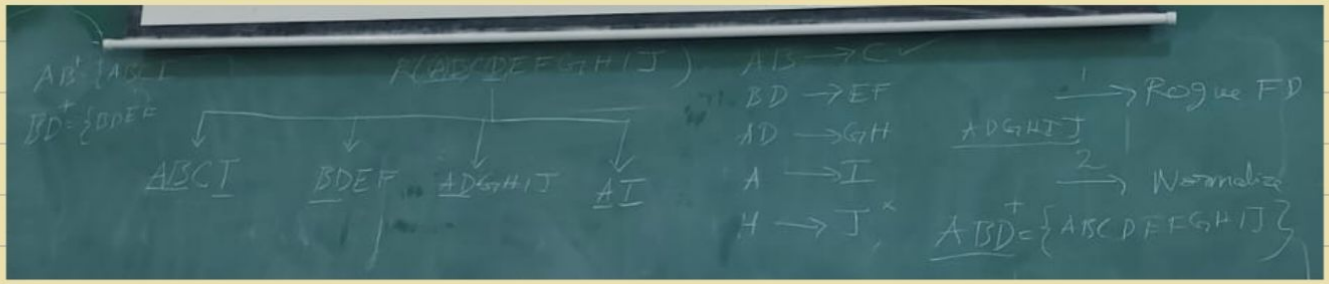
2NF Decomposition

For Each PD make a New Table

Always there will be one table with:

1. *Candidate Key*
2. *Any Non-Prime attribute fully dependent on that C.Key*

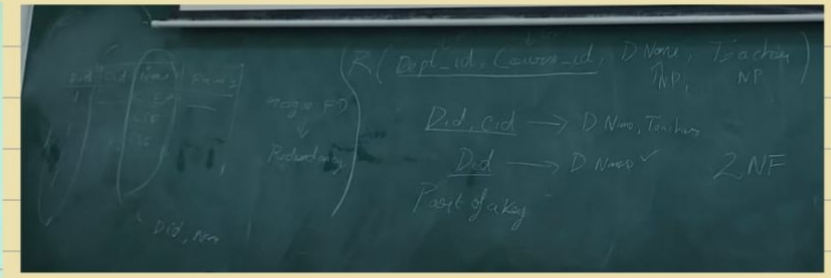
Case 2: Happens only if, There is a FD where a non-prime attribute is entirely dependent on the C.Key

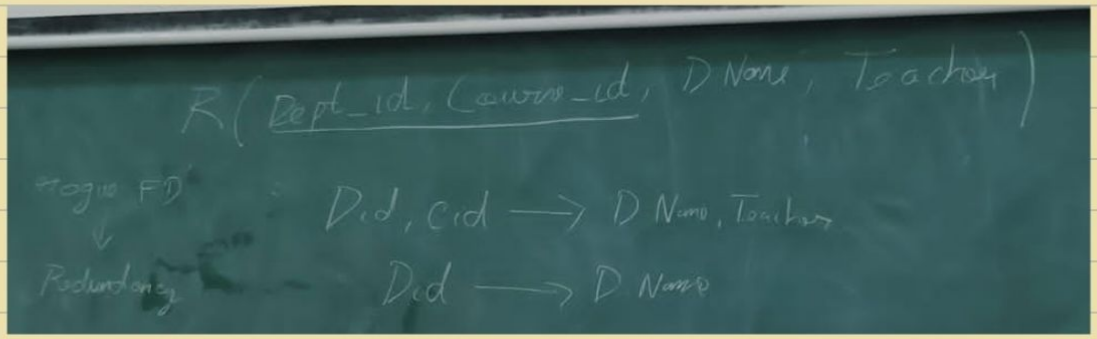


Second Normal Form

No Partial Dependency

A non-prime attribute dependent on a part of a candidate key





WHY 1NF ?

Makes further normalization levels possible which eliminate data redundancy and anomalies.

Second Normal Form

Check Whether it is in 2NF

If not Normalize it to 2NF

First Normal Form

Only Atomic Values are allowed.

No Multivalued, or Composite attributes allowed.

The very definition of Relational Model ensures that, a relation is by default is in 1NF

FD Preservation - Desirable but Not Mandatory

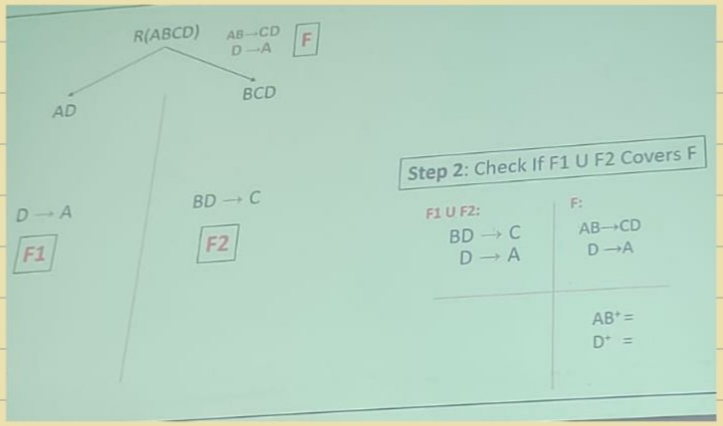
FD Preserving Ensures that all FDs of the original relation can still be enforced without needing to *recompute* joins.

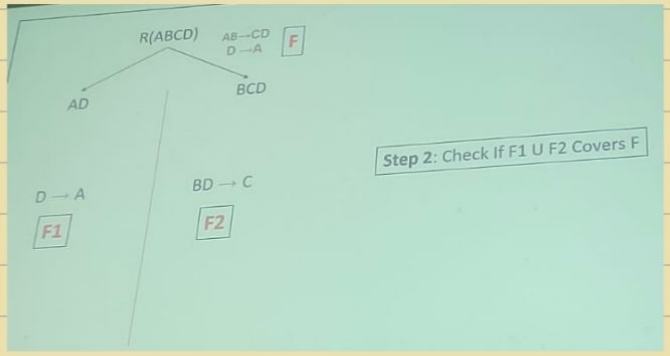
Without this, we may need to join multiple tables to check/enforce a constraint, which can be costly.

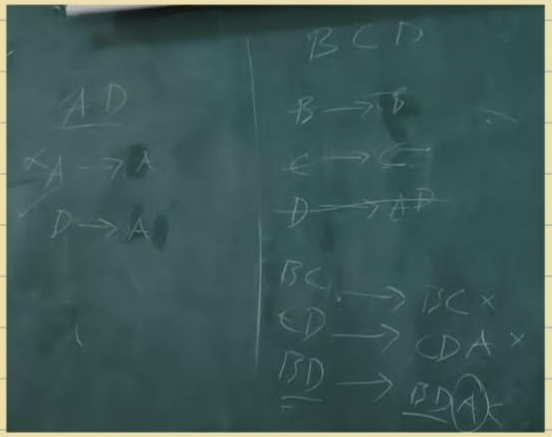
Customer	Customer ID
Abraham	1
Isaac	2
Jacob	3

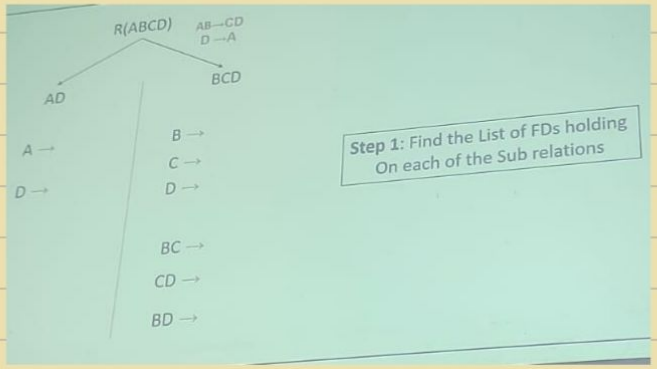
Customer ID	Transaction ID	Date	Amount
1	12890	2003-10-14	−87
1	12904	2003-10-15	−50
2	12898	2003-10-14	−21
3	12907	2003-10-15	−18
3	14920	2003-11-20	−70
3	15003	2003-11-27	−60

https://en.wikipedia.org/wiki/First_normal_form









Anatomy of Decomposition

R(A1, A2, ...AN)

FDs Holding on R (Say F)

R1 (.....)

$F_1 \subseteq F$

R2 (.....)

$F_2 \subseteq F$

$F_1 \cup F_2 = F$ → FD Preserving Decomposition

Desirable, but a Necessity

Foreign Key

Note: Dept_Id is a Primary Key

Dept No. refers Dept Id

E_Name	Emp_Id	Dept No
Eren	E_180	12
Mikasa	E_181	12

Referencing Relation

Related Tuples

Dept_Name	Dept_Id	Start Year
Scout Regimen	12	1450
Military Corps	13	1300

Referred Relation

This is Foreign Key

A F.Key should always refer to an attribute, which is a P.Key in that referred table.

Anatomy of Decomposition

A	B	C
A1	B1	C1
A2	B1	C1
A1	B2	C2

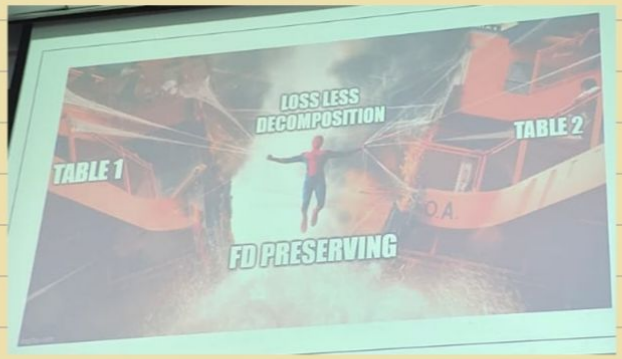
A | BC

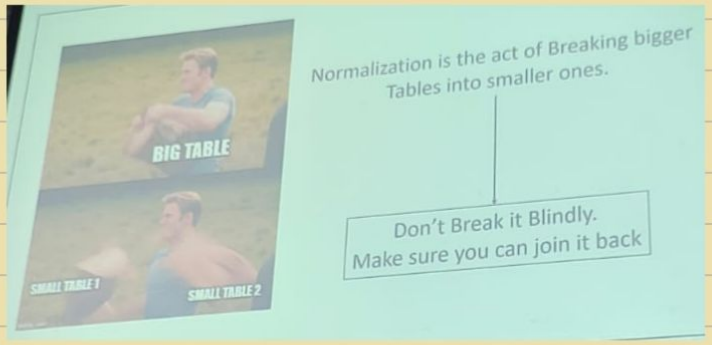
AB | AC

AB | BC

Ensure there is a common attribute (Other wise only Cross Product Possible)

The common attribute *should be a Key* in any one of the Table.

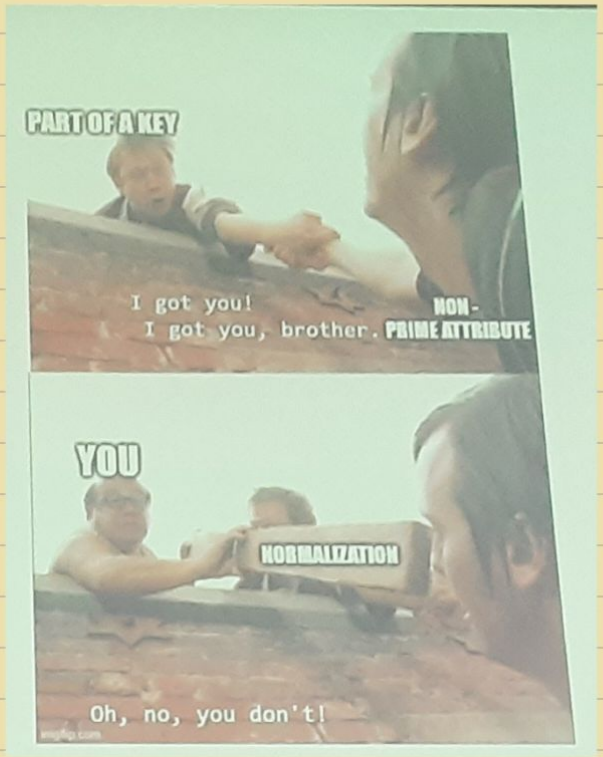

LOSS LESS
DECOMPOSITION
TABLE 1
TABLE 2
FD PRESERVING


BIG TABLE
SMALL TABLE 1
SMALL TABLE 2
Normalization is the act of Breaking bigger Tables into smaller ones.
Don't Break it Blindly.
Make sure you can join it back


BIG TABLE
SMALL TABLE 1
SMALL TABLE 2

P.T.O

3/4

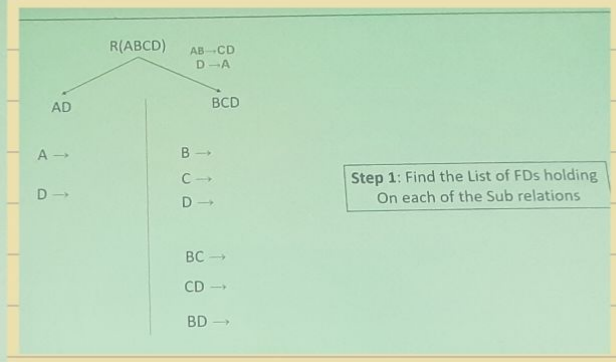


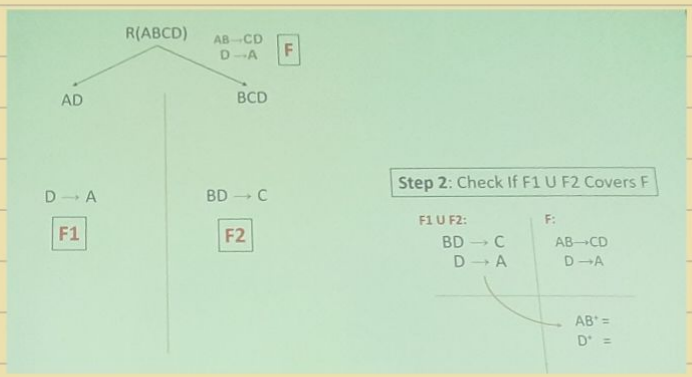
Normalization – A Broader View

Normalization is the act of Breaking bigger Tables into smaller ones.

Don't Break it Blindly.
Make sure you can join it back

- Lossless Decomposition (Common Attribute, C.Key in One Table)
- FD Preserving (Desirable, but not Mandatory)





First Normal Form

1NF, ensures that, its just plain simple table, that you are working with

Normalization – The Actual Process

For each Rogue FDs

Break the Table, such that, LHS (of the rogue FD) and all attributes Dependent on it are made into a new table

Repeat

What is Rogue ?

2NF:
Partial Dependencies

3NF:
Transitive Dependencies

1NF → 2NF → 3NF → 4NF → ....

You are progressively re-organizing or re-structuring the relational schema,
such that,
you limit the possibility of
Expressing redundant information.

Second Normal Form

Check for Partial Dependencies → Not in 2NF

Normalize it

Course_Id	Dept_id	Faculty	Dept_Name

R(ABCDEFGHIJ)

AB → C
BD → EF
AD → GH
A → I
H → J

Partial Dependencies

Step 1:
Identify the Candidate Key.

ABD is the candidate key

AB → C
BD → EF
AD → GH
A → I
H → J

AB⁺ = ABCI
BD⁺ = DBEF
AD⁺ = ADGHIJ
A⁺ = AI

R(ABCDEFGHIJ)

ABCI — AB → C, A → I — PD → ABC, AI

BDEF — BD → EF

ADGHIJ — AD → GH, A → I, H → J — PD → ADGHJ, AI

AI — A → I

ABD

ABCDEFGHIJ Whatever remains, make it into a new table

2NF Decomposition

For Each PD make a New Table

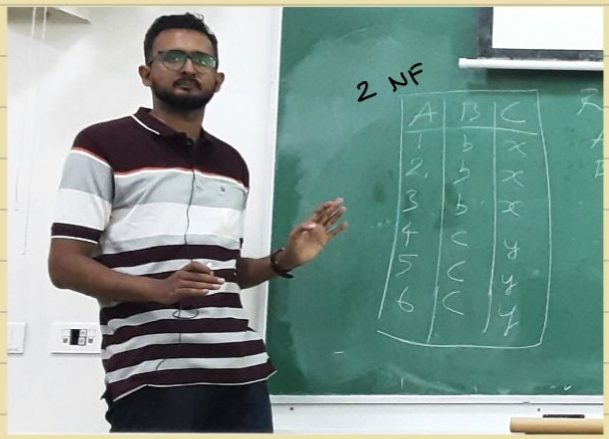
Always there will be one table with:
1. Candidate Key
2. Any Non-Prime attribute fully dependent on that C.Key

Case 2: Happens only if, There is a FD where a non-prime attribute is entirely dependent on the C.Key

We are effectively making sure that for each new table, all of its attributes are dependent on the Entire candidate key

2NF Decomposition – What's happening

For each PD (Part of a key determining, a non-prime attribute),
make a new table, such that,
the part of a key, becomes
The complete key for the new table.



3NF

What's Rogue Here ?

Transitive Dependency

A **transitive dependency** in a relation **R(A, B, C)** occurs when:

- **A → B** (A functionally determines B)
- **B → C** (B functionally determines C)
- **A does not directly determine C**, but C is indirectly dependent on A through B.

3NF

WAY 1:
First, Go to 2NF (i.e., Remove PDs)
Then, Go to 3NF (i.e., Remove TDs)

WAY 2:

A relational schema R is in 3NF only if in every Non-Trivial FD X → Y, either:

1. X is a superkey.
   [OR]
2. Y is a Prime attribute

} Eliminates both PD and TDs

4/4

BCNF

A relational schema R is in BCNF only if in every Non-Trivial FD X → Y, either:

1. X is a superkey.

BCNF (Boyce-Codd Normal Form)

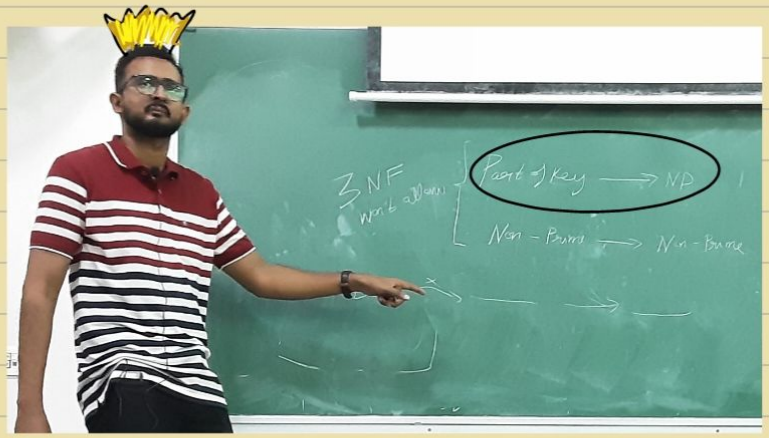
X → Y
(Part of) Prime Non - Prime

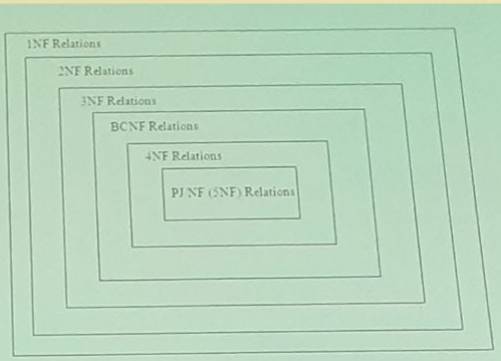
X → Y
Non - Prime Non - Prime

These two (PD & TD) are covered / removed by 2NF and 3NF

What about this Case ?

X → Y
(Part of) Prime Prime





- While higher level normal forms are available, normalization up to BCNF is often found to be adequate for business data.

The key difference between 3NF and BCNF revolves around the determinants (LHS) of FDs.

BCNF: Every determinant must be a super key.
(LHS must always be a super key)

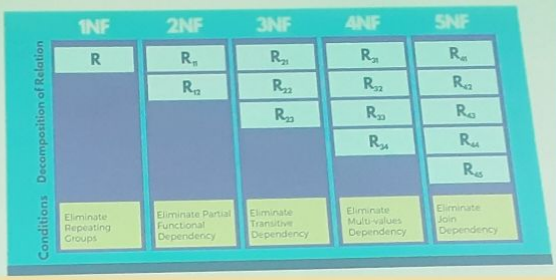
3NF: Allows determinants that are not super keys, as long as the dependent attribute is a prime attribute (part of a candidate key).

BCNF ≡ 3.5 NF

What About FD Preserving Property ?

Any decomposition of 2NF and 3NF need not be FD preserving, but you can always
Find a 3NF decomposition which is FD preserving.

How Far Should I Normalize ?



For BCNF, there need not be FD preserving always

Synthesis Algorithm:

Transactions :

→ Law of Conservation of Net Balance.

TRANSACTIONS

A transaction is a collection of instructions which
Operates upon the DB and may affect the state of the DB

It is a set of operations used to perform a logical unit of work

Real World Work

TRANSACTIONS

Transactions interacts with DB and make changes to it

T → DataBase

You have already ensured that DB is consistent, Via Proper ER/Relational Design. Normalization etc...

Make Transactions consistent → Whole System Consistent

5/4 Transaction (Practical definition) :

A Transaction is a sequence of One or More SQL Operations treated as a Unit.

TRANSACTIONS

Transactions interacts with DB and make "changes" to it

T → DataBase

Transactions can potentially make the DB inconsistent

You have already ensured that DB is consistent, Via Proper ER/Relational Design. Normalization etc...

Make Transactions consistent → Whole System Consistent

TRANSACTIONS

It is a set of operations used to perform a logical unit of work

What operations are Possible ?

Read (X)
Write (X)

Get → Modify → Put

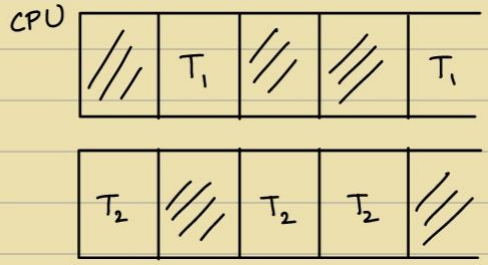
How does a Transaction Look

A = 100
B = 200

Say you want to transfer 50 Rs from A to B

```
Begin Transaction
R(A)      → I/O
A = A-50  → CPU
W(A)      → I/O
R(B)      → I/O
B = B+50  → CPU
W(B)      → I/O
End Transaction
Commit Transaction
```

Con : Two people withdrawing at same time



Running Transactions Simultaneously

Multi-programming

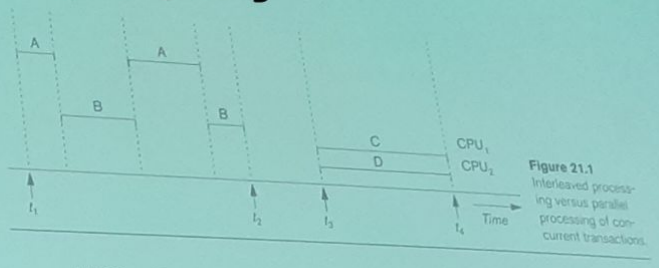


Figure 21.1 Interleaved processing versus parallel processing of concurrent transactions.

A,B,C,D are Transactions.

Potential problems with Running Transactions Concurrently

```
UPDATE college SET enrollment = enrollment + 1000 WHERE cname ='IIITDM';
```

Both are running Concurrently

```
UPDATE college SET enrollment = enrollment + 1500 WHERE cname = 'IIITDM';
```

Say current Enrollment is 2000.

∴ 3000 or 3500 or 4500

Potential problems with Running Transactions Concurrently

```
UPDATE Apply SET decision = 'Y' WHERE sID IN
                    (SELECT sID FROM Student WHERE GPA > 3.9);
```

Both are running Concurrently

```
UPDATE Student SET GPA = (1.1) * GPA WHERE Birth_Place = Mars;
```

Potential problems with Running Transactions Concurrently

```
INSERT INTO Archive
      SELECT * FROM Apply WHERE decision = 'N';
DELETE FROM Apply WHERE decision = 'N';
```

Both are running Concurrently

```
SELECT COUNT(*) FROM Apply;
SELECT COUNT(*) FROM Archive;
```

Recommended Properties of Transactions (AC)

- **Atomicity.** A transaction is an atomic unit of processing; it should either be performed in its entirety or not performed at all.
- **Consistency preservation.** A transaction should be consistency preserving, meaning that if it is completely executed from beginning to end without interference from other transactions, it should take the database from one consistent state to another.

Recommended Properties of Transactions (I)

- **Isolation.** A transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executing concurrently. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.

Imagine standing in a Queue at a hotel to order Food
VS
Sitting at your own table and ordering food

"Your Biryani, Your thing"
– Shahul Hameed Sir, 2025

Physical Isolation

→ Logical Isolation

Recommended Properties of Transactions

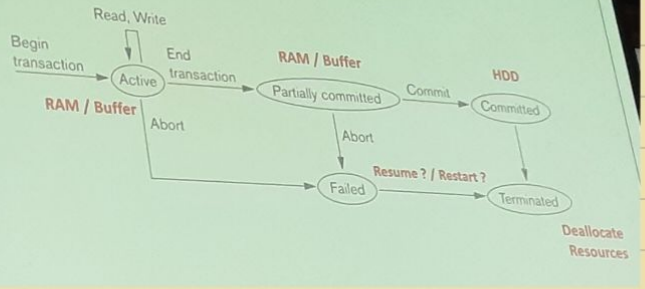
You should be able to run multiple transactions simultaneously, but it should
Appear to be running in Isolation

Recommended Properties of Transactions (D)

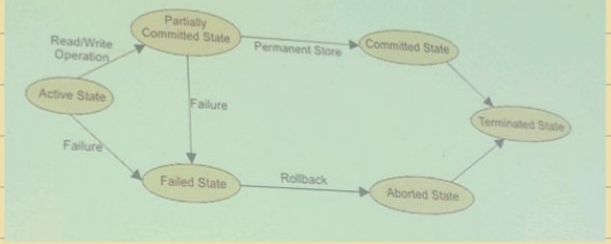
- **Durability or permanency.** The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure.

ACID Properties ↗

State Transition Diagram for Transactions



State Transition Diagram for Transactions



8/4

Theory	
Mid – Sem	30
HackerRank Assignment	10 10
End-Sem	50

Lab	
Mid – Sem	20
Cont Ass Project	20 20
End-Sem Viva	30 10

TRANSACTIONS

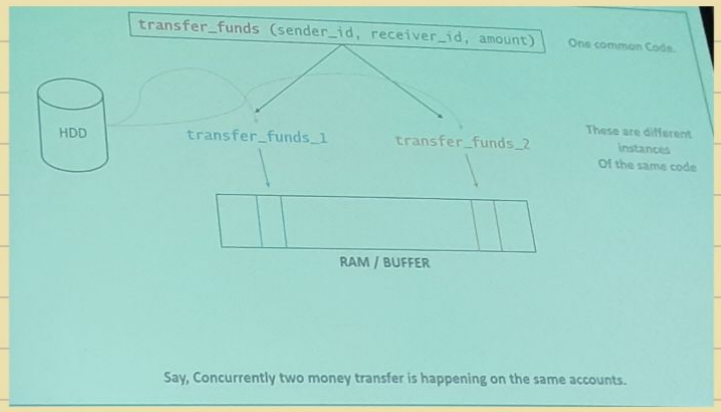
It is a set of operations used to perform a logical unit of work

A Transaction is a sequence of One or More SQL Operations treated as a Unit.

How does a Transaction actually look ?

```
def transfer_funds(sender_id, receiver_id, amount):
    try:
        conn = psycopg2.connect(
            dbname="bank", user="admin", password="secret",
host="localhost"
        )
        conn.autocommit = False  # Start transaction
        cur = conn.cursor()

        # Deduct from sender
        cur.execute("UPDATE accounts SET balance = balance - %s
WHERE id = %s", (amount, sender_id))

        # Add to receiver
        cur.execute("UPDATE accounts SET balance = balance + %s
WHERE id = %s", (amount, receiver_id))

        conn.commit()
        print("Transfer successful!")

    except Exception as e:
        conn.rollback()
        print("Transfer failed:", e)

    finally:
        cur.close()
        conn.close()
```

Typically a combination of Higher level programming language And Embedded SQL



Say, Concurrently two money transfer is happening on the same accounts.

How does a Transaction Look to US ?

```
Begin Transaction
R(A)
A = A-50
W(A)
R(B)
B = B+50
W(B)
End Transaction
Commit Transaction
```

Abstraction is the process of hiding complex details and showing only the essential features relevant to the context.

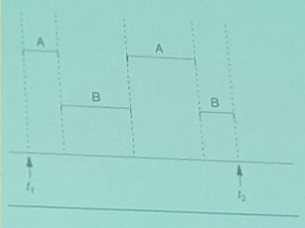
Who manages the Transaction the DBMS Software or the OS ?

A transaction is created & managed by the DBMS, but it relies on the OS for key operations like:

- Process scheduling
- Memory management
- File system access (reading/writing to disk)

So, while the DBMS handles the logic and rules of the transaction (like atomicity, consistency), the OS provides the infrastructure to execute it. It's a collaboration.

The Gray Dashed Lines are Switching



OS does this switching for you.
It figures out, when a process is busy in I/O
Pre-Empts it, and let some other process to run

OS does this for the Sake of Efficiency,
It does not care about your Bank Balance.

It's the Duty of the DBMS software to ensure that, ACID property is retained, Irrespective of the way in which the underlying OS handles the execution.

A - Atomicity
All or Nothing Transactions
Never Stop Midway (Undo Partial effects of Partial Transactions)

C - Consistency
Guarantees Committed Transaction State
The Net Balance Should be same.

I - Isolation
Transactions are Independent
Each user thinks he is the only one Using the system (While being unaware that others are using it)

D – Durability
Committed Data is Never Lost
If system crashes after a Transaction commits All effects of the transaction should be retained, When the system is back up and running

Property	Responsibility for maintaining properties
Atomicity	Transaction Manager
Consistency	Application programmer
Isolation	Concurrency Control Manager
Durability	Recovery Manager

What Kind of Failures Can Occur

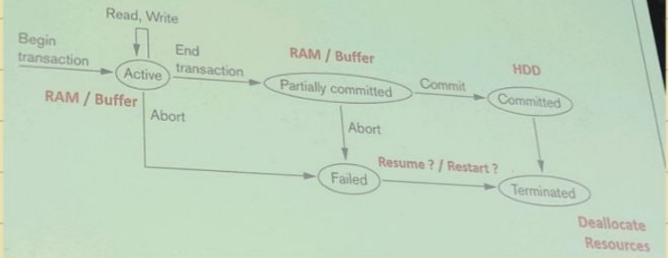
Types of Failures. Failures are generally classified as transaction, system, and media failures. There are several possible reasons for a transaction to fail in the middle of execution:

1. **A computer failure (system crash).** A hardware, software, or network error occurs in the computer system during transaction execution. Hardware crashes are usually media failures—for example, main memory failure.
2. **A transaction or system error.** Some operation in the transaction may cause it to fail, such as integer overflow or division by zero. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error.[3] Additionally, the user may interrupt the transaction during its execution.
3. **Local errors or exception conditions detected by the transaction.** During transaction execution, certain conditions may occur that necessitate cancellation of the transaction. For example, data for the transaction may not be found. An exception condition,[4] such as insufficient account balance in a banking database, may cause a transaction, such as a fund withdrawal, to be canceled. This exception could be programmed in the transaction itself, and in such a case would not be considered as a transaction failure.

State Transition Diagram for Transactions

1. **Active** – The transaction is being executed.
2. **Partially Committed** – All statements have executed, and the transaction is ready to commit.
3. **Committed** – Changes are made permanent in the database.
4. **Failed** – Some error occurred, and the transaction cannot proceed.
5. **Aborted** – All changes are rolled back, and the database is restored.

State Transition Diagram for Transactions



Failed State Vs Abort

Transaction Failure = An error occurred while executing the statements from transaction and hence it failed.

Transaction Abort = An explicit request to stop the ongoing transaction. It could very well also be because of transaction failure or user request.

→ Concurrency Issues :

①

The Lost Update Problem (W-W Conflict)

Transaction A	Transaction B
Read (X)	
X = X+15	
	Read (X)
	x = x-25
	WRITE (X)
WRITE (X)	

		X	
Transaction A	Transaction B	A	B
Read (X)		X = 100	
X = X+15		115	
	Read (X)		X = 100
	x = x-25		X = 75
	WRITE (X)		X = 75
WRITE (X)		X = 115	

The **Lost Update Problem** is called a **Write-Write Conflict** because:

- It happens when **two transactions write to the same data item** without proper synchronization.
- The **first write is overwritten** (or lost) by the second write.
- Both transactions **read the same initial value**, perform updates, and then write back — but the final value **only reflects one of the writes**, hence the term "lost update."

②

Concurrency Issues

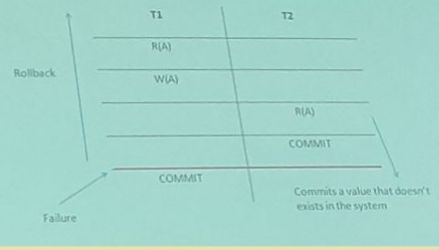
(Reading an uncommitted write)

The Temporary Update (or Dirty Read) Problem. This problem occurs when one transaction updates a database item and then the transaction fails for some reason (see Section 21.1.4). Meanwhile, the updated item is accessed (read) by another transaction before it is changed back to its original value. Figure 21.3(b) shows an example where T_1 updates item X and then fails before completion, so the system must change X back to its original value. Before it can do so, however, transaction T_2 reads the *temporary* value of X, which will not be recorded permanently in the database because of the failure of T_1. The value of item X that is read by T_2 is called *dirty data* because it has been created by a transaction that has not completed and committed yet; hence, this problem is also known as the *dirty read problem*.

[Not every dirty read necessarily leads to a problem.]

Dirty Read (W-R Conflict)

Dirty Read Problem



The **Dirty Read Problem** is called a **Read-Write Conflict** because it involves:

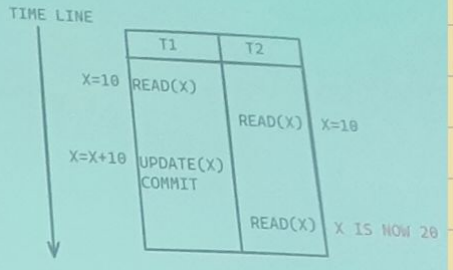
- One transaction **reading** a value that was **written by another uncommitted transaction**.
- If the writing transaction **rolls back**, the reading transaction has read an invalid or "dirty" value.

③

Concurrency Issues

The Unrepeatable Read Problem. Another problem that may occur is called *unrepeatable read*, where a transaction T reads the same item twice and the item is changed by another transaction T′ between the two reads. Hence, T receives *different values* for its two reads of the same item. This may occur, for example, if during an airline reservation transaction, a customer inquires about seat availability on several flights. When the customer decides on a particular flight, the transaction then reads the number of seats on that flight a second time before completing the reservation, and it may end up reading a different value for the item.

Unrepeatable Read (R-W Conflict)



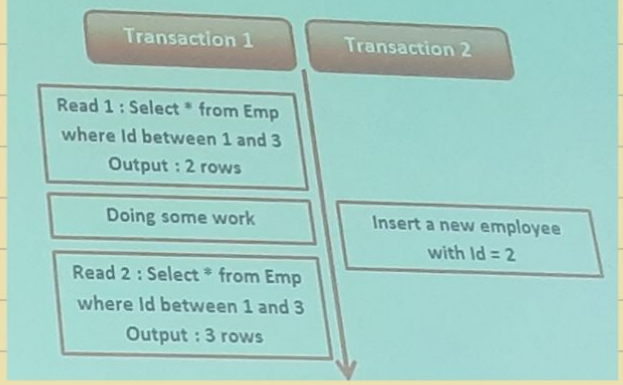
Unrepeatable Read (R-W Conflict)

Unrepeatable Read is called a **Read-Write Conflict** because:

- One transaction **reads** a value.
- Then, **another transaction writes** (updates) that same value and **commits**.
- When the first transaction **reads again**, it gets a **different value**, violating consistency.

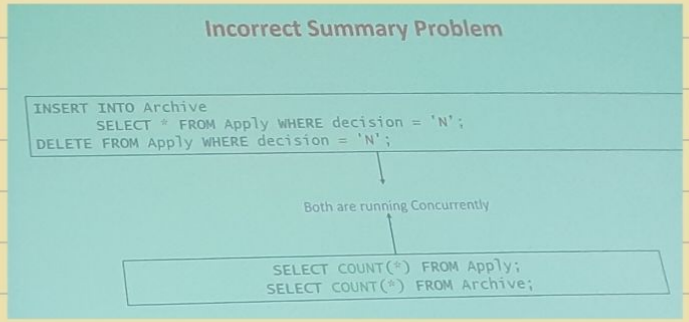
④

Phantom Tuple Problem



⑤

Incorrect Summary Problem



- **Unrepeatable Read**: A transaction reads the same row twice and gets different values because another transaction modified it in between.
- **Phantom Tuple Problem**: A transaction re-executes a query and finds new rows (phantoms) inserted by another transaction that weren't visible earlier.
- **Incorrect Summary**: A transaction computes an aggregate (like `SUM`) over a set of rows while another transaction is modifying those rows, leading to incorrect results.

Differences

- **Unrepeatable Read** happens when **the same row** is read twice and returns different values.
- **Phantom Tuple Problem** involves **new rows** appearing in a repeated query due to concurrent inserts/deletes—not changes to existing rows.
- **Incorrect Summary** happens when an **aggregate** is calculated while other transactions are modifying the data set (including inserts, updates, or deletes), which can affect the correctness of the summary.

Unrepeatable read → **row-level inconsistency.**

Phantom → **new/deleted row appears/disappears.**

Incorrect summary → **aggregate calculation inconsistency.**

8/4

```
sudo apt-get update
sudo apt-get install libmysqlclient-dev
```

```
sudo mysql -u root -p

    CREATE USER 'harith'@'localhost' IDENTIFIED BY 'harith';
    CREATE DATABASE IF NOT EXISTS library;
    GRANT ALL PRIVILEGES ON library.* TO 'harith'@'localhost';
    FLUSH PRIVILEGES;
    QUIT

mysql -u harith -p      // password : harith
```

```
For compiling:
g++ movies.cpp -o movies -lmysqlclient

For run:
./movies
```

```cpp
#include <iostream>
#include <mysql/mysql.h>

using namespace std;

const char* HOST = "localhost";
const char* USER = "harith";
const char* PASSWORD = "harith";
const char* DATABASE = "movies";

MYSQL* conn;

bool connectToDatabase() {
    conn = mysql_init(NULL);
    if (!conn)
        return false;
    conn = mysql_real_connect(conn, HOST, USER, PASSWORD, DATABASE, 0, NULL, 0);

    if (!conn)
        return false;
    return true;
}

void executeQuery(const string& query) {
    if (mysql_query(conn, query.c_str()))
        cout << "Query Error" << endl;
    else
        cout << "Query Success" << endl;
}

void createTables() {
    string createMoviesTable = "CREATE TABLE IF NOT EXISTS movies(movie_id INT AUTO_INCREMENT PRIMARY KEY,title VARCHAR(50),director
VARCHAR(50), year INT);";

    executeQuery(createMoviesTable);
}
```

```cpp
int main() {
    if (!connectToDatabase())
        return 1;

    createTables();
    if (0)
        addMovie();
    else
        displayResults();

    closeConnection();
    return 0;
}
```

```cpp
void addMovie() {

    string title;
    cout << "Enter title : ";
    getline(cin, title);

    string director;
    cout << "Enter director : ";
    getline(cin, director);

    int year;
    cout << "Enter Year of release : ";
    cin >> year;

    string query = "INSERT INTO movies (title, director, year) VALUES ('" + title + "', '" + director + "', " + to_string(year) + ");";

    executeQuery(query);
}
```

```cpp
void displayResults() {
    string query = "SELECT * FROM movies;";
    executeQuery(query);

    MYSQL_RES* result = mysql_store_result(conn);
    MYSQL_ROW row;
    while (row = mysql_fetch_row(result)) {
        cout << "Movie ID: " << row[0] << " | Title: " << row[1] << " | Director: " << row[2] << " | Year: " << row[3] << endl;
    }

    mysql_free_result(result);
}
```

```cpp
void closeConnection() {
    if (conn)
        mysql_close(conn);
}
```

11/4

Transaction - Schedules

A Schedule represents the order in which the operations of transactions are executed.

A schedule is a sequence of operations (like reads, writes, commits, and rollbacks) from different transactions, arranged in the order they occur. It shows how transactions are interleaved during execution

Transaction - Schedules

A Schedule represents the order in which the operations of transactions are executed.

T1: A, B
T2: X, Y

Possible Schedules:

T1	T2
B	
A	
	Y
	Z

Is this even legal ? → No

Transaction - Schedules ⇒ n!

A Schedule represents the order in which the operations of transactions are executed.

T1: A, B
T2: X, Y

Possible Schedules:

S3:

T1	T2
A	
	X
B	
	Y

S4:

T1	T2
A	
	X
	Y
B	

S5:

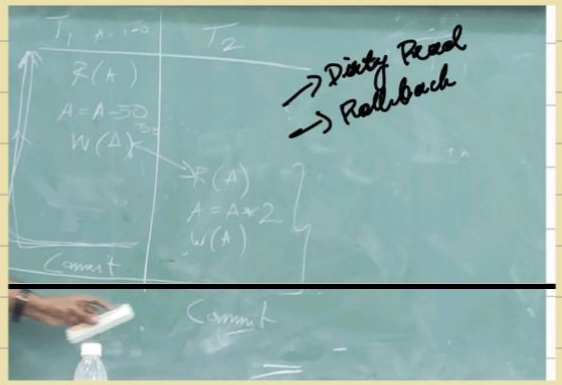
T1	T2
	X
A	
	Y
B	

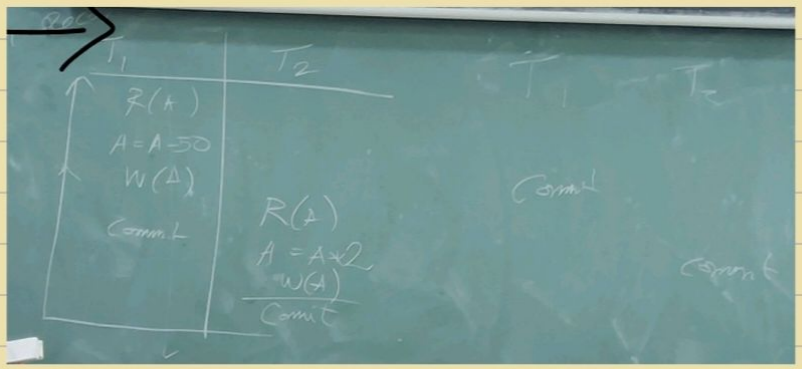
S6:

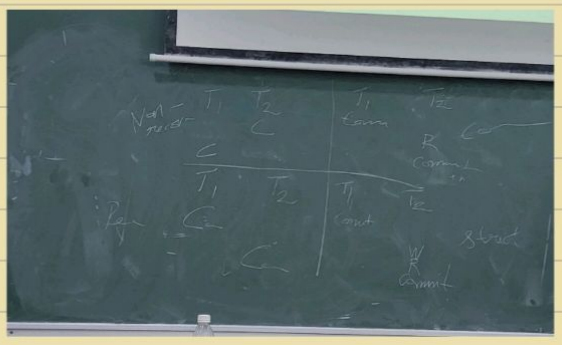
T1	T2
	X
A	
B	
	Y

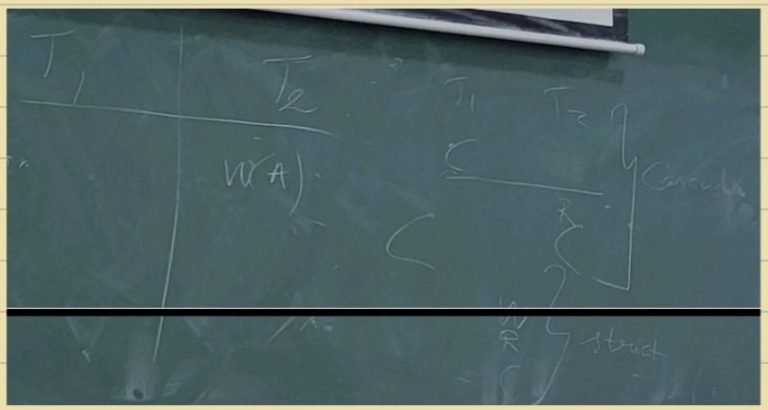
Serial = S1, 2

Non-Serial = S3, 4, 5, 6

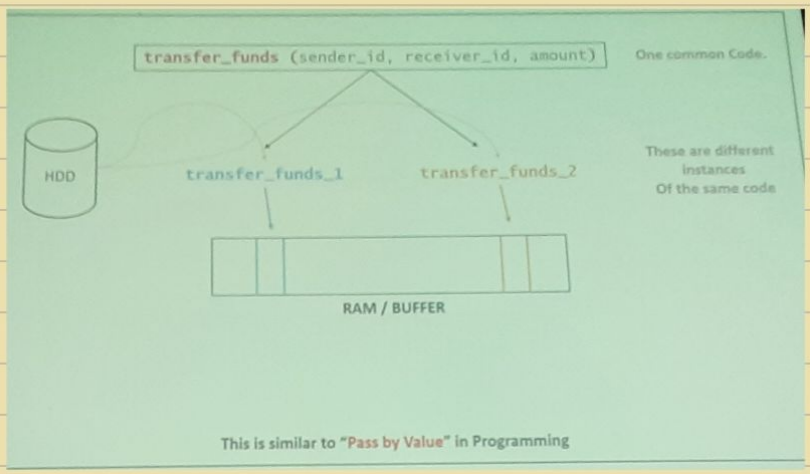








Schedules



This is similar to "Pass by Value" in Programming

Recoverable vs Non –Recoverable: Why are we doing it ?

For some schedules it is easy to recover from transaction and system failures, whereas for other schedules the recovery process can be quite involved. In some cases, it is even not possible to recover correctly after a failure

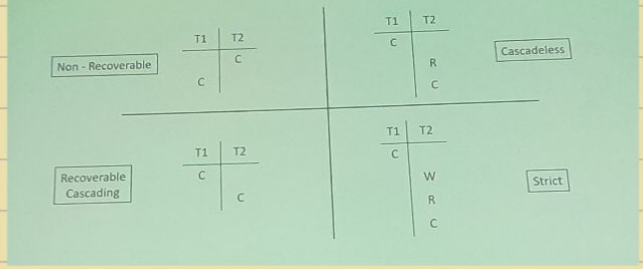
Hence, it is important to characterize the types of schedules for which recovery is possible, as well as those for which recovery is relatively simple

Recoverable vs Non –Recoverable: Why are we doing it ?

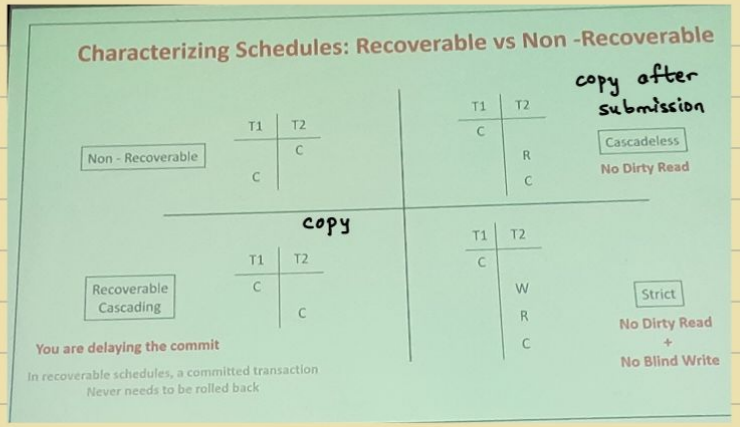
A Schedule should be recoverable to ensure Durability property.

Recoverable means, a committed transaction should never (be forced to) rollback, Whereas, an uncommitted transaction Should not be stopped from rolling back when the need arises.

Characterizing Schedules: Recoverable vs Non -Recoverable

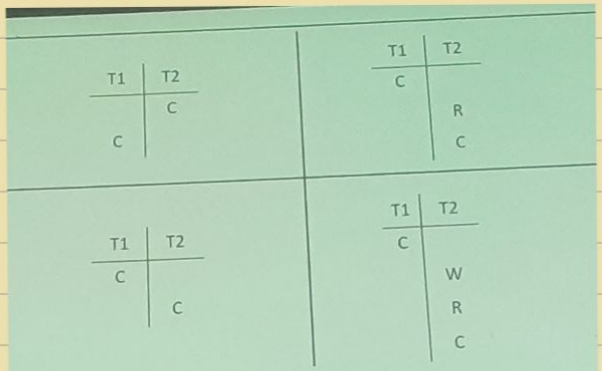


Characterizing Schedules: Recoverable vs Non -Recoverable



You are delaying the commit

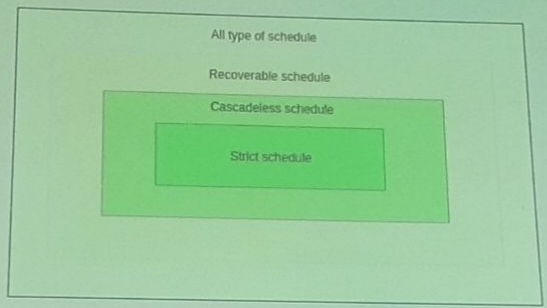
In recoverable schedules, a committed transaction Never needs to be rolled back



Note: The assumption here is – Some item X is first written by T1, Which is later read by T2.

order of commit follows this order

Characterizing Schedules: Recoverable vs Non -Recoverable



Characterizing Schedules: Recoverable vs Non -Recoverable

Non – Recoverable: A schedule where a transaction commits after reading uncommitted data from another transaction that later aborts, leading to inconsistency.

Recoverable: A schedule where a transaction commits only after all transactions whose data it has read also commit—ensuring no committed transaction relies on an aborted one.

Cascadeless: A stricter form of recoverable schedule where transactions only read data from committed transactions, thus avoiding cascading rollbacks.

Strict: A schedule where transactions can neither read nor write a data item until the transaction that last wrote it commits or aborts.

→ Correct Schedules:

Schedules Vs Consistency

S1:

T1	T2
A	
B	
	X
	Y

S2:

T1	T2
	X
	Y
A	
B	

Both of these schedules are considered to be consistent

Note that we are not worried about the actual real world meaning of the ordering, this is because The definition of consistency (in our case is): The system goes from one consistent state to another.

Transaction - Schedules

Serial Schedules are always consistent, but Slow. So we are not interested them

The interesting question is, which of these are consistent ?

S3:

T1	T2
A	
	X
B	
	Y

S4:

T1	T2
A	
	X
	Y
B	

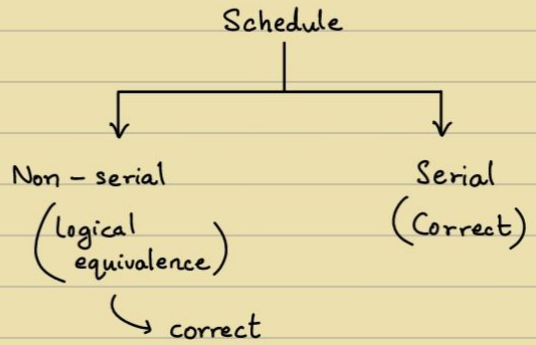
S5:

T1	T2
	X
A	
	Y
B	

S6:

T1	T2
	X
A	
B	
	Y

All serial schedules → consistent



Serial Schedule: A schedule where transactions are executed one after another, with no interleaving of operations.
It ensures correctness but may reduce concurrency.

Non-Serial Schedule: A schedule where operations of transactions are interleaved, allowing them to run concurrently.
It increases concurrency but needs careful control to maintain correctness

Serializability

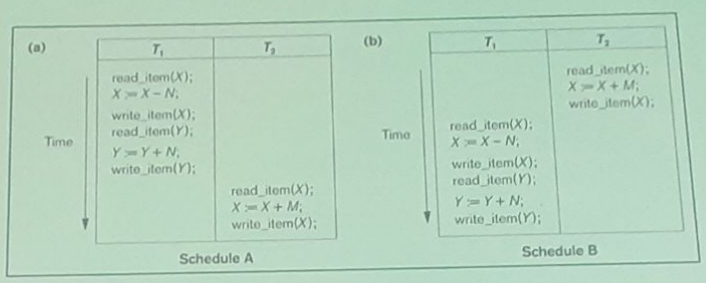
Operations may be interleaved, but execution must be equivalent to some serial/sequential order of all transactions

Interleaving concurrent transaction statements, so that the outcome is equivalent to a serial execution

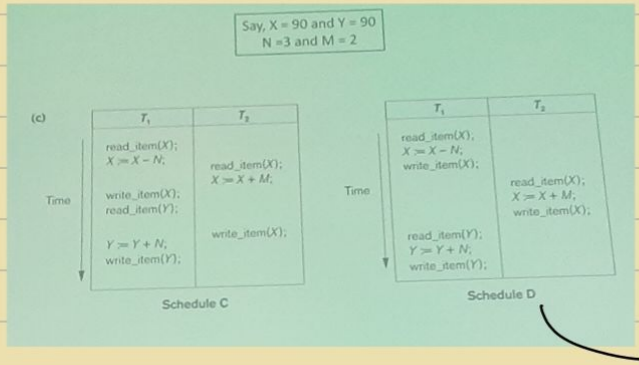
Serializability

Serializability guarantees that a concurrent schedule of transactions produces the same result as some serial schedule,
thereby avoiding problems like lost updates, dirty reads, and inconsistent data.

Say, X = 90 and Y = 90
N =3 and M = 2



X = 89 & Y = 93



→ Result Equivalent to A & B

X = 92 X = 89

Y = 93 Y = 93

(Serializable)

How to Check Equivalency ?

GOAL: To show that a Non-Serial Schedule is equivalent to any one of the Serial Schedule.

But Result Equivalence is not convenient.

We check for "Conflict Equivalence" or "View Equivalence".

16/4

Schedules

Recoverability

Concerned With Durability & Atomicity

Serializability

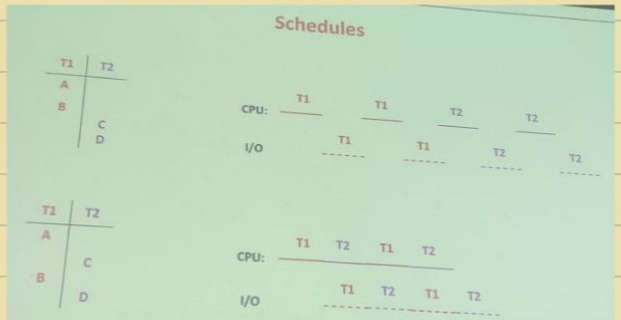
Concerned With Consistency & Isolation

Why Isolation ?

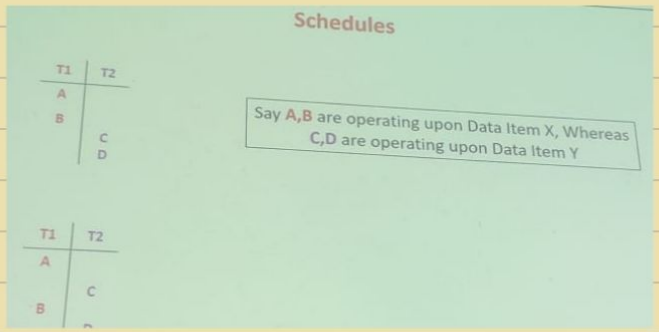
Serializability is all about ensuring that concurrent (non-serial) schedules behave as if they were executed serially.

This matches exactly with Isolation — the idea that transactions appear to run in isolation, even when they're actually interleaved

Schedules



Schedules

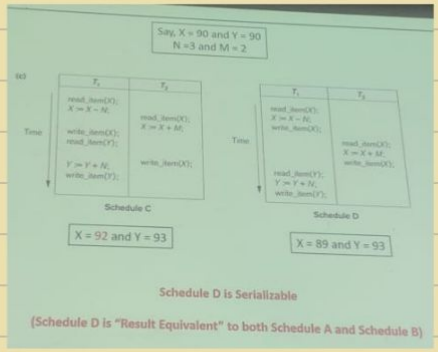


Schedules

T1	T2
A	
B	
	C
	D

Say **A,B** are operating upon Data Item X, Whereas
C,D are operating upon Data Item Y

T1	T2
A	
	C
B	
	D



Schedule D is Serializable

(Schedule D is "Result Equivalent" to both Schedule A and Schedule B)

What is Conflict Equivalence ?

Two Schedules are said to be conflict equivalent

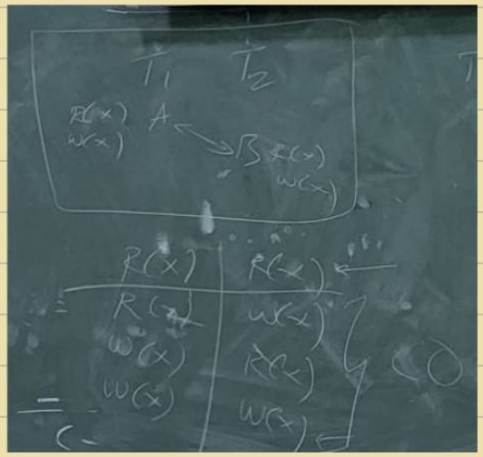
If the Conflicting operations in both the schedules are executed in the same order

Two schedules are conflict equivalent if one can be transformed into the other by swapping non-conflicting operations
(i.e., operations from different transactions on different data items), while keeping the order of conflicting operations
(like Read-Write, Write-Write, or Write-Read on the same item) unchanged

What is Conflict Equivalence ?

What are Conflicting Operations ?

T1	T2	
R(A)	W(A)	Conflicting Operations
W(A)	R(A)	
W(A)	W(A)	
R(A)	R(A)	Non - Conflicting Operations
Operating on Different Data		



How to Check Equivalency ?

GOAL: To show that a Non-Serial Schedule is equivalent to any one of the Serial Schedule.

But Result Equivalence is not convenient.

We check for "Conflict Equivalence" or "View Equivalence".

If a Non-Serial schedule is "Conflict Equivalent" to any one of the serial Schedule, then it is Called as "Conflict Serializable"

Serializability

Serializability guarantees that a concurrent schedule of transactions produces the same result as some serial schedule,
thereby avoiding problems like lost updates, dirty reads, and inconsistent data.

Serializability

Operations may be interleaved, but execution must be equivalent to some serial/sequential order of all transactions

If this property holds, then we call that Non-Serial Schedule as "Serializable"

What if you had a Non-Serial Schedule that is Logically equivalent to a serial Schedule ?

Schedules

T1	T2
A	
	C
B	
	D

Say **A and C** are operating upon Data Item X, Whereas
B and D are operating upon Data Item Y

Interleave them, in such a way that, the dependency order Is preserved in the Non-Serial Schedule

T1	T2
A	
	C
B	
	D

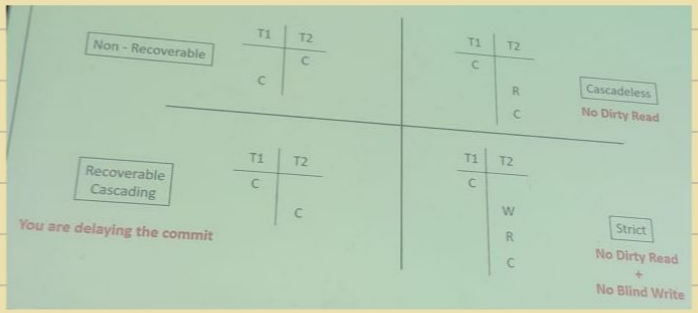
T1	T2
C	
	C

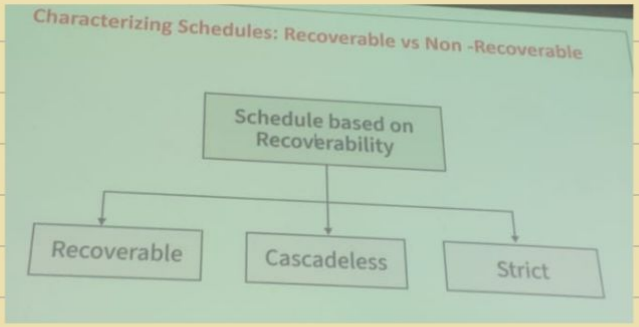
T1	T2
C	
	R
	C

T1	T2
C	
	C

T1	T2
C	
	W
	R
	C

Note: The assumption here is – Some item X is first written by T1, Which is later read by T2.





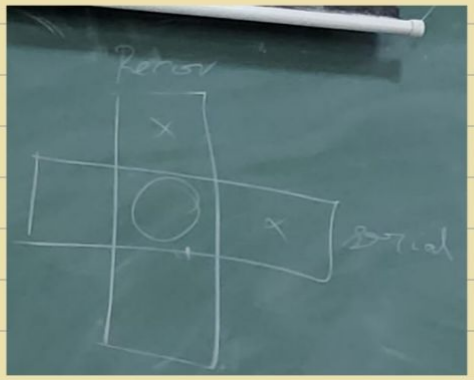
"Correct" Schedules

Any Serial Schedule is correct to us.

We don't care about the order. Even though different order may give different end results.

Rationale behind Ignoring the specific serial order of execution:

1. It makes things simple.
2. 95% of time in the real world order wont matter very much.
3. For well known cases, where order does matter, you can hardcode the system, to handle them explicitly



We are interested in :
A non-Serial schedule that is consistent (logically equivalent to some serial schedule) and Is casacdeless,

17/4

Schedules

Recoverability — Serializability

Both of them are completely separate Concerns.

We don't talk about the order of commit when, classifying schedules based on Serializability
Likewise, we don't talk about conflicting operations during the other case.

Serializability

Equivalence between Any two Arbitrary Schedules

A Non-Serial Schedule is "Conflict" or "View Equivalent" to a serial Schedule

Conflict Serializable — View Serializable

What is Conflict Equivalence ?

What are Conflicting Operations ?

T1	T2	
R(A)	W(A)	Conflicting Operations
W(A)	R(A)	
W(A)	W(A)	
R(A)	R(A)	Non - Conflicting Operations
Operating on Different Data		

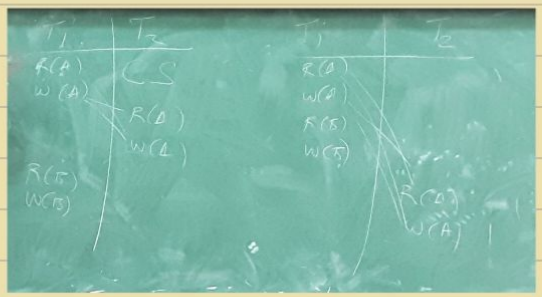
What is Conflict Equivalence ?

Two Schedules are said to be conflict equivalent

If the Conflicting operations in both the schedules are executed in the same order

Conflict Serializable: If a Non –Serial schedule is "Conflict Equivalent" to a serial schedule, Then it is called as Conflict Serializable

Finding if two schedules are CE or not ? - Is it easy or difficult



Finding if two schedules are CE or not ? - Is it easy or difficult

1

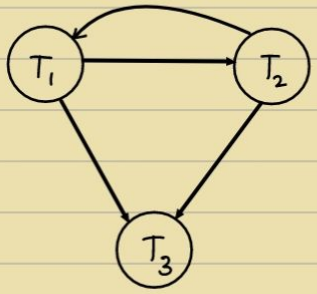
Finding if a Non – Serial Schedule is conflict serializable or not ? – Is it easy or difficult

n!

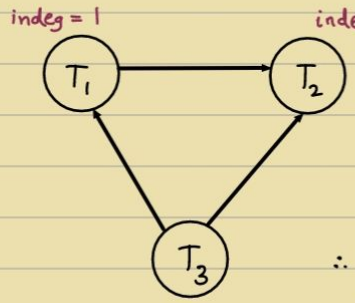
→ The DAG Approach for checking CS

Ex.

T_1	T_2	T_3
R(A)		
	W(A)	
W(A)		
		W(A)



T_1	T_2	T_3
R(X)		
		R(X)
W(X)		
	R(X)	
	W(X)	



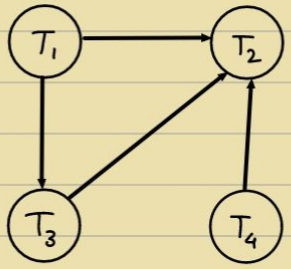
No loop / No cycle

∴ Conflict Serializable

∴ Serializable $T_3 \to T_1 \to T_2$

∴ Consistent

Ex.



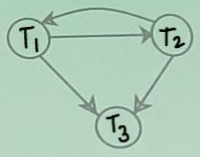
DAG or not ?

→ Do topological Sort

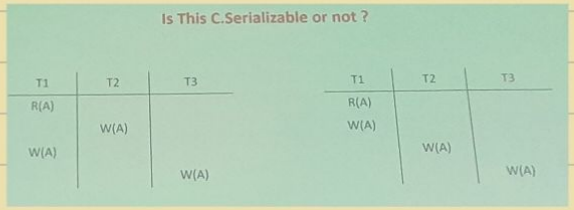
∴ T_4, T_1, T_3, T_2 → Schedule

Is This C.Serializable or not ?

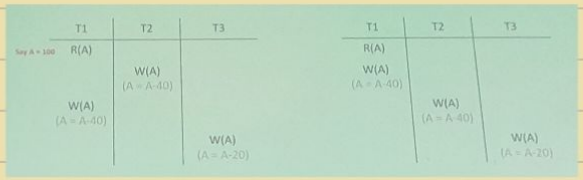
T1	T2	T3
R(A)		
	W(A)	
W(A)		
		W(A)

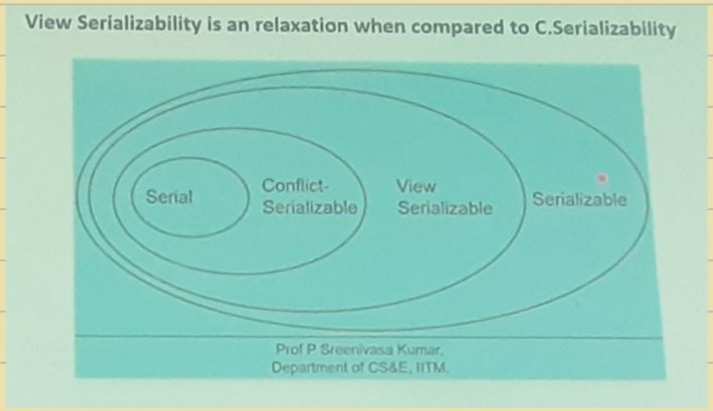


This Schedule cannot be "Conflict Equivalent" to Any Serial Schedule



Not Conflict Equivalent

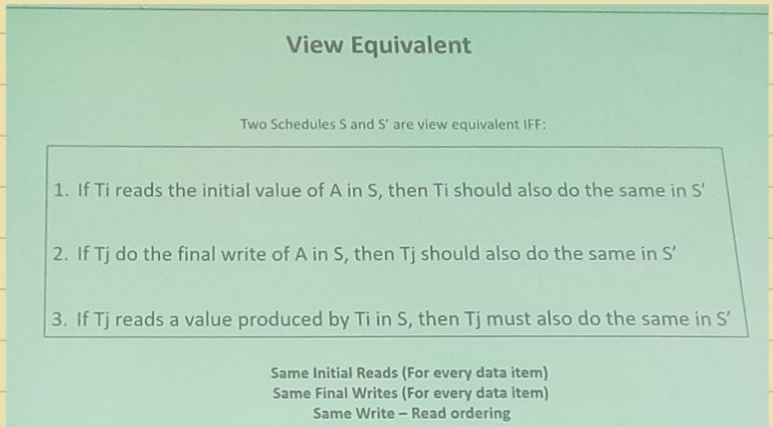


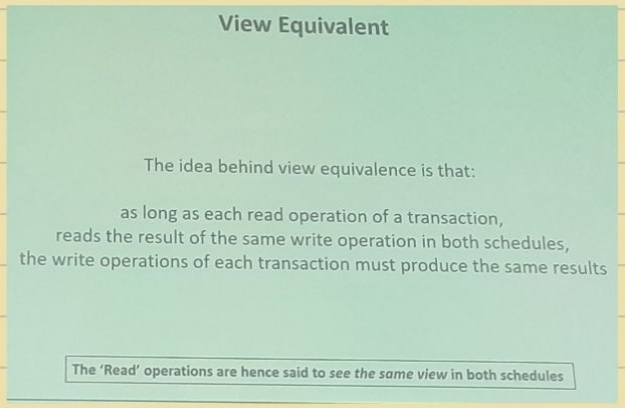


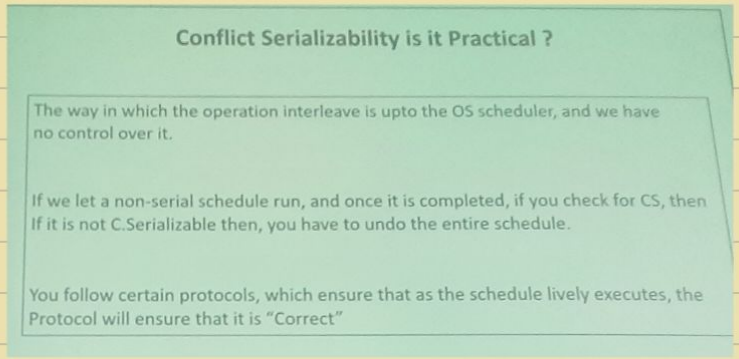
→ Non - serialisable

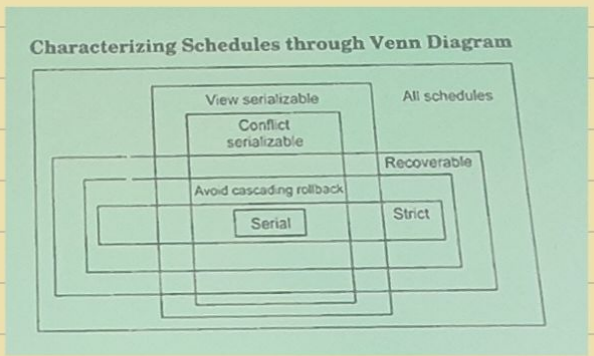
- Conflict Serializable
- Non - Conflict Serializable
  - Blind Write → Check
  - Not Blind Write → Not View Serializable

Shortcut

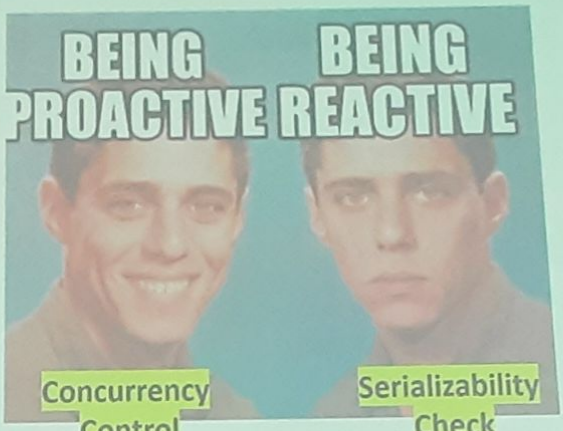








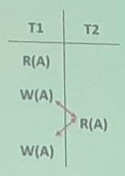
22/4



What Makes a Non-Serial Schedule Non –Serializable ?

What is it, that you should avoid to ensure that, a Non-Serial Schedule Is Serializable ?

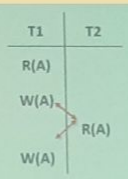
What Makes a Non-Serial Schedule Non –Serializable ?



Cyclic Dependency is the reason

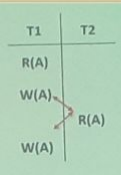
This happens because, two different transactior
Are allowed to operate on the same data item
Without any control

Develop a mechanism, which will pro-actively avoid such dependency



If T1 is accessing Data Item A, then, T2 Should not be allowed to access it – "Mutual Exclusion"

You are effectively "Isolating" the data item



T1 should successfully Lock Data Item A, and unlock it after it has done operating upon it.

(T2 Cannot access A ,when it is being locked by someone else.)

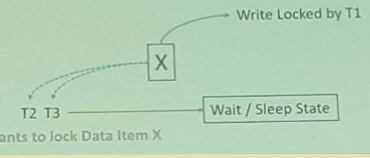
Read_Lock (X) ⟶ Shared Lock

Write_Lock (X) ⟶ Exclusive Lock

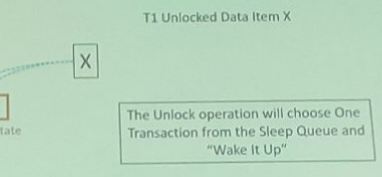
Locking a Data Item

Shared/Exclusive (or Read/Write) Locks. The preceding binary locking scheme is too restrictive for database items because at most, one transaction can hold a lock on a given item. We should allow several transactions to access the same item X if they all access X for *reading purposes only*. This is because read operations on the same item by different transactions are not conflicting (see Section 21.4.1). However, if a transaction is to write an item X, it must have exclusive access to X. For this purpose, a different type of lock called a **multiple-mode lock** is used. In this scheme—called **shared/exclusive** or **read/write** locks—there are three locking operations: read_lock(X), write_lock(X), and unlock(X). A lock associated with an item X, LOCK(X), now has three possible states: *read-locked*, *write-locked*, or *unlocked*. A **read-locked item** is also called **share-locked** because other transactions are allowed to read the item, whereas a **write-locked item** is called **exclusive-locked** because a single transaction exclusively holds the lock on the item.

Locking a Data Item



T2 and T3 wants to lock Data Item X



The Unlock operation will choose One Transaction from the Sleep Queue and "Wake It Up"

Lock request type →

State of the lock	Shared	Exclusive
Shared	Yes	No
Exclusive	No	No

(a)

T_1	T_2
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
unlock(Y);	unlock(X);
write_lock(X);	write_lock(Y);
read_item(X);	read_item(Y);
$X := X + Y$;	$Y := X + Y$;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

(b) Initial values: X=20, Y=30

Result serial schedule T_1 followed by T_2: X=50, Y=80

Result of serial schedule T_2 followed by T_1: X=70, Y=50

(c)

Initial values: X=20, Y=30

T_1	T_2
read_lock(Y);	
read_item(Y);	
unlock(Y);	
	read_lock(X);
	read_item(X);
	unlock(X);
	write_lock(Y);
	read_item(Y);
	$Y := X + Y$;
	write_item(Y);
	unlock(Y);
write_lock(X);	
read_item(X);	
$X := X + Y$;	
write_item(X);	
unlock(X);	

Time

Result of schedule S: X=50, Y=50 (nonserializable)

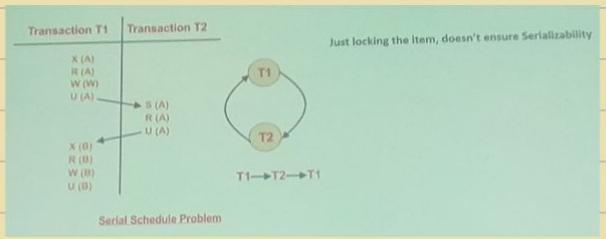
Why This Happens ?
Releasing the lock too early

Remember, to us Serializability = Correctness.

We should atleast ensure Serializability property. Additionally Recoverability, Deadlock free are some Desirable properties.

Time	T1	T2
Time1	Lock-S (A)	
Time2		Lock-S (A)
Time3	Lock-X (B) *	
Time4		
Time5	Unlock (A)	
Time6		Lock-X (C) *
Time7	Unlock (B)	
Time8		Unlock (A)
Time9		Unlock (C)

← Right way



Time	T1	T2
Time1	Lock-S (A)	
Time2		Lock-S (A)
Time3	Lock-X (B) *	
Time4		
Time5	Unlock (A)	
Time6		Lock-X (C) *
Time7	Unlock (B)	
Time8		Unlock (A)
Time9		Unlock (C)

T1 Growing Phase

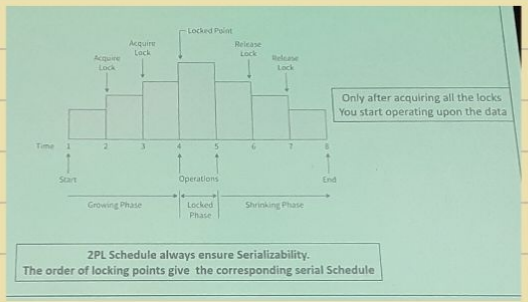
Lock Point Of T1

T1 Shrinking Phase

T2 Growing Phase

Lock Point Of T2

T2 Shrinking Phase

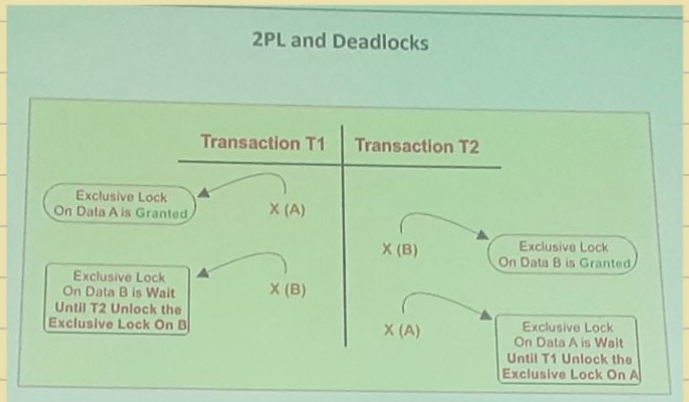


Two phase locking protocol

Transaction T1	Transaction T2
X (A)	
R (A)	
W (A)	
U (A)	
	S (A)
	R (A) // Dirty Read
	U (A)
	Commit
Failure	

Irrecoverable

2-Phase locking protocol



2PL and Cascading Roll Back

Time	T1	T2	T3
Time1	S (A)		
Time2	R (A)		
Time3	W (A)		
Time4	U (A)		
Time5		S (A)	
Time6		R (A)	
Time7			S (A)
Time8			R (A)
	Failure	Rollback	Rollback

2PL and Starvation

Time	T1	T2	T3	T4
Time1		S (A)		
Time2	X (A) Wait			
Time3			S (A)	
Time4		U (A)		
Time5				S (A)
Time6			U (A)	
Time7				U (A)
Time8	X (A) Granted			

2PL Properties

It always Gives Serializable Schedules.

May not be Deadlock Free.
Can be Irrecoverable.
Can be Cascading.
Can be starving

Strict 2PL

A schedule will be in Strict 2PL if

- It must satisfy the basic 2-PL.
- Each transaction should hold all **Exclusive(X) Locks** until the Transaction is Commited or aborted.

Strict 2PL

T1	T2
Lock-X(A)	
READ (A)	
WRITE (A)	Lock-S(A) // Wait
LOCK-X(B)	
READ (B)	
WRITE (B)	
Commit (A)	
Commit (B)	
Unlock (A)	
Unlock (B)	Lock-S(A) // Granted
	READ (A)
	WRITE (A)
	Commit (A)
	Unlock (A)

Delay unlocking after commiting
Deadlock ✗
Recoverability ✓

T1	T2
Lock-S(A)	
READ (A)	
WRITE (A)	Lock-S(A) // Granted
LOCK-X(B)	
READ (B)	
WRITE (B)	
Commit (A)	
Commit (B)	
Unlock (A)	
Unlock (B)	

Strict 2-PL

T1	T2
Lock-S(A)	
READ (A)	
WRITE (A)	Lock-S(A) // Wait
LOCK-X(B)	
READ (B)	
WRITE (B)	
Commit (A)	
Commit (B)	
Unlock (A)	
Unlock (B)	Lock-S(A) // Granted
	READ (A)
	WRITE (A)
	Commit (A)
	Unlock (A)

Rigorous 2-PL

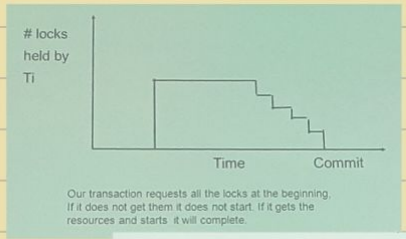
More strict

Conservative 2PL

Before starting the transaction, the transaction (i.e., T1) should hold all the locks on all data items(i.e., A, B, C etc). In this way, the rest of the transactions (T2, T3 ... Tn) will not access the data until T1 commits or aborts.

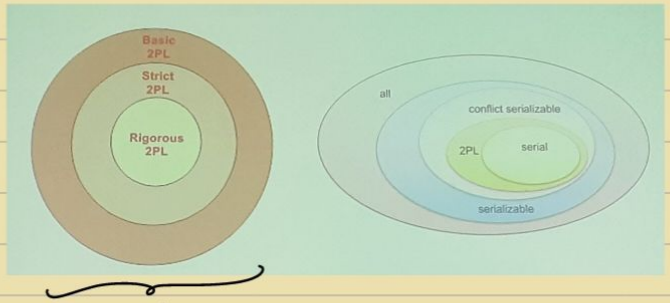
> **Conservative 2PL** prevents deadlocks along with recoverability and cascadless. Because when any transaction holds all the resources, then it will never go to a deadlock state.

Keep in mind: it is difficult to use in practice because of the need to pre-declare the read-set and the write-set, which is not possible in many situations.



Our transaction requests all the locks at the beginning. If it does not get them it does not start. If it gets the resources and starts it will complete.

No Growing Phase. The transaction only starts, when it has collected all the required locks



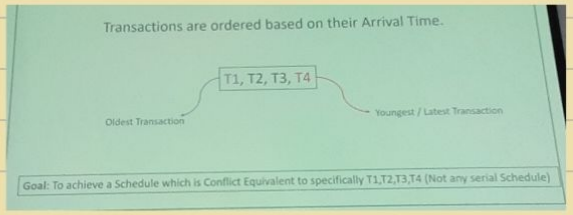
Time Stamp Ordering Protocol

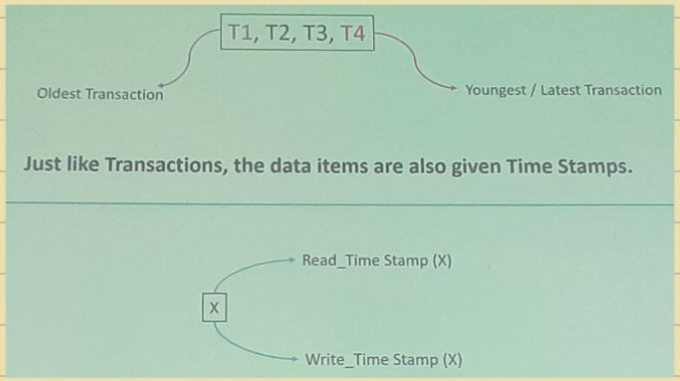
Transactions are ordered based on their Arrival Time.

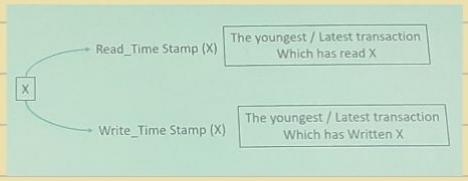
T1, T2, T3, T4

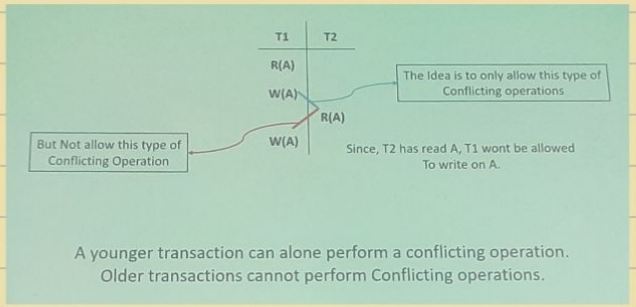
↳ Cannot escape deadlock ⟹ Solution : Don't use locks at all

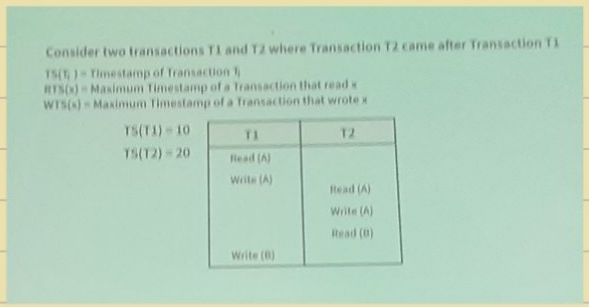
Conservative 2 PL can escape deadlock, but practically it is not possible

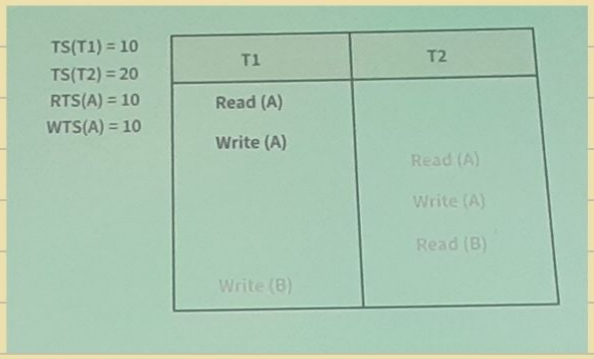


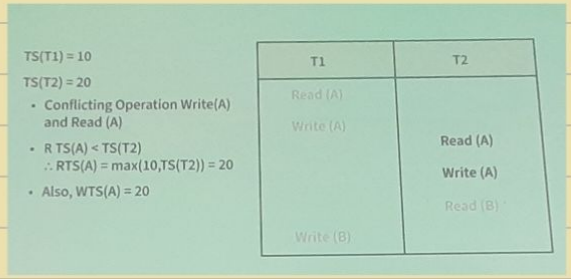












TS(T1) = 10

TS(T2) = 20

RTS(B) = 20

T1	T2
Read (A)	
Write (A)	
	Read (A)
	Write (A)
	Read (B)
Write (B)	

TS(T1) = 10

TS(T2) = 20

- Conflicting Operation Write(B) and Read (B)
- Write(B) operation of Transaction T1 is conflicting with Read (B) operation of transaction T2.
- RTS(B) > TS(T1) ∴ Rollback

T1	T2
Read (A)	
Write (A)	
	Read (A)
	Write (A)
Rollback	Read (B)
Write (B)	

Transaction T issues a read(x) operation

```
if (write-TS(x) > TS(T))
    abort T and Rollback;
else {
    read(x);
    read-TS(x) = TS(T);
}
```

If a younger transaction has already Performed a Write on X, then abort and rollback

Else, you read, and update the Read time stamp

Transaction T issues a read(x) operation

```
if (write-TS(x) > TS(T))
    abort T and Rollback;
else {
    read(x);
    If (read-TS(X) < TS(T))
      read-TS(x) = TS(T);
}
```

Else, you read; and update the read time stamp, IFF you Are the latest transaction

```
Transaction Ti issues a write(x):-

if (read-TS(x) > TS(Ti) OR write-TS(x) > TS(Ti))
    abort T and roll back;
else {
    write(x)
    write-TS(x) = TS(Ti);
}
```

If a younger transaction has already Performed either a Write or read on X, then abort and rollback

Else, if you are the latest transaction, Then write X, and update write time stamp of X