

COURSE FILE

DATABASE MANAGEMENT SYSTEMS

R13 REGULATION

III B Tech – I SEMESTER



COMPUTER SCIENCE ENGINEERING

VIGNAN'S INSTITUTE OF INFORMATION TECHNOLOGY

www.vignaniit.com (Approved by AICTE & Affiliated to JNTUK) Estd. – 2002
NAAC, NBA Accredited & ISO 9001:2008, ISO14001:2004, OHSAS 18001:2007 Certified
Institution
Beside VSEZ, Duvvada, Visakhapatnam-530049. A.P.

Phone : 0891- 2755222 / 333 / 444 :: Fax : 0891 - 2752333 :: E-Mail :
vignaniit@yahoo.com

TABLE OF CONTENT

| S.NO. | Content | Page No. |
|-------|---|----------|
| 1 | Course Objectives & Outcomes | 3 |
| 2 | Syllabus | 4 |
| 3 | Lecture Plan | 5 |
| 4 | Unit wise course Material (As per given format) | |
| | Unit-I: Introduction | 7 |
| | Unit-II: Relational Model & Basic SQL | 48 |
| | Unit-III: Entity Relationship Model & SQL | 116 |
| | Unit-IV: Schema Refinement (Normalization) | |
| | Unit-V: Transaction Management and Concurrency Control | |
| | Unit-VI: Storage and Indexing | |
| 5 | Reference text books/web material etc., | |
| 6 | Mid Question Paper + Schemes of Evaluation. | |
| 7 | Fast track material for Back-Log students. | |
| 8 | Sample Question Papers with solutions | |
| 9 | Virtual Labs if required | |
| 10 | Mapping of Assignments / Question Papers with course objective learning outcomes. | |
| 11 | Bloom's Taxonomy checklist | |

1. Course Objectives & Outcomes

1.1. Course Objectives

The objectives of the course are:

- Provides students with theoretical knowledge and practical skills in the use of databases and database management systems in information technology applications.
- The logic design, physical design and implementation of relational databases are covered.

1.2. Course Outcomes

After completing this Course, the student should be able to:

- Define a Database Management System
- Give a description of the Database Management Structure
- Understand the application of Databases
- Know the advantages and disadvantages of the different models
- Compare relational model with Structured Query Language (SQL)
- Know the constraints and controversies associated with relational database model
- Know the rules guiding the transaction ACID
- Understand the concept of data planning and Database design
- Identify the various functions of Database Administrator

2. Syllabus

Unit – I:

INTRODUCTION

Database system, Characteristics (Database Vs File System), Database Users(Actors on Scene, Workers behind the scene), Advantages of Data base systems, Database applications.

- Brief introduction of different Data Models; Concepts of Schema, Instance and data independence; Three tier schema architecture for data independence; Database system structure, environment, Centralized and Client Server architecture for the database.

Unit – II:

- .. **RELATIONAL MODEL :** Introduction to relational model, concepts of domain, attribute, tuple, relation, importance of null values, constraints (Domain, Key constraints, integrity constraints) and their importance

- **BASIC SQL :** Simple Database schema, data types, table definitions (create, alter), different DML operations (insert, delete, update), basic SQL querying (select and project) using where clause, arithmetic & logical operations, SQL functions(Date and Time, Numeric, String conversion).

Unit – III:

- **Entity Relationship Model:** Introduction, Representation of entities, attributes, entity set, relationship, relationship set, constraints, sub classes, super class, inheritance, specialization, generalization using ER Diagrams.

- **SQL :** Creating tables with relationship, implementation of key and integrity constraints, nested queries, sub queries, grouping, aggregation, ordering, implementation of different types of joins, view(updatable and non-updatable), relational set operations.

Unit – IV:

- .. **SCHEMA REFINEMENT (NORMALIZATION) :** Purpose of Normalization or schema refinement, concept of functional dependency, normal forms based on functional dependency(1NF, 2NF and 3 NF), concept of surrogate key, Boyce-codd normal form(BCNF), Lossless join and dependency preserving decomposition, Fourth normal form(4NF).

Unit – V:

- **TRANSACTION MANAGEMENT AND CONCURRENCY CONTROL :** Transaction, properties of transactions, transaction log, and transaction management with SQL using commit rollback and savepoint.

- Concurrency control for lost updates, uncommitted data, inconsistent retrievals and the Scheduler. Concurrency control with locking methods : lock granularity, lock types, two phase locking for ensuring serializability, deadlocks, Concurrency control with time stamp ordering : Wait/Die and Wound/Wait Schemes, Database Recovery management : Transaction recovery.

- SQL constructs that grant access or revoke access from user or user groups. Basic PL/SQL procedures, functions and triggers.

UNIT – VI:

- .. **STORAGE AND INDEXING :** Database file organization, file organization on disk,

heap files and sorted files, hashing, single and multi-level indexes, dynamic multilevel indexing using B-Tree and B+ tree, index on multiple keys.

3. Lecture Plan

| Lecture no. | Unit Number | Topic |
|-------------|-------------|---|
| Day1 | Unit-1 | Introduction, Database System, Characteristics (Database Vs File System). |
| Day2 | | Database Users(Actors on Scene, Workers behind the Scene) |
| Day3 | | Advantages of Database Systems, Database Applications |
| Day4 | | Brief introduction of different Data Models; Concepts of Schema, |
| Day5 | | Tutorial |
| Day6 | | Instance and data Independence |
| Day7 | | Three tier schema Architecture for data Independence |
| Day8 | | Database System Structure, Environment |
| Day8 | | Centralized and Client Server Architecture for the database |
| Day9 | | Tutorial |
| Day10 | Unit-2 | Relational Model: Introduction to relational model, Concepts of domain |
| Day11 | | Attribute, tuple |
| Day12 | | relation, importance of null values |
| Day13 | | Constraints (Domain, Key constraints, integrity constraints) and their importance |
| Day | | Tutorial |
| Day14 | | Basic SQL: Simple Database Schema, Data types, table Definitions (create, alter) |
| Day15 | | Different DML operations (insert, delete, update), basic SQL querying (select and project) using where clause |
| Day16 | | Arithmetical & logical operations |
| Day17 | | SQL functions (Date and Time, Numeric, String conversion). |
| Day18 | | Tutorial |
| Day19 | Unit-3 | Entity Relationship Model: Introduction, Representation of entities, attributes, entity set |
| Day20 | | relationship, relationship set, constraints, sub classes, |
| Day21 | | super class, inheritance, |
| Day22 | | specialization, generalization using ER Diagrams. |
| Day23 | | Tutorial |
| Day23 | | SQL: Creating tables with relationship |
| Day24 | | implementation of key and integrity constraints, |
| Day25 | | nested queries, sub queries, |
| Day26 | | grouping, aggregation, ordering |
| Day27 | | implementation of different types of joins |
| Day28 | | view(updatable and non-updatable), relational set |

| | | |
|-------|--------|--|
| | | operations |
| Day29 | | Tutorial |
| Day30 | Unit-4 | Schema Refinement (Normalization): Purpose of Normalization or schema refinement, concept of functional dependency, normal forms based on functional dependency(1NF) |
| Day31 | | normal forms based on functional dependency (2NF) |
| Day32 | | Tutorial |
| Day33 | | normal forms based on functional dependency (3NF) |
| Day34 | | concept of surrogate key, |
| Day35 | | Boyce-codd normal form(BCNF) |
| Day36 | Unit-4 | Lossless join and dependency preserving decomposition |
| Day37 | | Fourth normal form(4NF) |
| Day38 | | Tutorial |
| Day39 | | Transaction, properties of transactions |
| Day40 | | transaction log |
| Day41 | Unit-5 | transaction management with SQL using commit rollback and savepoint. |
| Day42 | | Concurrency control for lost updates |
| Day43 | | Tutorial |
| Day44 | | uncommitted data, inconsistent retrievals and the Scheduler |
| Day44 | | Concurrency control with locking methods |
| Day45 | | lock granularity, lock types |
| Day46 | | two phase locking for ensuring serializability |
| Day47 | | Tutorial |
| Day48 | | two phase locking for ensuring serializability |
| Day49 | | Deadlocks |
| Day50 | | Concurrency control with time stamp ordering : Wait/Die and Wound/Wait Schemes |
| Day51 | | Database Recovery management : Transaction recovery. |
| Day52 | | SQL constructs that grant access |
| Day53 | | revoke access from user or user groups |
| Day54 | Unit-6 | Basic PL/SQL procedures, functions and triggers |
| Day55 | | Tutorial |
| Day56 | | Storage and Indexing: Database file organization, |
| Day57 | | file organization on disk |
| Day58 | | heap files and sorted files |
| Day59 | | Hashing |
| Day60 | | Tutorial |
| Day60 | | single and multi-level indexes |
| Day61 | | dynamic multilevel indexing using B-Tree |
| Day62 | | dynamic multilevel indexing using B+ tree |
| Day63 | | index on multiple keys |

Day64

Tutorial

4. UNIT-WISE COURSE MATERIAL

Database System, Characteristics (Database Vs File system), Database Users (Actors on Scene, Workers behind the scene), Advantages of Database Systems, Database Applications.

Brief introduction of different data models; Concepts of Schema, Instance and data independence; Database System structure, Environment, Centralized and client server

4.1. Unit - I - Introduction

4.1.1. Unit Objectives:

After reading this Unit, you should be able to understand:

- To study the physical and logical database designs, database modeling, relational, hierarchical and network models
- To develop an understanding of essential DBMS concepts such as: database security, integrity, concurrency, distributed database, and intelligent database, Client/Server (Database Server), Data Warehousing

4.1.2. Unit Outcomes:

- Differentiate database systems from file systems by enumerating the features provided by database systems and describe each in both function and benefit
- Analyze an information storage problem and derive an information model expressed in the form of an entity relation diagram and other optional analysis forms, such as a data dictionary.
- Demonstrate an understanding of the relational data model

4.1.3. Unit Lecture Plan

| Lecture No. | Topic | Methodology | Quick Reference |
|-------------|---|---------------|----------------------|
| 1 | Introduction, Database System, Characteristics (Database Vs File System). | Chalk & Board | Text. Book with Page |
| 2 | Database Users (Actors on Scene, Workers behind the Scene) | Chalk & Board | Text. Book with Page |
| 3 | Advantages of Database Systems, Database Applications | Chalk & Board | Text. Book with Page |
| 4 | Brief introduction of different Data Models; Concepts of Schema | PPT | Text. Book with Page |
| 5 | Instance and data Independence | Chalk & Board | Text. Book with Page |
| 6 | Three tier schema Architecture for data Independence | Chalk & Board | Text. Book with Page |
| 7 | Database System Structure, Environment | Chalk & Board | Text. Book with Page |
| 8 | Centralized and Client Server Architecture for the database | Chalk & Board | Text. Book with Page |

4.1.4. Teaching Material / Teaching Aids as per above lecture plan.

4.1.4.1. Lecture-1

Introduction to Database:

Database is a collection of related data. Database management system is software designed to assist the maintenance and utilization of large scale collection of data. DBMS came into existence in 1960 by Charles. Integrated data store which is also called as the first general purpose DBMS. Again in 1960 IBM brought IMS-Information management system. In 1970 Edgor Codd at IBM came

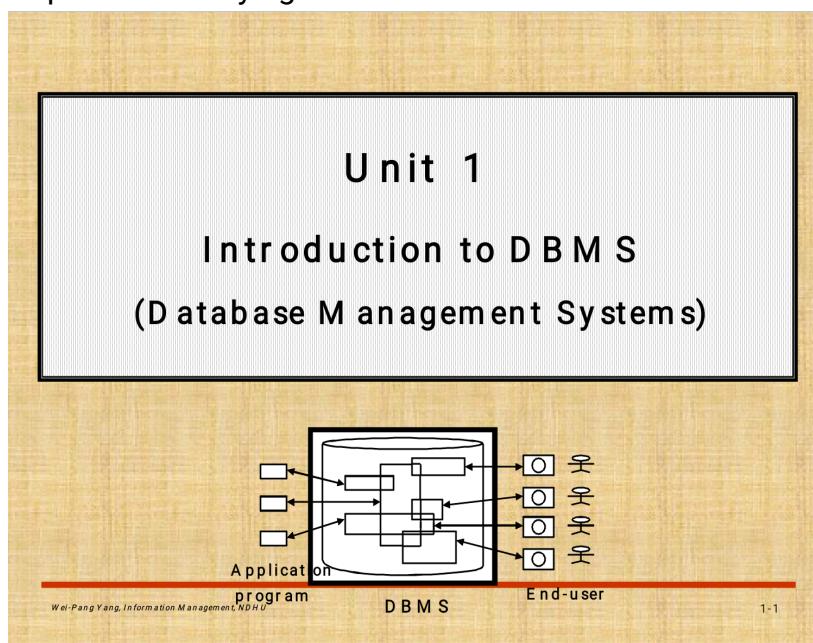
with new database called RDBMS. In 1980 then came SQL Architecture- Structure Query Language. In 1980 to 1990 there were advances in DBMS e.g. DB2, ORACLE.

Data:

- Data is raw fact or figures or entity.
- When activities in the organization takes place, the effect of these activities need to be recorded which is known as Data.

Information:

- Processed data is called information
- The purpose of data processing is to generate the information required for carrying out the business activities.



In general data management consists of following tasks

- Data capture: Which is the task associated with gathering the data as and when they originate.
- Data classification: Captured data has to be classified based on the nature and intended usage.
- Data storage: The segregated data has to be stored properly.
- Data arranging: It is very important to arrange the data properly
- Data retrieval: Data will be required frequently for further processing, hence it is very important to create some indexes so that data can be retrieved easily.

- Data maintenance: Maintenance is the task concerned with keeping the data up-to-date.
- Data Verification: Before storing the data it must be verified for any error.
- Data Coding: Data will be coded for easy reference.
- Data Editing: Editing means re-arranging the data or modifying the data for presentation.
- Data transcription: This is the activity where the data is converted from one form into another.
- Data transmission: This is a function where data is forwarded to the place where it would be used further.

Metadata (meta data, or sometimes meta information) is "data about data", of any sort in any media. An item of metadata may describe a collection of data including multiple content items and hierarchical levels, for example a database schema. In data processing, metadata is definitional data that provides information about or documentation of other data managed within an application or environment. The term should be used with caution as all data is about something, and is therefore metadata.

Database

- Database may be defined in simple terms as a collection of data
- A database is a collection of related data.
- The database can be of any size and of varying complexity.
- A database may be generated and maintained manually or it may be computerized

Database Management System

- A Database Management System (DBMS) is a collection of program that enables user to create and maintain a database.
- The DBMS is hence a general purpose software system that facilitates the process of defining constructing and manipulating database for various applications.

Characteristics of DBMS

- To incorporate the requirements of the organization, system should be designed for easy maintenance.
- Information systems should allow interactive access to data to obtain new information without writing fresh programs.
- System should be designed to co-relate different data to meet new

requirements.

- An independent central repository, which gives information and meaning of available data is required.
- Integrated database will help in understanding the inter-relationships between data stored in different applications.
- The stored data should be made available for access by different users simultaneously.
- Automatic recovery feature has to be provided to overcome the problems with processing system failure.

DBMS Utilities

- A data loading utility: Which allows easy loading of data from the external format without writing programs.
- A backup utility: Which allows to make copies of the database periodically to help in cases of crashes and disasters.
- Recovery utility: Which allows to reconstruct the correct state of database from the backup and history of transactions.
- Monitoring tools: Which monitors the performance so that internal schema can be changed and database access can be optimized.
- File organization: Which allows restructuring the data from one type to another?

Difference between File system & DBMS

File System:

1. File system is a collection of data. Any management with the file system, user has to write the procedures
2. File system gives the details of the data representation and Storage of data.
3. In File system storing and retrieving of data cannot be done efficiently.
4. Concurrent access to the data in the file system has many problems like
 - Reading the file while other deleting some information, updating some information
5. File system doesn't provide crash recovery mechanism.
Eg. While we are entering some data into the file if System crashes then content of the file is lost.
6. Protecting a file under file system is very difficult.

DBMS:

1. DBMS is a collection of data and user is not required to write the procedures for managing the database.
2. DBMS provides an abstract view of data that hides the details.
3. DBMS is efficient to use since there are wide varieties of sophisticated techniques to store and retrieve the data.
4. DBMS takes care of Concurrent access using some form of locking.
5. DBMS has crash recovery mechanism, DBMS protects user from the effects of system failures.
6. DBMS has a good protection mechanism.

4.1.4.2. Lecture -2

Database Users:

- ❖ There are four different types of database-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users.
- ❖ **Naive users** are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.
 - For example, a bank teller who needs to transfer \$50 from account A to account B invokes a program called transfer. This program asks the teller for the amount of money to be transferred, the account from which the money is to be transferred, and the account to which the money is to be transferred.
 - As another example, consider a user who wishes to find her account balance over the World Wide Web. Such a user may access a form, where she enters her account number. An application program at the Web server then retrieves the account balance, using the given account number, and passes this information back to the user. The typical user interface for naive users is a forms interface, where the user can fill in appropriate fields of the form.

Naive users may also simply read reports generated from the database.

- ❖ **Application programmers** are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. **Rapid application development (RAD)** tools are tools that enable an application programmer to construct forms and reports without writing a program. There are also special types of programming languages that combine imperative control structures (for example, for loops, while loops and if-then-else statements) with statements of the data manipulation language. These languages, sometimes called fourth-generation languages, often include special features to facilitate the generation of forms and the display of data on the screen. Most major commercial database systems include a fourth generation language.
- ❖ **Sophisticated users** interact with the system without writing programs. Instead, they form their requests in a database query language. They submit each such query to a **query processor**, whose function is to break down DML statements into instructions that the storage manager understands. Analysts who submit queries to explore data in the database fall in this category.
- ❖ **Online analytical processing (OLAP)** tools simplify analysts' tasks by letting them view summaries of data in different ways. For instance, an analyst can see total sales by region (for example, North, South, East, and West), or by product, or by a combination of region and product (that is, total sales of each product in each region). The tools also permit the analyst to select specific regions, look at data in more detail (for example, sales by city within a region) or look at the data in less detail (for example, aggregate products together by category).

Another class of tools for analysts is **data mining** tools, which help

them find certain kinds of patterns in data.

- ❖ **Specialized users** are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework.

Among these applications are computer-aided design systems, knowledge base and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems.

4.1.4.3 Lecture -3

Advantages of DBMS.

Due to its centralized nature, the database system can overcome the disadvantages of the file system-based system

1. Data independency:

- Application program should not be exposed to details of data representation and storage
- DBMS provides the abstract view that hides these details.

2. Efficient data access:

- DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently.

3. Data integrity and security:

- Data is accessed through DBMS, it can enforce integrity constraints.
E.g.: Inserting salary information for an employee.

4. Data Administration:

- When users share data, centralizing the data is an important task, Experience professionals can minimize data redundancy and perform fine tuning which reduces retrieval time.

5. Concurrent access and Crash recovery:

- DBMS schedules concurrent access to the data. DBMS protects user from the effects of system failure.

6. Reduced application development time.

- DBMS supports important functions that are common to many applications.

4.1.4.4 Lecture -4

Database Applications & Introduction of different Data Model

Traditional Applications:

- ✓ Numeric and Textual Databases

More Recent Applications:

- ✓ Multimedia Databases
- ✓ Geographic Information Systems (GIS)
- ✓ Data Warehouses
- ✓ Real-time and Active Databases

Model:

A model is an abstraction process that hides superfluous details. Data modeling is used for representing entities of interest and their relationship in the database. Data model and different types of Data Model Data model is a collection of concepts that can be used to describe the structure of a database which provides the necessary means to achieve the abstraction.

The structure of a database means that holds the data.

- Data types
- Relationships
- Constraints

Types of Data Models

1. High Level- Conceptual data model.
 2. Low Level – Physical data model.
 3. Relational or Representational
 4. Object-oriented Data Models:
 5. Object-Relational Models:
-
1. High Level-conceptual data model: User level data model is the high level or conceptual model. This provides concepts that are close to the way that many users perceive data.
 2. Low level-Physical data model : provides concepts that describe the details of how data is stored in the computer model. Low level data model is only for Computer specialists not for end-user.
 3. Representation data model: It is between High level & Low level data model Which provides concepts that may be understood by end-user but that are not too far removed from the way data is organized by within the computer.

PPT ON INTRODUCTION TO DATA MODELS

Data Models

The most common data models are

1. Relational Model

The Relational Model uses a collection of tables both data and the relationship among those data. Each table have multiple column and each column has a unique name .

Relational database comprising of two tables
Customer –Table.

| Customer-Name | Security Number | Address | City | Account-Number |
|---------------|-----------------|--------------|-----------|----------------|
| Preethi | 111-222-3456 | Yelhanka | Bangalore | A-101 |
| Sharan | 111-222-3457 | Hebbal | Bangalore | A-125 |
| Preethi | 112-123-9878 | Jaynagar | Bangalore | A-456 |
| Arun | 123-987-9909 | MG road | Bangalore | A-987 |
| Preethi | 111-222-3456 | Yelhanka | Bangalore | A-111 |
| Rocky | 222-232-0987 | Sanjay Nagar | Bangalore | A-111 |

Account –Table

| Account-Number | Balance |
|----------------|---------|
| A-101 | 1000.00 |
| A-125 | 1200.00 |
| A-456 | 5000.00 |
| A-987 | 1234.00 |
| A-111 | 3000.00 |

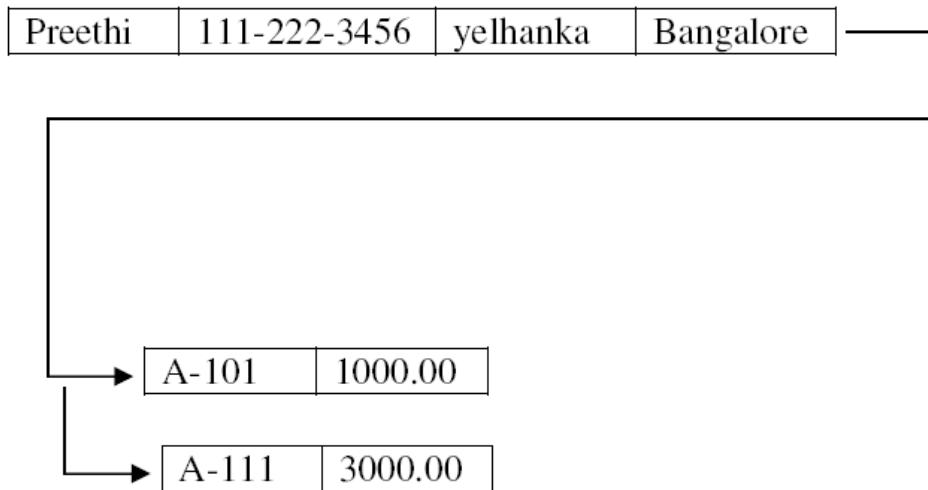
Customer Preethi and Rocky share the same account number A-111

Advantages:

1. The main advantage of this model is its ability to represent data in a simplified format.
2. The process of manipulating record is simplified with the use of certain key attributes used to retrieve data.
3. Representation of different types of relationship is possible with this model.

2. Network Model

The data in the network model are represented by collection of records and relationships among data are represented by links, which can be viewed as pointers.

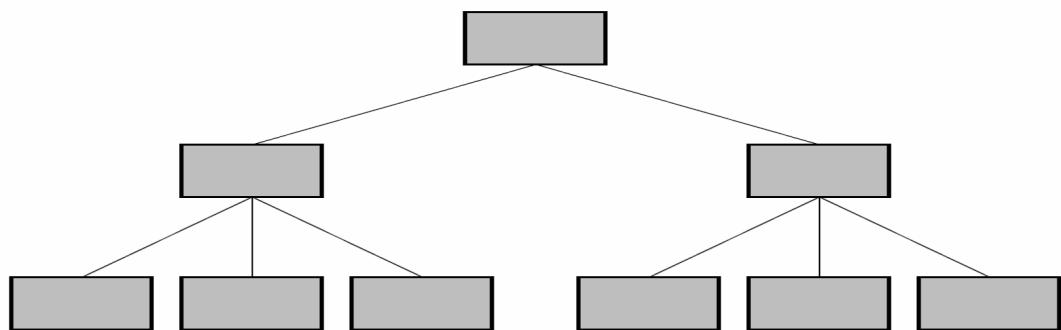


The records in the database are organized as collection of arbitrary groups.

Advantages:

1. Representation of relationship between entities is implemented using pointers which allows the representation of arbitrary relationship
 2. Unlike the hierarchical model it is easy.
 3. Data manipulation can be done easily with this model.
- 3. Hierarchical Model**

A hierarchical data model is a data model which the data is organized into a tree like structure. The structure allows repeating information using parent/child relationships: each parent can have many children but each child only has one parent. All attributes of a specific record are listed under an entity type.



Advantages:

1. The representation of records is done using an ordered tree, which is

natural method of implementation of one-to-many relationships.

2. Proper ordering of the tree results in easier and faster retrieval of records.
3. Allows the use of virtual records. This result in a stable database especially when modification of the data base is made.

4. Object-oriented Data Models

- Several models have been proposed for implementing in a database system.
- One set comprises models of persistent O-O Programming Languages such as C++ (e.g., in OBJECTSTORE or VERSANT), and Smalltalk (e.g., in GEMSTONE).
 - Additionally, systems like O2, ORION (at MCC – then ITASCA), IRIS (at H.P.- used in Open OODB).

5. Object-Relational Models

- Most Recent Trend. Started with Informix
- Universal Server.
- Relational systems incorporate concepts from object databases leading to object relational.
- Object Database Standard: ODMG-93, ODMG-version 2.0, ODMG-version 3.0.
- Exemplified in the latest versions of Oracle-10i, DB2, and SQL Server and other DBMSs.
- Standards included in SQL-99 and expected to be enhanced in future SQL standards.

4.1.4.5 Lecture -5

Concepts of Schema, Instance and Data Independence:

Schemas versus Instances

Database Schema:

The description of a database. Includes descriptions of the database structure, data types, and the constraints on the database.

• Schema Diagram:

An *illustrative display of (most aspects of) a* database schema.

- **Schema Construct:**

A *component of the schema or an object within* the schema, e.g., STUDENT, COURSE.

- **Database State:**

The actual data stored in a database at a particular moment in time. This includes the collection of all the data in the database. Also called database instance (or occurrence or snapshot)

- The term instance is also applied to individual database components, e.g. record instance, table instance, entity instance

Database Schema vs. Database State

- **Database State:** Refers to the *content of a database at a moment* in time.

- **Initial Database State:** Refers to the database state when it is initially loaded into the system.

- **Valid State:** A state that satisfies the structure and constraints of the database.

- **Distinction:** The *database schema changes very infrequently*.

The *database state changes every time the* database is updated

- Schema is also called intension
- State is also called extension

Example of a Database Schema

STUDENT

| Name | Student_number | Class | Major |
|------|----------------|-------|-------|
|------|----------------|-------|-------|

COURSE

| Course_name | Course_number | Credit_hours | Department |
|-------------|---------------|--------------|------------|
|-------------|---------------|--------------|------------|

PREREQUISITE

| Course_number | Prerequisite_number |
|---------------|---------------------|
|---------------|---------------------|

SECTION

| Section_identifier | Course_number | Semester | Year | Instructor |
|--------------------|---------------|----------|------|------------|
|--------------------|---------------|----------|------|------------|

Example of a database state

COURSE

| Course_name | Course_number | Credit_hours | Department |
|---------------------------|---------------|--------------|------------|
| Intro to Computer Science | CS1310 | 4 | CS |
| Data Structures | CS3320 | 4 | CS |
| Discrete Mathematics | MATH2410 | 3 | MATH |
| Database | CS3380 | 3 | CS |

SECTION

| Section_identifier | Course_number | Semester | Year | Instructor |
|--------------------|---------------|----------|------|------------|
| 85 | MATH2410 | Fall | 04 | King |
| 92 | CS1310 | Fall | 04 | Anderson |
| 102 | CS3320 | Spring | 05 | Knuth |
| 112 | MATH2410 | Fall | 05 | Chang |
| 119 | CS1310 | Fall | 05 | Anderson |
| 135 | CS3380 | Fall | 05 | Stone |

GRADE_REPORT

| Student_number | Section_identifier | Grade |
|----------------|--------------------|-------|
| 17 | 112 | B |
| 17 | 119 | C |
| 8 | 85 | A |
| 8 | 92 | A |
| 8 | 102 | B |
| 8 | 135 | A |

PREREQUISITE

| Course_number | Prerequisite_number |
|---------------|---------------------|
| CS3380 | CS3320 |
| CS3380 | MATH2410 |
| CS3320 | CS1310 |

Data Independence

Data independence can be defined as the capacity to change the schema at one

level without changing the schema at next higher level. There are two types of data Independence. They are

1. Logical data independence.
2. Physical data independence.

1. Logical data independence is the capacity to change the conceptual schema without having to change the external schema.
2. Physical data independence is the capacity to change the internal schema without changing the conceptual schema.

4.1.4.6 Lecture -6

Three tier schema Architecture for data Independence

Database management systems are complex software's which were often developed and optimized over years. From the view of the user, however, most of them have a quite similar basic architecture. The discussion of this basic architecture shall help to understand the connection with data modeling and the introduction to this module postulated 'data independence' of the database approach.

Three-Scheme Architecture

Knowing about the conceptual and the derived logical scheme (discussed in unit [Database Models, Schemes and Instances](#) this unit explains two additional schemes - the external scheme and the internal scheme - which help to understand the DBMS architecture

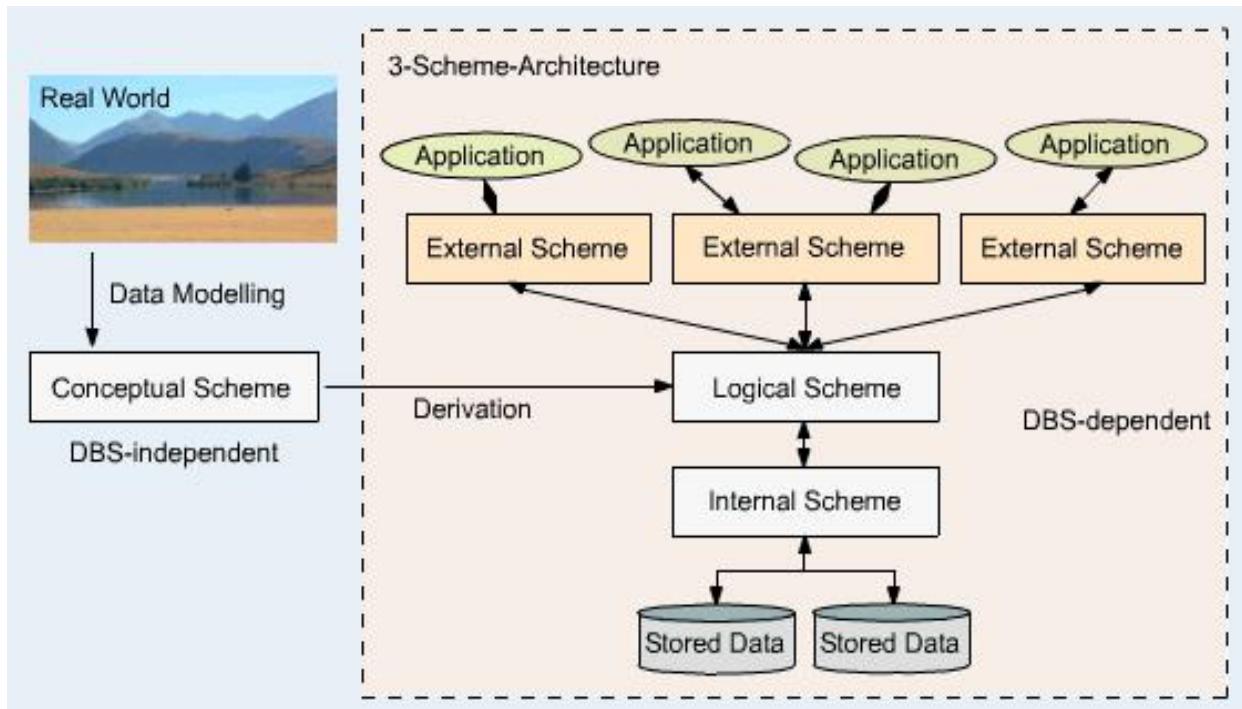
External Scheme:

An external data scheme describes the information about the user view of specific users (single users and user groups) and the specific methods and constraints connected with this information. (Translated) (ZEHNDER 1998)

Internal Scheme:

The internal data scheme describes the content of the data and the required service functionality which is used for the operation of the DBMS. (Translated) (ZEHNDER 1998)

Therefore, the internal scheme describes the data from a view very close to the computer or system in general. It completes the logical scheme with data technical aspects like storage methods or help functions for more efficiency.



The right hand side of the representation above is also called the three-schemes architecture: internal, logical and external scheme.

While the internal scheme describes the physical grouping of the data and the use of the storage space, the logical scheme (derived from the conceptual scheme) describes the

basic construction of the data structure. The external scheme of a specific application, generally, only highlights that part of the logical scheme which is relevant for its application. Therefore, a database has exactly one internal and one logical scheme but may have several external schemes for several applications using this database.

The aim of the three-schemes architecture is the separation of the user applications from the physical database, the stored data. Physically the data is only existent on the internal level while other forms of representation are calculated or derived respectively if needed. The DBMS has the task to realize this representation between each of these levels

Data Independence

With knowledge about the three-schemes architecture the term data independence can be explained as followed: Each higher level of the data architecture is immune to changes of the next lower level of the architecture.

Physical Independence:

Therefore, the logical scheme may stay unchanged even though the storage space or type of some data is changed for reasons of optimization or reorganization.

Logical Independence:

Also the external scheme may stay unchanged for most changes of the logical scheme. This is especially desirable as in this case the application software does not need to be modified or newly translated.

4.1.4.6 Lecture -7

Database System Structure, Environment

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the storage manager and the query processor components.

The storage manager is important because databases typically require a large amount of storage space. Corporate databases range in size from hundreds of gigabytes to, for the largest databases, terabytes of data. A gigabyte is 1000 megabytes (1 billion bytes), and a terabyte is 1 million megabytes (1 trillion bytes). Since the main memory of computers cannot store this much information, the information is stored on disks. Data are moved between disk storage and main memory as needed. Since the movement of data to and from disk is slow relative to the speed of the central processing unit, it is imperative that the database system structure the data so as to minimize the need to move data between disk and main memory.

The query processor is important because it helps the database system simplify and facilitate access to data. High-level views help to achieve this goal; with them, users of the system are not burdened unnecessarily with the physical details of the implementation of the

system. However, quick processing of updates and queries is important. It is the job of the database system to translate updates and queries written in a nonprocedural language, at the logical level, into an efficient sequence of operations at the physical level.

Storage Manager

A *storage manager* is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the interaction with the file manager. The raw data are stored on the disk using the file system, which is usually provided by a conventional operating system. The storage manager translates the various DML statements into low-level file-system commands. Thus, the storage manager is responsible for storing, retrieving, and updating data in the database.

The storage manager components include:

Authorization and integrity manager, which tests for the satisfaction of integrity constraints and checks the authority of users to access data.

Transaction manager, which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.

File manager, which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.

Buffer manager, which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory. The buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory.

The storage manager implements several data structures as part of the physical system implementation:

Data files, which store the database itself.

Data dictionary, which stores metadata about the structure of the database, in particular the schema of the database.

Indices, which provide fast access to data items that hold

particular values.

The Query Processor

The query processor components include

DDL interpreter, which interprets DDL statements and records the definitions in the data dictionary.

DML compiler, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.

A query can usually be translated into any of a number of alternative evaluation plans that all give the same result. The **DML compiler** also performs **query optimization**, that is, it picks the lowest cost evaluation plan from among the alternatives.

Query evaluation engine, which executes low-level instructions generated by the **DML compiler**.

Figure shows these components and the connections among them.

Application Architectures

Most users of a database system today are not present at the site of the database system, but connect to it through a network. We can therefore differentiate between **client** machines, on which remote database users work, and **server** machines, on which the database system runs.

Database applications are usually partitioned into two or three parts, as in Figure. In **two-tier architecture**, the application is partitioned into a component that resides at the client machine, which invokes database system functionality at the server machine through query language statements. Application program interface standards like ODBC and JDBC are used for interaction between the client and the server.

In contrast, in a **three-tier architecture**, the client machine acts as merely a front end and does not contain any direct database calls. Instead, the client end communicates with an **application server**, usually through a forms interface. The application server in turn communicates with a database system to access data. The **business logic** of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead of being distributed across multiple clients. Three-tier applications are more appropriate for large applications, and for applications that run on the World Wide Web.

Data processing drives the growth of computers, as it has from the

earliest days of commercial computers. In fact, automation of data processing tasks predates computers. Punched cards, invented by Hollerith, were used at the very beginning of the twentieth century to record U.S. census data, and mechanical systems were used to process the cards and tabulate results. Punched cards were later widely used as a means of entering data into computers.

4.1.4.6 Lecture -8

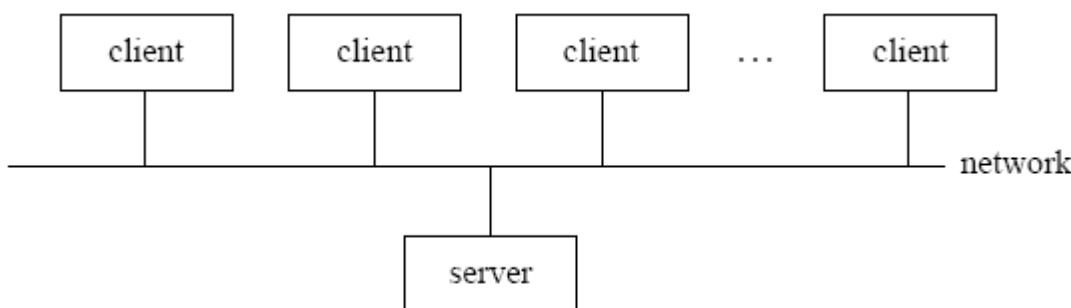
Centralized and Client and Server architecture for the database

Centralized Systems

- Run on a single computer system and do not interact with other computer systems.
- General-purpose computer system: one to a few CPUs and a number of device controllers that are connected through a common bus that provides access to shared memory.
- Single-user system (e.g., personal computer or workstation): desk-top unit, single user, usually has only one CPU and one or two hard disks; the OS may support only one user.
- Multi-user system: more disks, more memory, multiple CPUs, and a multi-user OS. Serve a large number of users who are connected to the system via terminals. Often called server systems.

Client-Server Systems

Server systems satisfy requests generated at client systems, whose general structure is shown below:



- Database functionality can be divided into:
 1. Back-end: manages access structures, query evaluation and

- optimization, concurrency control and recovery.
2. Front-end: consists of tools such as forms, report-writers, and graphical user interface facilities.
 The interface between the front-end and the back-end is through SQL or through an application program interface.
- Advantages of replacing mainframes with networks of workstations or personal computers connected to back-end server machines:
 1. Better functionality for the cost
 2. Flexibility in locating resources and expanding facilities
 3. Better user interfaces\
 4. Easier maintenance
 - Server systems can be broadly categorized into two kinds:
 1. **Transaction servers** which are widely used in relational database systems, and
 2. **Data servers**, used in object-oriented database systems

Transaction Servers

- Also called query server systems or SQL server systems; clients send requests to the server system where the transactions are executed, and results are shipped back to the client.
- Requests specified in SQL, and communicated to the server through a remote procedure call (RPC) mechanism.
 Transactional RPC allows many RPC calls to collectively form a transaction.
- Open Database Connectivity (ODBC) is an application program interface standard from Microsoft for connecting to a server, sending SQL requests, and receiving results.

Data Servers

- Used in LANs, where there is a very high speed connection between the clients and the server, the client machines are comparable in processing power to the server machine, and the tasks to be executed are compute intensive.
- Ship data to client machines where processing is performed, and then ship results back to the server machine.
- This architecture requires full back-end functionality at the clients.
- Used in many object-oriented database systems

4.1.5. Test Questions

a) Fill in the blanks type of questions :

1. _____ program module, which is responsible for fetching data from disk storage into main memory and deciding what data to be cache in memory.
2. One of the responsibilities of a _____ is to create database schema.
3. Data is raw whereas information is _____.
4. Two important languages in the database system are (a) _____ and (b) _____.
5. To access information from a database, one needs a _____.
6. SQL stands for_____.
7. CODASYL stands for_____.
8. _____language enables users to access or manipulate data.
9. _____ is a language for specifying the database schema and as well as other properties of the data.
10. The __ subsystem provides the interface between the low level data stored in the database and the application programs and queries submitted to the system.
11. The __ subsystem compiles and executes DDL and DML statements.
12. _____management ensures that the database remains in a consistent state despite of system failures.
13. Under the function _____ the DBA creates the original database schema by executing a set of data definition statements in the DDL.
14. Granting of authorization for data access is done by _____ user.
15. _____users are unsophisticated users.
16. _____files stores the database itself.

b) Multiple choice questions

1. The view of total database content is

- (A) Conceptual view. (B) Internal view. (C) External view. (D) Physical View.**

Ans: A

2. In the relational modes, cardinality is termed as:

- (A) Number of tuples. (B) Number of attributes. (C) Number of tables. (D) Number of constraints.**

Ans: A

3. The view of total database content is

- (A) Conceptual view. (B) Internal view. (C) External view. (D) Physical View.**

Ans: A

4. Architecture of the database can be viewed as

- (A) Two levels. (B) Four levels. (C) Three levels. (D) One level.**

Ans: C

5. In the architecture of a database system external level is the

- A) Physical level. (B) Logical level. (C) Conceptual level (D) views level.**

Ans: D

6. In a Hierarchical model records are organized as

- (A) Graph. (B) List. (C) Links. (D) Tree.**

Ans: D

7. A logical schema

- (A) is the entire database.
(B) is a standard way of organizing information into accessible parts.
(C) describes how data is actually stored on disk.
(D) both (A) and (C)**

Ans: A

8. The database environment has all of the following components except:
(A) users. (B) separate files.(C) database. (D) database administrator.

Ans: A

9. The property / properties of a database is / are :
**(A) It is an integrated collection of logically related records.
(B) It consolidates separate files into a common pool of data records.
(C) Data stored in a database is independent of the application programs using it.
(D) All of the above.**

Ans: D

10. A relational database developer refers to a record as

- (A) a criteria. (B) a relation. (C) a tuple. (D) an attribute.**

Ans: C

(c) True or False questions

1. Relations are related to each other by sharing a common entity characteristic.(T)
2. A relational DBMS is a single data repository in which data independence is maintained.(T)
3. An entity is a person, place, or thing about which data are to be collected and stored.(T)
4. A model is an exact representation of a real-world event.(F)
5. A data model is built out of tables, rows, and columns.(F)
6. Business rules are policies written in the employee handbook.(F)
7. Business rules help you determine the relationships that exist between entities.(T)
8. Many-to-many relationships are easily represented using a hierarchical

model.(F)

9. One of the advantages of the hierarchical model is the ease of maintaining data integrity.(T)

10. Hierarchical databases became popular in the 1980s.(F)

11. The only relationship type supported by the hierarchical model is 1:1.(F)

12. The hierarchical model was the first to define a standard DML and DDL.(F)

13. The most important advantage of an RDBMS is the ability to let the designer operate in a

human logical environment.(T)

14. Only 1:M and M:N relationships can be represented in a relational schema.(F)

15. Ad hoc query capability was introduced in the relational model.(T)

4.1.6. *Review Questions*

d) Objective type of questions (Very short notes)

1. What do you mean by data inconsistency?

Ans: Data inconsistency exists when different and conflicting versions of the same data appear in different places. Data inconsistency creates unreliable information, because it will be difficult to determine which version of the information is correct.

Data inconsistency is likely to occur when there is data redundancy.

Data redundancy occurs when the data file/database file contains redundant unnecessarily duplicated data. That's why one major goal of good database design is to eliminate data redundancy.

2. What is the importance of atomicity?

Ans: Atomicity is a feature of databases systems dictating where a transaction must be all-or-nothing. That is, the transaction must either fully happen, or not happen at all. It must not complete partially.

3. What does durability mean?

Ans: In database systems, **durability** is the ACID property which guarantees that transactions that have committed will survive

permanently. For example, if a flight booking reports that a seat has successfully been booked, then the seat will remain booked even if the system crashes.

4. What are the disadvantages in File Processing System?

Ans: 1) Data redundancy and inconsistency.

- 2) Difficult in accessing data.
- 3) Data isolation.
- 4) Data integrity.
- 5) Concurrent access is not possible.
- 6) Security Problems.

5. What is physical, logical and view level of data abstraction?

Ans: We have three levels of abstraction:

Physical level: This is the lowest level of data abstraction. It describes how data is actually stored in database. You can get the complex data structure details at this level.

Logical level: This is the middle level of 3-level data abstraction architecture. It describes what data is stored in database.

View level: Highest level of data abstraction. This level describes the user interaction with database system.

6. Why does database system offer different level of abstraction?

Ans: At **physical level** these records can be described as blocks of storage (bytes, gigabytes, terabytes etc.) in memory. These details are often hidden from the programmers.

At the **logical level** these records can be described as fields and attributes along with their data types, their relationship among each other can be logically implemented. The programmers generally work at this level because they are aware of such things about database systems.

At **view level**, user just interact with system with the help of GUI and enter the details at the screen, they are not aware of how the data is stored and what data is stored; such details are hidden from them.

7. What is a data model?

Ans: Data model is Relatively simple representation, usually graphical, of complex real world data Structures and Communications tool to facilitate interaction among the designer, the applications programmer,

and the end user.

8. What is the importance of data model?

Ans: End users have different views and needs for data and Data model organizes data for various users

9. What is Data Independence?

Ans: Data independence means that "the application is independent of the storage structure and access strategy of data". In other words, The ability to modify the schema definition in one level should not affect the schema definition in the next higher level.

Two types of Data Independence:

1. **Physical Data Independence:** Modification in physical level should not affect the logical level.
2. **Logical Data Independence:** Modification in logical level should affect the view level.

10. State the role of buffer manager.

Ans: Buffer manager intelligently shuffles data from main memory to disk: It is transparent to higher levels of DBMS operation

11. Explain five duties of Database Administrator.

Ans:

- ✓ Deciding the information content of the database
- ✓ Deciding the storage structure and access strategy
- ✓ Liaising with the users
- ✓ Defining authorization checks and validation procedures
- ✓ Defining a strategy for backup and recovery
- ✓ Monitoring performance and responsibilities to changes in requirements

12. What is the purpose of Transaction Manager?

Ans: A user's program may carry out many operations on the data retrieved from the database, but the DBMS is only concerned about what data is read/written from/to the database. Transaction Manager controls the execution of transactions.

13. State the function of file manager.

Ans: Atomicity of updates

✓Failures may leave database in an inconsistent state with partial updates carried out

- ✓E.g. transfer of funds from one account to another should either complete or not happen at all Concurrent access by multiple users
- ✓Concurrent accessed needed for performance
- ✓Uncontrolled concurrent accesses can lead to inconsistencies
 - E.g. two people reading a balance and updating it at the same time
 - Security problems Database systems offer solutions to all the above problems

14. Define the terms 1) physical schema 2) logical schema.

Ans: Physical schema: The physical schema describes the database design at the physical level, which is the lowest level of abstraction describing how the data are actually stored. Logical schema: The logical schema describes the database design at the logical level, which describes what data are stored in the database and what relationship exists among the data.

e) Essay type Questions <As per requirements>

1. How does conventional file processing system works?

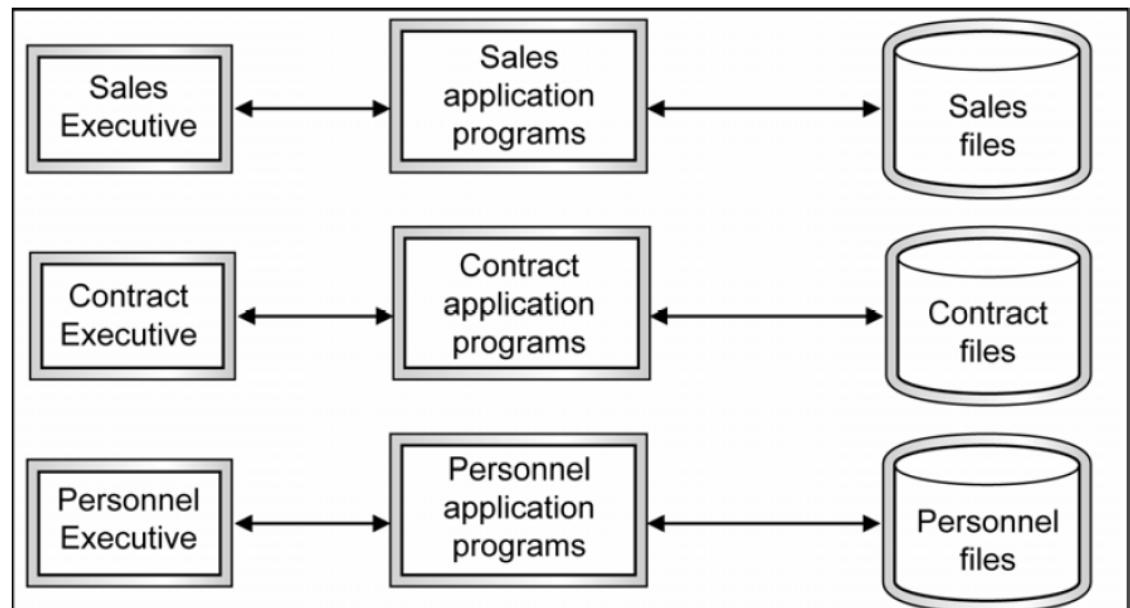
Ans: A file-based system is a collection of application programs that perform services for the end-users such as students reports for the academic office and lecturers report for the dean's office. Each program defines and manages its own data.

Traditionally, manual files are being used to store all internal and external data within an organization. These files are being stored in cabinets and for security purposes, the cabinets are locked or located in a secure area. When any information is needed, you may have to search starting from the first page until you found the information that you are looking for. To speed up the searching process, you may create an indexing system to help you locate the information that you are looking for quickly. You may have such system that store all your results or important documents.

The manual filing system works well if the number of items stored is not large. However, this kind of system may fail if you want to do a cross-reference or process any of the information in the file. Then, computer based data processing emerge and it replaces the traditional filing system with computer-based data processing system or file-based system. However, instead of having a centralized store for the organizations operational access, a decentralized approach was taken. In this approach, each department would have their own file-based system where they would monitor and control separately.

Lets refer to the following example.

File processing system at Make-Believe real estate company Make-Believe real estate company has three departments, that are, Sales, Contract and Personnel. Each of these departments were physically located in the same building, but in separate floors, and each has its own file-based system. The function of the Sales department is to sell and rent properties. The function of the Contract department is handle the lease agreement associated with properties for rent. The function of the Personnel department is to store the information about the staff. Figure illustrates the file-based system for Make-Believe real estate company. Each department has its own application program that handles similar operations like data entry, file maintenance and generation of reports.



File-based system for Make-Believe real estate company

2. What are the anomalies related to traditional file processing systems?

Ans: the anomalies related to traditional file processing systems:

Data Redundancy: Data Redundancy means same information is duplicated in several files. This makes data redundancy.

Data Inconsistency: Data Inconsistency means different copies of the same

data are not matching. That means different versions of same basic data are existing. This occurs as the result of update operations that are not updating the same data stored at different places.

Example: Address Information of a customer is recorded differently in different files.

Difficulty in Accessing Data: It is not easy to retrieve information using a conventional file processing system. Convenient and efficient information retrieval is almost impossible using conventional file processing system.

Data Isolation: Data are scattered in various files, and the files may be in different format, writing new application program to retrieve data is difficult.

Integrity Problems: The data values may need to satisfy some integrity constraints. For example the balance field Value must be grater than 5000. We have to handle this through program code in file processing systems. But in database we can declare the integrity constraints along with definition itself.

Atomicity Problem: It is difficult to ensure atomicity in file processing system. For example transferring \$100 from Account A to account B. If a failure occurs during execution there could be situation like \$100 is deducted from Account A and not credited in Account B.

Concurrent Access anomalies: If multiple users are updating the same data simultaneously it will result in inconsistent data state. In file processing system it is very difficult to handle this using program code. This results in concurrent access anomalies.

Security Problems: Enforcing Security Constraints in file processing system is very difficult as the application programs are added to the system in an ad-hoc manner.

3. What are the disadvantages of File Processing System? What are the advantages of DBMS?

Ans: The database approach offers a number of potential advantages compared to traditional file processing systems.

1. Program-Data Independence: The separation of data descriptions from the application programs that use the data is called Data independence. With the database approach. Data descriptions are stored in a central location called the repository. This property of database systems allows an organization's data to change without changing the application programs that process the data.

2. Data Redundancy and Inconsistency: In File-processing System, files having different formats and application programs may created by different programmers. Similarly different programs may be written in

several programming languages. The same information placed at different files which cause redundancy and inconsistency consequently higher storage and access cost. For Example, the address and telephone number of a person may exist in two files containing saving account records and checking account records. Now a change in person's address may reflect the saving account records but not anywhere in the whole system. This results the data inconsistency. One solution to avoid this data redundancy is keeping the multiple copies of same information, replace it by a system where the address and telephone number stored at just one place physically while it is accessible to all applications from this itself. DBMS can handle Data Redundancy and Inconsistency.

3. Difficulty in accessing Data: - In Classical file organization the data is stored in the files. Whenever data has to be retrieved as per the requirements then a new application program has to be written. This is tedious process.

4. Data isolation: - Since data is scattered in various files, and files may be in different formats, it is difficult to write new application programs to retrieve the appropriate data.

5. Concurrent access: - There is no central control of data in classical file organization. So, the concurrent access of data by many users is difficult to implement.

6. Security Problems: - Since there is no centralized control of data in classical file organization. So, security, enforcement is difficult in File-processing system.

7. Integrity Problem: - The data values stored in the database must satisfy certain types of consistency constraints. For example, the balance of a bank account may never fall below a prescribed amount. These constraints are enforced in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.

8. Improved Data Sharing: - A database is designed as a shared corporate resource. Authorized internal and external users are granted permission to use the database, and each user is provided one or more user views to facilitate this use. A user view is a logical description of some portion of the database that is required by a user to perform

some task.

9. Increased Productivity of Application Development: - A major advantage of the database approach is that it greatly reduces the cost and time of developing new business applications.

There are two important reasons that data base applications can often be developed much more rapidly than conventional file applications.

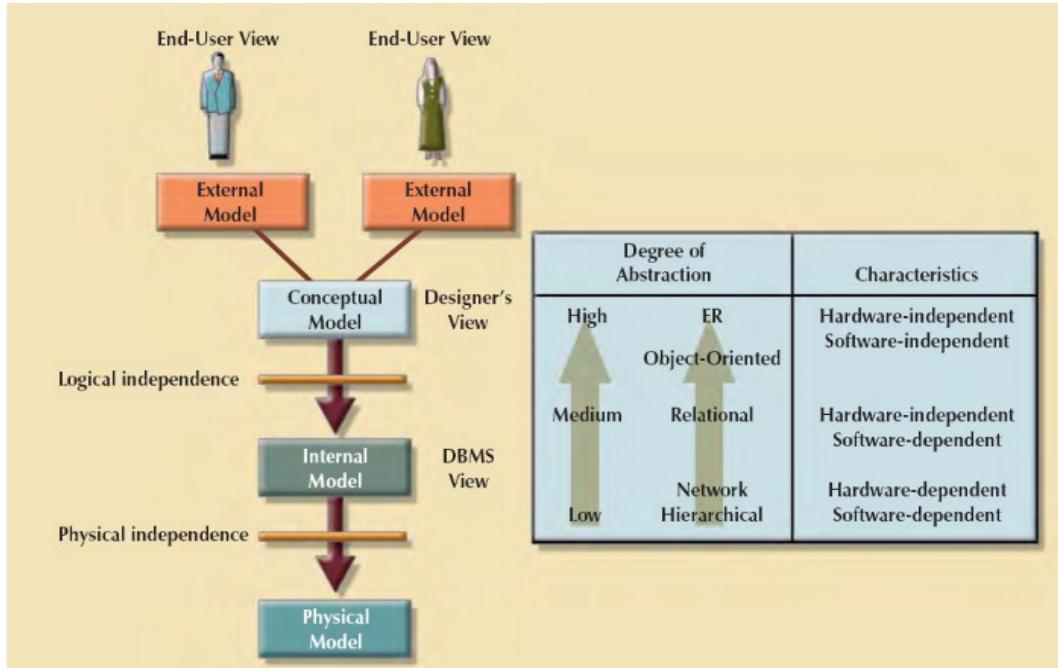
- a) Assuming that the database and the related data capture and maintenance applications have already been designed and implemented, the programmer can concentrate on the specific functions required for the new application, without having to worry about file design or low-level implementation details.
- b) The data base management system provided a number of high-level productivity tools such as forms and reports generators and high-level languages that automate some of the activities of database design and implementation.

Disadvantages:-

1. It occupies more amount of space. It is generic
2. More time they access data.
3. More complex and expensive hardware and software resources are needed.
4. Sophisticated security measures must be implemented to prevent unauthorized access of sensitive data in online storage.

4. Explain all different levels of data abstraction?

Ans: The database designer starts with an abstract view of the overall data environment and adds details as the design comes closer to implementation. Using levels of abstraction can also be very helpful in integrating multiple (and sometimes conflicting) views of data as seen at different levels of an organization. The ANSI/SPARC architecture (as it is often referred to) defines three levels of data abstraction: external, conceptual, and internal. You can use this framework to better understand database models, as shown in Figure



The External Model:

The external model is the end users' view of the data environment. The term end users refers to people who use the application programs to manipulate the data and generate information. End users usually operate in an environment in which an application has a specific business unit focus. Companies are generally divided into several business units, such as sales, finance, and marketing. Each business unit is subject to specific constraints and requirements, and each one uses a data subset of the overall data in the organization. Therefore, end users working within those business units view their data subsets as separate from or external to other units within the organization.

The use of external views representing subsets of the database has some important advantages:

It makes it easy to identify specific data required to support each business unit's operations.

It makes the designer's job easy by providing feedback about the model's adequacy. Specifically, the model can be checked to ensure that it supports all processes as defined by their external models, as well as all operational requirements and constraints.

It helps to ensure security constraints in the database design. Damaging an entire database is more difficult when each business unit works with only a subset of data.

It makes application program development much simpler.

The Conceptual Model:

Having identified the external views, a conceptual model is used, graphically represented by an ERD to integrate all external views into a single view. The conceptual model represents a global view of the entire database as viewed by the entire organization. That is, the conceptual model integrates all external views (entities, relationships, constraints, and processes) into a single global view of the data in the enterprise. The conceptual model yields some very important advantages.

First, it provides relatively easily understood bird's eye (macro level) view of the data environment.

Second, the conceptual model is independent of both software and hardware. Software independence means that the model does not depend on the DBMS software used to implement the model.

Hardware independence means that the model does not depend on the hardware used in the implementation of the model. Therefore, changes in either the hardware or the DBMS software will have no effect on the database design at the conceptual level. Generally, the term logical design is used to refer to the task of creating a conceptual data model that could be implemented in any DBMS.

The Internal Model:

Once a specific DBMS has been selected, the internal model maps the conceptual model to the DBMS. The internal model is the representation of the database as "seen" by the DBMS. In other words, the internal model requires the designer to match the conceptual model's characteristics and constraints to those of the selected implementation model. An internal schema depicts a specific representation of an internal model, using the database constructs supported by the chosen database. Because the internal model depends on specific database software, it is said to be software dependent. Therefore, a change in the DBMS software requires that the internal model be changed to fit the characteristics and requirements of the implementation database model.

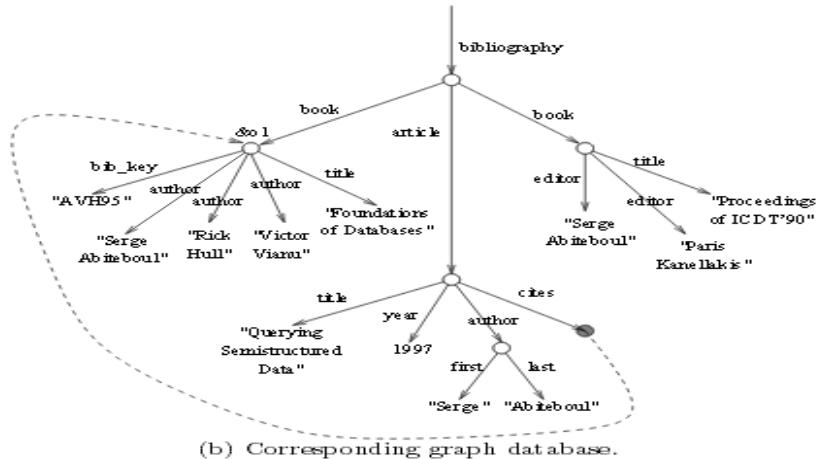
The Physical Model: The physical model operates at the lowest level of abstraction, describing the way data are saved on storage media such as disks or tapes. The storage structures used are dependent on the software (the DBMS and the operating system) and on the type of storage devices that the computer can handle. The precision required in the physical model's definition demands that database designers who work at this level have a detailed knowledge of the hardware and software used to implement the database design. As noted earlier, the physical model is dependent on the DBMS, methods of accessing files, and types of hardware storage devices

supported by the operating system. When you can change the physical model without affecting the internal model, you have physical independence. Therefore, a change in storage devices or methods and even a change in operating system will not affect the internal model.

5. Explain in brief object based and semi-structured databases.

Ans:

A semi structured database (SSDB) is a collection of objects in which attribute labels may be associated with values of different types, even within the same complex object. For this reason, it is usually difficult or impossible to define a schema for the data that has a complete, independent description of all objects in the database. Instead, typically each object has its type annotated to itself, i.e., a “local schema”. As a consequence, the actual schema of the database, i.e., the union of all local schemas, is relatively large compared to the data alone, thus blurring the line that separates data from metadata. Figure 1(a) depicts a textual description of a hypothetical semi structured database of bibliographic entries, which we use as an example throughout the paper. This example is not intended to cover all aspects of semi structured data, rather to give a flavor of the kinds of constructs we will be dealing with. In particular, note the irregular type for book objects and the reference from the article object to the first book object, via its oid



Example of a semistructured database.

Semi structured data models. Semi structured databases are often modeled by rooted directed graphs, in which vertices represent objects and edges represent relationships among objects: attribute edges represent the attributes of the objects, while reference edges represent references between objects (e.g., the *cites* attribute in our example). Reference edges are useful for avoiding duplicating the description of objects. Figure 1(b) shows a graph that models our example database.

6. Explain Database System Architecture.

Ans:

Components of a DBMS

These functional units of a database system can be divided into two parts:

1. Query Processor Units(Components)
2. Storage Manager Units

Query Processor Units:

Query processor units deal with execution of DDL and DML statements.

- **DDL Interpreter**— Interprets DDL statements into a set of tables containing metadata.
- **DML Compiler**— Translates DML statements into low level instructions that the query evaluation engine understands.
- **Embedded DML Pre-compiler**— Converts DML statements embedded in an application program into normal procedure calls in the host language.
- **Query Evaluation Engine**—Executes low level instructions generated by DML compiler.

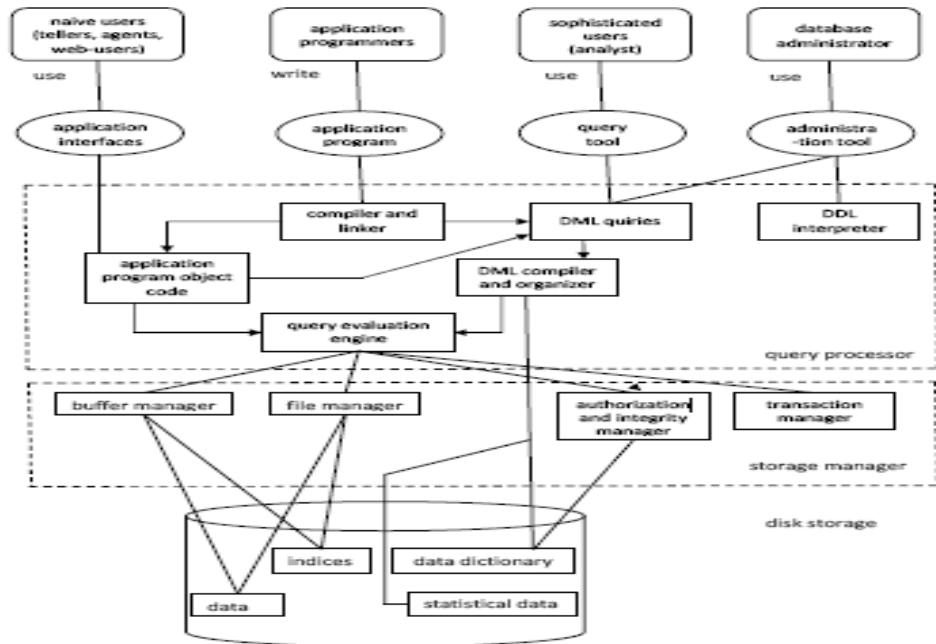
Storage Manager Units:

Storage manager units provide interface between the low level data stored in database and the application programs & queries submitted to the system.

- **Authorization Manager**—Checks the authority of users to access data.
- **Integrity Manager**—Checks for the satisfaction of the integrity constraints.
- **Transaction Manager**—Preserves atomicity and controls concurrency.
- **File Manager**—Manages allocation of space on disk storage.
- **Buffer Manager**—Fetches data from disk storage to memory for being used.

In addition to these functional units ,several data structures are required to implement physical storage system. These are described below:

- **Data Files**— To store user data.
- **Data Dictionary and System Catalog**— To store meta data. It issued heavily, almost for each and every data manipulation operation. So, it should be accessed efficiently.
- **Indices**—To provide faster access to data items.
- **Statistical Data**— To store statistical information about the data in the database. This information issued by the query processor to select efficient ways to execute a query.



7. Write short note on followings:

(i) Relational Constraints

Ans: \clubsuit Constraints

- Restrictions on the permitted values in a database state
- Derived from the rules in the mini world that the database represents
- \clubsuit Inherent model-based constraints or implicit constraints
 - Inherent in the data model
 - e.g., duplicate tuples are not allowed in a relation
- \clubsuit Schema-based constraints or explicit constraints
 - Can be directly expressed in schemas of the data model
 - e.g., films have only one director
- \clubsuit Application-based or semantic constraints
 - Also called business rules
 - Not directly expressed in schemas
 - Expressed and enforced by application program
 - e.g., this year's salary increase can be no more than last year's

(ii) Disadvantages of Relational Approach

Ans: It is convenient to divide the limitations up into two categories.

First of all, there are always very special types of data which require special forms of representation

and/or inference. Some examples are the following.

Limitations regarding special forms of data:

- Temporal data
- Spatial data
- Multimedia data
- Unstructured data (warehousing/mining)
- Document libraries (digital libraries)

Limitations regarding SQL as the query language:

· Recursive queries (e.g., compute the ancestor relation from the parent relation):

· Although part of the SQL:1999 standard, recursive queries are still not supported by many systems (e.g. PostgreSQL).

Support for recursive queries in SQL:1999 is weak in any case. (Only so-called linear queries are supported.)

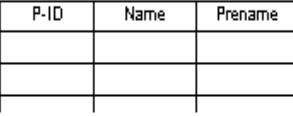
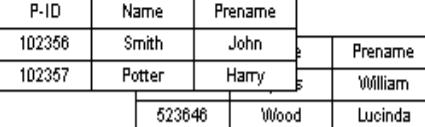
(iii) Instances and Schemas

Ans: Independent from the database model it is important to differentiate between the description of the database and the database itself. The description of the database is called **database scheme** or also metadata. The database scheme is defined during the database design process and changes very rarely afterwards.

The actual content of the database, the data, changes often over the years. A database state at a specific time defined through the currently existing content and relationship and their attributes is called a **database instance**

The following illustration shows that a database scheme could be looked at like a template or building plan for one or several database instances.

Analogy

| | Real World | Database |
|-----------|--|--|
| Scheme |  Plan for a Standard-House |  Template for a Table |
| Instances |  Built Standard-Houses |  Data-filled Tables |

Analogy Database Schemes and Building Plans

7. Describe the differences between data and database administration.

Ans: A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data. This is a collection of related data with an implicit meaning and hence is a database. The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient. By data, we mean known facts that can be recorded and that have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people you know. You may have recorded this data in an indexed address book, or you may have stored it on a diskette, using a personal computer and software such as DBASE IV or V, Microsoft ACCESS, or EXCEL. A datum – a unit of data – is a symbol or a set of symbols which is used to represent something. This relationship between symbols and what they represent is the essence of what we mean by information. Hence, information is interpreted data – data supplied with semantics. Knowledge refers to the practical use of information. While information can be transported, stored or shared without many difficulties the same can not be said about knowledge. Knowledge necessarily involves a personal experience. Referring back to the scientific experiment, a third person reading the results will have information about it, while the person who conducted the experiment personally will have knowledge about it. Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the information stored, despite

system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results. Because information is so important in most organizations, computer scientists have developed a large body of concepts and techniques for managing data.

One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a **database administrator (DBA)**. The functions of a DBA include: Schema definition. The DBA creates the original database schema by executing a set of data definition statements in the DDL. Storage structure and access-method definition. Schema and physical-organization modification. The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance. Granting of authorization for data access. By granting different types of authorization, the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system. Routine maintenance. Examples of the database administrator's routine maintenance activities are: Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding. Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required. Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

8. How different users interact with the database system?

Ans: A primary goal of a database system is to retrieve information from and store new information in the database. People who work with a database can be categorized as database users or database administrators.

Database Users and User Interfaces:

There are four different types of database-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users.

Naive users are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously. For example, a bank teller who needs to transfer \$50 from account A to account B invokes a program called transfer. This program asks the teller for the amount of money to be transferred, the account from which the money is to be transferred, and the account to which the money is to be transferred. As another example, consider a user who wishes to find her account balance over the World Wide Web. Such a user may access a form, where she enters her account number. An application program at the Web server then retrieves the account balance, using the given account number,

and passes this information back to the user. The typical user interface for naive users is a forms interface, where the user can fill in appropriate fields of the form. Naive users may also simply read reports generated from the database.

Application programmers are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. **Rapid application development (RAD)** tools are tools that enable an application programmer to construct forms and reports without writing a program. There are also special types of programming languages that combine imperative control structures (for example, for loops, while loops and if-then-else statements) with statements of the data manipulation language. These languages, sometimes called fourth-generation languages, often

include special features to facilitate the generation of forms and the display of data on the screen. Most major commercial database systems include a fourth generation language.

Sophisticated users interact with the system without writing programs. Instead, they form their requests in a database query language. They submit each such query to a **query processor**, whose function is to break down DML statements into instructions that the storage manager understands. Analysts who submit queries to explore data in the database fall in this category.

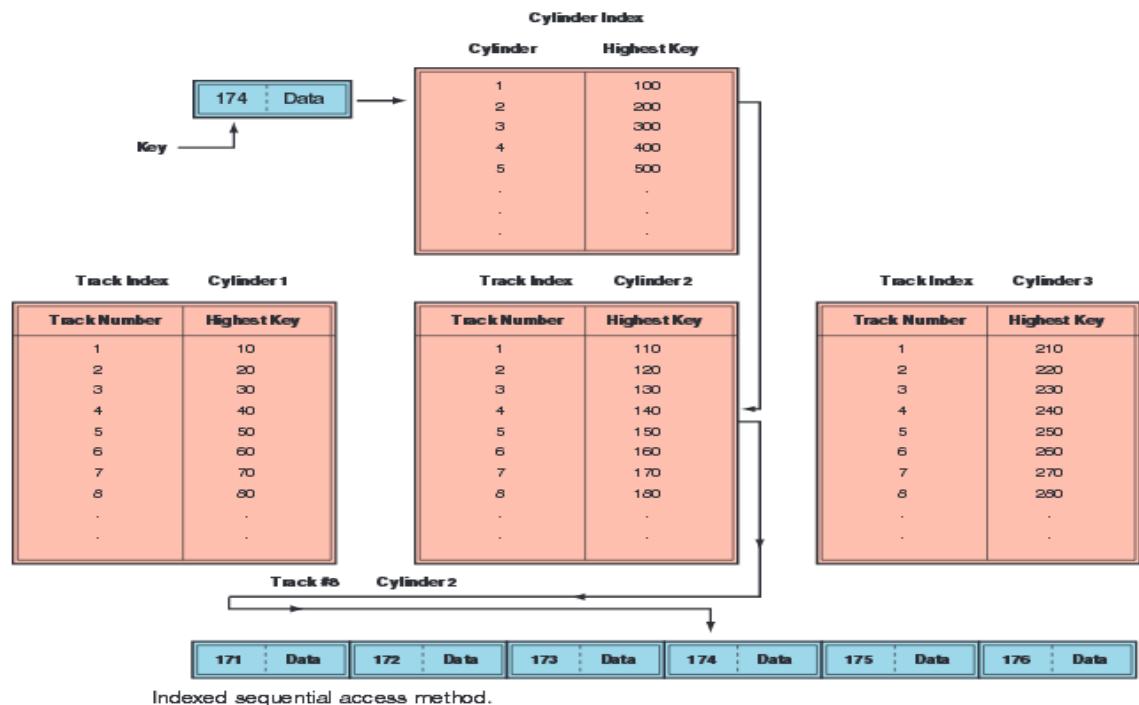
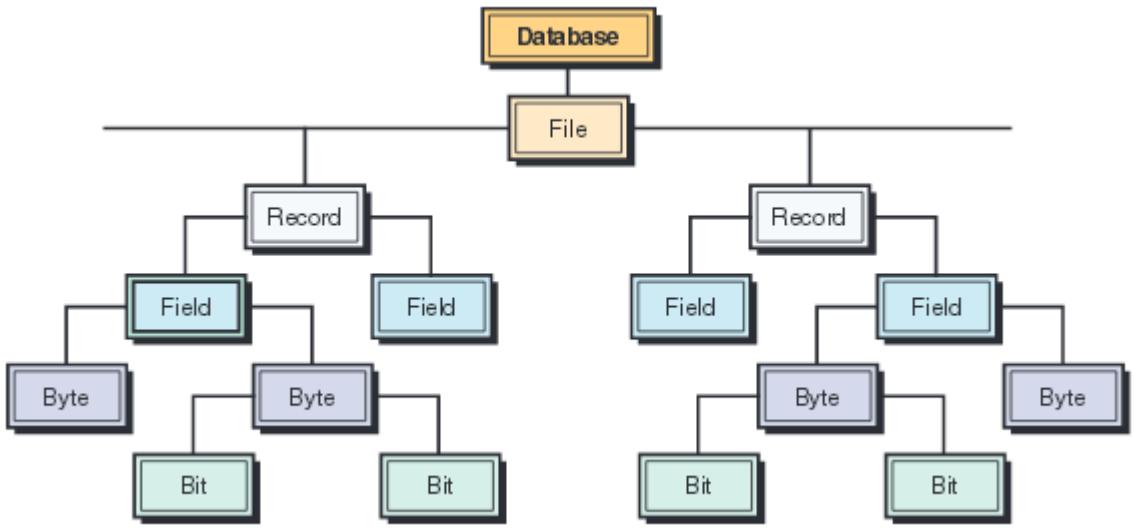
Online analytical processing (OLAP) tools simplify analysts' tasks by letting them view summaries of data in different ways. For instance, an analyst can see total sales by region (for example, North, South, East, and West), or by product, or by a combination of region and product (that is, total sales of each product in each region). The tools also permit the analyst to select specific regions, look at data in more detail (for example, sales by city within a region) or look at the data in less detail (for example, aggregate products together by category). Another class of tools for analysts is **data mining tools**, which help them find certain kinds of patterns in data.

Specialized users are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework. Among these applications are computer-aided design systems, knowledge base and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems.

9. Explain with suitable example data redundancy and inconsistency problem.

Ans: Records can be arranged in several ways on a storage medium, and the arrangement determines the manner in which individual records can be accessed. In sequential file organization, data records must be retrieved in the same physical sequence in which they are stored. (The operation is like a tape recorder.) In direct or random file organization, users can retrieve records in any sequence, without regard to actual physical order on the

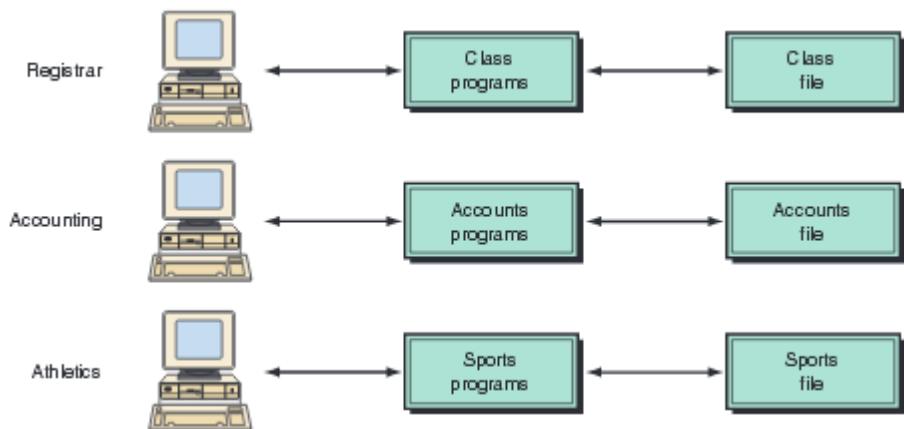
storage medium.



The operation is like a CD drive.) Magnetic tape utilizes sequential file organization, whereas magnetic disks use direct file organization. The indexed sequential access method (ISAM) uses an index of key fields to locate individual records (see Figure T3.2). An index to a file lists the key field of each record and where that record is physically located in storage. Records are stored on disks in their key sequence. A track index shows the highest value of the key field that can be found on a specific track. To locate a specific record, the track index is searched to locate the cylinder and the

track containing the record. The track is then sequentially read to find the record.

The direct file access method uses the key field to locate the physical address of a record. This process employs a mathematical formula called a transform algorithm to translate the key field directly into the record's storage location on disk. The algorithm performs a mathematical calculation on the record key, and the result of that calculation is the record's address. The direct access method is most appropriate when individual records must be located directly and rapidly for immediate processing, when a few records in the file need to be retrieved at one time, and when the required records are found in no particular sequence.



Organizations typically began automating one application at a time. These systems grew independently, without overall planning. Each application required its own data, which were organized into a data file. This approach led to redundancy, inconsistency, data isolation, and other problems. uses a university file environment as an example. The applications (e.g., registrar, accounting, or athletics) would share some common core functions, such as input, report generation, querying, and data browsing. However, these common functions would typically be designed, coded, documented, and tested, at great expense, for each application. Moreover, users must be trained to use each application. File environments often waste valuable resources creating and maintaining similar applications, as well as in training users how to use them.

Other problems arise with file management systems. The first problem is **data redundancy**: As applications and their data files were created by different programmers over a period of time, the same data could be duplicated in several files. In the university example, each data file will contain records about students, many of whom will be represented in other data files. Therefore, student files in the aggregate will contain some amount of duplicate data. This wastes physical computer storage media, the students' time and effort, and the clerks' time needed to enter and maintain the data. Data redundancy leads to the potential for data inconsistency. **Data inconsistency** means that the actual values across various copies of

the data no longer agree or are not synchronized. For example, if a student changes his or her address, the new address must be changed across all applications in the university that require the address. File organization also leads to difficulty in accessing data from different applications, a problem called **data isolation**. With applications uniquely designed and implemented, data files are likely to be organized differently, stored in different formats (e.g., height in inches versus height in centimeters), and often physically inaccessible to other applications. In the university example, an administrator who wanted to know which students taking advanced courses were also starting players on the football team would most likely not be able to get the answer from the computer-based file system. He or she would probably have to manually compare printed output data from two data files. This process would take a great deal of time and effort and would ignore the greatest strengths of computers—fast and accurate processing.

4.1.7. Assignments: 1

1. Draw and explain the detailed system architecture of DBMS.
2. Compare the database system with conventional file system.
3. Describe in detail about two-tier and three-tier client-server architectures

4.1.8. Previous Questions (Asked by JNTUK from the concerned Unit)

1. a) Draw and explain the detailed system architecture of DBMS.
b) What are the advantages of DBMS?
c) Describe the concept of client/server model.
2. a) Discuss the main characteristics of the database approach and specify how it differs from traditional file system.
b) Explain in detail about the three tier schema architecture of DBMS.
3. Discuss the activities of different database users. b) Briefly describe various architectures of database systems.
4. Compare the database system with conventional file system. [8M] b)
Describe in detail about two-tier and three-tier client-server architectures

5. a) What is data model? List and explain different data models.
b) Explain the difference between external, internal, and conceptual schemas. How are these different schema layers related to the concepts of logical and physical data independence?
6. a) What are the responsibilities of a DBA? If we assume that the DBA is never interested in running his or her own queries, does the DBA still need to understand query optimization? Why?
b) Which of the following plays an important role in representing information about the real world in a database? Explain briefly about:
 - i) The data definition language.
 - ii) The data manipulation language.
 - iii) The buffer manager.
 - iv) The data model.
7. a) What are application programs? Discuss in detail about database access for application programs.
b) Explain the difference between logical and physical data independence. What is logical data independence and why is it important?
8. a) Why would choose a database system instead of simply storing data in operating system files? When would it make sense not to use a database system?
b) Shows the structures of a typical DBMS based on the relational data model and explain in detail.

4.1.9. GATE Questions (Where relevant) –No Relevant Questions

4.1.10. Interview questions (which are frequently asked in a Technical round - Placements)

1. What is database?
2. What are the disadvantages in File Processing System?
3. Describe the three levels of data abstraction?

4. Define the "integrity rules"?
5. What is Data Independence?
6. What is a view? How it is related to data independence?
7. What is Data Model?

4.1.11. Real-Word (Live) Examples / Case studies wherever applicable –

Not applicable in this unit

4.1.12. Suggested "Expert Guest Lectures" (both from in and outside of the campus)

Not applicable in this unit

4.1.13. Literature references of Relevant NPTEL Videos/Web/YouTube videos etc.

- a. <https://www.youtube.com/watch?v=1057YmExS-I>
- b. <https://www.youtube.com/watch?v=FR4QleZaPeM>

4.1.14. Any Lab requirements; if so link it to Lab Lesson Plan.

Not applicable in this unit

4.1.15. Reference Text Books / with Journals Chapters etc.

Textbooks:

1. Database Management Systems, 3/e Raghuram Krishnan, Johannes Gehrke, TMH
2. Database Management System, 6/e Ramez Elmasri, Shamkant B. Navathe, PEA
3. Database Principles Fundamentals of Design Implementation and Management, Corlos Coronel, Steven Morris, Peter Robb, Cengage Learning

Introduction to relational model, concepts of domain, attribute, tuple, relation, importance of null values, constraints (Domain, key constraints, integrity constraints) and their importance BASIC SQL: Simple Database schema, data types, table definitions (create, alter), different DML operations (insert, delete, update), basic SQL querying (select and project) using where clause, logical operations, SQL functions (Date and Time, Numeric, String conversion)

4.2. Unit-II-Relational model and Basic SQL

4.2.1. Unit Objectives:

After reading this Unit, you should be able to understand:

- Understand the relational model's logical structure.
- Understand components of a relational database and their characteristics and relationships
- Understand basic concepts of table design

4.2.2. Unit Outcomes:

- Explain how relational database enables data to be viewed logically

rather than physically.

- Explain the characteristics of relational tables
- Define the following key terms: key, super key, candidate key, primary key, secondary key, foreign key, referential integrity.
- Identify and apply entity integrity and referential integrity rules
- Differentiate between Data Definition Language (DDL) and Data Manipulation Language

4.2.3. Unit Lecture Plan:

| Lecture No. | Topic | Methodology | Quick reference |
|--------------------|--|--------------------|------------------------|
| 1 | Relational Model: Introduction to relational model, Concepts of domain | Chalk & Board | Text. Book with Page |
| 2 | Attribute, tuple | Chalk & Board | Text. Book with Page |
| 3 | relation, importance of null values | Chalk & Board | Text. Book with Page |
| 4 | Constraints (Domain, Key constraints, integrity constraints) and their importance | Chalk & Board | Text. Book with Page |
| 5 | Basic SQL: Simple Database Schema, Data types, table Definitions (create, alter) | Chalk & Board | Text. Book with Page |
| 6 | Different DML operations (insert, delete, update), basic SQL querying (select and project) using | Chalk & Board | Text. Book with Page |

| | | | |
|---|--|---------------|----------------------|
| | where clause | | |
| 7 | Arthimetic & logical operations | Chalk & Board | Text. Book with Page |
| 8 | SQL functions (Date and Time, Numeric, String conversion). | Chalk & Board | Text. Book with Page |

4.2.4. Teaching Material / Teaching Aids as per above lecture plan.

4.2.4.1. Lecture-1

Introduction to Relational Model

Relational Model Concepts

The relational Model of Data is based on the concept of a Relation. A

Relation is a mathematical concept based on the ideas of sets. The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations. The model was first proposed by Dr. E.F. Codd of IBM in 1970 in the following paper: "A Relational Model for Large Shared Data Banks," Communications of the ACM, June 1970.

Informal Definitions

Relation:

A Relation is table of values. A relation may be thought of as a set of rows. A relation may alternately be thought of as a set of columns. Each row represents a fact that corresponds to a real-world entity or relationship. Each row has a value of an item or set of items that uniquely identifies that row in the table. Sometimes row-ids or sequential numbers are assigned to identify the rows in the table. Each column typically is called by its column name or column header or attribute name.

Formal definitions

A Relation may be defined in multiple ways. The **Schema** of a Relation: $R(A_1, A_2, \dots, A_n)$ Relation schema R is defined over **attributes** A_1, A_2, \dots, A_n .

For Example -

CUSTOMER (Cust-id, Cust-name, Address, Phone#)

Here, CUSTOMER is a relation defined over the four attributes Cust-id, Cust-name, Address, Phone#, each of which has a **domain** or a set of valid values. For example, the domain of Cust-id is 6 digit numbers.

A tuple is an ordered set of values. Each value is derived from an appropriate domain. Each row in the CUSTOMER table may be referred to as a tuple in the table and would consist of four values. <632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000"> is a tuple belonging to the CUSTOMER relation.

A relation may be regarded as a set of tuples (rows). Columns in a table are also called attributes of the relation. A domain has a logical definition: e.g., "USA_phone_numbers" are the set of 10 digit phone numbers valid in the U.S.

A domain may have a data-type or a format defined for it. The USA_phone_numbers may have a format: (ddd)-ddd-dddd where each d is a decimal digit. E.g., Dates have various formats such as month name, date, year or yyyy-mm-dd, or dd mm,yyyy etc.

An attribute designates the role played by the domain. E.g., the domain Date may be used to define attributes "Invoice-date" and "Payment-date".

The relation is formed over the Cartesian product of the sets; each set has values from a domain; that domain is used in a specific role which is conveyed by the attribute name.

For example, attribute Cust-name is defined over the domain of strings of 25 characters. The role these strings play in the CUSTOMER relation is that of the name of customers.

Formally, Given R(A₁, A₂,, A_n)

r(R) ⊂ dom (A₁) X dom (A₂) XX dom(A_n) R: schema of the relation

r of R: a specific "value" or population of R. R is also called the intension of a relation

r is also called the extension

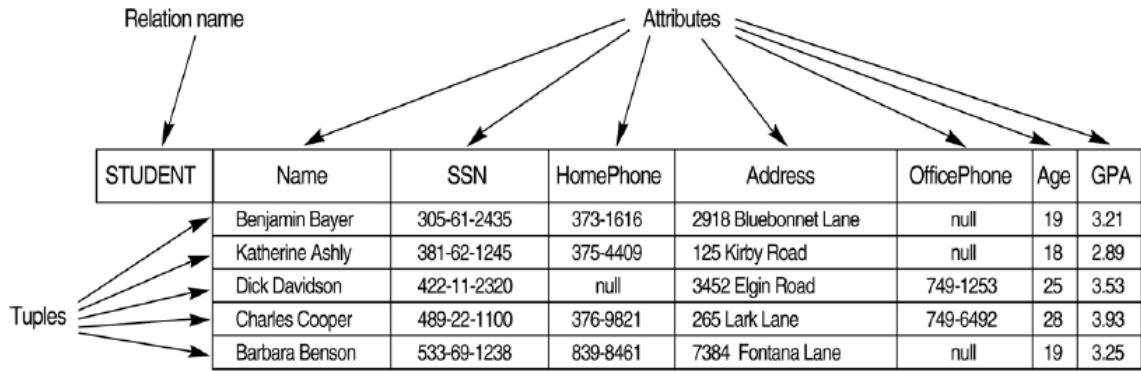
Let S₁ = {0,1}

Let S₂ = {a,b,c}

Let R ⊂ S₁ X S₂

Then for example: r(R) = {<0,a>, <0,b>, <1,c>} "population" or "extension" r of the relation has three tuples.

Example



Characteristics of Relations

Ordering of tuples in a relation $r(R)$: The tuples are not considered to be ordered, even though they appear to be in the tabular form.

Ordering of attributes in a relation schema R (and of values within each tuple): We will consider the attributes in R (A_1, A_2, \dots, A_n) and the values in $t = \langle v_1, v_2, \dots, v_n \rangle$ to be ordered .

(However, a more general alternative definition of relation does not require this ordering).

Values in a tuple: All values are considered atomic (indivisible). A special null value is used to represent values that are unknown or inapplicable to certain tuples.

Notation:

We refer to component values of a tuple t by $t[A_i] = v_i$ (the value of attribute A_i for tuple t).

Similarly, $t[A_u, A_v, \dots, A_w]$ refers to the subtuple of t containing the values of attributes A_u, A_v, \dots, A_w , respectively.

| STUDENT | Name | SSN | HomePhone | Address | OfficePhone | Age | GPA |
|---------|-----------------|-------------|-----------|----------------------|-------------|-----|------|
| | Dick Davidson | 422-11-2320 | null | 3452 Elgin Road | 749-1253 | 25 | 3.53 |
| | Barbara Benson | 533-69-1238 | 839-8461 | 7384 Fontana Lane | null | 19 | 3.25 |
| | Charles Cooper | 489-22-1100 | 376-9821 | 265 Lark Lane | 749-6492 | 28 | 3.93 |
| | Katherine Ashly | 381-62-1245 | 375-4409 | 125 Kirby Road | null | 18 | 2.89 |
| | Benjamin Bayer | 305-61-2435 | 373-1616 | 2918 Bluebonnet Lane | null | 19 | 3.21 |

4.2.4.2. Lecture -2

Concepts of Domain, attributes, Tuples, relation

Relational Schema

A Relational Database Schema comprises

1. The definition of all domains
2. The definition of all relations, specifying for each
 - a) Its intension (all attribute names), and
 - b) a primary key

An example of such a schema which has all the components mentioned above. The primary keys are designated by shading the component attribute names. Of course, this is only an informal view of a schema. Its formal definition must rely on the use of a specific DDL whose syntax may vary from one DBMS to another.

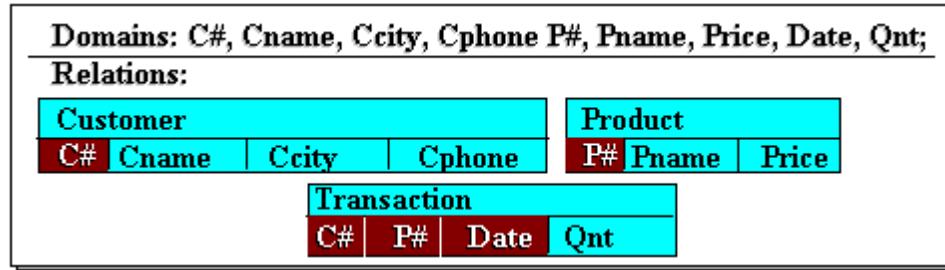


Fig :An example relational schema

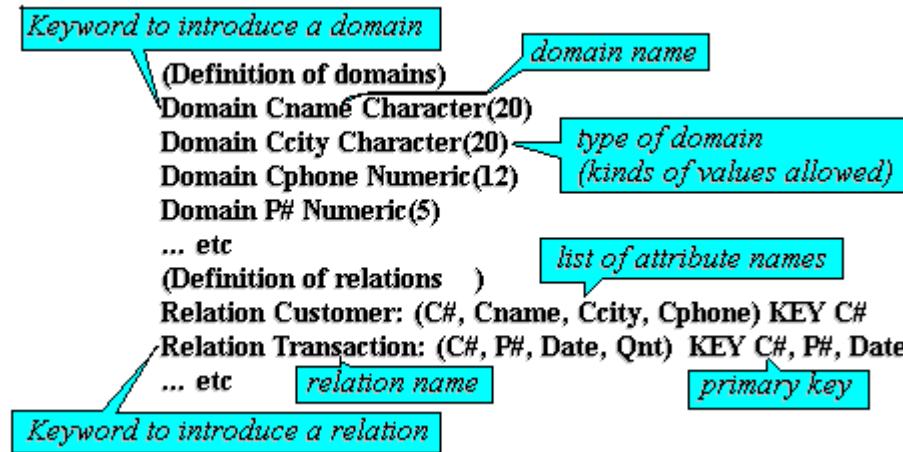
There is, however, a useful notation for relational schemas commonly adopted to document and communicate database designs free of any specific DDL. It takes the simple form:

<relation name> : <list of attribute names>

Additionally, attributes that are part of the primary key are underlined. Thus, for the example in Figure , the schema would be written as follows:

Customer: (C#, Cname, Ccity, Cphone)
Transaction: (C#, P#, Date, Qnt)
Product: (P#, Pname, Price)

This notation is useful in clarifying the overall organization of the database but omits some details, particularly the properties of domains. As an example of a more complete definition using a more concrete DDL, we rewrite some the schema above using Codd's original notation. The principal components of his notation are annotated alongside.



Domains, Attributes, Tuples, and Relations:

- **Domain D**

Set of atomic values

- **Atomic**

Each value indivisible

- Specifying a domain

Data type specified for each domain

- **Relation schema R**

1. Denoted by $R(A_1, A_2, \dots, A_n)$
2. Made up of a relation name R and a list of attributes, A_1, A_2, \dots, A_n
 - **Attribute A_i**
 1. Name of a role played by some domain D in the relation schema R
 - **Degree (or arity)** of a relation
1. Number of attributes n of its relation schema
 - **Relation (or relation state)**
 1. Set of n -tuples $r = \{t_1, t_2, \dots, t_m\}$
 2. Each n -tuple t
 - 3.
- Ordered list of n values $t = \langle v_1, v_2, \dots, v_n \rangle$
- Each value $v_i, 1 \leq i \leq n$, is an element of $\text{dom}(A_i)$ or is a special NULL value
 - **Relation (or relation state) $r(R)$**
1. **Mathematical relation** of degree n on the domains $\text{dom}(A_1), \text{dom}(A_2), \dots, \text{dom}(A_n)$
2. **Subset** of the **Cartesian product** of the domains that define R :
 - $r(R) \subseteq (\text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n))$

- **Cardinality**

Total number of values in domain

- **Current relation state**

1. Relation state at a given time
2. Reflects only the valid tuples that represent a particular state of the real world

- **Attribute names**

1. Indicate different **roles**, or interpretations, for the domain

4.2.4.3 Lecture -3

Importance of null values

Values and NULLs in tuples

1. Each value in a tuple is atomic
 - **Flat relational model**
1. Composite and multivalued attributes not allowed
2. **First normal form** assumption
 - Multivalued attributes
1. Must be represented by separate relations
 - Composite attributes
1. Represented only by simple component attributes in basic relational model

NULL values

Represent the values of attributes that may be unknown or may not apply to a tuple

Meanings for NULL values

- *Value unknown*
- *Value exists but is not available*
- *Attribute does not apply to this tuple (also known as value undefined)*

Interpretation (meaning) of a relation

Assertion

- Each tuple in the relation is a **fact** or a particular instance of the assertion

Predicate

- Values in each tuple interpreted as values that satisfy predicate

4.2.4.4 Lecture -4

Constraints (Domain, Key constraints, integrity constraints) and their importance

Domain Constraints: Attributes have associated domains *Domain*—set of atomic data values of a specific type. *Constraint*—stipulates that the actual values of an attribute in any tuple must belong to the declared domain.

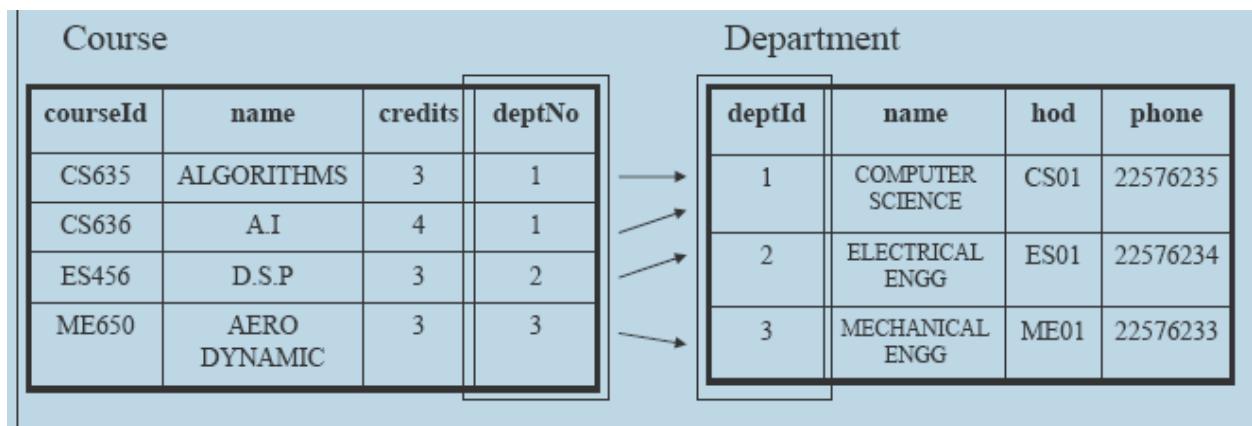
•**Key Constraint:** Relation scheme –associated keys Constraint –if K is supposed to be a key for scheme R , any relation instance r on R should not have two tuples that have identical values for attributes in K . Also, none of the key attributes can have null value.

Foreign Keys

- Tuples in one relation, say $r_1(R_1)$, often need to refer to tuples in another relation, say $r_2(R_2)$ •to capture relationships between entities
- Primary Key of R_2 : $K = \{B_1, B_2, \dots, B_j\}$
- A set of attributes $F = \{A_1, A_2, \dots, A_j\}$ of R_1 such that $\text{dom}(A_i) = \text{dom}(B_i)$, $1 \leq i \leq j$ and whose values are used to refer to tuples in r_2 is called a *foreign key* in R_1 referring to R_2 .
- R_1, R_2 can be the same scheme also.
- There can be more than one foreign key in a relation scheme

Foreign Key –Examples

Foreign key attribute *deptNo* of *course* relation refers to Primary key attribute *deptID* of *department* relation



It is possible for a foreign key in a relation to refer to the primary key of the relation itself An

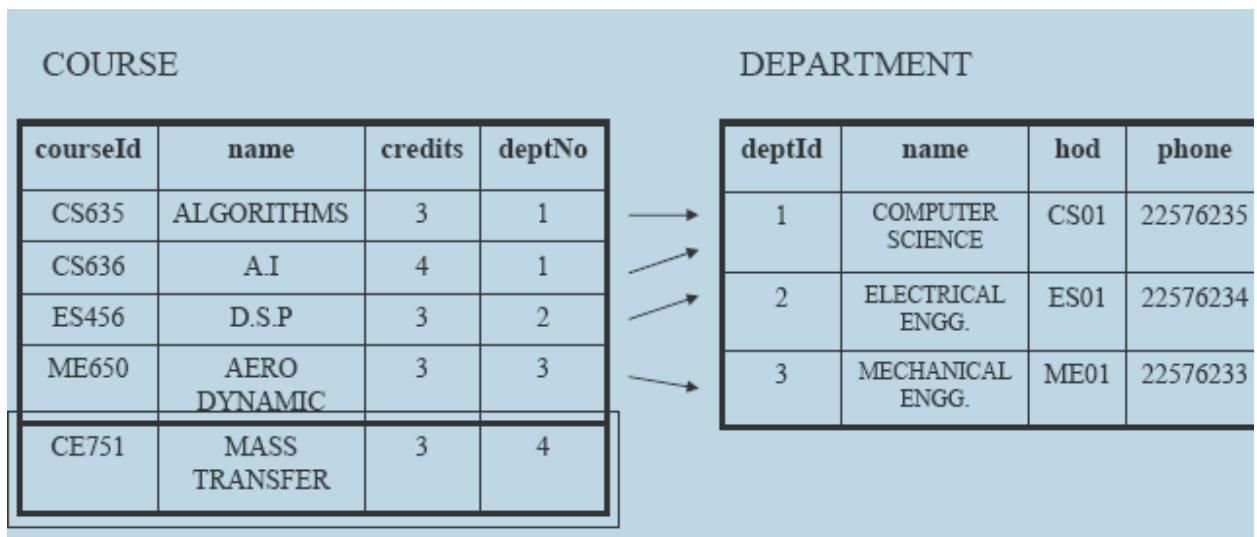
Example:

univEmployee (empNo, name, sex, salary, dept, reportsTo)

reportsTo is a foreign key referring to *empNo* of the same relation Every employee in the university reports to some other employee for administrative purposes-except the *vice-chancellor*, of course!

Referential Integrity Constraint (RIC)

- Let F be a foreign key in scheme R1 referring to scheme R2 and let K be the primary key of R2.
- RIC: any relational instance r_1 on R1, r_2 on R2 must be for any tuple t_1 in r_1 , either its F -attribute values are *null* or they are identical to the K -attribute values of *some* tuple in r_2 .
- RIC ensures that references to tuples in r_2 are for *currently existing* tuples.
- That is, there are no *dangling* references.



The new course refers to a non-existent department and thus violates the RIC

Example Relational Scheme course (courseId, cname, credits, deptNo) Here, *deptNo* indicates the department that offers the course.

enrollment (rollNo, courseId, sem, year, grade) Here, *sem* can be either "odd" or "even" indicating the two semesters of an academic year. The value of *grade* will be null for the current semester and non-null for past semesters.

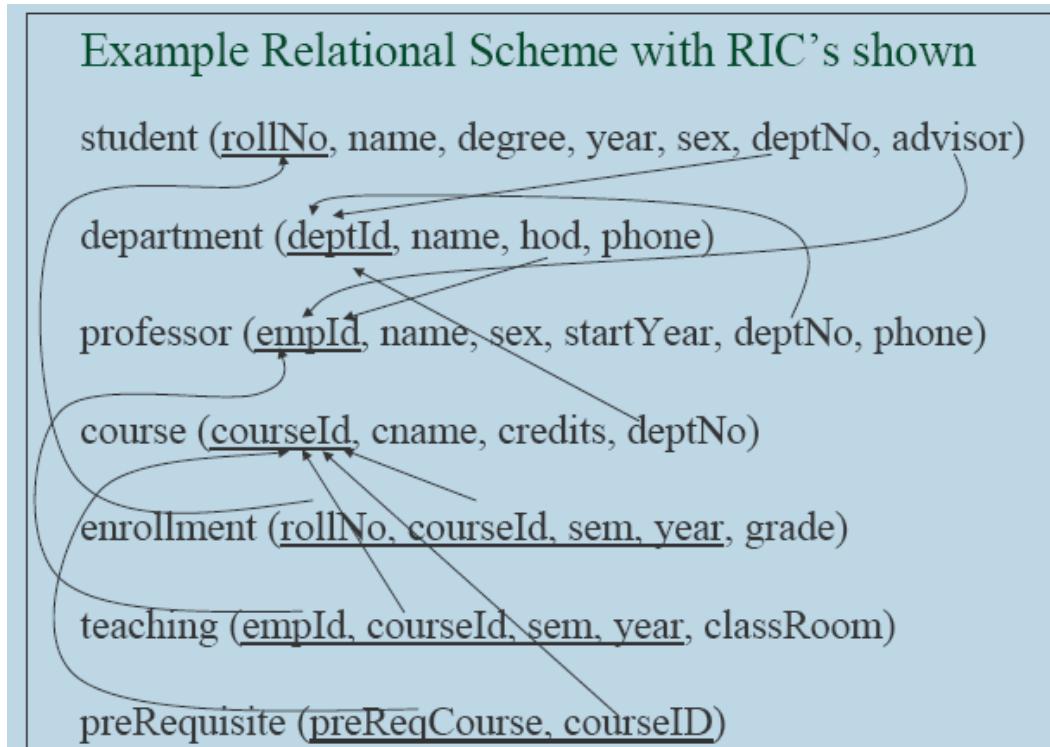
Teaching (empld, courseId, sem, year, classRoom)

preRequisite(preReqCourse, courseId)

Here, if (c_1, c_2) is a tuple, it indicates that c_1 should be successfully completed before enrolling for c_2 .

Example Relational Scheme

student (rollNo, name, degree, year, sex, deptNo, advisor)
 department (deptId, name, hod, phone)
 professor (empId, name, sex, startYear, deptNo, phone)
 course (courseId, cname, credits, deptNo)
 enrollment (rollNo, courseId, sem, year, grade)
 teaching (empId, courseId, sem, year, classRoom)
 preRequisite(preReqCourse, courseId)



4.2.4.5 Lecture -5

Basic SQL:

A Brief History of SQL

The history of SQL begins in an IBM laboratory in San Jose, California, where SQL was developed in the late 1970s. The initials stand for Structured Query Language, and the language itself is often referred to as "sequel." It was originally developed for IBM's DB2 product (a relational database management system, or RDBMS, that can still be bought today for various platforms and environments). In fact, SQL makes an RDBMS possible. SQL is a nonprocedural language, in contrast to the procedural or third generation languages (3GLs) such as COBOL and C that had been created up to that time.

NOTE:

Nonprocedural means what rather than how. For example, SQL describes what data to retrieve, delete, or insert, rather than how to perform the operation.

A Brief History of Databases

A little background on the evolution of databases and database theory will help you understand the workings of SQL. Database systems store information in every conceivable business environment. From large tracking databases such as airline reservation systems to a child's baseball card collection, database systems store and distribute the data that we depend on. Until the last few years, large database systems could be run only on large mainframe computers. These machines have traditionally been expensive to design, purchase, and maintain. However, today's generation of powerful, inexpensive workstation computers enable programmers to design software that maintains and distributes data quickly and inexpensively.

Codd's 12 Rules for a Relational Database Model:

Rule 1: Information rule

This rule states that all information (data), which is stored in the database, must be a value of some table cell. Everything in a database must be stored in table formats. This information can be user data or meta-data.

Rule 2: Guaranteed Access rule

This rule states that every single data element (value) is guaranteed to be accessible logically with combination of table-name, primary-key (row value) and attribute-name (column value). No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL values

This rule states the NULL values in the database must be given a systematic treatment. As a NULL may have several meanings, i.e. NULL can be interpreted as one of the following: data is missing, data is not known, data is not applicable etc.

Rule 4: Active online catalog

This rule states that the structure description of whole database must be stored in an online catalog, i.e. data dictionary, which can be accessed by the authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

Rule 5: Comprehensive data sub-language rule

This rule states that a database must have a support for a language which has linear syntax which is capable of data definition, data manipulation and transaction management operations. Database can be accessed by means of this language only, either directly or by means of some application. If the database can be accessed or manipulated in some way without any help of this language, it is then a violation.

Rule 6: View updating rule

This rule states that all views of database, which can theoretically be updated, must also be updatable by the system.

Rule 7: High-level insert, update and delete rule

This rule states the database must employ support high-level insertion, updating and deletion. This must not be limited to a single row that is, it must also support union, intersection and minus operations to yield sets of data records.

Rule 8: Physical data independence

This rule states that the application should not have any concern about how the data is physically stored. Also, any change in its physical structure must not have any impact on application.

Rule 9: Logical data independence

This rule states that the logical data must be independent of its user's view (application). Any change in logical data must not imply any change in the application using it. For example, if two tables are merged or one is split into two different tables, there should be no impact the change on user application. This is one of the most difficult rule to apply.

Rule 10: Integrity independence

This rule states that the database must be independent of the application using it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes database independent of the front-end application and its interface.

Rule 11: Distribution independence

This rule states that the end user must not be able to see that the data is distributed over various locations. User must also see that data is located at one site only. This rule has been proven as a foundation of distributed

database systems.

Rule 11: Distribution independence

This rule states that the end user must not be able to see that the data is distributed over various locations. User must also see that data is located at one site only. This rule has been proven as a foundation of distributed database systems.

Rule 12: Non-subversion rule

This rule states that if a system has an interface that provides access to low level records, this interface then must not be able to subvert the system and bypass security and integrity constraints.

Simple database scheme:

```
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) );
CREATE TABLE DEPARTMENT
(
    Dname        VARCHAR(15)      NOT NULL,
    Dnumber      INT             NOT NULL,
    Mgr_ssn     CHAR(9)         NOT NULL,
    Mgr_start_date DATE,
PRIMARY KEY (Dnumber),
UNIQUE (Dname),
FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
```

```

CREATE TABLE DEPT_LOCATIONS
(
    Dnumber          INT           NOT NULL,
    Dlocation        VARCHAR(15)   NOT NULL,
    PRIMARY KEY (Dnumber, Dlocation),
    FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber);
)
CREATE TABLE PROJECT
(
    Pname            VARCHAR(15)   NOT NULL,
    Pnumber          INT           NOT NULL,
    Plocation        VARCHAR(15),
    Dnum             INT           NOT NULL,
    PRIMARY KEY (Pnumber),
    UNIQUE (Pname),
    FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber);
)
CREATE TABLE WORKS_ON
(
    Essn            CHAR(9)       NOT NULL,
    Pno              INT           NOT NULL,
    Hours            DECIMAL(3,1)  NOT NULL,
    PRIMARY KEY (Essn, Pno),
    FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
    FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber);
)
CREATE TABLE DEPENDENT
(
    Essn            CHAR(9)       NOT NULL,
    Dependent_name  VARCHAR(15)   NOT NULL,
    Sex              CHAR,
    Bdate            DATE,
    Relationship     VARCHAR(8),
    PRIMARY KEY (Essn, Dependent_name),
    FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn);
)

```

Basic data types

1. Numeric data types

- Integer numbers: INTEGER, INT, and SMALLINT
- Floating-point (real) numbers: FLOAT or REAL, and DOUBLE PRECISION

2. Character-string data types

- Fixed length: CHAR(n), CHARACTER(n)
- Varying length: VARCHAR(n), CHAR VARYING(n), CHARACTER VARYING(n)

3. Bit-string data types

- Fixed length: BIT(n)
- Varying length: BIT VARYING(n)

4. Boolean data type

- Values of TRUE or FALSE or NULL

5. DATE data type

- Ten positions
- Components are YEAR, MONTH, and DAY in the form YYYY-MM-DD

4.2.4.6 Lecture -6

Creating and Maintaining Tables:

- Create key fields
- Create a database with its associated tables

- Create, alter, and drop a table
- Add data to the database
- Modify the data in a database
- Drop databases

The CREATE TABLE Statement

The process of creating a table is far more standardized than the CREATE DATABASE statement. Here's the basic syntax for the CREATE TABLE statement:

SYNTAX:

```
CREATE TABLE table_name
( field1 datatype [ NOT NULL ],
field2 datatype [ NOT NULL ],
field3 datatype [ NOT NULL ]...)
```

A simple example of a CREATE TABLE statement follows

INPUT/OUTPUT:

```
SQL> CREATE TABLE BILLS (
2 NAME CHAR(30),
3 AMOUNT NUMBER,
4 ACCOUNT_ID NUMBER);
Table created.
```

Examples

INPUT/OUTPUT:

```
SQL> create database PAYMENTS;
Statement processed.
SQL> create table BILLS (
2 NAME CHAR(30) NOT NULL,
3 AMOUNT NUMBER,
4 ACCOUNT_ID NUMBER NOT NULL);
Table created.
```

```
SQL> create table BANK_ACCOUNTS (
2 ACCOUNT_ID NUMBER NOT NULL,
3 TYPE CHAR(30),
4 BALANCE NUMBER,
5 BANK CHAR(30));
Table created.
```

```
SQL> create table COMPANY (
2 NAME CHAR(30) NOT NULL,
3 ADDRESS CHAR(50),
```

4 CITY CHAR(30),
5 STATE CHAR(2));
Table created.

Table . Sample data for the BILLS table.

| Name | Amount | Account_ID |
|------------------|--------|------------|
| Phone Company | 125 | 1 |
| Power Company | 75 | 1 |
| Record Club | 25 | 2 |
| Software Company | 250 | 1 |
| Cable TV Company | 35 | 3 |

Table . Sample data for the BANK_ACCOUNTS table.

| Account_ID | Type | Balance | Bank |
|------------|--------------|---------|------------------|
| 1 | Checking | 500 | First Federal |
| 2 | Money Market | 1200 | First Investor's |
| 3 | Checking | 90 | Credit Union |

Table Sample data for the COMPANY table.

| Name | Address | City | State |
|------------------|----------------|---------------|-------|
| Phone Company | 111 1st Street | Atlanta | GA |
| Power Company | 222 2nd Street | Jacksonville | FL |
| Record Club | 333 3rd Avenue | Los Angeles | CA |
| Software Company | 444 4th Drive | San Francisco | CA |
| Cable TV Company | 555 5th Drive | Austin | TX |

The ALTER TABLE Statement

Many times your database design does not account for everything it should. Also, requirements for applications and databases are always subject to change. The ALTER TABLE statement enables the database administrator or designer to change the structure of a table after it has been created.

The ALTER TABLE command enables you to do two things:
Add a column to an existing table

Modify a column that already exists

The syntax for the ALTER TABLE statement is as follows:

SYNTAX:

```
ALTER TABLE table_name  
<ADD column_name data_type; |  
MODIFY column_name data_type;>
```

The following command changes the NAME field of the BILLS table to hold 40 characters:

INPUT/OUTPUT:

```
SQL> ALTER TABLE BILLS  
2 MODIFY NAME CHAR(40);  
Table altered.
```

Here's a statement to add a new column to the NEW_BILLS table:

INPUT/OUTPUT:

```
SQL> ALTER TABLE NEW_BILLS  
2 ADD COMMENTS CHAR(80);  
Table altered.
```

SYNTAX:

```
ALTER TABLE table_name MODIFY (column_name data_type NULL);
```

To change a column from NULL to NOT NULL, you might have to take several steps:

1. Determine whether the column has any NULL values.
2. Deal with any NULL values that you find. (Delete those records, update the column's value, and so on.)
3. Issue the ALTER TABLE command.

Different DML operations:

Objectives

Today we discuss data manipulation. By the end of the day, you should understand:

- How to manipulate data using the INSERT, UPDATE, and DELETE commands
- The importance of using the WHERE clause when you are manipulating data
- The basics of importing and exporting data from foreign data sources

Introduction to Data Manipulation Statements

Up to this point you have learned how to retrieve data from a database using every selection criterion imaginable. After this data is retrieved, you can use it in an application program or edit it. Week 1 focused on retrieving data. However, you may have wondered how to enter data into the database in the

first place. You may also be wondering what to do with data that has been edited. Today we discuss three SQL statements that enable you to manipulate the data within a database's table. The three statements are as follows:

- The INSERT statement
- The UPDATE statement
- The DELETE statement

The INSERT Statement

The INSERT statement enables you to enter data into the database. It can be broken down into two statements:

[INSERT...VALUES](#)

and

[INSERT...SELECT](#)

The INSERT...VALUES Statement

The INSERT...VALUES statement enters data into a table one record at a time. It is useful for small operations that deal with just a few records. The syntax of this statement is as follows:\

SYNTAX:

```
INSERT INTO table_name  
(col1, col2...)  
VALUES(value1, value2...)
```

The basic format of the INSERT...VALUES statement adds a record to a table using the columns you give it and the corresponding values you instruct it to add. You must follow three rules when inserting data into a table with the INSERT...VALUES statement:

The values used must be the same data type as the fields they are being added to.

The data's size must be within the column's size. For instance, you cannot add an 80-character string to a 40-character column.

The data's location in the VALUES list must correspond to the location in the column list of the column it is being added to. (That is, the first value must be entered into the first column, the second value into the second column, and so on.)

Example

Assume you have a COLLECTION table that lists all the important stuff you have collected. You can display the table's contents by writing

INPUT:

[SQL> SELECT * FROM COLLECTION;](#)

which would yield this:

OUTPUT:

| ITEM | WORTH | REMARKS |
|----------------------|-------|------------------------------|
| NBA ALL STAR CARDS | 300 | SOME STILL IN BIKE SPOKES |
| MALIBU BARBIE | 150 | TAN NEEDS WORK |
| STAR WARS GLASS | 5.5 | HANDLE CHIPPED |
| LOCK OF SPOUSES HAIR | 1 | HASN'T NOTICED BALD SPOT YET |

INPUT/OUTPUT:

```
SQL> INSERT INTO COLLECTION
2 (ITEM, WORTH, REMARKS)
3 VALUES('SUPERMANS CAPE', 250.00, 'TUGGED ON IT');
1 row created.
```

You can execute a simple SELECT statement to verify the insertion:

INPUT/OUTPUT:

```
SQL> SELECT * FROM COLLECTION;
```

| ITEM | WORTH | REMARKS |
|----------------------|-------|------------------------------|
| NBA ALL STAR CARDS | 300 | SOME STILL IN BIKE SPOKES |
| MALIBU BARBIE | 150 | TAN NEEDS WORK |
| STAR WARS GLASS | 5.5 | HANDLE CHIPPED |
| LOCK OF SPOUSES HAIR | 1 | HASN'T NOTICED BALD SPOT YET |
| SUPERMANS CAPE | 250 | TUGGED ON IT |

The UPDATE Statement

The purpose of the UPDATE statement is to change the values of existing records. The syntax is

SYNTAX:

```
UPDATE table_name
SET columnname1 = value1
[, columnname2 = value2]...
WHERE search_condition
```

This statement checks the WHERE clause first. For all records in the given table in which the WHERE clause evaluates to TRUE, the corresponding value is updated.

Example

This example illustrates the use of the UPDATE statement:

INPUT:

```
SQL> UPDATE COLLECTION
2 SET WORTH = 900
```

3 WHERE ITEM = 'STRING';

OUTPUT:

1 row updated.

To confirm the change, the query

INPUT/OUTPUT:

```
SQL> SELECT * FROM COLLECTION  
2 WHERE ITEM = 'STRING';
```

Yields

| ITEM | WORTH | REMARKS |
|--------|-------|------------------------------|
| STRING | 900 | SOME DAY IT WILL BE VALUABLE |

INPUT:

```
SQL> UPDATE COLLECTION  
2 SET WORTH = WORTH * 0.005;
```

that changes the table to this:

INPUT/OUTPUT:

```
SQL> SELECT * FROM COLLECTION;  
  
ITEM          WORTH REMARKS  
-----  
NBA ALL STAR CARDS    2.775 SOME STILL IN BIKE SPOKES  
MALIBU BARBIE        2.775 TAN NEEDS WORK  
STAR WARS GLASS       2.775 HANDLE CHIPPED  
LOCK OF SPOUSES HAIR   2.775 HASN'T NOTICED BALD SPOT YET  
SUPERMANS CAPE        2.775 TUGGED ON IT  
STRING                 2.775 SOME DAY IT WILL BE VALUABLE  
  
6 rows selected.
```

The DELETE Statement

In addition to adding data to a database, you will also need to delete data from a database. The syntax for the DELETE statement is

SYNTAX:

```
DELETE FROM tablename  
WHERE condition
```

The first thing you will probably notice about the DELETE command is that it doesn't have a prompt. Users are accustomed to being prompted for assurance when, for instance, a directory or file is deleted at the operating system level. Are you sure? (Y/N) is a common question asked before the operation is performed. Using SQL, when you instruct the DBMS to delete a group of records from a table, it obeys your command without asking. That is, when you tell SQL to delete a group of records, it will really do it!

Depending on the use of the DELETE statement's WHERE clause, SQL can do the following:

- Delete single rows
- Delete multiple rows
- Delete all rows
- Delete no rows

Here are several points to remember when using the DELETE statement:

- The DELETE statement cannot delete an individual field's values (use UPDATE instead). The DELETE statement deletes entire records from a single table.
- Like INSERT and UPDATE, deleting records from one table can cause referential integrity problems within other tables. Keep this potential problem area in mind when modifying data within a database.
- Using the DELETE statement deletes only records, not the table itself. Use the DROP TABLE statement (see Day 9) to remove an entire table.

This example shows you how to delete all the records from COLLECTION where WORTH is less than 275.

INPUT:

SQL> **DELETE FROM COLLECTION**

2 WHERE WORTH < 275;

4 rows deleted.

The result is a table that looks like this:

INPUT/OUTPUT:

```
SQL> SELECT * FROM COLLECTION;

ITEM          WORTH REMARKS
-----        -----
NBA ALL STAR CARDS    300 SOME STILL IN BIKE SPOKES
STRING          1000 SOME DAY IT WILL BE VALUABLE
```

WARNING: Like the UPDATE statement, if you omit a WHERE clause from the DELETE statement, all rows in that particular table will be deleted.

Basic SQL Querying

Objectives

- Write an SQL query
- Select and list all rows and columns from a table
- Select and list selected columns from a table
- Select and list columns from multiple tables

General Rules of Syntax

As you will find, syntax in SQL is quite flexible, although there are rules to follow as in any programming language. A simple query illustrates the basic syntax of an SQL select statement. Pay close attention to the case, spacing, and logical separation of the components of each query by SQL keywords.

```
SELECT NAME, STARTTERM, ENDTERM  
FROM PRESIDENTS  
WHERE NAME = 'LINCOLN';
```

In this example everything is capitalized, but it doesn't have to be. The preceding query would work just as well if it were written like this:

```
select name, startterm, endterm  
from presidents  
where name = 'LINCOLN';
```

Notice that LINCOLN appears in capital letters in both examples. Although actual SQL statements are not case sensitive, references to data in a database are. For instance, many companies store their data in uppercase. In the preceding example, assume that the column name stores its contents in uppercase. Therefore, a query searching for 'Lincoln' in the name column would not find any data to return. Check your implementation and/or company policies for any case requirements.

NOTE: Commands in SQL are not case sensitive.

Take another look at the sample query. Is there something magical in the spacing? Again the answer is no. The following code would work as well:

```
select name, startterm, endterm from presidents where name = 'LINCOLN';
```

If the magic isn't in the capitalization or the format, then just which elements are important? The answer is keywords, or the words in SQL that are reserved as a part of syntax. (Depending on the SQL statement, a keyword can be either a mandatory element of the statement or optional.) The keywords in the current example are

- SELECT
- FROM
- WHERE

Check the table of contents to see some of the SQL keywords you will learn and on what days.

The Building Blocks of Data Retrieval: SELECT and FROM

As your experience with SQL grows, you will notice that you are typing the words SELECT and FROM more than any other words in the SQL vocabulary. They aren't as glamorous as CREATE or as ruthless as DROP, but they are indispensable to any conversation you hope to have with the computer concerning data retrieval. And isn't data retrieval the reason that you entered mountains of information into your very expensive database in the first place? This discussion starts with SELECT because most of your statements will also start with SELECT:

SYNTAX:

SELECT <COLUMN NAMES>

Before going any further, look at the sample database that is the basis for the following examples. This database illustrates the basic functions of SELECT and FROM. In the real world you would use the techniques described on Day 8, "Manipulating Data," to build this database, but for the purpose of describing how to use SELECT and FROM, assume it already exists. This example uses the CHECKS table to retrieve information about checks that an individual has written. The CHECKS table:

| CHECK# | PAYEE | AMOUNT | REMARKS |
|--------|--------------------|--------|---------------------|
| 1 | Ma Bell | 150 | Have sons next time |
| 2 | Reading R.R. | 245.34 | Train to Chicago |
| 3 | Ma Bell | 200.32 | Cellular Phone |
| 4 | Local Utilities | 98 | Gas |
| 5 | Joes Stale \$ Dent | 150 | Groceries |
| 6 | Cash | 25 | Wild Night Out |
| 7 | Joans Gas | 25.1 | Gas |

Your First Query

INPUT:

SQL> select * from checks

| queries | CHECK# | PAYEE | AMOUNT | REMARKS |
|---------|--------|--------------------|--------|---------------------|
| | 1 | Ma Bell | 150 | Have sons next time |
| | 2 | Reading R.R. | 245.34 | Train to Chicago |
| | 3 | Ma Bell | 200.32 | Cellular Phone |
| | 4 | Local Utilities | 98 | Gas |
| | 5 | Joes Stale \$ Dent | 150 | Groceries |
| | 6 | Cash | 25 | Wild Night Out |
| | 7 | Joans Gas | 25.1 | Gas |

7 rows selected.

Terminating an SQL Statement

In some implementations of SQL, the semicolon at the end of the statement tells the interpreter that you are finished writing the query. For example,

Oracle's SQL*PLUS won't execute the query until it finds a semicolon (or a slash). On the other hand, some implementations of SQL do not use the semicolon as a terminator. For example, Microsoft Query and Borland's ISQL don't require a terminator, because your query is typed in an edit box and executed when you push a button.

Changing the Order of the Columns

The preceding example of an SQL statement used the * to select all columns from a table, the order of their appearance in the output being determined by the database. To specify the order of the columns, you could type something like:

INPUT:

```
SQL> SELECT payee, remarks, amount, check# from checks;
```

Notice that each column name is listed in the SELECT clause. The order in which the columns are listed is the order in which they will appear in the output. Notice both the commas that separate the column names and the space between the final column name and the subsequent clause (in this case FROM). The output would look like this:

OUTPUT:

| PAYEE | REMARKS | AMOUNT | CHECK# |
|--------------------|---------------------|--------|--------|
| Ma Bell | Have sons next time | 150 | 1 |
| Reading R.R. | Train to Chicago | 245.34 | 2 |
| Ma Bell | Cellular Phone | 200.32 | 3 |
| Local Utilities | Gas | 98 | 4 |
| Joes Stale \$ Dent | Groceries | 150 | 5 |
| Cash | Wild Night Out | 25 | 6 |
| Joans Gas | Gas | 25.1 | 7 |

7 rows selected.

INPUT:

```
SELECT payee, remarks, amount, check#
FROM checks;
```

Notice that the FROM clause has been carried over to the second line. This convention is a matter of personal taste when writing SQL code. The output would look like this:

OUTPUT:

| PAYEE | REMARKS | AMOUNT | CHECK# |
|--------------------|---------------------|--------|--------|
| Ma Bell | Have sons next time | 150 | 1 |
| Reading R.R. | Train to Chicago | 245.34 | 2 |
| Ma Bell | Cellular Phone | 200.32 | 3 |
| Local Utilities | Gas | 98 | 4 |
| Joes Stale \$ Dent | Groceries | 150 | 5 |
| Cash | Wild Night Out | 25 | 6 |

Selecting Individual Columns

Suppose you do not want to see every column in the database. You used `SELECT *` to find out what information was available, and now you want to concentrate on the check number and the amount. You type

INPUT:

`SQL> SELECT CHECK#, amount from checks;`
which returns

OUTPUT:

| CHECK# | AMOUNT |
|--------|--------|
| 1 | 150 |
| 2 | 245.34 |
| 3 | 200.32 |
| 4 | 98 |
| 5 | 150 |
| 6 | 25 |
| 7 | 25.1 |

`7 rows selected.`

4.2.4.7 Lecture -7

Arithmetic & logical operations

Expressions

The definition of an expression is simple: An *expression* returns a value. Expression types are very broad, covering different data types such as String, Numeric, and Boolean. In fact, pretty much anything following a clause (`SELECT` or `FROM`, for example) is an expression. In the following example `amount` is an expression that returns the value contained in the `amount` column.

`SELECT amount FROM checks;`

In the following statement `NAME`, `ADDRESS`, `PHONE` and `ADDRESSBOOK` are expressions:

`SELECT NAME, ADDRESS, PHONE
FROM ADDRESSBOOK;`

Now, examine the following expression:

WHERE NAME = 'BROWN'

It contains a condition, NAME = 'BROWN', which is an example of a Boolean expression. NAME = 'BROWN' will be either TRUE or FALSE, depending on the condition =.

Conditions

If you ever want to find a particular item or group of items in your database, you need one or more conditions. Conditions are contained in the WHERE clause. In the preceding example, the condition is

NAME = 'BROWN'

To find everyone in your organization who worked more than 100 hours last month, your condition would be

NUMBEROFGHOURS > 100

Conditions enable you to make selective queries. In their most common form, conditions comprise a variable, a constant, and a comparison operator. In the first example the variable is NAME, the constant is 'BROWN', and the comparison operator is =. In the second example the variable is NUMBEROFGHOURS, the constant is 100, and the comparison operator is >. You need to know about two more elements before you can write conditional queries: the WHERE clause and operators.

The WHERE Clause

The syntax of the WHERE clause is

SYNTAX:

WHERE <SEARCH CONDITION>

SELECT, FROM, and WHERE are the three most frequently used clauses in SQL. WHERE simply causes your queries to be more selective. Without the WHERE clause, the most useful thing you could do with a query is display all records in the selected table(s). For example:

INPUT:

SQL> SELECT * FROM BIKES;

lists all rows of data in the table BIKES.

OUTPUT:

| NAME | FRAMESIZE | COMPOSITION | MILESRIDDEN | TYPE |
|-------------|-----------|--------------|-------------|----------|
| TREK 2300 | 22.5 | CARBON FIBER | 3500 | RACING |
| BURLEY | 22 | STEEL | 2000 | TANDEM |
| GIANT | 19 | STEEL | 1500 | COMMUTER |
| FUJI | 20 | STEEL | 500 | TOURING |
| SPECIALIZED | 16 | STEEL | 100 | MOUNTAIN |
| CANNONDALE | 22.5 | ALUMINUM | 3000 | RACING |

6 rows selected.

If you wanted a particular bike, you could type

INPUT/OUTPUT:

```
SQL> SELECT *
  FROM BIKES
 WHERE NAME = 'BURLEY';
which would yield only one record:
```

| NAME | FRAMESIZE | COMPOSITION | MILESRIDDEN | TYPE |
|--------|-----------|-------------|-------------|--------|
| BURLEY | 22 | STEEL | 2000 | TANDEM |

Operators

Operators are the elements you use inside an expression to articulate how you want specified conditions to retrieve data. Operators fall into six groups: arithmetic, comparison, character, logical, set, and miscellaneous.

Arithmetic Operators

The arithmetic operators are plus (+), minus (-), divide (/), multiply (*), and modulo (%). The first four are self-explanatory. Modulo returns the integer remainder of a division. Here are two examples:

5 % 2 = 1

6 % 2 = 0

INPUT/OUTPUT:

```
SQL> SELECT ITEM, WHOLESALE, WHOLESALE + 0.15
  FROM PRICE;
```

Here the + adds 15 cents to each price to produce the following:

| ITEM | WHOLESALE | WHOLESALE+0.15 |
|----------|-----------|----------------|
| TOMATOES | .34 | .49 |
| POTATOES | .51 | .66 |
| BANANAS | .67 | .82 |
| TURNIPS | .45 | .60 |
| CHEESE | .89 | 1.04 |
| APPLES | .23 | .38 |

6 rows selected.

Type the following:

INPUT/OUTPUT:

```
SQL> SELECT ITEM, WHOLESALE, (WHOLESALE + 0.15) RETAIL
  FROM PRICE;
```

Here's the result:

| ITEM | WHOLESALE | RETAIL |
|----------|-----------|--------|
| TOMATOES | .34 | .49 |
| POTATOES | .51 | .66 |
| BANANAS | .67 | .82 |
| TURNIPS | .45 | .60 |
| CHEESE | .89 | 1.04 |
| APPLES | .23 | .38 |

For example, the query

INPUT/OUTPUT:

**SQL> SELECT ITEM PRODUCE, WHOLESALE, WHOLESALE + 0.25 RETAIL
FROM PRICE;**

renames the columns as follows

| PRODUCE | WHOLESALE | RETAIL |
|----------|-----------|--------|
| TOMATOES | .34 | .59 |
| POTATOES | .51 | .76 |
| BANANAS | .67 | .92 |
| TURNIPS | .45 | .70 |
| CHEESE | .89 | 1.14 |
| APPLES | .23 | .48 |

Minus (-)

Minus also has two uses. First, it can change the sign of a number. You can use the table HILOW to demonstrate this function.

INPUT:

SQL> SELECT * FROM HILOW;

OUTPUT:

| STATE | HIGHTEMP | LOWTEMP |
|-------|----------|---------|
| CA | -50 | 120 |
| FL | 20 | 110 |
| LA | 15 | 99 |
| ND | -70 | 101 |
| NE | -60 | 100 |

INPUT/OUTPUT:

**SQL> SELECT STATE, -HIGHTEMP LOWS, -LOWTEMP HIGHS
FROM HILOW;**

| STATE | LOWS | HIGHS |
|-------|------|-------|
| CA | 50 | -120 |
| FL | -20 | -110 |
| LA | -15 | -99 |
| ND | 70 | -101 |
| NE | 60 | -100 |

INPUT/OUTPUT:

```
SQL> SELECT STATE,
2 HIGTEMP LOWS,
3 LOWTEMP HIGHS,
4 (LOWTEMP - HIGTEMP) DIFFERENCE
5 FROM HILOW;
```

| STATE | LOWS | HIGHS | DIFFERENCE |
|-------|------|-------|------------|
| CA | -50 | 120 | 170 |
| FL | 20 | 110 | 90 |
| LA | 15 | 99 | 84 |
| ND | -70 | 101 | 171 |
| NE | -60 | 100 | 160 |

You can also create a new column, REMAINDER, to hold the values of NUMERATOR % DENOMINATOR:

INPUT/OUTPUT:

```
SQL> SELECT NUMERATOR,
DENOMINATOR,
NUMERATOR%DENOMINATOR REMAINDER
FROM REMAINS;
```

| NUMERATOR | DENOMINATOR | REMAINDER |
|-----------|-------------|-----------|
| 10 | 5 | 0 |
| 8 | 3 | 2 |
| 23 | 9 | 5 |
| 40 | 17 | 6 |
| 1024 | 16 | 0 |
| 85 | 34 | 17 |

6 rows selected.

Precedence

This section examines the use of precedence in a SELECT statement. Using the database PRECEDENCE, type the following:

```
SQL> SELECT * FROM PRECEDENCE;
```

| N1 | N2 | N3 | N4 |
|----|----|----|----|
| 1 | 2 | 3 | 4 |
| 13 | 24 | 35 | 46 |
| 9 | 3 | 23 | 5 |
| 63 | 2 | 45 | 3 |
| 7 | 2 | 1 | 4 |

Use the following code segment to test precedence:

INPUT/OUTPUT:

```
SQL> SELECT
2 N1+N2*N3/N4,
3 (N1+N2)*N3/N4,
4 N1+(N2*N3)/N4
5 FROM PRECEDENCE;
```

| N1+N2*N3/N4 | (N1+N2)*N3/N4 | N1+(N2*N3)/N4 |
|-------------|---------------|---------------|
| 2.5 | 2.25 | 2.5 |
| 31.26087 | 28.152174 | 31.26087 |
| 22.8 | 55.2 | 22.8 |
| 93 | 975 | 93 |
| 7.5 | 2.25 | 7.5 |

Here's an example of NULL: Suppose an entry in the PRICE table does not contain a value for WHOLESALE. The results of a query might look like this:

INPUT:

```
SQL> SELECT * FROM PRICE;
```

OUTPUT:

| ITEM | WHOLESALE |
|----------|-----------|
| TOMATOES | .34 |
| POTATOES | .51 |
| BANANAS | .67 |
| TURNIPS | .45 |
| CHEESE | .89 |
| APPLES | .23 |
| ORANGES | |

Notice that nothing is printed out in the WHOLESALE field position for oranges. The value for the field WHOLESALE for oranges is NULL. The NULL is noticeable in this case because it is in a numeric column. However, if the NULL appeared in the ITEM column, it would be impossible to tell the difference between NULL and a blank.

Try to find the NULL:

INPUT/OUTPUT:

```
SQL> SELECT *
2 FROM PRICE
3 WHERE WHOLESALE IS NULL;
```

| ITEM | WHOLESALE |
|---------|-----------|
| ORANGES | |

Equal (=)

Earlier today you saw how some implementations of SQL use the equal sign in the SELECT clause to assign an alias. In the WHERE clause, the equal sign is the most commonly used comparison operator. Used alone, the equal sign is a very convenient way of selecting one value out of many. Try this:

INPUT:

```
SQL> SELECT * FROM FRIENDS;
```

OUTPUT:

| LASTNAME | FIRSTNAME | AREACODE | PHONE | ST | ZIP |
|----------|-----------|----------|----------|----|-------|
| BUNDY | AL | 100 | 555-1111 | IL | 22333 |
| MEZA | AL | 200 | 555-2222 | UK | |
| MERRICK | BUD | 300 | 555-6666 | CO | 80212 |
| MAST | JD | 381 | 555-6767 | LA | 23456 |
| BULHER | FERRIS | 345 | 555-3223 | IL | 23332 |

Let's find JD's row. (On a short list this task appears trivial, but you may have more friends than we do--or you may have a list with thousands of records.)

INPUT/OUTPUT:

```
SQL> SELECT *
FROM FRIENDS
WHERE FIRSTNAME = 'JD';
```

| LASTNAME | FIRSTNAME | AREACODE | PHONE | ST | ZIP |
|----------|-----------|----------|----------|----|-------|
| MAST | JD | 381 | 555-6767 | LA | 23456 |

Greater Than (>) and Greater Than or Equal To (>=)

The greater than operator (>) works like this:

INPUT:

```
SQL> SELECT *
FROM FRIENDS
WHERE AREACODE > 300;
```

OUTPUT:

| LASTNAME | FIRSTNAME | AREACODE | PHONE | ST | ZIP |
|----------|-----------|----------|---------------|----|-------|
| MAST | JD | | 381 555 -6767 | LA | 23456 |
| BULHER | FERRIS | | 345 555 -3223 | IL | 23332 |

INPUT/OUTPUT:

```
SQL> SELECT *
2 FROM FRIENDS
3 WHERE AREACODE >= 300;
```

| LASTNAME | FIRSTNAME | AREACODE | PHONE | ST | ZIP |
|----------|-----------|----------|---------------|----|-------|
| MERRICK | BUD | | 300 555 -6666 | CO | 80212 |
| MAST | JD | | 381 555 -6767 | LA | 23456 |
| BULHER | FERRIS | | 345 555 -3223 | IL | 23332 |

Less Than (<) and Less Than or Equal To (<=)

As you might expect, these comparison operators work the same way as > and >= work, only in reverse:

INPUT:

```
SQL> SELECT *
2 FROM FRIENDS
3 WHERE STATE < 'LA';
```

OUTPUT:

| LASTNAME | FIRSTNAME | AREACODE | PHONE | ST | ZIP |
|----------|-----------|----------|---------------|----|-------|
| BUNDY | AL | | 100 555 -1111 | IL | 22333 |
| MERRICK | BUD | | 300 555 -6666 | CO | 80212 |
| BULHER | FERRIS | | 345 555 -3223 | IL | 23332 |

Inequalities (< > or !=)

When you need to find everything except for certain data, use the inequality symbol, which can be either < > or !=, depending on your SQL implementation. For example, to find everyone who is not AL, type this:

INPUT:

```
SQL> SELECT *
2 FROM FRIENDS
3 WHERE FIRSTNAME <> 'AL';
```

OUTPUT:

| LASTNAME | FIRSTNAME | AREACODE | PHONE | ST | ZIP |
|----------|-----------|----------|---------------|----|-------|
| MERRICK | BUD | | 300 555 -6666 | CO | 80212 |
| MAST | JD | | 381 555 -6767 | LA | 23456 |
| BULHER | FERRIS | | 345 555 -3223 | IL | 23332 |

To find everyone not living in California, type this:

INPUT/OUTPUT:

```
SQL> SELECT *
2 FROM FRIENDS
3 WHERE STATE != 'CA';
```

| LASTNAME | FIRSTNAME | AREACODE | PHONE | ST | ZIP |
|----------|-----------|----------|----------|----|-------|
| BUNDY | AL | 100 | 555-1111 | IL | 22333 |
| MEZA | AL | 200 | 555-2222 | UK | |
| MERRICK | BUD | 300 | 555-6666 | CO | 80212 |
| MAST | JD | 381 | 555-6767 | LA | 23456 |
| BULHER | FERRIS | 345 | 555-3223 | IL | 23332 |

Logical Operators

logical operators Logical operators separate two or more conditions in the WHERE clause of an SQL statement. Vacation time is always a hot topic around the workplace. Say you designed a table called VACATION for the accounting department:

INPUT:

```
SQL> SELECT * FROM VACATION;
```

OUTPUT:

| LASTNAME | EMPLOYEEENUM | YEARS | LEAVETAKEN |
|----------|--------------|-------|------------|
| ABLE | 101 | 2 | 4 |
| BAKER | 104 | 5 | 23 |
| BLEDSOE | 107 | 8 | 45 |
| BOLIVAR | 233 | 4 | 80 |
| BOLD | 210 | 15 | 100 |
| COSTALES | 211 | 10 | 78 |

6 rows selected.

AND

AND means that the expressions on both sides must be true to return TRUE. If either expression is false, AND returns FALSE. For example, to find out which employees have been with the company for 5 years or less and have taken more than 20 days leave, try this:

INPUT:

```
SQL> SELECT LASTNAME
2 FROM VACATION
3 WHERE YEARS <= 5
```

**4 AND
5 LEAVETAKEN > 20 ;
OUTPUT:**

```
LASTNAME
-----
BAKER
BOLIVAR
```

If you want to know which employees have been with the company for 5 years or more and have taken less than 50 percent of their leave, you could write:

INPUT/OUTPUT:

```
SQL> SELECT LASTNAME WORKAHOLICS
2 FROM VACATION
3 WHERE YEARS >= 5
4 AND
5 ((YEARS *12)-LEAVETAKEN)/(YEARS * 12) < 0.50;
```

```
WORKAHOLICS
-----
BAKER
BLEDSOE
```

Check these people for burnout. Also check out how we used the AND to combine these two conditions.

OR

You can also use OR to sum up a series of conditions. If any of the comparisons is true, OR returns TRUE. To illustrate the difference, conditions run the last query with OR instead of with AND:

INPUT:

```
SQL> SELECT LASTNAME WORKAHOLICS
2 FROM VACATION
3 WHERE YEARS >= 5
4 OR
5 ((YEARS *12)-LEAVETAKEN)/(YEARS * 12) >= 0.50;
```

OUTPUT:

```
WORKAHOLICS
-----
ABLE
BAKER
BLEDSOE
BOLD
COSTALES
```

NOT

NOT means just that. If the condition it applies to evaluates to TRUE, NOT make it FALSE. If the condition after the NOT is FALSE, it becomes TRUE.

For example, the following SELECT returns the only two names not beginning with B in the table:

INPUT:

```
SQL> SELECT *
2 FROM VACATION
3 WHERE LASTNAME NOT LIKE 'B%';
```

OUTPUT:

| LASTNAME | EMPLOYEEENUM | YEARS | LEAVETAKEN |
|----------|--------------|-------|------------|
| ABLE | 101 | 2 | 4 |
| COSTALES | 211 | 10 | 78 |

To find the non-NULL items, type this:

INPUT/OUTPUT:

```
SQL> SELECT *
2 FROM PRICE
3 WHERE WHOLESALE IS NOT NULL;
```

| ITEM | WHOLESALE |
|----------|-----------|
| TOMATOES | .34 |
| POTATOES | .51 |
| BANANAS | .67 |
| TURNIPS | .45 |
| CHEESE | .89 |

4.2.4.8 Lecture-8

SQL functions (Date and Time, Numeric, String conversion).

Date and time functions

We live in a civilization governed by times and dates, and most major implementations of SQL have functions to cope with these concepts. This section uses the table PROJECT to demonstrate the time and date functions.

INPUT:

```
SQL> SELECT * FROM PROJECT;
```

OUTPUT:

| TASK | STARTDATE | ENDDATE |
|---------------|-----------|-----------|
| KICKOFF MTG | 01-APR-95 | 01-APR-95 |
| TECH SURVEY | 02-APR-95 | 01-MAY-95 |
| USER MTGS | 15-MAY-95 | 30-MAY-95 |
| DESIGN WIDGET | 01-JUN-95 | 30-JUN-95 |
| CODE WIDGET | 01-JUL-95 | 02-SEP-95 |
| TESTING | 03-SEP-95 | 17-JAN-96 |

6 rows selected.

ADD_MONTHS

This function adds a number of months to a specified date. For example, say something extraordinary happened, and the preceding project slipped to the right by two months. You could make a new schedule by typing

INPUT:

```
SQL> SELECT TASK,
2 STARTDATE,
3 ENDDATE ORIGINAL_END,
4 ADD_MONTHS(ENDDATE,2)
5 FROM PROJECT;
```

OUTPUT:

| TASK | STARTDATE | ORIGINAL_ | ADD_MONTH |
|---------------|-----------|-----------|-----------|
| KICKOFF MTG | 01-APR-95 | 01-APR-95 | 01-JUN-95 |
| TECH SURVEY | 02-APR-95 | 01-MAY-95 | 01-JUL-95 |
| USER MTGS | 15-MAY-95 | 30-MAY-95 | 30-JUL-95 |
| DESIGN WIDGET | 01-JUN-95 | 30-JUN-95 | 31-AUG-95 |
| CODE WIDGET | 01-JUL-95 | 02-SEP-95 | 02-NOV-95 |
| TESTING | 03-SEP-95 | 17-JAN-96 | 17-MAR-96 |

6 rows selected.

Not that a slip like this is possible, but it's nice to have a function that makes it so easy. ADD_MONTHS also works outside the SELECT clause. Typing

INPUT:

```
SQL> SELECT TASKS_SHORTER_THAN_ONE_MONTH
2 FROM PROJECT
3 WHERE ADD_MONTHS(STARTDATE,1) > ENDATE;
```

produces the following result:

OUTPUT:

| TASKS_SHORTER_THAN_ONE_MONTH |
|------------------------------|
| KICKOFF MTG |
| TECH SURVEY |
| USER MTGS |
| DESIGN WIDGET |

LAST_DAY

LAST_DAY returns the last day of a specified month. It is for those of us who haven't mastered the "Thirty days has September..." rhyme--or at least those of us who have not yet taught it to our computers. If, for example, you need to know what the last day of the month is in the column ENDDATE, you would type

INPUT:

```
SQL> SELECT ENDDATE, LAST_DAY(ENDDATE)
2 FROM PROJECT;
```

Here's the result:

OUTPUT:

```
ENDDATE      LAST_DAY(ENDDATE)
----- -----
01-APR-95    30-APR-95
01-MAY-95    31-MAY-95
30-MAY-95    31-MAY-95
30-JUN-95    30-JUN-95
02-SEP-95    30-SEP-95
17-JAN-96    31-JAN-96
```

```
6 rows selected.
```

MONTHS_BETWEEN

If you need to know how many months fall between month x and month y, use MONTHS_BETWEEN like this:

INPUT:

```
SQL> SELECT TASK, STARTDATE,
ENDDATE,MONTHS_BETWEEN(STARTDATE,ENDDATE)
DURATION
2 FROM PROJECT;
```

OUTPUT:

```
TASK          STARTDATE  ENDDATE      DURATION
-----        -----
KICKOFF MTG   01-APR-95  01-APR-95      0
TECH SURVEY   02-APR-95  01-MAY-95  -.9677419
USER MTGS    15-MAY-95  30-MAY-95  -.483871
DESIGN WIDGET 01-JUN-95  30-JUN-95  -.9354839
CODE WIDGET   01-JUL-95  02-SEP-95 -2.032258
TESTING       03-SEP-95 17-JAN-96  -4.451613
```

```
6 rows selected.
```

NEXT_DAY

NEXT_DAY finds the name of the first day of the week that is equal to or later than another specified date. For example, to send a report on the Friday following the first day of each event, you would type

INPUT:

```
SQL> SELECT STARTDATE,  
2 NEXT_DAY(STARTDATE, 'FRIDAY')  
3 FROM PROJECT;
```

which would return

OUTPUT:

```
STARTDATE NEXT_DAY(  
----- -----  
01-APR-95 07-APR-95  
02-APR-95 07-APR-95  
15-MAY-95 19-MAY-95
```

```
01-JUN-95 02-JUN-95  
01-JUL-95 07-JUL-95  
03-SEP-95 08-SEP-95
```

```
6 rows selected.
```

SYSDATE

SYSDATE returns the system time and date:

INPUT:

```
SQL> SELECT DISTINCT SYSDATE  
2 FROM PROJECT;
```

OUTPUT:

```
SYSDATE  
-----  
18-JUN-95 1020PM
```

Conversion Functions

These three conversion functions provide a handy way of converting one type of data to another. These examples use the table CONVERSIONS.

INPUT:

```
SQL> SELECT * FROM CONVERSIONS;
```

OUTPUT:

| NAME | TESTNUM |
|------|---------|
| 40 | 95 |
| 13 | 23 |
| 74 | 68 |

TO_CHAR

The primary use of TO_CHAR is to convert a number into a character. Different implementations may also use it to convert other data types, like

Date, into a character, or to include different formatting arguments. The next example illustrates the primary use of TO_CHAR:

INPUT:

```
SQL> SELECT TESTNUM, TO_CHAR(TESTNUM)
2 FROM CONVERT;
```

OUTPUT:

| TESTNUM | TO_CHAR (TESTNUM) |
|---------|-------------------|
| 95 | 95 |
| 23 | 23 |
| 68 | 68 |

Not very exciting, or convincing. Here's how to verify that the function returned a character string:

INPUT:

```
SQL> SELECT TESTNUM, LENGTH(TO_CHAR(TESTNUM))
2 FROM CONVERT;
```

OUTPUT:

| TESTNUM | LENGTH (TO_CHAR (TESTNUM)) |
|---------|-----------------------------|
| 95 | 2 |
| 23 | 2 |

TO_NUMBER

TO_NUMBER is the companion function to TO_CHAR, and of course, it converts a string into a number. For example:

INPUT:

```
SQL> SELECT NAME, TESTNUM, TESTNUM*TO_NUMBER(NAME)
2 FROM CONVERT;
```

OUTPUT:

| NAME | TESTNUM | TESTNUM*TO_NUMBER (NAME) |
|------|---------|--------------------------|
| 40 | 95 | 3800 |
| 13 | 23 | 299 |
| 74 | 68 | 5032 |

4.2.5. Test Questions

a) Fill in the blanks type of questions:

1. A _____ is a set of data elements that is organized using a model of vertical columns and horizontal rows.

2. A _____ is a set of data values of a particular simple type, one for each row of the table.
3. A _____ represents a single, data item in a table.
4. A _____ is a unique value that identifies a row in a table.
5. _____ are used to identify which type of data we are going to store in the database.
6. Types of languages used for creating and manipulating the data in the Database are _____ & _____.
7. A _____ is a standard for commands that define the different structures in a database.
8. A _____ is a part of DML involving information retrieval only.
9. Common DDL statements are _____, _____ and _____ .
10. A popular data manipulation language is _____.

b) Multiple choice questions

1. You can add a row using SQL in a database with which of the following?

- | | |
|--------------------------|--------------------------|
| A.ADD | B.CREATE |
| C.INSERT | D.MAKE |

Ans C

2. The command to remove rows from a table 'CUSTOMER' is:

- [A.REMOVE FROM CUSTOMER ...](#)

- [B.DROP FROM CUSTOMER ...](#)

- [C.DELETE FROM CUSTOMER WHERE ...](#)

- [D.UPDATE FROM CUSTOMER ...](#)

Ans: C

3. The SQL WHERE clause:

- [A.limits the column data that are](#)

returned.

- [B.](#) limits the row data are returned.
- [C.](#) Both A and B are correct.
- [D.](#) Neither A nor B are correct.

Ans:B

4. The wildcard in a WHERE clause is useful when?

- An exact match is necessary in a
- [A.](#) SELECT statement.
- An exact match is not possible in a
- [B.](#) SELECT statement.
- An exact match is necessary in a
- [C.](#) CREATE statement.
- An exact match is not possible in a
- [D.](#) CREATE statement.

Ans: B

5. the command to eliminate a table from a database is:

- [A.](#) REMOVE TABLE CUSTOMER;
- [B.](#) DROP TABLE CUSTOMER;
- [C.](#) DELETE TABLE CUSTOMER;
- [D.](#) UPDATE TABLE CUSTOMER;

Ans: B

6. SQL data definition commands make up a(n) _____ .

- [A.](#) DDL
- [B.](#) DML
- [C.](#) HTML
- [D.](#) XML

Ans: A

7. The SQL keyword(s) _____ is used with wildcards.
- A.LIKE only
B.IN only
C.NOT IN only
D.IN and NOT IN

Ans: A

8. The result of a SQL SELECT statement is a(n) _____ .
- A.report B.form
C.file D.table

Ans: D

9. In the relational modes, cardinality is termed as:
- (A) Number of tuples.
(B) Number of attributes.
(C) Number of tables.
(D) Number of constraints.

Ans: A

10. Architecture of the database can be viewed as
- (A) two levels.
(B) four levels.
(C) three levels.
(D) one level.

Ans: C

(c)True or False questions

1. The condition in a WHERE clause can refer to only one value.
- A.True B.False

Ans: B

2. SQL provides the AS keyword, which can be used to assign meaningful column names to the results of queries using the SQL built-in functions.
- A.True B.False

Ans: A

3. The SELECT command, with its various clauses, allows users to query the data contained in the tables and ask many different questions or ad hoc queries.

[A.True](#) [B.False](#)

Ans: A

4. A SELECT statement within another SELECT statement and enclosed in square brackets [...] is called a subquery.

[A.True](#) [B.False](#)

Ans: B

5. The rows of the result relation produced by a SELECT statement can be sorted, but only by one column.

[A.True](#) [B.False](#)

Ans: B

6. SQL is a programming language.

[A.True](#) [B.False](#)

Ans: B

7. Most companies keep at least two versions of any database they are using.

[A.True](#) [B.False](#)

Ans: A

8. The SQL statement: SELECT Number1 + Number 2 AS Total FROM NUMBER_TABLE; adds two numbers from each row together and lists the results in a column named Total.

[A.True](#) [B.False](#)

Ans: A

9. In a relation, the rows are sometimes called "records."

[A.True](#) [B.False](#)

Ans: A

10. In a relation, the columns are sometimes called "attributes."

[A.True](#) [B.False](#)

Ans: A

4.2.6. Review Questions

d) Objective type of questions (Very short notes)

1. Explain entity integrity and referential integrity rules in relational model. Show how these are realized in SQL.

Ans: Entity Integrity Rule –No primary key value can be null.

Referential Integrity Rule –In referential integrity, it is ensured that a value that appears in one relation for a given set of attributes also appears for a certain set of

attributes in another relation. In SQL, entity integrity and referential integrity rules are implemented as constraints on the relation called as primary key constraint and reference key constraint respectively.

These constraints can be specified with relation at the time of creation of the relations or after the creation of the relations by altering the definition of the relations. For example:

```
CREATE TABLE DEPT
```

```
(DEPTNO NUMBER PRIMARY KEY, DNAME VARCHAR2(15));
```

```
CREATE TABLE EMP (EMPNO NUMBER PRIMARY KEY, ENAME VARCHAR2(15),  
JOB VARCHAR2(10), DEPTNO NUMBER REFERENCES DEPT(DEPTNO));
```

2. Consider the following relations:

S (S#, SNAME, STATUS, CITY)

SP (S#, P#, QTY)

P (P#, PNAME, COLOR, WEIGHT, CITY)

Give an expression in SQL for each of queries below:

(i)Get supplier names for supplier who supply at least one red part

(ii)Get supplier names for supplier who do not supply part P2.

Ans:

(i) SELECT SNAME FROM S WHERE S# IN (SELECT S# FROM SP
WHERE P# IN (SELECT P# FROM P WHERE COLOR = RED')) ;

(ii) SELECT SNAME FROM S WHERE S# NOT IN (SELECT S# FROM SP WHERE
P# = 'P2');

3. Give an expression in SQL for each of queries below:

(i)Find the names of all employees who work for first Bank Corporation.

(ii)Find the names and company names of all employees sorted in
ascending order of company name and descending order of employee
names of that company.

(iii)Change the city of First Bank Corporation to 'New Delhi'

Ans:

(i)SELECT EMPLOYEE_NAME FROM WORKS
WHERE COMPANYNAME = 'First Bank Corporation';

(ii)SELECT EMPLOYEE_NAME, COMPANYNAME FROM WORKS
ORDER BY COMPANYNAME, EMPLOYEE_NAME DESC;

(iii)UPDATE COMPANY SET CITY = 'New Delhi'
WHERE COMPANY_NAME = 'First Bank Corporation';

4. Explain the concepts of relational data model. Also discuss its
advantages and disadvantages.

Ans: Relational Data Model – The relational model was first introduced by

Prof. E.F. Codd of the IBM Research in 1970 and attracted immediate attention due to its simplicity and D C mathematical foundation. The model uses the concept of a mathematical relation (like a table of values) as its basic building block, and has its theoretical basis in set theory and first-order predicate logic. The relational model represents the database as a collection of relations. The relational model like all other models consists of three basic components: a set of domains and a set of relations operation on relations integrity rules

Advantages

- Ease of use
- Flexibility
- Security
- Data Independence
- Data Manipulation Language

Disadvantages

- Performance
- Unsuitable for Hierarchies

5. Consider the relational schema:

employee (person_name, street, city)
works (person_name, company_name, salary)
company (Company_name, city)
manager (person_name, manager_name) Where primary keys are underlined.
Give an expression in SQL to express each of the following query:

- (a) Find the name of all employees in this database who do not work for "Yes Bank".
- (b) Find the name of all employees who earn more than every employee of "AXIS Bank".

6. What are DML and DDL statements?

Ans:

DML stands for Data Manipulation Statements. They update data values in table. Below are the

most important DDL statements:-

=>SELECT - gets data from a database table

=> UPDATE - updates data in a table

=> DELETE - deletes data from a database table

=> INSERT INTO - inserts new data into a database table

DDL stands for Data definition Language. They change structure of the database objects like table, index etc. Most important DDL statements are as shown below:-

=>CREATE TABLE - creates a new table in the database.
=>ALTER TABLE – changes table structure in database.
=>DROP TABLE - deletes a table from database
=> CREATE INDEX - creates an index
=> DROP INDEX - deletes an index

7. How do we select distinct values from a table?

Ans:

DISTINCT keyword is used to return only distinct values. Below is syntax:-

Column age and

Table pcdsEmp

```
SELECT DISTINCT age FROM pcdsEmp
```

8. What is Like operator for and what are wild cards?

Ans:

LIKE operator is used to match patterns. A "%" sign is used to define the pattern.

Below SQL statement will return all words with letter "S"

```
SELECT * FROM pcdsEmployee WHERE EmpName LIKE 'S%'
```

Below SQL statement will return all words which end with letter "S"

```
SELECT * FROM pcdsEmployee WHERE EmpName LIKE '%S'
```

Below SQL statement will return all words having letter "S" in between

```
SELECT * FROM pcdsEmployee WHERE EmpName LIKE '%S%
```

"_" operator (we can read as "Underscore Operator"). "_" operator is the character defined at

that point. In the below sample fired a query Select name from pcdsEmployee where name like
'_s%' So all name where second letter is "s" is returned.

9. Can you explain Insert, Update and Delete query?

Ans:

Insert statement is used to insert new rows in to table. Update to update existing data in the

table. Delete statement to delete a record from the table. Below code snippet for Insert, Update and Delete.

```
INSERT INTO pcdsEmployee SET name='rohit',age='24';
```

```
UPDATE pcdsEmployee SET age='25' where name='rohit';
```

```
DELETE FROM pcdsEmployee WHERE name = 'sonia';
```

10. What's the difference between DELETE and TRUNCATE ?

Ans:

Following are difference between them:

=>>DELETE TABLE syntax logs the deletes thus making the delete operations low. TRUNCATE table does not log any information but it logs information about deallocation of data page of the table. So TRUNCATE table is faster as compared to delete table.

=>>DELETE table can have criteria while TRUNCATE cannot.
=>> TRUNCATE table cannot have triggers.

e) Analytical type questions

1. Explain in detail how you resolved the problem by Data Analysis with SQL

Ans:

Today information management systems along with operational applications need to support a wide variety of business requirements that typically involve some degree of analytical processing. These requirements can range from data enrichment and transformation during ETL workflows, creating time-based calculations like moving average and moving totals for sales reports, performing real-time pattern searches within logs files to building what-if data models during budgeting and planning exercises. Developers, business users and project teams can choose from a wide range of languages to create solutions to meet these requirements.

Over time many companies have found that the use so many different programming languages to drive their data systems creates five key problems:

1. Decreases the ability to rapidly innovate
2. Creates data silos
3. Results in application-level performance bottlenecks that are hard to trace and rectify
4. Drives up costs by complicating the deployment and management processes
5. Increases the level of investment in training

Development teams need to quickly deliver new and innovative applications that provide significant competitive advantage and drive additional revenue streams. Anything that stifles innovation needs to be urgently reviewed and resolved. The challenge facing many organizations is to find the right

platform and language to securely and efficiently manage the data and analytical requirements while at the same time supporting the broadest range of tools and applications to maximize the investment in existing skills.

IT and project managers need an agile platform to underpin their projects and applications so that developers can quickly and effectively respond to ever-changing business requirements without incurring the issues listed above.

2. Illustrating how you transformed ideas into practical solutions, and how you followed up with database analytical SQL with Oracle Database

Ans:

Processing concepts behind analytical SQL

Oracle's in-database analytical SQL – first introduced in Oracle Database 8i Release 1- has introduced several new elements to SQL processing. These elements build on existing SQL features to provide developers and business users with a framework that is both flexible and powerful in terms of its ability to support sophisticated calculations. There are four essential concepts used in the processing of Oracle's analytic SQL:

- Processing order
- Result set partitions
- Calculation windows
- Current Row

This four-step process is internally optimized and completely transparent but it does provide a high degree of flexibility in terms of being able to layer analytical features to create the desired result set without having to resort to long and complicated SQL statements. The following sections will explore these four concepts in more detail.

1. Processing order

Query processing using analytic SQL takes place in three stages:

Stage 1: All joins, WHERE, GROUP BY and HAVING clauses¹ are performed. Where customers are using Exadata storage servers the initial join and filtering operations for the query will be managed by the storage cells. This step can result in a significant reduction in the volume of data passed to the analytic function, which helps improve performance.

Stage 2: The result set is made available to the analytic function, and all the calculations are applied.

Stage 3: If the query has an ORDER BY clause then this is processed to allow for precise control of the final output.

2. Partitions – organizing your data sets

Analytic SQL allows users to divide query result sets into ordered groups of rows called "partitions"². Any aggregated results such as SUM's, AVG's etc. are available to the analytical functions. Partitions can be based upon any column(s) or expression. A query result set may have just one partition holding all the rows, a few large partitions, or many small partitions with each holding just a few rows.

3. Calculation windows

Within each partition, a sliding window of data can be defined. The window determines the range of rows used to perform the calculations for the "current row" (defined in the next section). Window sizes can be based on either a physical number of rows or a logical interval such as time.

The window has a starting row and an ending row. Depending on its definition, the window may move at one or both ends.

For instance, a window defined for a cumulative sum function would have its starting row fixed at the first row of its window, and its ending row would slide from the starting point all the way to the last row of the window.

SELECT

Qtrs

```

, Months
, Channels
, Revenue
, SUM(Revenue) OVER (PARTITION BY Qtrs) AS Qtr_Sales
, SUM(Revenue) OVER () AS Total_Sales
FROM sales_table

```

In contrast, a window defined for a moving average would have both its starting and end points slide so that they

| QTRS | MONTHS | CHANNELS | REVENUE | QTR_SALES | TOTAL_SALES |
|---------|---------|--------------|------------|------------|-------------|
| 1998-01 | 1998-01 | Direct Sales | 1630685.07 | 6480684 | 12074678.14 |
| 1998-01 | 1998-01 | Internet | 222086.08 | 6480684 | 12074678.14 |
| 1998-01 | 1998-01 | Partners | 424649.34 | 6480684 | 12074678.14 |
| 1998-01 | 1998-02 | Direct Sales | 1627680 | 6480684 | 12074678.14 |
| 1998-01 | 1998-02 | Internet | 267127.38 | 6480684 | 12074678.14 |
| 1998-01 | 1998-02 | Partners | 477883.49 | 6480684 | 12074678.14 |
| 1998-01 | 1998-03 | Direct Sales | 1257087.65 | 6480684 | 12074678.14 |
| 1998-01 | 1998-03 | Internet | 205005.52 | 6480684 | 12074678.14 |
| 1998-01 | 1998-03 | Partners | 315084.1 | 6480684 | 12074678.14 |
| 1998-01 | 1998-03 | Tele Sales | 53395.37 | 6480684 | 12074678.14 |
| 1998-02 | 1998-04 | Direct Sales | 1313683.91 | 5593994.14 | 12074678.14 |
| 1998-02 | 1998-04 | Internet | 223332.56 | 5593994.14 | 12074678.14 |
| 1998-02 | 1998-04 | Partners | 396249.81 | 5593994.14 | 12074678.14 |
| 1998-02 | 1998-04 | Tele Sales | 42712.02 | 5593994.14 | 12074678.14 |
| 1998-02 | 1998-05 | Direct Sales | 1228571.99 | 5593994.14 | 12074678.14 |
| 1998-02 | 1998-05 | Partners | 321134.55 | 5593994.14 | 12074678.14 |
| 1998-02 | 1998-06 | Direct Sales | 1243938.66 | 5593994.14 | 12074678.14 |
| 1998-02 | 1998-06 | Internet | 232485.87 | 5593994.14 | 12074678.14 |
| 1998-02 | 1998-06 | Partners | 393304.08 | 5593994.14 | 12074678.14 |

maintained a constant physical or logical range. The example below demonstrates the use of physical window to create a 6-month moving average:

SELECT

```

calendar_month_desc
as "Month",
amount_sold as "Sales"

```

```

AVG(amount_sold) OVER(ORDER BY calendar_month_desc rows
between 5 preceding and current row) as "6M_Avg_Sales"

```

FROM ...

Alternatively, if the data set contains a date column then it is possible to use logical windows by taking advantage of Oracle's built-in time awareness. In this case we can use the "range interval '5' month preceding" syntax to create the 6-month moving average:

SELECT

```

calendar_month_desc as "Month"

```

```

, amount_sold as "Sales"

```

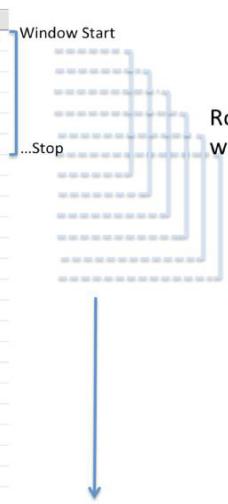
```

, AVG(amount_sold) OVER(ORDER BY calendar_month_desc range

```

interval '5' month preceding) as "6M_Avg_Sales"
 FROM ...

The output from these queries is shown here:



The diagram shows a table of monthly sales data from 1998-01 to 1999-10. A vertical double-headed arrow labeled 'Window Start' points to the first row (1998-01). A vertical double-headed arrow labeled '...Stop' points to the last row (1999-10). A large blue bracket labeled 'Rolling 6-month window' spans six rows at a time, starting from the 'Window Start' and ending at the '...Stop' point. The window moves sequentially down the table.

| | Month | Sales | 6M_Avg_Sales |
|----|---------|------------|--------------|
| 1 | 1998-01 | 2277420.49 | 2277420.49 |
| 2 | 1998-02 | 2372690.87 | 2325055.68 |
| 3 | 1998-03 | 1830572.64 | 2160228.00 |
| 4 | 1998-04 | 1975978.3 | 2114165.58 |
| 5 | 1998-05 | 1748287.23 | 2040989.91 |
| 6 | 1998-06 | 1869728.61 | 2012446.36 |
| 7 | 1998-07 | 1932282.28 | 1954923.32 |
| 8 | 1998-08 | 1972532.63 | 1888230.28 |
| 9 | 1998-09 | 2167008.19 | 1944302.87 |
| 10 | 1998-10 | 2236464.53 | 1987717.25 |
| 11 | 1998-11 | 1959664.22 | 2022946.74 |
| 12 | 1998-12 | 1741284.96 | 2001539.47 |
| 13 | 1999-01 | 2077439.76 | 2025732.38 |
| 14 | 1999-02 | 2357629.26 | 2089915.15 |
| 15 | 1999-03 | 1658678.19 | 2005193.49 |
| 16 | 1999-04 | 1573272.66 | 1894661.51 |
| 17 | 1999-05 | 1711727.87 | 1853338.78 |
| 18 | 1999-06 | 1640471.1 | 1836536.47 |
| 19 | 1999-07 | 1891215.57 | 1805499.11 |
| 20 | 1999-08 | 1904916.61 | 1730047.00 |
| 21 | 1999-09 | 2030917.97 | 1792086.96 |
| 22 | 1999-10 | 1722615.48 | 1816977.43 |
| 23 | 1999-11 | 1710127.18 | 1818211.40 |

The concept of a window is very powerful and provides a lot of flexibility in terms of being able to interact with the data. A window can be set as large as all the rows in a partition. At the other extreme it could be just a single row. Users may specify a window containing a constant number of rows, or a window containing all rows where a column value is in a specified numeric range. Windows may also be defined to hold all rows where a date value falls within a certain time period, such as the prior month.

4. Current Row

Each calculation performed with an analytic function is based on a current row within a window. The current row serves as the reference point determining the start and end of the window. In the example below the calculation of a running total would be the result of the current row plus the values from the preceding two rows. At the end of the window the running total will be reset. The example shown below creates running totals within a result set showing the total sales for each channel within a product category within year:

```

SELECT
    calenda
    r_year
    , prod_category_desc

```

```

    , channel_desc
    , country_name
    , sales
    , units
    , SUM(sales) OVER (PARTITION BY calendar_year,
prod_category_desc, channel_desc order by
country_name) sales_tot_cat_by_channel

FROM ...

```

| CALENDAR_YEAR | PROD_CATEGORY_DESC | CHANNEL_DESC | COUNTRY_NAME | SALES | UNITS | SALES_TOT_CAT |
|--|--------------------|--------------|--------------------------|-----------|-------|---------------|
| 1998 | Electronics | Internet | United Kingdom | 8312.73 | 15 | 66238.19 |
| 1998 | Electronics | Internet | United States of America | 89978.66 | 162 | 156216.85 |
| 1998 | Electronics | Partners | Australia | 12741.75 | 50 | 12741.75 |
| 1998 | Electronics | Partners | Brazil | 544.49 | 1 | 13286.24 |
| 1998 | Electronics | Partners | Canada | 9675.24 | 54 | 22961.48 |
| 1998 | Electronics | Partners | Denmark | 4075.33 | 29 | 27036.81 |
| Current row: calculations based on window contents | | | | | | |
| 1998 | Electronics | Partners | Germany | 21610.7 | 106 | 55794.57 |
| 1998 | Electronics | Partners | Italy | 12450.01 | 58 | 68244.58 |
| 1998 | Electronics | Partners | Japan | 24194.48 | 133 | 92439.06 |
| 1998 | Electronics | Partners | Singapore | 20055.11 | 82 | 112494.17 |
| 1998 | Electronics | Partners | Spain | 5581.34 | 26 | 118075.51 |
| 1998 | Electronics | Partners | United Kingdom | 15184.35 | 93 | 133259.86 |
| 1998 | Electronics | Partners | United States of America | 180014.84 | 928 | 313274.70 |
| 1998 | Electronics | Tele Sales | Argentina | 47.94 | 6 | 47.94 |
| 1998 | Electronics | Tele Sales | Brazil | 39.95 | 5 | 87.89 |
| 1998 | Electronics | Tele Sales | Denmark | 7.99 | 1 | 95.88 |
| 1998 | Electronics | Tele Sales | France | 7.99 | 1 | 103.87 |
| 1998 | Electronics | Tele Sales | Germany | 47.94 | 6 | 151.81 |
| 1998 | Electronics | Tele Sales | New Zealand | 7.99 | 1 | 159.80 |
| 1998 | Electronics | Tele Sales | Poland | 7.99 | 1 | 167.79 |
| 1998 | Electronics | Tele Sales | United States of America | 1530.09 | 191 | 1697.88 |
| 1998 | Hardware | Direct Sales | Australia | 205882.38 | 144 | 205882.38 |
| 1998 | Hardware | Direct Sales | France | 100.00 | 1 | 100.00 |
| 1998 | Hardware | Direct Sales | Germany | 78926.81 | 53 | 78926.81 |
| 1998 | Hardware | Direct Sales | United Kingdom | 55160.07 | 60 | 55160.07 |
| 1998 | Hardware | Direct Sales | United States of America | 360811.00 | 111 | 360811.00 |

f) Essay type Questions

1. Explain about Database Languages?

Ans:

In many DBMSs where no strict separation of levels is maintained, one language, called the data definition language (DDL), is used by the DBA and by database designer's to define both schemas. In DBMSs where a clear separation is maintained between the conceptual and internal levels, the DDL is used to specify the conceptual schema only. Another language, the storage definition language (SDL), is used to specify the internal schema.

The mappings between the two schemas may be specified in either one of these languages. For a true three-schema architecture a third language,

the view definition language (VDL), to Specify user views, and their mappings to the conceptual schema, but in most DBMSs the DDL is used to define both conceptual and external schemas. Once the database schemas are complied and the database is populated with data, users must have some means to manipulate the database. The DBMS provides a set of operations or a language called the data manipulation language (DML) for manipulations include retrieval, insertion, deletion, and modification of the data.

The Data Definition Language (DDL):

A language that allows the DBA or user to describe and name the entities, attributes, and relationships required for the application, together with any associated integrity and security constraints is called DDL. The storage structure and access methods used by the database system by a set of statements in a special type of DDL called a data storage and definition language. These statements define the implementation details of the database schemas, which are usually hidden from the users. The data values stored in the database must satisfy certain consistency constraints. The DDL provides facilities to specify the following constraints. The database systems check these constraints every time the database is updated.

Domain Constraints:

A domain of possible values must be associated with every attribute. Domain constraints are the most elementary form integrity constraint. They are tested easily by the system whenever a new data item is entered into the database. Referential Integrity There is cases to ensure that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation. Assertions an assertion is any condition that the database must always satisfy. Domain constraints and referential integrity constraints are special forms of assertions. When an assertion is created, the system tests it for validity. If the assertion is valid then any Future modification to the database is allowed only if it does not cause that assertion to be violated. Authorization Read authorization, which allows reading, but not modification of data. Insert authorization, which allows insertion of new data, but not modification of existing data. Update authorization, which allows modification, but not deletion, of data. Delete authorization, which allows deletion of data. We may assign the user all none or a combination of these types of authorization. The output of the DDL is placed in the data dictionary, which contains metadata that is, data about data.

The Data Manipulation Language (DML)

DML is a language that provides a set of operations to support the

basic data manipulation operations on the data held in the database. Data Manipulation operations usually include the following:

- Insertion of new data into the database
- Modification of data stored in the database
- Retrieval of data contained in the database
- Deletion of data from the database

Data manipulation applied to the external, conceptual and internal level. The part of a DML that involves data retrieval is called a Query language. A Query is a statement requesting the retrieval of information. There are basically two types of DML

- Procedural DMLs.
- Declarative DMLs (or) nonprocedural DMLs.

Procedural DML: A language that allows the user to tell the system what data is needed and exactly how to retrieve the data. Non Procedural DML: A language that allows the user to state what data is needed rather than how it is to be retrieved.

2. Explain about Relational Databases

Ans:

A relational database is based on the relational model and uses a collection of tables to represent both data and the relationships among those data. It also includes a DML and DDL. The relational model is an example of a record-based model. Record-based models are so named because the database is structured in fixed-format records of several types. A relational database consists of a collection of tables, each of which is assigned a unique name. A row in a table represents a relationship among a set of values. A table is an entity set, and a row is an entity. Example: a simple relational database.

Department database Dept:

| Dept no | Deptname | Budget |
|---------|-------------|--------|
| D1 | Marketing | 10M |
| D2 | Development | 12M |
| D3 | Research | 5M |

Employee database: Emp

| Empno | Empname | Deptno | Salary |
|-------|---------|--------|--------|
| E1 | John | D1 | 40K |
| E2 | Peter | D1 | 42K |
| E3 | David | D2 | 30K |
| E4 | Sachin | D2 | 35K |

Columns in relations (table) have associated data types. The relational model includes an open-ended set of data types, i.e. users will be able to define their own types as

well as being able to
use system-defined or built in types. Every relation value has two pairs

- 1) A set of column-name: type-name pairs.
- 2) A set of rows The optimizer is the system component that determines how to implement user requests.

The process of navigating around the stored data in order to satisfy the user's request is performed automatically by the system, not manually by the user. For this reason, relational systems are sometimes said to perform automatic navigation. Every DBMS must provide a catalog or dictionary function. The catalog is a place where all of the various schemas (external, conceptual, internal) and all of the corresponding mappings (external/conceptual, conceptual/internal) are kept. In other words, the catalog contains detailed information (sometimes called descriptor information or metadata) regarding the various objects that are of interest to the system itself. Example: Relation variables, indexes, users, integrity constraints, security constraints, and so on. The catalog itself consists of relvars. (System relvars). The catalog will typically include two system relvars called TABLE and COLUMN. The purpose of which is to describe the tables in the database and the columns in those tables.

3. Consider the following relational database:

STUDENT (name, student#, class, major)
COURSE (course name, course#, credit hours, department)
SECTION (section identifier, course#, semester, year, instructor)
GRADE_REPORT (student#, section identifier, grade)
PREREQUISITE (course#, prequisite#)

Specify the following queries in SQL on the above database schema.

- (i) Retrieve the names of all students majoring in 'CS'(Computer Science).
- (ii) Retrieve the names of all courses taught by Professor King in 1998
- (iii) Delete the record for the student whose name is 'Smith' and whose student number is 17.
- (iv) Insert a new course <'Knowledge Engineering', 'CS4390', 3, 'CS'>

Ans:

(i) SELECT NAME FROM STUDENT WHERE MAJOR = 'CS'

(ii) SELECT COURSE_NAME FROM COURSE C, SECTION S
WHERE C.COURSE# = S.COURSE# AND INSTRUCTOR = 'KING' AND YEAR =
1998

OR

SELECT COURSE_NAME FROM COURSE

WHERE COURSE# IN (SELECT COURSE# FROM SECTION
WHERE INSTRUCTOR = 'KING' AND YEAR = 1998)

(iii)DELETE FROM STUDENT WHERE NAME = 'Smith' AND STUDENT# = 17

(iv)INSERT INTO COURSE VALUES('Knowledge Engineering', 'CS4390', 3, 'CS')

4. Explain the concept of a data model. What data models are used in database management systems?

Ans:

Data Model –Model is an abstraction process that hides irrelevant details while highlighting details relevant to the applications at hand. Similarly, a data model is a collection of concepts that can be used to describe structure of a database and provides the necessary means to achieve this abstraction. Structure of database means the data types, relationships, and constraints that should hold for the data. In general a data model consists of two elements:

- A mathematical notation for expressing data and relationships.
- Operations on the data that serve to express queries and other manipulations of the data

Data Models used in DBMSs:

Hierarchical Model -

It was developed to model many types of hierarchical

Data Models used in DBMSs:

Hierarchical Model - It was developed to model many types of hierarchical organizations that exist in the real world. It uses tree structures to represent relationship among records. In hierarchical model, no dependent record can occur without its parent record occurrence and no dependent record occurrence may be connected to more than one parent record occurrence.

Network Model - It was formalized in the late 1960s by the Database Task Group of the Conference on Data System Language (DBTG/CODASYL). It uses two different data structures to represent the database entities and relationships between the entities, namely record type and set type. In the network model, the relationships as well as the navigation through the database are predefined at database creation time.

Relational Model - The relational model was first introduced by E.F. Codd of the IBM Research in 1970. The model uses the concept of a mathematical relation (like a table of values) as its basic building block, and has its theoretical basis in set theory and first-order predicate logic. The relational model represents the database as a collection of relations.

Object Oriented Model –This model is based on the object-oriented programming language paradigm. It includes the features of OOP like inheritance, object-identity, encapsulation, etc. It also supports a rich type system, including structured and collection types.

Object Relational Model –This model combines the features of both

relational model and object oriented model. It extends the traditional relational model with a variety of features such as structured and collection types.

5. Given the following relations

TRAIN (NAME, START, DEST)

TICKET (PNRNO., START, DEST, FARE)

PASSENGER (NAME, ADDRESS, PNRNO.)

Write SQL expressions for the following queries:

Note: Assume NAME of Train is a column of Ticket.

- (i) List the names of passengers who are travelling from the start to the destination station of the train.
- (ii) List the names of passengers who have a return journey ticket.
- (iii) Insert a new Shatabdi train from Delhi to 'Bangalore'.
- (iv) Cancel the ticket of Tintin.

Ans:

(i) SELECT P.NAME FROM TRAIN T, TICKET I, PASSENGER P
WHERE P.PNRNO = I.PNRNO AND T.NAME = I.NAME AND T.START =
I.START AND T.DEST = I.DEST

(ii) SELECT NAME FROM PASSENGER
WHERE PNRNO IN (SELECT DISTINCT A.PNRNO FROM TICKET A, TICKET B
WHERE A.PNRNO = B.PNRNO AND A.START = B.DEST AND A.DEST =
B.START)

(iii) INSERT INTO TRAIN VALUES('Shatabdi', 'Delhi', 'Bangalore')

(iv) DELETE FROM TICKET
WHERE PNRNO = (SELECT PNRNO FROM PASSENGER WHERE NAME =
'Tintin')

- 6. Explain the three data models namely relational, network and hierarchical and compare their relative advantages and disadvantages.**

Ans:

Hierarchical Model: In hierarchical model, data elements are connected to one another through links. Records are arranged in a top-down structure that resembles a tree or genealogy chart. The top node is called the root, the bottom nodes are called leaves, and intermediate nodes have one parent node and several child nodes. The root can have any number of child nodes but a child node can have only one parent node. Data are related in a nested, one-to-many set of relationships, while many-to-many relationship cannot be directly expressed. A child record occurrence must have a parent record occurrence; deleting a parent record occurrence requires deleting all its child record occurrences.

A network data model can be regarded as an extended form of the

hierarchical model; the principle distinction between the two being that in a hierarchical model, a child record has exactly one parent whereas in network model, a child record can have any number of parents. It may have zero also. Data in the network model is represented by collection of records and relationship among data is represented by links, which can be viewed, as pointers. The records in the database are organized as collection of arbitrary graphs, which allows to have one-to-many as well as many-to-many relationship is a collection of data items which can be retrieved from a database, or which can be stored in a database as an undivided object.

Thus, A DBMS may STORE, DELETE or MODIFY records within a database. In this way, a number of records within a network database are dynamically changed. The network model can be graphically represented as follows:

A labeled rectangle represents the corresponding entity or record type. An arrow represents the set type, which denotes the relationship between the owner record type and member record. The arrow direction is from the owner record type to the member record type. A labeled rectangle represents the corresponding entity or record type. An arrow represents the set type, which denotes the relationship between the owner record type and member record. The arrow direction is from the owner record type to the member record type. Each many to many relationship is handled by introducing a new record type to represent the relationship wherein the attributes, if any, of the relationship are stored. We when create two symmetrical 1:M sets with the member in each of the sets being the newly introduced record type. In this model, the relationships as well as the navigation through the database are predefined at database creation time.

In **relational model** the data and the relations among them are represented by a collection of tables. A table is a collection of records and each record in a table contains the same fields. The attractiveness of the relational approach arises from the simplicity in the data organization and the availability of ably simple to very powerful query languages. The relational model is based on a technique called "Normalization" proposed by E.F. Codd. This model reduces the complexity of the Network and Hierarchical Models. This model uses the certain mathematical operations from relational algebra and relational calculus on the relation such as projection, union and joins etc. where fields in two different tables take values from the same set, a join operation can be performed to select related records in the two tables by matching values in those fields.

A description of data in terms of a data model is called a schema. In relation model, the schema for a relation specifies its name, the name of each field and the type of each field. Navigation through relations they represent an M:N relationship is just as simple as through a 1:M relationship. This leads us to conclude that it is easier to specify how to manipulate a relational database than a network or hierarchical one. This in turn leads to a query language for the relational model that is correct, clear, and effective in specifying the required operations. Unfortunately, the join operation is inherently inefficient and demands a considerable amount of processing

and retrieval of unnecessary data. The structure for the network and hierarchical model can be implemented efficiently. Such an implementation would mean that navigating through these databases, though awkward, requires the retrieval of relatively little unnecessary data.

7.

(i) Consider employee (*e_no*, *e_name*, *e_salary*, *d_code*), dept (*d_code*, *d_name*) and dependent (*depndt_name*, *e_no*, *relation*). Show the names of employees in purchase and accounts departments with at least one dependent.

(ii) Consider student (*std_id*, *std_name*, *date_of_birth*, *phone*, *dept_name*). Put the data for a student with student id200, name arun, birth date 1stFebruary, 1985, phone number (01110 32818 and dept name English in the student table.

(iii) A constraint named less_than_20 was defined on the field *date_of_birth* of table student. Delete this constraint.

(iv) Consider the table student and list names of students in the departments other than maths and computer.

(v) Consider employee table of (i) and list names of department(s) for which average salary for department is more than 10,000.

(vi) Create role named role_table that allows a user to create tables. Using role_table allow users kripa and reena to create tables.

(vii) Create a view emp_dep containing *e_name* and numberof dependents from the tables employee and dependent of (i)

Ans:

(i) SELECT *e_name* FROM employee, dependent, dept
WHERE employee.*d_code*=dept.*d_code* AND
employee.*e_no*=dependant.*e_no*
AND *d_name* IN ('ACCOUNTS', 'PURCHASE')
GROUP BY *e_name*
HAVING COUNT (*e_no*)>=1 ;

(ii) INSERT INTO student VALUES (200,'arun,"01-FEB-85','(0111)32818','English');

(iii) ALTER TABLE student DROP CONSTRAINT less_than_20;

(iv)SELECT *std_name* FRM student WHERE *dept_name* NOT
IN('Maths','Computer');

- (v) (SELECT d_name FROM dept
 WHERE d_code IN (SELECT d_code
 FROM emp GROUP BY d_code
 HAVING AVG(sal) ? 10000);
- (vi) CREATE ROLE role_table;
 GRANT CREATE ANY TABLE TO role_table;
 GRANT role_table TO kripa, reena;
- (vii) CREATE VIEW emp_dept AS SELECT ename,
 COUNT(*) FROM employee, dependent
 WHERE employee.e_no = dependant.e_no GROUP BY e_name;

8. Explain the SQL operators BETWEEN-AND, IN, LIKE and IS_NULL by taking suitable examples.

Ans:

| Comparison Operators | Description |
|----------------------|--|
| LIKE | column value is similar to specified character(s). % and _ are two special characters to match any substring and character respectively in a string. Eg. 'Perry%' matches any string with Perry. |
| IN | column value is equal to any one of a specified set of values. Eg. SQL query to find all customers who are borrowers also from a bank as well as account holders in the bank is : Select cust# from borrower where cust# in(Select cust# in depositor) |
| BETWEEN...AND | column value is between two values, including the end values specified in the range. Eg. SQL query to find all employees hiredate between 1-JAN-2008 to 31-DEC-2008: Select * from employee where hiredate between 1-JAN-2008 and '31-DEC-2008' |
| IS NULL | column value does not exist. Eg. SQL statement to find all salesperson who are not earning any commission: Select * from employee where job= "Sales" and commission is null. |

9. Define the following terms: *relation schema*, *relational database schema*, *domain*, *attribute*, *attribute domain*, *relation instance*, *relation cardinality*, and *relation degree*.

Ans:

A *relation schema* can be thought of as the basic information describing a table or *relation*. This includes a set of column names, the data types associated with each column, and the name associated with the entire table.

For example, a relation schema for the relation called Students could be expressed using the following representation:

Students(*sid*: string, *name*: string, *login*: string,
age: integer, *gpa*: real)

There are five fields or columns, with names and types as shown above.
A *relational database schema* is a collection of relation schemas, describing one or more relations.

Domain is synonymous with *data type*. *Attributes* can be thought of as columns in a table.

Therefore, an *attribute domain* refers to the data type associated with a column.

A *relation instance* is a set of tuples (also known as *rows* or *records*) that each conform to the schema of the relation.

The *relation cardinality* is the number of tuples in the relation.

The *relation degree* is the number of fields (or columns) in the relation.

10. Consider the following relations:

Student(*snum*: integer, *sname*: string, *major*: string, *level*: string, *age*: integer)
Class(*name*: string, *meets at*: string, *room*: string, *fid*: integer)
Enrolled(*snum*: integer, *cname*: string)
Faculty(*fid*: integer, *fname*: string, *deptid*: integer)

The meaning of these relations is straightforward; for example, Enrolled has one record per student-class pair such that the student is enrolled in the class.

Write the following queries in SQL. No duplicates should be printed in any of the answers.

1. Find the names of all Juniors (*level* = JR) who are enrolled in a class taught by I. Teach.
2. Find the age of the oldest student who is either a History major or enrolled in a course taught by I. Teach.
3. Find the names of all classes that either meet in room R128 or have five or more students enrolled.
4. Find the names of all students who are enrolled in two classes that meet at the same time.
5. Find the names of faculty members who teach in every room in which some class is taught.
6. Find the names of faculty members for whom the combined enrollment of the courses that they teach is less than five.
7. For each level, print the level and the average age of students for that level.
8. For all levels except JR, print the level and the average age of students for that level.

9. For each faculty member that has taught classes only in room R128, print the faculty member's name and the total number of classes she or he has taught.
10. Find the names of students enrolled in the maximum number of classes.
11. Find the names of students not enrolled in any class.
12. For each age value that appears in Students, find the level value that appears most often. For example, if there are more FR level students aged 18 than SR, JR, or SO students aged 18, you should print the pair (18, FR).

Ans:

1. SELECT DISTINCT S.Sname
 FROM Student S, Class C, Enrolled E, Faculty F
 WHERE S.snum = E.snum AND E cname = C.name AND C.fid = F.fid AND
 F.fname = 'I.Teach' AND S.level = 'JR'

2. SELECT MAX(S.age)
 FROM Student S
 WHERE (S.major = 'History') OR S.snum IN
 (SELECT E.snum FROM Class C, Enrolled E, Faculty F
 WHERE E cname = C.name AND C.fid = F.fid
 AND F.fname = 'I.Teach')

3. SELECT C.name
 FROM Class C
 WHERE C.room = 'R128' OR C.name IN
 (SELECT E cname FROM Enrolled E
 GROUP BY E cname
 HAVING COUNT (*) >= 5)

4. SELECT DISTINCT S.sname
 FROM Student S
 WHERE S.snum IN (SELECT E1.snum
 FROM Enrolled E1, Enrolled E2, Class C1, Class C2
 WHERE E1.snum = E2.snum AND E1 cname <> E2 cname
 AND E1 cname = C1 name
 AND E2 cname = C2 name AND C1 meets at = C2 meets at)

5. SELECT DISTINCT F.fname
 FROM Faculty F
 WHERE NOT EXISTS ((SELECT * FROM Class C)
 EXCEPT
 (SELECT C1 room
 FROM Class C1
 WHERE C1 fid = F fid))

6. SELECT DISTINCT F.fname

```
FROM Faculty F
WHERE 5 > (SELECT COUNT (E.snum)
FROM Class C, Enrolled E
WHERE C.name = E cname AND C.fid = F.fid)
```

```
7. SELECT S.level, AVG(S.age)
FROM Student S
GROUP BY S.level
```

```
8. SELECT S.level, AVG(S.age)
FROM Student S
WHERE S.level <> 'JR'
GROUP BY S.level
```

```
9. SELECT F.fname, COUNT(*) AS CourseCount
FROM Faculty F, Class C
WHERE F.fid = C.fid
GROUP BY F.fid, F.fname
HAVING EVERY ( C.room = 'R128' )
```

```
10. SELECT DISTINCT S.sname
FROM Student S
WHERE S.snum IN (SELECT E.snum
FROM Enrolled E
GROUP BY E.snum
HAVING COUNT (*) >= ALL (SELECT COUNT (*)
FROM Enrolled E2
GROUP BY E2.snum ))
```

```
11. SELECT DISTINCT S.sname
FROM Student S
WHERE S.snum NOT IN (SELECT E.snum
FROM Enrolled E )
```

```
12. SELECT S.age, S.level
FROM Student S
GROUP BY S.age, S.level,
HAVING S.level IN (SELECT S1.level
FROM Student S1
WHERE S1.age = S.age
GROUP BY S1.level, S1.age
HAVING COUNT (*) >= ALL (SELECT COUNT (*)
FROM Student S2
WHERE s1.age = S2.age
GROUP BY S2.level, S2.age))
```

4.2.7. Assignments:

1. Define a table constraint on Emp that will ensure that every employee

makes at least \$10,000.

2. Define a table constraint on Dept that will ensure that all managers have $age > 30$.
3. Define an assertion on Dept that will ensure that all managers have $age > 30$.
30. Compare this assertion with the equivalent table constraint. Explain which is better.
4. Write SQL statements to delete all information about employees whose salaries exceed that of the manager of one or more departments that they work in. Be sure to ensure that all the relevant integrity constraints are satisfied after your updates.

4.b.viii. Previous Questions (Asked by JNTUK from the concerned Unit)

1. For the following relational database write the expressions in SQL.

Branch Schema (branch name, Branchcity, Assets)

Customer schema(customername, customerstreet, customercity)

Loan schema(Branchname, loan number, Amount)

Borrower schema(customername, loan number)

Account schema (Branchname, Account number, balance)

Depositor schema(customername, Account number)

i. Find the names of all branches in Loan Schema?

ii. Find all customers having loan, account or both at bank?

iii. Display customernames in alphabetical order who have a loan at the Perry
ridge branch?

iv. Find set of all customers who have an account at the bank?

2. Write the SQL expressions for the following relational database?

sailor schema (sailor id, Boat id, sailortname, rating, age)

Receives (Sailor id, Boat id, Day)

Boat Schema (boat id, Boatname, color)

i. Find the age of the youngest sailor for each rating level?

ii. Find the age of the youngest sailor who is eligible to vote for each rating level with at least two such sailors?

iii. Find the No.of reservations for each red boat?

iv. Find the average age of sailor for each rating level that at least 2 sailors.

3. (a) What is a relational database query? Explain with an example.

(b) What are the SQL constructs to modify the structure of tables, views and to destroy the tables and views?

4. a) What is data model? List and explain different data models.
 b) Explain the difference between external, internal, and conceptual schemas. How are these different schema layers related to the concepts of logical and physical data independence?
- 5.a) Consider the following relational schema. An employee can work in more than one department; the pct time field of the Works relation shows the percentage of time that a given employee works in a given department.
- ```

 Emp(eid: integer, ename: string, age: integer, salary: real)
 Works(eid: integer, did: integer, pct time: integer)
 Dept(did: integer, budget: real, managerid: integer)

```
- Write the following queries in SQL:
- Print the names and ages of each employee who works in both the Hardware department and the Software department.
  - For each department with more than 20 full-time-equivalent employees (i.e., where the part-time and full-time employees add up to at least that many full-time employees), print the did together with the number of employees that work in that department.
  - Print the name of each employee whose salary exceeds the budget of all of the departments that he or she works in.
  - Find the manager ids of managers who manage only departments with budgets greater than \$1,000,000.
  - Find the enames of managers who manage the departments with the largest budget.

- 6.Which of the following plays an important role in representing information about the real world in a database? Explain briefly about:
- The data definition language.
  - The data manipulation language.
  - The buffer manager.
  - The data model.

- 7.The following relations keep track of airline flight information:
- ```

  Flights(flno: integer, from: string, to: string, distance: integer, departs: time,
  arrives: time,
  price: integer) Aircraft(aid: integer, aname: string, cruisingrange: integer)
  Certified(eid: integer, aid: integer)
  Employees(eid: integer, ename: string, salary: integer)

```
- Note that the Employees relation describes pilots and other kinds of employees as well; every pilot is certified for some aircraft, and only pilots are certified to fly. Write each of the following queries in SQL.
- Find the names of aircraft such that all pilots certified to operate them earn more than 80,000.
 - For each pilot who is certified for more than three aircraft, find the eid and the maximum cruisingrange of the aircraft that he (or she) is certified for.
 - Find the names of pilots whose salary is less than the price of the cheapest

route from Los Angeles to Honolulu.

iv) For all aircraft with cruising range over 1,000 miles, find the name of the aircraft and the average salary of all pilots certified for this aircraft.

4.2.9.GATE Questions (Where relevant)

1. Given relation r(w,x) and s(y,z), the result of select distinct w,x from r,s is guaranteed to be same as r, provided [GATE 200]
 - A. r and s have the same number of tuples
 - B. s has no duplicates and r is non empty
 - C. r and s has no duplicates
 - D. r has no duplicates and s is non empty

Ans: D

2. Consider the table employee(empid, name, department, salary) and the two queries Q1 and Q2 below. Assuming that department 5 has more than one employee, and we want to find the employees who get the higher salary than anyone in the department 5, which one of the statements is true for any arbitrary employee table

Q1: select e.empid from employee e Where non exists (select * From employee s where s.department= '5' and s.salary>= e.salary)

Q2: Select e.empid from employee e where e.salary >any
(select distinct salary from employee s where s.department='5')

- A. Q2 is correct query
- B. Both Q1 and Q2 are incorrect
- C. Both Q1 and Q2 gives same results
- D. Q1 is correct query

Ans: A

3. Database table by name Loan_Records is given below.

| Borrower | Bank_Manager | Loan_Amount |
|----------|--------------|-------------|
| Ramesh | Sunderrajan | 100000 |
| Suresh | Ramgopal | 5000 |
| Mahesh | Sunderrajan | 7000 |

| | | |
|--|--|--|
| | | |
|--|--|--|

What is the output of the following SQL query ?

```
SELECT count(*) From(SELECT Borrower.Bank_Manager From
Loan_Records) AS S Natural Join ( SELECT BANK_manager, Loan_Amount
From Loan_Records ) As T ;
```

- A. 6
- B. 5
- C. 9
- D. 3

Ans: B

4. In SQL relations can contain null values, and comparisons with null values are treated as unknown. Suppose all comparisons with a null value are treated as false. Which of the following pairs is not equivalent ? [GATE 200]

- A. $x=5$ not ($\text{not}(x=5)$)
- B. $x=5$ $x>4$ and $x<$, where x is an integer
- C. none of the above
- D. $x \neq 5$ not ($x=5$)

Ans: D

5. Let E1 and E2 be two entities in an E/R diagram with simple single valued attributes. R1 and R2 are two relationships between E1 and E2, where R1 is one to many and R2 is many to many. R1 and R2 do not have any attributes of their own. What is the minimum number of tables required to represent this situation in the relational model ? [GATE2005]

- A. 5
- B. 3
- C. 2
- D. 4

Ans: C

6. The relation book(title,price) contains the titles and prices of different books. Assuming that no two books have the same price, what does the following sql query list ?[GATE 2005]

```
Select title from book as B where (select count(*) from book as T
where T.price >B.price)<5
```

- A. title of the 5th most expensive books
- B. titles of the 5 most expensive books

- C. titles of the 4 most expensive books
- D. title of the 5th most inexpensive books

Ans: B

- 7. From the following instance of a relational schema R(A,B,C) We can conclude that

| A | B | C |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 0 |
| 2 | 3 | 2 |
| 2 | 3 | 2 |

- A. A functionally determine B and B functionally determines C
- B. B does not functionally determines C
- C. A functionally determine B and B does not functionally determines C
- D. A does not functionally determine B and B does not functionally determines C

Ans: C

- 8. The employee information in a company is stored in the relation Employee(name,sex,salary,deptname)
Consider the following sql query
Select deptname from employee where sex='m' group by deptName having avg(salary)>(select avg(salary) from employee)
It returns the name of the department in which
- A. the average salary is more than the average salary in the company
 - B. the average salary of male employers is more than the average salary in the company
 - C. the average salary of male employees is more than the average salary of the employees in the same department
 - D. the average salary of male employee is more than the average salary of all male employee in the company

Ans: B

- 9. Consider the following relational schema pertaining to a student's

database: [GATE 2004]

Students (*rollno*, name, address)

Enroll(*rollno*, *courseno*, *coursename*)

Where primary keys are shown in italics. The number of tuples in the student and Enroll tables are 120 and 8 respectively. What are the maximum and minimum number of tuples that can be present in (Student *Enroll) , where * denotes natural join ?

- A. 8,8 B. 120,8 C. 960,8 D.960,120

Ans: A

10. The employee information in a company is stored in the relation GATE 2004 Employee (name, sex, salary, deptName) Consider the following SQL query Select deptName From Employee Where sex = 'M' Group by deptName Having avg(salary) > (select avg (salary) from Employee)

It returns the names the department in which

- (a) The average salary is more than the average salary in the company
- (b) The average salary of male employees is more than the average salary of all male employees in the company
- (c) The average salary of male employees is more than the average salary of employees in the same department.
- (d) The average salary of male employees is more than the average salary in the company

Ans: D

4.2.10. Interview questions

1. What are DML and DDL?
2. How do we select distinct values from a table?
3. What is Like operator for and what are wild cards?
4. Can you explain Insert, Update and Delete query?
5. What is order by clause?
6. What is the SQL " IN "
7. Can you explain the between clause?
8. we have an employee salary table how do we find the second highest

from it?

9. How to select the first record in a given set of rows?
10. What is a self-
11. What's the difference between DELETE and TRUNCATE
12. What is " Group by "
13. What is the difference between "HAVING" and "WHERE" clause?
14. Can you explain the SELECT INTO
15. What is Data & What are Data?
16. What are fact tables and Dimension Tables? What is Dimensional Modeling and Star Schema Design

4.2.11. Real-Word (Live) Examples / Case studies wherever applicable

Supporting Real-world Activities in Database Management Systems

Mohamed Y. Eltabakh ^{#1}, Walid G. Aref ^{#2}, Ahmed K. Elmagarmid ^{#3}, Yasin N. Silva ^{#4}, Mourad Ouzzani ^{*5}

[#]Computer Science Department, Purdue University

West Lafayette, IN, USA

¹meltabak@cs.purdue.edu

²aref@cs.purdue.edu

³ake@cs.purdue.edu

⁴ysilva@cs.purdue.edu

^{*}Cyber Center, Purdue University

West Lafayette, IN, USA

⁵mourad@cs.purdue.edu

Abstract— The cycle of processing the data in many application domains is complex and may involve real-world activities that are external to the database, e.g., wet-lab experiments, instrument readings, and manual measurements. These real-world activities may take long time to prepare for and to perform, and hence introduce inherently long time delays between the updates in the database. The presence of these long delays between the updates, along with the need for the intermediate results to be instantly available, makes supporting real-world activities in the database engine a challenging task. In this paper, we address these challenges through a system that enables users to reflect their updates immediately into the database while keeping track of the dependent and *potentially invalid* data items until they are re-validated. The proposed system includes: (1) semantics and syntax for interfaces through which users can express the dependencies among data items, (2) new operators to alert users when the returned query results contain potentially invalid or out-of-date data, and to enable evaluating queries on either valid data only, or both valid and potentially invalid data, and (3) mechanisms for data invalidation and re-validation. The proposed system is being realized via extensions to PostgreSQL.

I. INTRODUCTION

In many application domains, e.g., scientific experimentation, the cycle of processing the data to generate new results is complex and may involve sequences of real-world activities that cannot be coded within the database system, e.g., wet-lab experiments, instrument readings, and manual measurements. As illustrated in Figure 1, it is typical in scientific applications to have the following scenario: (1) scientists retrieve values from the database system, (2) perform one or more real-world activities that depend on the retrieved values, and then (3) store the output results from the performed activities back into the database. Current database technologies do not capture the dependencies among the data items that involve real-world activities. Therefore, updating a database value will render all the dependent and derived values invalid until the external activities involved in the dependency, e.g., the wet-lab experiment, are re-executed and the output results are reflected back into the database. Because of this mandated unbounded delay in propagating the updates, parts of the underlying

This research was partially supported by NSF Grant Numbers IIS 0916614 and IIS 0811954, and by Purdue Cyber Center.

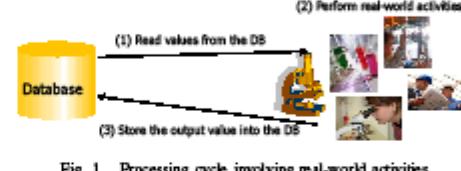


Fig. 1. Processing cycle involving real-world activities

database may remain inconsistent while it still needs to be available for querying.

With current database technologies, scientists may consider one of the following two options: (1) postpone the database updates until all dependent processes are executed and their outputs become consistent, and then reflect these updates into the database at once (may involve unbounded long delays), or (2) reflect the updates immediately into the database and *temporarily* compromise the consistency of the derived data (in this case, users may lose track of the outdated data that need re-evaluation). Both options have serious problems as they delegate tracking the dependencies and maintaining the data consistency to the end-users instead of the database management system. In this paper, we propose a third, more appealing, option where scientists can reflect the updates immediately into the database while the DBMS keeps track of the derived data by marking them as *potentially invalid (outdated)* and reflecting them in the queries' results until the external manual activities are re-executed and the outdated values are updated.

The theory of functional dependencies (FDs) [4], [8] and its usage in ordinary databases address an orthogonal problem to the problem highlighted in this paper. That is, even with a good schema design of the database and following the decomposition and normalization rules, the inconsistency problem of the derived data still exists. Several extensions to functional dependencies have been proposed to address other issues, e.g., [7], [5]. However, none of this past work can be applied directly to the problem at hand. Other works have been proposed to support long-running transactions, e.g., [6], by loosening the ACID properties and avoiding locks,

4.2.12. Suggested “Expert Guest Lectures” (both from in and outside of the campus) NA

4.2.13. Literature references of Relevant NPTEL Videos/Web/You

Tube

videos etc.

- <https://www.youtube.com/watch?v=gGGHjYbQMvw&list=PLCB23FA85B073E3C8&index=4>
- <https://www.youtube.com/watch?v=nc1yivH1Yac&list=PLCB23FA85B073E3C8&index=5>
- <https://www.youtube.com/watch?v=64szTfLNu3o&index=6&list=PLCB23FA85B073E3C8>

4.2.14. Any Lab requirements; if so link it to Lab Lesson Plan.

NA

4.2.15. Reference Text Books / with Journals Chapters etc.

Text Book:

1. Alexis Leon, Mathews Leon, Fundamentals of Database Management Systems, Tata McGraw Hill Publishers, Second Print, 2010

Reference Books:

1. Mark Gillenson, Fundamentals of Database Management Systems, Wiley Publications. Reprint 2008
2. Raghu Ramakrishnan, Johannes Gehrke , Database Management Systems, McGraw-Hill, 2003
3. Isrd Group, Introduction to Database Management Systems, Tata McGraw-Hill Education, 2005
4. Michael V. Mannino , Database Application Development and Design, McGrawHill/Irvin, 2001
5. Connolly, Database Systems: A Practical Approach to Design, Implementation and Management, 4/E, Pearson Education India, 2008
6. Patricia Ward, George Dafoulas, Database Management Systems, Cengage Learning EMEA, 2006
7. Post, Database Management Systems 3e with Cd, Tata McGraw-Hill Education
8. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, Database System Concepts, Mcgraw-Hill, 2010
9. Colin Ritchie, Database Principles and Design, Cengage Learning EMEA, 2008
10. Peter Rob, Carlos Coronel, Database Systems, - 2007

Journals Chapters:

1. International Journal of Database Management Systems (IJDMS), Vol.3, No.2, May 2011 "A COMPARATIVE STUDY BETWEEN THE PERFORMANCE OF RELATIONAL & OBJECT ORIENTED DATABASE IN DATA WAREHOUSING"
2. E. F. CODD The Relational Model for Database Management • Version 2

E. F. CODD

The
RELATIONAL
MODEL
for
DATABASE
MANAGEMENT

VERSION 2

4.3. Unit-III-Entity Relationship model & SQL

Entity Relationship Model: Introduction, Representation of entities, attributes, entity set, relationship, relationship set, constraints, sub classes, super class, inheritance, specialization, generalization using ER Diagrams.

SQL : Creating tables with relationship, implementation of key and integrity constraints, nested queries, sub queries, grouping, aggregation, ordering, implementation of different types of joins, view(updatable and non-updatable), relational set operations.

4.3.1. Unit Objectives:

- Define terms related to entity relationship modeling, including entity, entity instance, attribute, relationship, cardinality and foreign key.
- Describe the entity modeling process
- Discuss how to draw an entity relationship diagram.
- Describe how to recognize entities, attributes, relationships and cardinalities

4.3.2. Unit Outcomes:

- The Entity-relationship modeling to develop a conceptual model of data
- How to organize data required in an application as relations.
- Define the following terms: entities, attributes, domain, composite primary key, simple attribute, composite attribute, single-valued attributes, multi-valued attributes and derived attributes.
- Identify and provide suitable name that is descriptive of the relationship.
- Differentiate between weak and strong relationships
- Explain and use SQL functions to manipulate dates, strings, and

other data.

- Create and use sequences
- Create and use simple triggers and stored procedures.

4.3.3. Unit Lecture Plan:

| Lecture No. | Topic | Methodology | Quick reference |
|--------------------|---|--------------------|-------------------------|
| 1 | Entity Relationship Model: Introduction, Representation of entities, attributes, entity set | Chalk & Board | Text. Book with Page |
| 2 | relationship, relationship set, constraints, sub classes | Chalk & Board | Text. Book with Page |
| 3 | super class, inheritance, | Chalk & Board | Text. Book with Page |
| 4 | specialization, generalization using ER Diagrams. | Chalk & Board | Text. Book with Page |
| 5 | SQL: Creating tables with relationship | Chalk & Board | Text. Book with Page |
| 6 | implementation of key and integrity constraints, | Chalk & Board | Text. Book with Page |
| 7 | nested queries, sub queries, | Chalk & Board | Text. Book with Page |
| 8 | grouping, aggregation, ordering | Chalk & Board | Text. Book with Page |
| 9 | implementation of different types of joins | Chalk & Board | Text. Book with Page |
| 10 | view(updatable and non- updatable), relational set operations | Chalk & Board | Text. Book with Page |

4.3.4. Teaching Material / Teaching Aids as per above lecture plan.

4.3.4.1. Lecture -1

Introduction Entity Relationship Model

Entity Relationship Model PPT

Entity-Relationship Model

E/R Diagrams

Weak Entity Sets

Converting E/R Diagrams to Relations

1

Database Design

- Overall goal of DBMS usage: Efficiently develop programs to support given

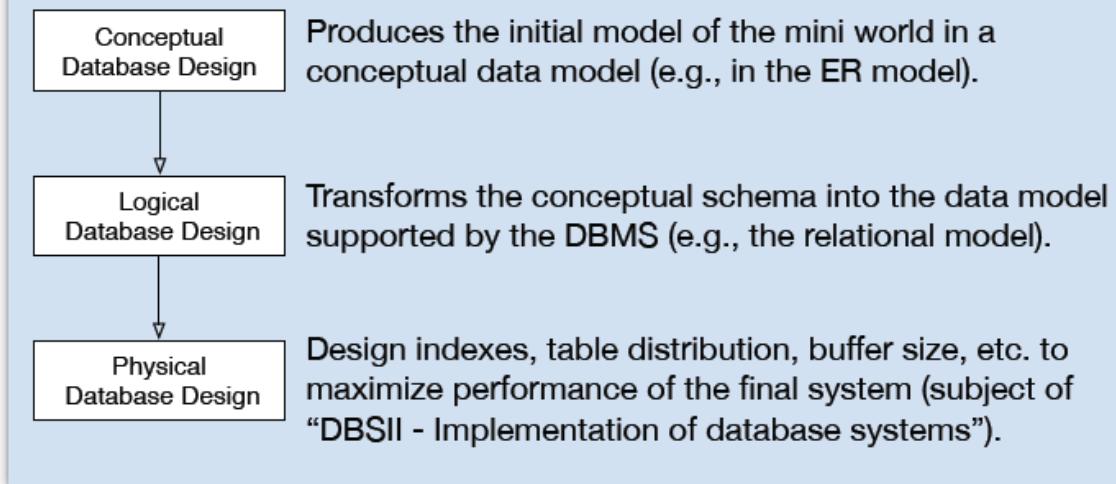
real-world tasks.

- These programs need to **store data persistently**.
- To develop these programs, apply proven methods of **software engineering**—specialized to support **data-intensive** programs.

Database design

- Database design is the process of developing a database schema for a given application.
- Database design is a subtask of the overall software engineering effort.
- The specification of programs and data is intertwined:
 - The schema should contain the data needed by the programs.
 - Programs are often easy to develop once the structure of the data to be manipulated has been specified.
- Data, however, is an **independent resource**:
 - Typically, additional programs will be developed later based on the collected data.
 - Also, ad-hoc queries will be posed against the DB.
- During DB design, a **formal model of the relevant aspects** of the real world (“mini world”, “domain of discourse”) must be built.
- Once the DB is up and running, questions will be posed against the DB. Knowledge of these questions beforehand is important input to the DB design process and helps to identify the relevant parts of the real world.
- In some sense, the real world is the **measure of correctness** for the DB schema: the possible DB states should correspond to the states of the real world.
- DB design is not easy for a variety of reasons:
 - **Expertise**: The designer must become an expert for the application domain or needs to talk to such experts. Depending on the domain, these experts might not be used to give a complete and formal account of their field.
 - **Flexibility**: The real world typically permits exceptions and corner cases. Does the DB schema need to reflect these?
 - **Size**: Database schemas may become huge (in terms of number of relations, attributes, constraints).
- Due to this complexity, DB design is a multi-step process.

Three phases of DB design



Why multiple design phases?

- **Partition** the problem, attack one sub-problem after the other. For example, during conceptual design there is no need to worry about performance aspects or limitations of the specific SQL dialect of the RDBMSs.
- DBMS features do not influence the conceptual design and only partially influence the logical design.
 - Thus, the conceptual design work is not invalidated if a different DBMS is used later on.

4.2.4.2. Lecture -2

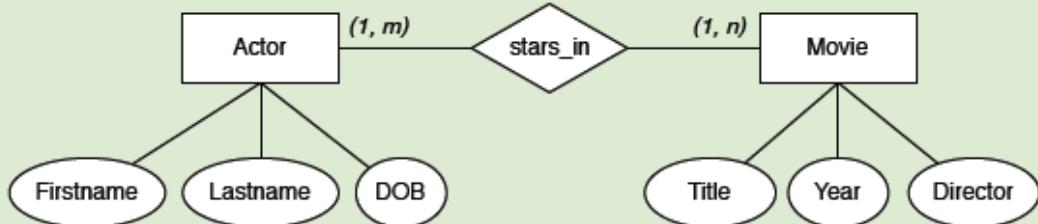
Representation of Entities, attributes, entity sets, relationship, relationship set.

The ER Model

- Proposed by Peter Chen (1976), today at Louisiana State University.
- The ER model comes with a **graphical notation** which helps to establish quick overviews of database application schemas.
- Such **ER diagrams** are also a great way to communicate schemas to non-expert/future DB users.

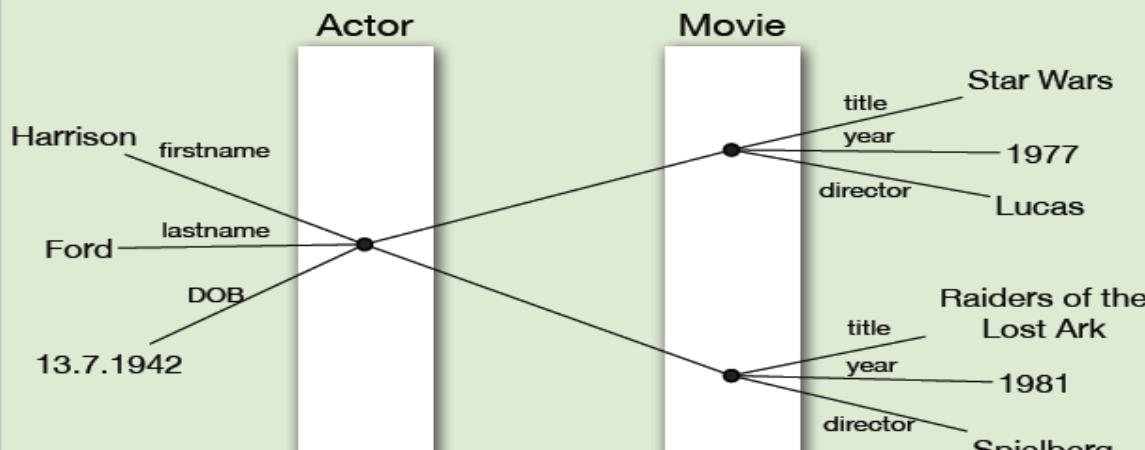
Example

ER schema in graphical notation



- This mini world talks about **actors** and **movies**.
- Actors **star** in movies.
- Actors have a **firstname**, a **lastname**, and a date of birth (**DOB**)
- Movies have a **title**, a **year**, and a **director**.

A possible state of this mini world



- The **Entity-Relationship Model** is often referred to as a semantic data model, because it more closely resembles real world scenarios than, e.g., the relational model.
 - In the ER model, we model the concept of "Actor". In the relational model we deal with names and dates of birth.
 - In the ER model, there is a distinction between **entities** (objects) and **relationships** between such entities. In the relational model, both concepts are represented by relations.
- There are no "ER Model DBMS". Instead, we will describe a translation of ER model concepts into the relational model.

Basic ER Model Concepts

Entity

- An **entity** is an **object** in the mini world about which information is to be stored. Examples: actors, movies, studios, awards.
- Note: entities do not have to correspond to objects of physical existence. Entities may also represent conceptual objects like, e.g., vacation.
- The mini world that has to be modeled can contain only a **finite number of objects**.
- It must be possible to distinguish entities from each other, i.e., objects must have some **identity**.
Examples: entity book identified by ISBN number, entity vacation identified by travel agency booking number.

Attribute:

- An **attribute** is a **property** or feature of an entity (or relationship, see below). Example: the title of this movie entity is "Star Wars".
- The **value** of an attribute is an element of a data type like string, integer, date.
- These values have a **printable representation** (which entities have not).

Relationship:

- A relationship represents a **relation**—not in the strict relational model sense—between **pairs** of entities (a binary relationship). Example: Ford (an entity) stars in (a relationship) the movie "Star Wars" (an entity).
- "Relationship" is often used as an abbreviation for "Relationship type" (see next slide).

Entity and Relationship Type

Entity Type:

A set of similar entities (similar with respect to the information which has to be stored about them), i.e., entities which have the same attributes. Example: All actors vs. all directors.

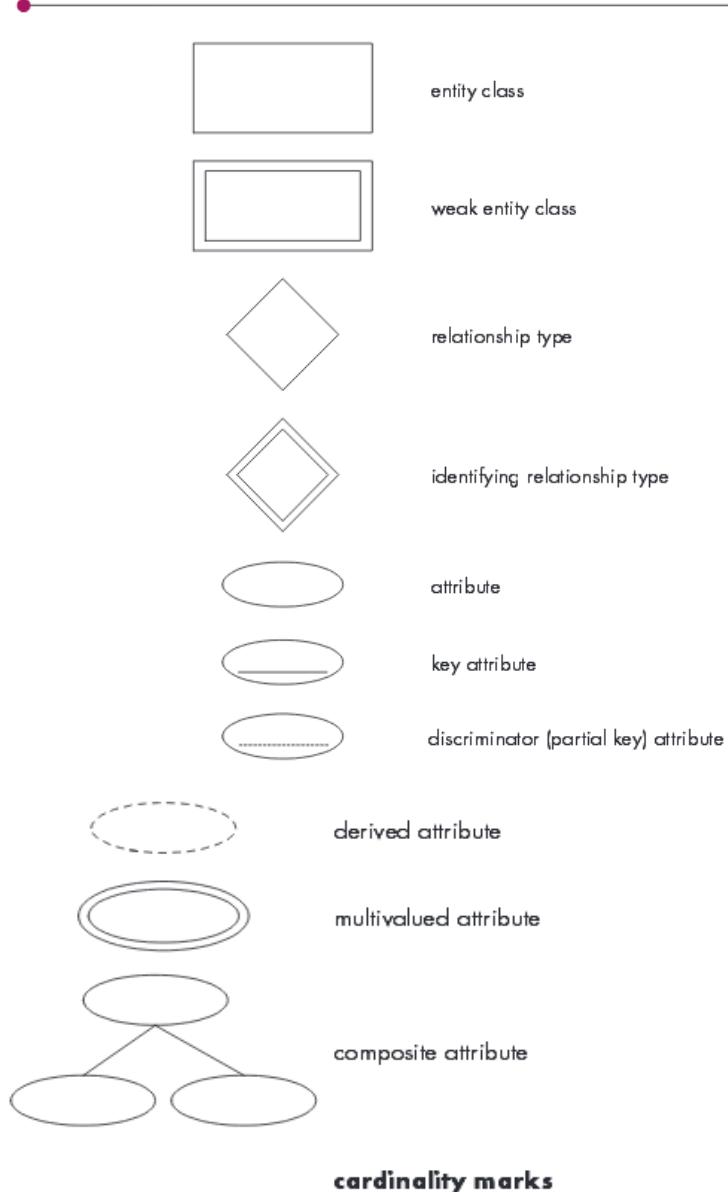
Relationship Type:

A set of similar relationships, i.e., relationships which have the same attributes. Example: X stars in movie Y.

Basic Graphical Syntax

FIGURE

Examples of E-R Diagram
Symbols



cardinality marks

- | | |
|-------|---|
| 1 | no more than one related entity |
| M | many (zero or more) related entities |
| i...j | at least i but not more than j related entities |
| — | must participate in the relationship |
| — | may participate in the relationship |

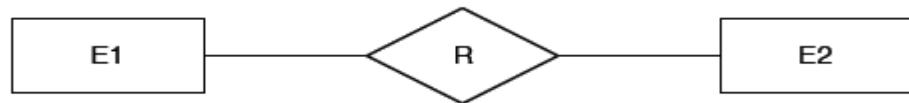
- Entity type E



- Attribute A of entity type E



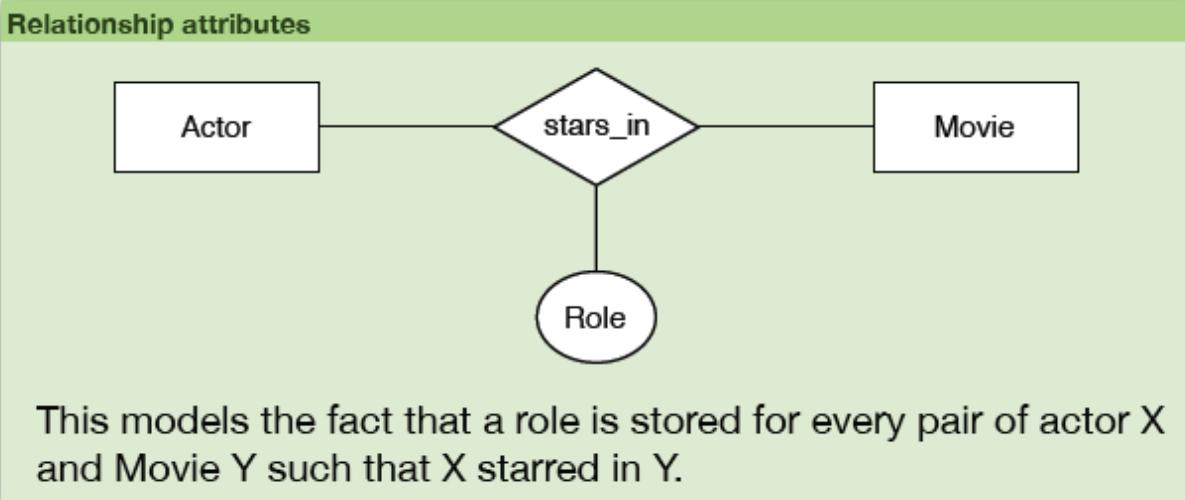
- Binary relationship R between entity types E1 and E2



ER Diagrams

Attributes of Relationships

- Relationships may have attributes, too:



Graphical Syntax

- An ER diagram contains **boxes**, **diamonds**, **ovals**, plus **interconnecting lines**.
- Boxes, diamonds, and ovals are each **labelled** by a string.
- Box labels are unique in the entire diagram.
- Oval labels are unique for a single box or diamond.
- Diamond labels are unique for a pair of connected boxes.
- **Interconnecting lines** are only allowed between box–diamond, box–oval, diamond–oval.

- A diamond has exactly **two connecting lines** to boxes. There may be any number of connections to ovals.
- An oval has exactly one connecting line.

ER Example

Modeling a mini world: define an ER diagram

- Information about movies is to be stored.
- For each movie, the title, release date, genre, and homepage URI are relevant.
- Movies may be granted awards (e.g., Oscars, Palme d'Or, ...)
- Relevant award information are the name, the category, and the year of the award.
- Each movie is produced by a studio (of a given name and address).

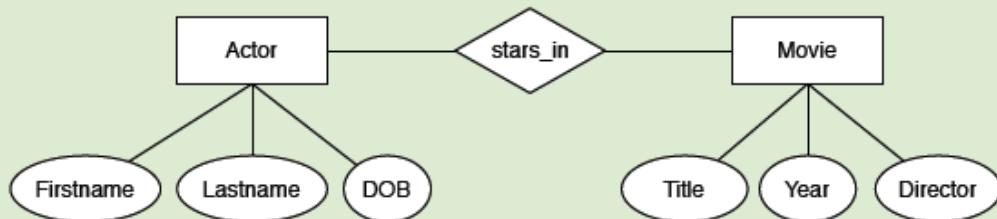
ER Schemas: Semantics

Definition

A DB state I interprets the symbols in the ER schema by defining

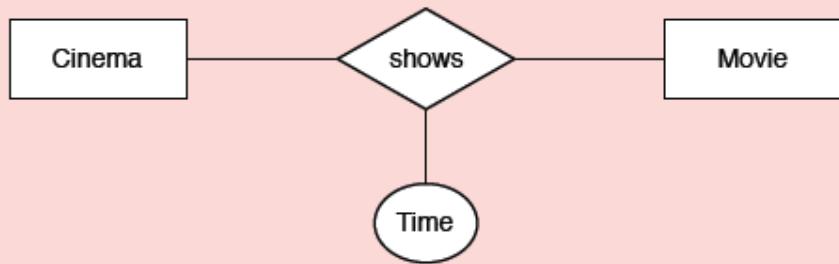
- A finite set $I(E)$ for every entity type E ,
- A mapping $I(A):I(E) \rightarrow val(D)$ for every attribute A of an entity type E , where D is the data type of A and $val(D)$ is the domain of D ,
- A relation $I(R) \subseteq I(E1) \times I(E2)$ for every relationship R between entity types $E1$ and $E2$.
- A mapping $I(A):I(R) \rightarrow val(D)$ for every attribute A of relationship R (D is the data type of A).

Example state corresponding to ER schema



- $I(Actor) = \{JD, HBC\}$
- $I(Firstname) = f_1$ with $f_1(JD) = "Jonny"$ and $f_1(HBC) = "Helena Bonham"$
- $I(Lastname) = f_2$ with $f_2(JD) = "Depp"$ and $f_2(HBC) = "Carter"$
- $I(DOB) = f_3$ with $f_3(JD) = "09.06.1963"$ and $f_3(HBC) = "26.05.1966"$
- $I(Movie) = \{CB, ST, PC\}$
- $I(Title) = g_1$ with $g_1(CB) = "A Corpse Bride"$, $g_1(ST) = "Sweeney Todd"$,
and $g_1(PC) = "Pirates of the Caribbean"$
- $I(Year) = g_2$ with $g_2(CB) = 2003$, $g_2(ST) = 2007$, and $g_2(PC) = 2006$
- $I(Director) = g_3$ with $g_3(CB) = "Tim Burton"$, $g_3(ST) = "Tim Burton"$, $g_3(PC) = "Gore Verbinski"$

Consequences of the ER semantics

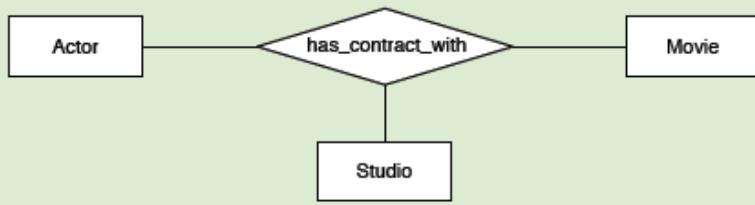


- Consider the above ER diagram.
- Suppose a cinema shows the same movie twice a day (at 3pm and 6pm).
- Can this information be stored in the given schema?

From n-ary Relationships to Binary Relationships

- We have seen that an ER model represents binary relationships only.
- However, in the real world, a relationship may relate more than two entities to each other.

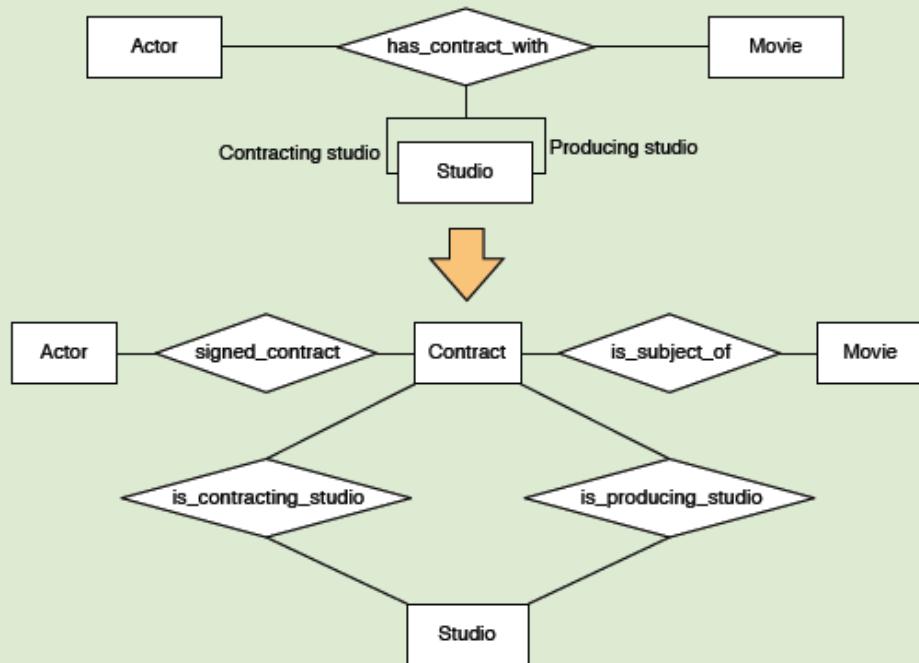
An n-ary relationship



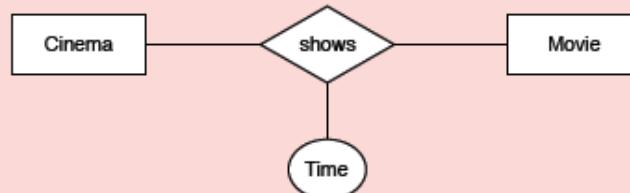
From n-ary Relationships to Binary Relationships

- To transform an n-ary relationship to a binary relationship, we
 1. Create a new **association entity type**.
 2. Introduce a relationship between the new entity type and the old entity types.
 3. If there is an entity with multiple roles, create a **relationship type per role**.
 4. Attributes of the old relationship type are appended to the new entity type.

An n-ary relationship



Modeling that a cinema shows the same movie twice a day



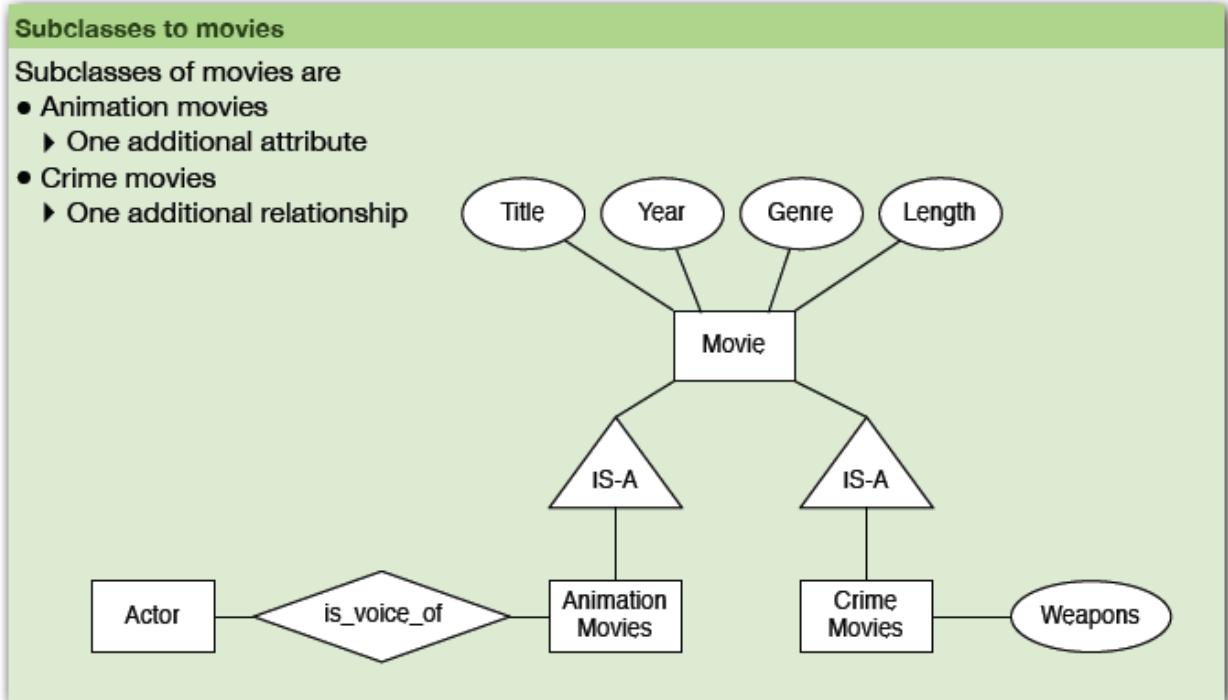
4.2.4.3. Lecture -3

subclass, superclass, inheritance

IS-A: A Special Type of Relationship

- Subclass
 - Special case of the (super)class
 - Less entities fall into the subclass (compared to the entities falling in the superclass)
 - More (specific) attributes
 - Possibly more relationships to other entities.
- The ER model allows to specify a subclass in the form of an IS-A relationship
 - Represented as a triangle
 - Tip is showing to the superclass
 - The relationship implicitly is a 1:1 relationship (and is not explicitly

represented)

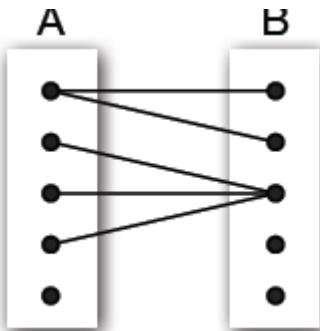


- The IS-A relationship forms **trees**, a class may have **multiple subclasses**, but a class may **not** have **multiple superclasses**.
- An entity may consist of **several components** of the IS-A-tree
 - “Star Wars” has four attributes.
 - “Cindarella” has four attributes plus a “Voices” relationship.
 - “Der Dritte Mann” has four attributes plus the “Weapons” attribute.
 - “Roger Rabbit” has four attributes, plus the “Weapons”-Attribute and the “Voices”-Relationship.

Cardinalities

M:N Relationships

- In general, an **arbitrary number of entities** of type A can **relate** to an **arbitrary number of entities** of type B via a binary relationship.
 - An actor can star in numerous movies.
 - A movie set comprises many actors.
 - An entity of type A **may not relate to any entity** of type B, and vice versa.
 - Meet the many “actors” sitting in LA cafes that have never played in any movie.
 - National geographic movies do not have actors

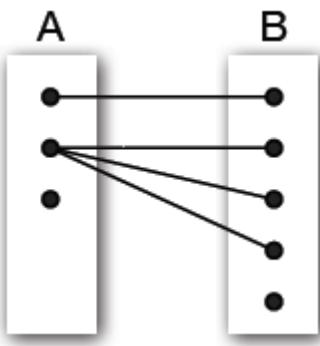


► Many-to-Many or m:n relationship

- Special cases are one-to-many (1:n) and one-to-one (1:1) relationships..

Cardinalities

1:N Relationships

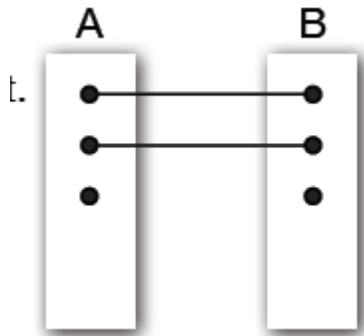


- An entity of type A can relate to any number of entities of type B.
- An entity of type B can relate to at most one entity of type A.
- Example
- A movie may win many awards.
- An award is granted to one movie.
- Again, an entity of type A **may not relate to any entity of type B**, and vice versa.
- A movie may not win any award.

Cardinalities

1:1 Relationships

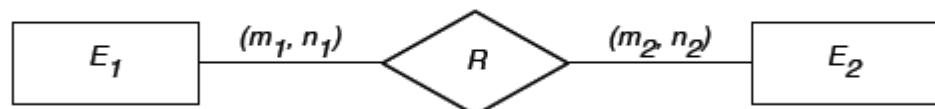
- An entity of type A can relate to at most one entity of type B.
- An entity of type B can relate to at most one entity of type A.
- Example
- A studio may only have one President.
- One President may only head one studio.
- Again, an entity of type A **may not relate to any entity of type B** and vice versa.
- A studio may temporarily not have any president



Cardinalities

The (min, max) notation

- The ER model introduces the (min, max) notation to specify an interval of possible participations in a relationship:



- An entity of type E1 may be related to at least m_1 and at most n_1 entities of type E2.
- Likewise, m_2 is the minimum number and n_2 is the maximum number of E1 entities to which an E2 entity is related.

4.2.4.4 Lecture -4

Specialization and generalization using ER Diagram

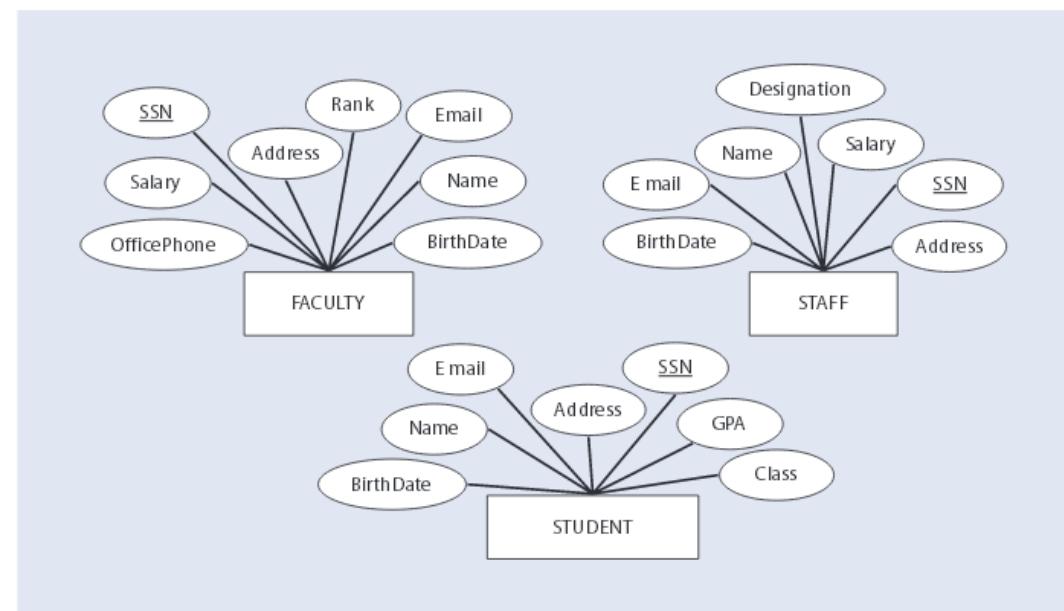
Generalization:

Generalization is the process of defining general entity types from a set of specialized entity types by identifying their common characteristics. In other words, this process minimizes the differences between entities by identifying a general entity type that features the common attributes of specialized entities. Generalization is a bottom-up approach as it starts with the specialized entity types (subclasses) and forms a generalized entity type (superclass).

For example, suppose that someone has given us the specialized entity types FACULTY, STAFF, and STUDENT, and we want to represent these entity types separately in the E-R model in Figure. However, if we examine them closely, we can observe that a number of attributes are common to all entity types, while others are specific to a particular entity. For example, FACULTY, STAFF, and STUDENT all share the attributes Name, SSN, BirthDate, Address, and Email. On the other hand, attributes such as GPA, Class, and MajorDept are specific to the STUDENTS; OfficePhone is specific to FACULTY, and

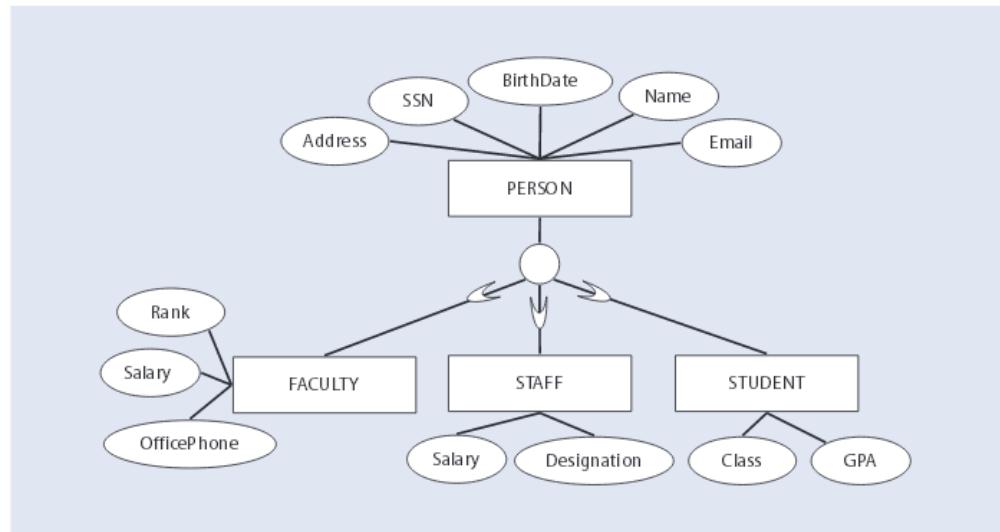
Designation is specific to STAFF. Common attributes suggest that each of these three entity types is a form of a more general entity type. This general entity type is simply a PERSON superclass entity with common attributes of three subclasses.

Thus, in the generalization process, we group specialized entity types to form one general entity type and identify common attributes of specialized entities as attributes of a general entity type. The general entity type is a superclass of specialized entity types or subclasses.



(continues)

Figure (a) STAFF, FACULTY, and STUDENT entities before generalization; (b) PERSON superclass and FACULTY, STAFF, and STUDENT subclasses after generalization.



(b)

Figure (continued)

Specialization

Specialization is the process of defining one or more subclasses of a superclass by identifying its distinguishing characteristics. Unlike generalization, specialization is thus a top-down approach. It starts with the general entity (superclass) and forms specialized entity types (subclasses) based on specialized attributes or relationships specific to a subclass.

For example, consider Figure 3.26(a). **LIBRARY ITEM** is an entity type with several attributes such as **IdentificationNo**, **RecordingDate**, **Frequency**, and **Edition**. After careful review of these items, it should become clear that some items such as books do not have values for attributes such as **Frequency**, **RecordingDate**, and **CourseNo**, while Video CDs do not have an **Author** or an **Edition**. In addition, all items have common attributes such as **IdentificationNo**, **Location**, and **Subject**. Someone creating a library database, then, could use the specialization process to identify superclass and subclass relationships. In this case, the original entity **LIBRARY ITEM** forms a superclass entity type made up of attributes shared by all items, while specialized items with distinguishing attributes, such as **BOOK**, **JOURNALS**, and **VIDEOCD**, form subclasses.

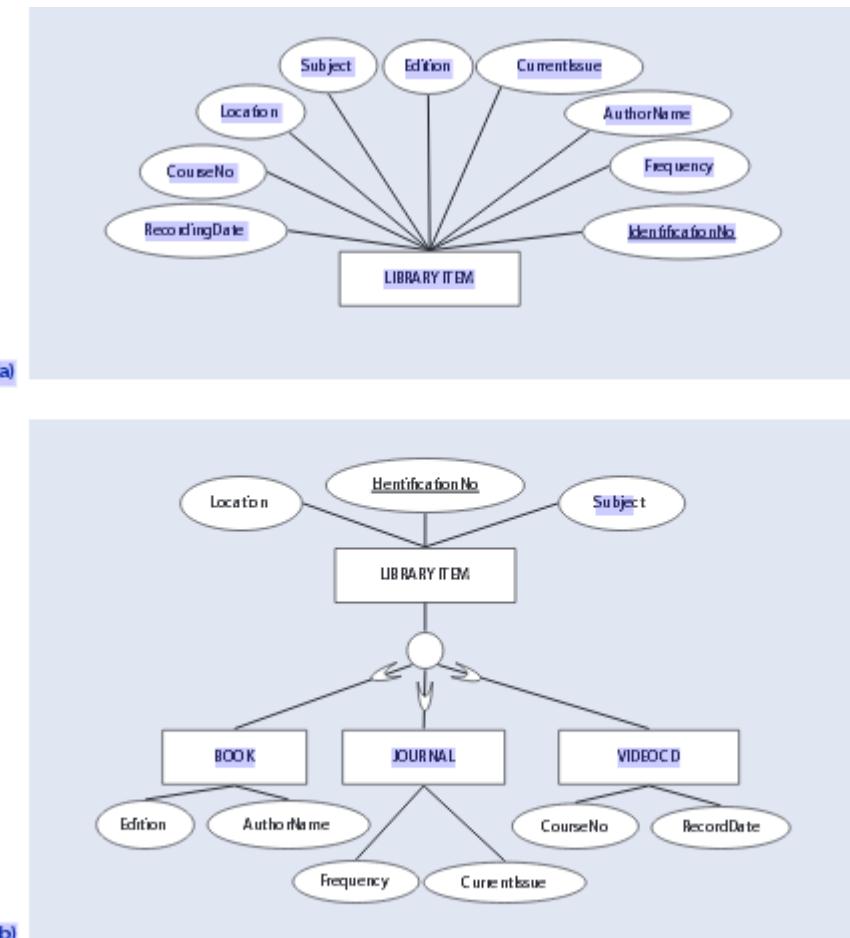


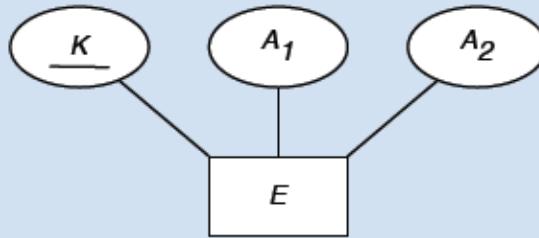
Figure (a) LIBRARY ITEM entity before specialization; (b) LIBRARY ITEM superclass and BOOK, JOURNAL, and VIDEOCD subclasses after specialization.

Constraints

Keys

ER key

A **key** K of an **entity type** E is an attribute of E which **uniquely identifies** the entities of this type. No two different entities share the same value for the key attribute, i.e., $I(K)$ is injective for all DB states I . **Composite** keys are allowed.



Weak Entities

• In many schemas, some entities describe a kind of detail that cannot exist without a master (or owner) entity.

- ▶ If entities are part of an IS-A hierarchy
- ▶ If entities are association entities that have been created to eliminate n-ary relationships.
- ▶ If the existence of the detail entity (e.g., a room) requires the existence of the master entity (e.g., a building).

• In such a case,

1. There is a relationship with cardinality (1, 1) on the detail entity side, and in addition
2. The key of the master is inherited and becomes part of the key of the detail entity.

• Without a specific ER construct for this case, we would require the following additional constraint:

- ▶ If two entities are in "has" relationship, e.g., an invoice and a position,
- ▶ Then their attribute "Inv No" are required to have identical values. For example, invoice #12 cannot have position 2 in invoice #42 as detail.
- Such constraints occur if an entity does not have a key by itself, but it is only unique in the context of some other (master) entity.

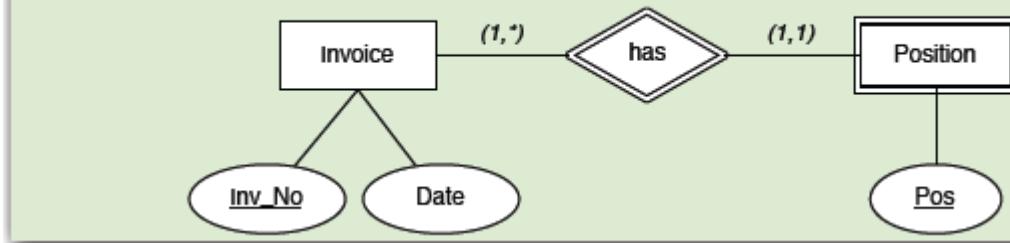
• Note: In such cases, keys are always composite.

• Examples:

- ▶ A classroom is identified by a building and a room number.
- ▶ A section in a book is identified by a chapter and a section title.
- ▶ A web page URI is composed of a web server DNS address and a path on that server.

- There is also an **existence dependency**. For instance, if the building is pulled down, the classrooms automatically disappear. If the web server is shut down, all URIs on that server cease to function.
- In the ER model, such scenarios are modeled via **weak entities** (non weak entities are also called strong entities).
- In ER diagrams, weak entities and their identifying relationships are indicated by **double lines**.
- For the weak entity, the **inherited part of the key is not shown**.
- The relationship type between the weak entity and the **identifying owner** is called **identifying relationship** and always describes a 1:N relationship .

Notation of weak entities

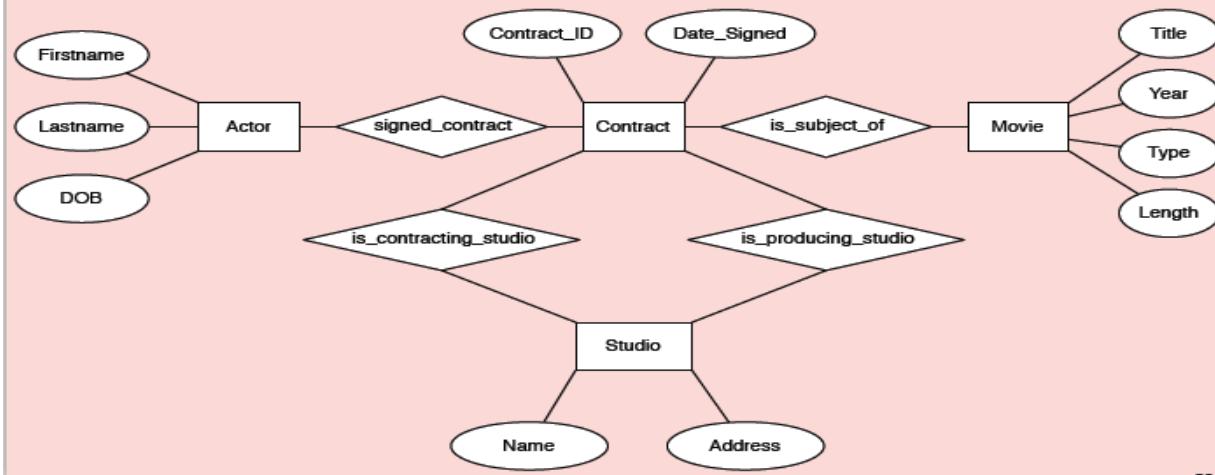


- Usage of weak entities is required only if the **key of one entity contains the key of a related entity**. A weak entity adds further key attributes to the master key.
- Weak entities are normal entities except that their key is constructed in a special way.
- Weak entities may themselves be masters of other weak entities (building an entire hierarchy of dependencies).

Weak Entities in Movie Example

Weak entities in our movie scenario

Assuming keys and relationships identified earlier, do we have weak entities here? If not, what would need to be changed to obtain at least one?



4.2.4.5 Lecture -5

SQL PPT



S Q L

SQL CREATE TABLE Statement

The CREATE TABLE statement is used to create a table in a database.

Tables are organized into rows and columns; and each table must have a name.

SQL CREATE TABLE Syntax

```
CREATE TABLE table_name
(
  column_name1 data_type(size),
  column_name2 data_type(size),
  column_name3 data_type(size),
  ....
);
```

The column_name parameters specify the names of the columns of the table.

The data_type parameter specifies what type of data the column can hold (e.g. varchar, integer, decimal, date, etc.).

The size parameter specifies the maximum length of the column of the table.

Tip: For an overview of the data types available in MS Access, MySQL, and SQL Server, go to our complete [Data Types Reference](#).

SQL CREATE TABLE Example

Now we want to create a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City.

We use the following CREATE TABLE statement:

Example

```
CREATE TABLE Persons
(
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

The PersonID column is of type int and will hold an integer.

The LastName, FirstName, Address, and City columns are of type varchar and will hold characters, and the maximum length for these fields is 255 characters.

The empty "Persons" table will now look like this:

| PersonID | LastName | FirstName | Address | City |
|----------|----------|-----------|---------|------|
| | | | | |

4.2.4.6 Lecture -6

SQL Constraints

SQL constraints are used to specify rules for the data in a table.

If there is any violation between the constraint and the data action, the action is aborted by the constraint.

Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).

SQL CREATE TABLE + CONSTRAINT Syntax

```
CREATE TABLE table_name
(
column_name1 data_type(size) constraint_name,
column_name2 data_type(size) constraint_name,
column_name3 data_type(size) constraint_name,
....;
);
```

In SQL, we have the following constraints:

- **NOT NULL** - Indicates that a column cannot store NULL value
- **UNIQUE** - Ensures that each row for a column must have a unique value
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have a unique identity which helps to find a particular record in a table more easily and quickly
- **FOREIGN KEY** - Ensure the referential integrity of the data in one table to match values in another table
- **CHECK** - Ensures that the value in a column meets a specific condition
- **DEFAULT** - Specifies a default value for a column

SQL UNIQUE Constraint

The UNIQUE constraint uniquely identifies each record in a database table.

The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.

Note that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

SQL UNIQUE Constraint on CREATE TABLE

The following SQL creates a UNIQUE constraint on the "P_Id" column when the "Persons" table is created:

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
    P_Id int NOT NULL UNIQUE,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
)
```

SQL PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a database table.

Primary keys must contain UNIQUE values.

A primary key column cannot contain NULL values.

Most tables should have a primary key, and each table can have only ONE primary key.

SQL PRIMARY KEY Constraint on CREATE TABLE

The following SQL creates a PRIMARY KEY on the "P_Id" column when the "Persons" table is created:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
    P_Id int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255),
```

```
CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,LastName)
)
```

Note: In the example above there is only ONE PRIMARY KEY (pk_PersonID). However, the VALUE of the primary key is made up of TWO COLUMNS (P_Id + LastName).

SQL FOREIGN KEY Constraint

A FOREIGN KEY in one table points to a PRIMARY KEY in another table.

Let's illustrate the foreign key with an example. Look at the following two tables:

The "Persons" table:

| P_Id | LastName | FirstName | Address | City |
|------|-----------|-----------|--------------|-----------|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

The "Orders" table:

| O_Id | OrderNo | P_Id |
|------|---------|------|
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 2 |
| 4 | 24562 | 1 |

Note that the "P_Id" column in the "Orders" table points to the "P_Id" column in the "Persons" table.

The "P_Id" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "P_Id" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

SQL FOREIGN KEY Constraint on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "P_Id" column when the "Orders" table is created:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Orders
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
PRIMARY KEY (O_Id),
CONSTRAINT fk_PerOrders FOREIGN KEY (P_Id)
REFERENCES Persons(P_Id)
)
```

SQL CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a single column it allows only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

SQL CHECK Constraint on CREATE TABLE

The following SQL creates a CHECK constraint on the "P_Id" column when the "Persons" table is created. The CHECK constraint specifies that the column "P_Id" must only include integers greater than 0.

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
```

```

P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT chk_Person CHECK (P_Id>0 AND City='Sandnes')
)

```

4.2.4.7 Lecture -7

Introduction to Nested Queries (Sub query)

A nested query is a query that has another query embedded within it; the embedded query is called a sub query. The embedded query can of course be a nested query itself.

- A **Scalar Sub query** returns a single column and a single row; i.e., is a single value. In principal, a scalar sub query can be used whenever a single value is needed.
- A **row sub query** returns multiple columns, but again only a single row. A row sub query can be used whenever a row value constructor is needed.
- A **table sub query** returns one or more columns and multiple rows. A table sub query can be used whenever a table is needed.

Examples

Q. Find the names of sailors who have reserved boat 103.

A. Select S.sname from Sailors S where S.sid in (Select R.sid from Reserves R where R.bid=103);

Q. Find the names of Sailors who have reserved a red boat.

A. Select S.sname from Sailors S where S.sid in (select R.sid from Reserves R where R.bid in (select b.bid from Boats b where b.color='red'));

Q. Find the names of Sailors who have not reserved a red boat.

A. Select S.sname from Sailors S where S.sid NOT IN (select R.sid from Reserves R where R.bid in (select b.bid from Boats b where b.color='red'));

Q. Find the details of student who have paid the highest amount so far in their course.

A. select sno, sname, course, totalfee, feepaid from students where (course, feepaid) in (select course, max (feepaid) from students group by course);

Q. Find the details of students whose TOTALFEE is more than average TOTALFEE.

A. select sno, sname from students where totalfee > (select avg (totalfee) from students).

Correlated Nested Queries

A Correlated sub query is a nested query that receives a value from the main query and then sends a value back to main query.

Q. Find the details of student whose total fee is more than the average total fee of their course. That means, if totalfee of a student is 10000 and the average totalfee of his/her course is less than 10000 then the details of that student are to be displayed.

A. select sno, sname, totalfee, course from students x where totalfee > (select avg (totalfee) from students where course=x.course);

Set-Comparison Operators

The following list gives the Set-Comparison Operators

- EXISTS
- NOT EXISTS
- IN
- UNIQUE
- ANY
- ALL

EXISTS and NOT EXISTS operators

These two operators are exclusively used in correlated sub query. EXISTS checks whether any row is existing in the sub query, and NOT EXISTS does the opposite.

EXISTS is different from other operators like IN, ANY etc. because it doesn't compare values of columns, instead, it checks whether any row is retrieved from sub query or not. If any row is retrieved from sub query the EXISTS

returns true otherwise it returns false.

The following will select student who have paid at least one instalment.

Select sno, sname from students where exists (select sno from instalments where students.sno=sno);

IN operator

It Checks a single value against a list of values.

Q. Find the details of subjects if version is either 3.0 or 5.0 or 6.0

A. select subcode, subdesc, version from subjects where version in ('3.0','5.0','6.0');

UNIQUE

It is closely related to EXISTS is the UNIQUE predicate. When we apply UNIQUE to a subquery, the resulting condition returns true if no row appears twice in the answer to the subquery, that is, there are no duplicates; in particular, it returns true if the answer is empty.

ANY and ALL Operators

Both are used for comparing one value against a set of values. ALL specifies that all the values given in the list should be taken into account, whereas ANY specifies the condition be satisfied when any of the satisfied the condition.

SYNTAX

Operator ANY list

Operator ALL list

The operator can be any one of the standard relational operators (=,>=,>,<,<=,!)=) and list is a series of values.

Example

Q. Find the sailors with highest rating.

A. select S.sid from sailors S where S.rating >= ALL (select S2.rating from sailors S2);

Select sno, sname, totalfee, feepaid from students where feepaid > ANY

(select feepaid from students);

The following list illustrates the result of ANY and ALL operator.

Rate ANY/ALL operator Result

10

10

10

10

Rate > Any (15, 20)

Rate > ANY (5, 15)

Rate > ALL (10, 20)

Rate > ALL (5, 7)

False

True

False

True

| Subquery | Correlated Subquery |
|--|--|
| Executed only for once before main-query | Executed once for each row of main-query |
| Sends a value to main-query | Receives a value from main query and sends a value to main-query |

4.2.4.8 Lecture -8

SQL Aggregate Functions

SQL aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:

- `AVG()` - Returns the average value
- `COUNT()` - Returns the number of rows
- `FIRST()` - Returns the first value
- `LAST()` - Returns the last value
- `MAX()` - Returns the largest value
- `MIN()` - Returns the smallest value
- `SUM()` - Returns the sum

The `AVG()` Function

The `AVG()` function returns the average value of a numeric column.

SQL `AVG()` Syntax

```
SELECT AVG(column_name) FROM table_name
```

Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Products" table:

| ProductID | ProductName | SupplierID | CategoryID | Unit | Price |
|-----------|------------------------------|------------|------------|---------------------|-------|
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 - 12 oz bottles | 19 |
| 3 | Aniseed Syrup | 1 | 2 | 12 - 550 ml bottles | 10 |
| 4 | Chef Anton's Cajun Seasoning | 2 | 2 | 48 - 6 oz jars | 21.35 |
| 5 | Chef Anton's Gumbo Mix | 2 | 2 | 36 boxes | 25 |

SQL `AVG()` Example

The following SQL statement gets the average value of the "Price" column from the "Products" table:

Example

```
SELECT AVG(Price) AS PriceAverage FROM Products;
```

SQL `COUNT(column_name)` Syntax

The `COUNT(column_name)` function returns the number of values (NULL values will not be counted) of the specified column:

```
SELECT COUNT(column_name) FROM table_name;
```

SQL COUNT(*) Syntax

The COUNT(*) function returns the number of records in a table:

```
SELECT COUNT(*) FROM table_name;
```

SQL COUNT(DISTINCT column_name) Syntax

The COUNT(DISTINCT column_name) function returns the number of distinct values of the specified column:

```
SELECT COUNT(DISTINCT column_name) FROM table_name;
```

Note: COUNT(DISTINCT) works with ORACLE and Microsoft SQL Server, but not with Microsoft Access.

Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Orders" table:

| OrderID | CustomerID | EmployeeID | OrderDate | ShipperID |
|---------|------------|------------|------------|-----------|
| 10265 | 7 | 2 | 1996-07-25 | 1 |
| 10266 | 87 | 3 | 1996-07-26 | 3 |
| 10267 | 25 | 4 | 1996-07-29 | 1 |

SQL COUNT(column_name) Example

The following SQL statement counts the number of orders from "CustomerID"=7 from the "Orders" table:

Example

```
SELECT COUNT(CustomerID) AS OrdersFromCustomerID7 FROM Orders  
WHERE CustomerID=7;
```

The MAX() Function

The MAX() function returns the largest value of the selected column.

SQL MAX() Syntax

```
SELECT MAX(column_name) FROM table_name;
```

Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Products" table:

| ProductID | ProductName | SupplierID | CategoryID | Unit | Price |
|-----------|------------------------------|------------|------------|---------------------|-------|
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 - 12 oz bottles | 19 |
| 3 | Aniseed Syrup | 1 | 2 | 12 - 550 ml bottles | 10 |
| 4 | Chef Anton's Cajun Seasoning | 2 | 2 | 48 - 6 oz jars | 21.35 |
| 5 | Chef Anton's Gumbo Mix | 2 | 2 | 36 boxes | 25 |

SQL MAX() Example

The following SQL statement gets the largest value of the "Price" column from the "Products" table:

Example

```
SELECT MAX(Price) AS HighestPrice FROM Products;
```

The MIN() Function

The MIN() function returns the smallest value of the selected column.

SQL MIN() Syntax

```
SELECT MIN(column_name) FROM table_name;
```

Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Products" table:

| ProductID | ProductName | SupplierID | CategoryID | Unit | Price |
|-----------|------------------------------|------------|------------|---------------------|-------|
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 - 12 oz bottles | 19 |
| 3 | Aniseed Syrup | 1 | 2 | 12 - 550 ml bottles | 10 |
| 4 | Chef Anton's Cajun Seasoning | 2 | 2 | 48 - 6 oz jars | 21.35 |
| 5 | Chef Anton's Gumbo Mix | 2 | 2 | 36 boxes | 25 |

SQL MIN() Example

The following SQL statement gets the smallest value of the "Price" column from the "Products" table:

Example

```
SELECT MIN(Price) AS SmallestOrderPrice FROM Products;
```

The SUM() Function

The SUM() function returns the total sum of a numeric column.

SQL SUM() Syntax

```
SELECT SUM(column_name) FROM table_name;
```

Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "OrderDetails" table:

| OrderDetailID | OrderID | ProductID | Quantity |
|---------------|---------|-----------|----------|
| 1 | 10248 | 11 | 12 |
| 2 | 10248 | 42 | 10 |
| 3 | 10248 | 72 | 5 |
| 4 | 10249 | 14 | 9 |
| 5 | 10249 | 51 | 40 |

SQL SUM() Example

The following SQL statement finds the sum of all the "Quantity" fields for the "OrderDetails" table:

Example

```
SELECT SUM(Quantity) AS TotalItemsOrdered FROM OrderDetails;
```

The GROUP BY Statement

The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

SQL GROUP BY Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name;
```

Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Orders" table:

| OrderID | CustomerID | EmployeeID | OrderDate | ShipperID |
|---------|------------|------------|------------|-----------|
| 10248 | 90 | 5 | 1996-07-04 | 3 |
| 10249 | 81 | 6 | 1996-07-05 | 1 |
| 10250 | 34 | 4 | 1996-07-08 | 2 |

And a selection from the "Shippers" table:

| ShipperID | ShipperName | Phone |
|-----------|------------------|----------------|
| 1 | Speedy Express | (503) 555-9831 |
| 2 | United Package | (503) 555-3199 |
| 3 | Federal Shipping | (503) 555-9931 |

And a selection from the "Employees" table:

| EmployeeID | LastName | FirstName | BirthDate | Photo | Notes |
|------------|-----------|-----------|------------|------------|-----------------------------|
| 1 | Davolio | Nancy | 1968-12-08 | EmpID1.pic | Education includes a BA.... |
| 2 | Fuller | Andrew | 1952-02-19 | EmpID2.pic | Andrew received his BTS.... |
| 3 | Leverling | Janet | 1963-08-30 | EmpID3.pic | Janet has a BS degree.... |

SQL GROUP BY Example

Now we want to find the number of orders sent by each shipper.

The following SQL statement counts as orders grouped by shippers:

Example

```
SELECT Shippers.ShipperName,COUNT(Orders.OrderID) AS
NumberOfOrders FROM Orders
LEFT JOIN Shippers
ON Orders.ShipperID=Shippers.ShipperID
GROUP BY ShipperName;
```

The HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

SQL HAVING Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value;
```

Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Orders" table:

| OrderID | CustomerID | EmployeeID | OrderDate | ShipperID |
|---------|------------|------------|------------|-----------|
| 10248 | 90 | 5 | 1996-07-04 | 3 |
| 10249 | 81 | 6 | 1996-07-05 | 1 |
| 10250 | 34 | 4 | 1996-07-08 | 2 |

And a selection from the "Employees" table:

| EmployeeID | LastName | FirstName | BirthDate | Photo | Notes |
|------------|-----------|-----------|------------|------------|-----------------------------|
| 1 | Davolio | Nancy | 1968-12-08 | EmpID1.pic | Education includes a BA.... |
| 2 | Fuller | Andrew | 1952-02-19 | EmpID2.pic | Andrew received his BTS.... |
| 3 | Leverling | Janet | 1963-08-30 | EmpID3.pic | Janet has a BS degree.... |

SQL HAVING Example

Now we want to find if any of the employees has registered more than 10 orders.

We use the following SQL statement:

Example

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees
ON Orders.EmployeeID=Employees.EmployeeID)
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 10;
```

The SQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set by one or more columns.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in a descending order, you can use the DESC keyword.

SQL ORDER BY Syntax

```
SELECT column_name, column_name
FROM table_name
ORDER BY column_name ASC|DESC, column_name ASC|DESC;
```

Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

| CustomerID | CustomerName | ContactName | Address | City | PostalCode | Country |
|------------|------------------------------------|--------------------|-------------------------------|-------------|------------|---------|
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

ORDER BY Example

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" column:

Example

```
SELECT * FROM Customers
ORDER BY Country;
```

ORDER BY DESC Example

The following SQL statement selects all customers from the "Customers" table, sorted DESCENDING by the "Country" column:

Example

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

ORDER BY Several Columns Example

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column:

Example

```
SELECT * FROM Customers  
ORDER BY Country, CustomerName;
```

ORDER BY Several Columns Example 2

The following SQL statement selects all customers from the "Customers" table, sorted ascending by the "Country" and descending by the "CustomerName" column:

Example

```
SELECT * FROM Customers  
ORDER BY Country ASC, CustomerName DESC;
```

4.2.4.9 Lecture -9

Implementation of different types of joins

SQL JOIN

An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.

The most common type of join is: **SQL INNER JOIN (simple join)**. An SQL INNER JOIN returns all rows from multiple tables where the join condition is met.

Let's look at a selection from the "Orders" table:

| OrderID | CustomerID | OrderDate |
|---------|------------|------------|
| 10308 | 2 | 1996-09-18 |
| 10309 | 37 | 1996-09-19 |
| 10310 | 77 | 1996-09-20 |

Then, have a look at a selection from the "Customers" table:

| CustomerID | CustomerName | ContactName | Country |
|------------|------------------------------------|----------------|---------|
| 1 | Alfreds Futterkiste | Maria Anders | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mexico |

Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

Then, if we run the following SQL statement (that contains an INNER JOIN):

Example

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers
ON Orders.CustomerID=Customers.CustomerID;
```

Different SQL JOINS

Before we continue with examples, we will list the types of the different SQL JOINS you can use:

- **INNER JOIN:** Returns all rows when there is at least one match in BOTH tables
- **LEFT JOIN:** Return all rows from the left table, and the matched rows from the right table
- **RIGHT JOIN:** Return all rows from the right table, and the matched rows from the left table
- **FULL JOIN:** Return all rows when there is a match in ONE of the tables

SQL INNER JOIN Keyword

The INNER JOIN keyword selects all rows from both tables as long as there

is a match between the columns in both tables.

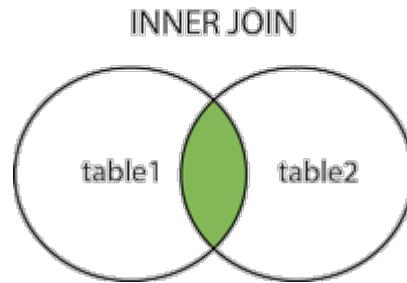
SQL INNER JOIN Syntax

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name=table2.column_name;
```

or:

```
SELECT column_name(s)
FROM table1
JOIN table2
ON table1.column_name=table2.column_name;
```

PS! INNER JOIN is the same as JOIN.



Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

| CustomerID | CustomerName | ContactName | Address | City | PostalCode | Country |
|------------|---------------------------------------|-------------------|----------------------------------|----------------|------------|---------|
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

And a selection from the "Orders" table:

| OrderID | CustomerID | EmployeeID | OrderDate | ShipperID |
|---------|------------|------------|------------|-----------|
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

SQL INNER JOIN Example

The following SQL statement will return all customers with orders:

Example

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

SQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.

SQL LEFT JOIN Syntax

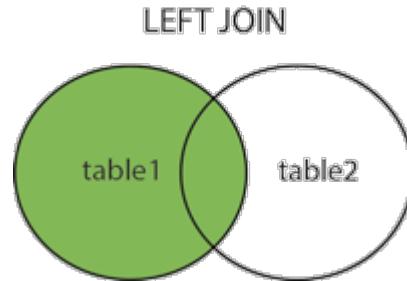
```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name=table2.column_name;
```

or:

```
SELECT column_name(s)
FROM table1
```

```
LEFT OUTER JOIN table2  
ON table1.column_name=table2.column_name;
```

PS! In some databases LEFT JOIN is called LEFT OUTER JOIN.



Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

| CustomerID | CustomerName | ContactName | Address | City | PostalCode | Country |
|------------|---------------------------------------|-------------------|----------------------------------|----------------|------------|---------|
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

And a selection from the "Orders" table:

| OrderID | CustomerID | EmployeeID | OrderDate | ShipperID |
|---------|------------|------------|------------|-----------|
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

SQL LEFT JOIN Example

The following SQL statement will return all customers, and any orders they might have:

Example

```
SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
LEFT JOIN Orders  
ON Customers.CustomerID=Orders.CustomerID  
ORDER BY Customers.CustomerName;
```

SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.

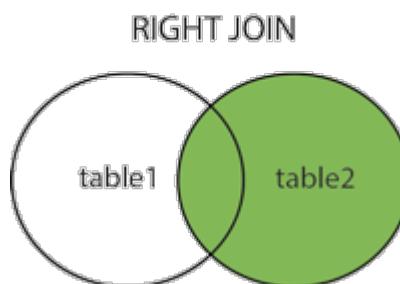
SQL RIGHT JOIN Syntax

```
SELECT column_name(s)  
FROM table1  
RIGHT JOIN table2  
ON table1.column_name=table2.column_name;
```

or:

```
SELECT column_name(s)  
FROM table1  
RIGHT OUTER JOIN table2  
ON table1.column_name=table2.column_name;
```

PS! In some databases RIGHT JOIN is called RIGHT OUTER JOIN.



Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Orders" table:

| OrderID | CustomerID | EmployeeID | OrderDate | ShipperID |
|---------|------------|------------|------------|-----------|
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

And a selection from the "Employees" table:

| EmployeeID | LastName | FirstName | BirthDate | Photo | Notes |
|------------|-----------|-----------|-----------|------------|--|
| 1 | Davolio | Nancy | 12/8/1968 | EmpID1.pic | Education includes a BA in psychology..... |
| 2 | Fuller | Andrew | 2/19/1952 | EmpID2.pic | Andrew received his BTS commercial and.... |
| 3 | Leverling | Janet | 8/30/1963 | EmpID3.pic | Janet has a BS degree in chemistry.... |

SQL RIGHT JOIN Example

The following SQL statement will return all employees, and any orders they have placed:

Example

```
SELECT Orders.OrderID, Employees.FirstName
FROM Orders
RIGHT JOIN Employees
ON Orders.EmployeeID=Employees.EmployeeID
ORDER BY Orders.OrderID;
```

SQL FULL OUTER JOIN Keyword

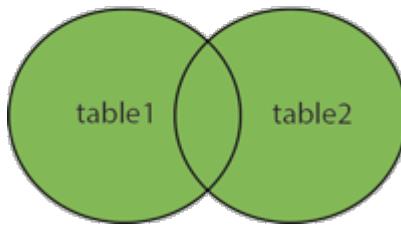
The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).

The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.

SQL FULL OUTER JOIN Syntax

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

FULL OUTER JOIN



Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

| CustomerID | CustomerName | ContactName | Address | City | PostalCode | Country |
|------------|---------------------------------------|-------------------|----------------------------------|----------------|------------|---------|
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

And a selection from the "Orders" table:

| OrderID | CustomerID | EmployeeID | OrderDate | ShipperID |
|---------|------------|------------|------------|-----------|
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

SQL FULL OUTER JOIN Example

The following SQL statement selects all customers, and all orders:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

A selection from the result set may look like this:

| CustomerName | OrderID |
|------------------------------------|---------|
| Alfreds Futterkiste | |
| Ana Trujillo Emparedados y helados | 10308 |
| Antonio Moreno Taquería | 10365 |
| | 10382 |
| | 10351 |

4.2.4.10 Lecture -10

View (updatable and non-updatable), relational set operations

A view is a virtual table.

This chapter shows how to create, update, and delete a view.

SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

SQL CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

Note: A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

SQL CREATE VIEW Examples

If you have the Northwind database you can see that it has several views installed by default.

The view "Current Product List" lists all active products (products that are not discontinued) from the "Products" table. The view is created with the following SQL:

```
CREATE VIEW [Current Product List] AS  
SELECT ProductID,ProductName  
FROM Products  
WHERE Discontinued=No
```

We can query the view above as follows:

```
SELECT * FROM [Current Product List]
```

Another view in the Northwind sample database selects every product in the "Products" table with a unit price higher than the average unit price:

```
CREATE VIEW [Products Above Average Price] AS  
SELECT ProductName,UnitPrice  
FROM Products  
WHERE UnitPrice > (SELECT AVG(UnitPrice) FROM Products)
```

We can query the view above as follows:

```
SELECT * FROM [Products Above Average Price]
```

Another view in the Northwind database calculates the total sale for each category in 1997. Note that this view selects its data from another view called "Product Sales for 1997":

```
CREATE VIEW [Category Sales For 1997] AS  
SELECT DISTINCT CategoryName,Sum(ProductSales) AS CategorySales  
FROM [Product Sales for 1997]  
GROUP BY CategoryName
```

We can query the view above as follows:

```
SELECT * FROM [Category Sales For 1997]
```

We can also add a condition to the query. Now we want to see the total sale only for the category "Beverages":

```
SELECT * FROM [Category Sales For 1997]
WHERE CategoryName='Beverages'
```

SQL Updating a View

You can update a view by using the following syntax:

SQL CREATE OR REPLACE VIEW Syntax

```
CREATE OR REPLACE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

Now we want to add the "Category" column to the "Current Product List" view. We will update the view with the following SQL:

```
CREATE OR REPLACE VIEW [Current Product List] AS
SELECT ProductID,ProductName,Category
FROM Products
WHERE Discontinued='No'
```

SQL Dropping a View

You can delete a view with the DROP VIEW command.

SQL DROP VIEW Syntax

```
DROP VIEW view_name
```

The SQL UNION operator combines the result of two or more SELECT statements.

The SQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

Notice that each SELECT statement within the UNION must have the same number of columns. The columns must also have similar data types. Also, the columns in each SELECT statement must be in the same order.

SQL UNION Syntax

```
SELECT column_name(s) FROM table1  
UNION  
SELECT column_name(s) FROM table2;
```

Note: The UNION operator selects only distinct values by default. To allow duplicate values, use the ALL keyword with UNION.

SQL UNION ALL Syntax

```
SELECT column_name(s) FROM table1  
UNION ALL  
SELECT column_name(s) FROM table2;
```

PS: The column names in the result-set of a UNION are usually equal to the column names in the first SELECT statement in the UNION.

Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

| CustomerID | CustomerName | ContactName | Address | City | PostalCode | Country |
|------------|---------------------------------------|-------------------|----------------------------------|----------------|------------|---------|
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

And a selection from the "Suppliers" table:

| SupplierID | SupplierName | ContactName | Address | City | PostalCode | Country |
|------------|-------------------------------|---------------------|-------------------|----------------|------------|---------|
| 1 | Exotic Liquid | Charlotte Cooper | 49 Gilbert St. | London | EC1 4SD | UK |
| 2 | New Orleans Cajun Delights | Shelley Burke | P.O. Box 78934 | New Orleans | 70117 | USA |
| 3 | Grandma Kelly's Homestead | Regina Murphy | 707 Oxford Rd. | Ann Arbor | 48104 | USA |

SQL UNION Example

The following SQL statement selects all the **different** cities (only distinct values) from the "Customers" and the "Suppliers" tables:

Example

```
SELECT City FROM Customers  
UNION  
SELECT City FROM Suppliers  
ORDER BY City;
```

SQL UNION ALL Example

The following SQL statement uses UNION ALL to select **all** (duplicate values also) cities from the "Customers" and "Suppliers" tables:

Example

```
SELECT City FROM Customers  
UNION ALL  
SELECT City FROM Suppliers  
ORDER BY City;
```

SQL UNION ALL With WHERE

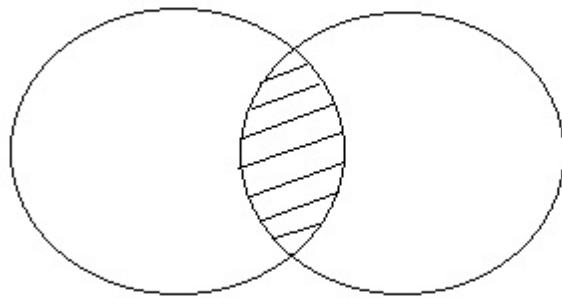
The following SQL statement uses UNION ALL to select **all** (duplicate values also) **German** cities from the "Customers" and "Suppliers" tables:

Example

```
SELECT City, Country FROM Customers  
WHERE Country='Germany'  
UNION ALL  
SELECT City, Country FROM Suppliers  
WHERE Country='Germany'  
ORDER BY City;
```

Intersect

Intersect operation is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements. In case of **Intersect** the number of columns and datatype must be same. MySQL does not support INTERSECT operator.



Example of Intersect

The **First** table,

| ID | NAME |
|----|------|
| 1 | abhi |
| 2 | adam |

The **Second** table,

| ID | NAME |
|----|---------|
| 2 | adam |
| 3 | Chester |

Intersect query will be,

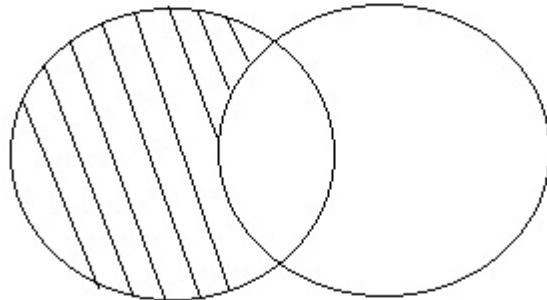
```
select * from First  
INTERSECT  
select * from second
```

The result table will look like

| ID | NAME |
|----|------|
| 2 | adam |

Minus

Minus operation combines result of two Select statements and return only those result which belongs to first set of result. MySQL does not support INTERSECT operator.



Example of Minus

The **First** table,

| ID | NAME |
|----|------|
| 1 | abhi |
| 2 | adam |

The **Second** table,

| ID | NAME |
|----|---------|
| 2 | adam |
| 3 | Chester |

Minus query will be,

```
select * from First
MINUS
select * from second
```

The result table will look like,

| ID | NAME |
|----|------|
| 1 | abhi |

4.3.5. Test Questions

a. *Fill in the blanks type of questions*

1. An attribute which can be used to uniquely identify the individual instances of the entity is ____.
2. A relationship that has a cardinality constraint depicted as is named ____.
3. An attribute whose value can be calculated from related attribute values is ____.
4. A relationship that relates instances of an entity to instances of the same entity is ____.
5. The relationship between a weak entity type and its owner is called ____.
6. E-R model was introduced by _____ in _____.
7. A relationship is an _____ among two or more _____ that is of interest to the enterprise.
8. The database model uses the _____, _____ and _____ to construct representation of the real world system.
9. The relationship is joined by _____ to the entities that participate in the relationship.
10. An association among three entities is called _____.
11. A relationship between the instances of a single entity type is called _____.
_____.
12. the association between the two entities is called _____.
13. The actual count of elements associated with the connectivity is called _____ of the relationship connectivity.
14. An attribute is a property of _____ or a _____ type.

15. The components of an entity or the qualifiers that describe it are called _____ of the entity.

b. Multiple choice questions

1. Which of the following indicates the maximum number of entities that can be involved in a relationship?
 - a. Minimum cardinality
 - b. Maximum cardinality
 - c. ERD
 - d. Greater Entity Count (GEC)
2. What is specified as an association among several entities?
a) Relationship b) Entity c) Entity Set d) Relationship Set

3. Which type of entity cannot exist in the database unless another type of entity also exists in the database, but does not require that the identifier of that other entity be included as part of its own identifier?

- a) Weak entity
- b) Strong entity
- c) ID-dependent entity
- d) ID-independent entity

In a one-to-many relationship, the entity that is on the one side of the relationship
4. is called a(n)

_____ entity.

- a) parent
- b) child
- c) instance
- d) subtype

Which type of entity represents an actual occurrence of an associated
5. generalized entity?

- a) Super type entity
- b) Subtype entity
- c) Archetype entity
- d) Instance entity

6. A recursive relationship is a relationship between an entity and _____.

- a) itself
- b) a subtype entity

-
- c) an archetype entity
 - d) an instance entity

7. The entity set that participates in a relationship are
- a) distinct b) need not be distinct
may or may not be distinct d) none
8. In which of the following is a single-entity instance of one type related to many entity instances of another type?
- a) One-to-One Relationship b) One-to-Many Relationship
c) Many-to-Many Relationship d) Composite Relationship
9. Properties that describe the characteristics of entities are called:
- a) entities b) attributes c) Identifiers d) relationships.
10. Entities can be associated with one another in which of the following?
- a) Entities b) Attributes c) Identifiers d) Relationships
11. Pick the relationship from the following:
- a) a classroom b) teacher c) attends d) cost per dozen
12. Pick the meaningful relationship between entities
- a) vendor supplies goods b) vendor talks with customers
c) vendor complains to vendor d) vendor asks prices
13. The entity set is a
- a) set of entities b) collection of different entities
c) collection of related entities d) collection of similar entities
14. Pick entity set from the following
- a) all vendors supplying to an organization b) vendors and organizations they supply
c) vendors and d) a vendor supplying to many

transporters

organizations

15. Attributes are

- (i) properties of relationship
 - (ii) attributed to entities
 - (iii) properties of members of an entity set
- a) I b) i and ii c) i and iii d) iii

c) True or False questions

1. E-R models are expressed using a single standardized set of universally accepted symbols.
2. A schema is a representation of something of interest to the modeler.
3. An internal schema is a representation of how users view the database.
4. A conceptual schema is a complete logical view of the database.
5. An entity is something in the users' work environment that the users want to track.
6. Entities of a given type are grouped into entity classes.
7. An entity class is described by the structure of the entities in that class.
8. An entity instance of an entity class is the representation of a particular entity and is described by the values of the attributes of the entity.
9. In E-R modeling, entities within an entity class may have different attributes.
10. In E-R modeling, an attribute may be either composite or multi-value, but it cannot be both.
11. An identifier of an entity instance must consist of one and only one attribute.
12. A "composite identifier" is defined as a composite attribute that is an identifier.
13. An identifier may be either unique or non-unique.
14. E-R modeling recognizes both relationship classes and relationship instances.
15. Relationships do not have attributes.
16. A single relationship class involves only one entity class.

4.3.6. Review Questions

d.Objective type of questions(Very short notes)

1. You have been given a set of tables with data and asked to create a

new database to store them. When you examine the data values in the tables, what are you looking for?

2. What are descriptive attributes?
3. What do you mean by arity (predicate) of a relationship?
4. What do you mean by participation constraints?
5. What is extension and intension?
6. What is DML Compiler?
7. What is query evaluation engine?
8. What is DDL Interpreter?
9. How is a super key different from candidate key?
10. What is domain constraint?
11. How can we define the primary key of a weak entity set?
12. Define: Identifying relationship
13. Why is a database considered to be "self-describing"?

e. Analytical type questions

1. Draw an E_R Model for the following:

An organization uses number of items of a equipment to produce goods. Each item is at one LOCATION, of one TYPE and has a DETAILED_DESCRIPTION. Faults on the equipment are identified by a unique FAULT_ID and are reported at a TIME_REPORTED. Any number of persons may be assigned to a fault and work on the fault until it is fixed. The TIME_FIXED is recorded as is the TIME_SPENT by each person on a fault. Any number of parts may be used to repair a fault. The QTY_USED of each part is recorded against the fault. Each part is identified by a PART_ID and has a given weight and MAX_DIMENSION and can have any number of colors.

2. What is an E-R model? Draw an E-R Diagram for the company database with following descriptions:

The company is organized into departments. Each department has a unique name and a unique number with several locations. A department controls a number of projects, each of which has a unique name, unique number and a single location. We store each employees name, social security number, address, and salary. An employee is assigned to one department but may work on several projects, which are not necessarily controlled by the same departments. We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's name, age and relationship to the employee.

3. Discuss the correspondence between the E-R model construct and the relation model construct. Show how each E-R model construct can be mapped to the relational model using the suitable example?

f. Essay type Questions

1. Which are the different types of relationships? Give an example for each.
2. Explain all four types of mapping cardinality with appropriate examples.
3. Explain the terms primary key, candidate key, alternate key and secondary key. In the given table identify each key.
STUDENT(SID, Regno, Name, City)
4. How strong and weak entity set are different from each other?
5. Describe various relationship constraints by giving suitable examples.
6. Explain the concept of keys in relational model and specify its importance.
7. Show with example the transformation of an E-R diagram into database design.
8. Give an appropriate example of total participation with proper explanation.
9. Explain the concept of generalization and aggregation in E-R diagrams. Give one example for each one of them.

g. Problems

h. Case study

1. Design of a Hospital-Based Database System (A Case Study of BIRDEM)

Design of a Hospital-Based Database System (A Case Study of BIRDEM)

Rosina Surovi Khan

Department of Computer Science and Engineering
Ahsanullah University of Science and Technology (AUST)
Dhaka, Bangladesh
Email: surovi99@yahoo.com

Mohammad Saber

Department of Computer Science and Engineering
Ahsanullah University of Science and Technology (AUST)
Dhaka, Bangladesh
Email: saber@aust.edu

Abstract— As technology advances, information in different organizations of Bangladesh can no more be maintained manually. There is a growing need for the information to become computerized so that it can be suitably stored. This is where databases come into the picture. Databases are convenient storage systems which can store large amounts of data and together with application programs such as interfaces they can aid in faster retrieval of data. An initiative was taken to design a complete database system for a hospital management such as Bangladesh Institute of Research and rehabilitation in Diabetes, Endocrine and Metabolic disorders (BIRDEM) in Dhaka so that its information can be stored, maintained, updated and retrieved conveniently and efficiently. The existing information in BIRDEM is partly computerized via databases only in patients' admissions, doctors' appointments and medical tests and reports sections. A partly slow and tedious manual system still exists in BIRDEM for example, in record of ambulances in service, assigning ward boys and nurses to rooms, the billing process and record of doctors' prescriptions etc. However, this paper outlines one complete database design for the entire BIRDEM hospital in which data maintenance and retrieval are in perfect harmony and speedy. Sample SQL-based queries executed on the designed system are also demonstrated.

Keywords- Database system, entity relationship diagram, relational model, normalization, SQL-based queries.

Introduction

Bangladesh Institute of Research and rehabilitation in Diabetes, Endocrine and Metabolic disorders, abbreviated as BIRDEM, is a Research Center for Diabetes and also a hospital. There are doctors, patients, and employees like nurses, ward boys, ambulance carriers which are considered as entities in the designed system. It becomes a tough, tedious and comparatively slow process to store the information partly manually and partly computerized. But having all the necessary information stored in one database, it not only helps in orderly maintenance but very speedy retrieval of data.

The work in this paper has been supervised by Rosina Surovi Khan for her student Syed Mahboob Nur and his group as a thesis work in Year 4, Semester 1, CSE, AUST in Fall 2009.

I. EXTRACTION OF INFORMATION FROM HOSPITAL MANAGEMENT (BIRDEM)

BIRDEM was visited and information was gathered; demos were seen and its websites visited from the internet. Now Ibrahim Medical College is founded in BIRDEM for the MBBS degree. A lot of students pass MBBS from this medical college. [3,4]

A database system was designed based on a case study of BIRDEM Hospital via Entity Relationship Diagram (ERD), Relational Model, Normalization of tables and Implementation in SQL server. [1,2,5] The ERD is outlined below.

II. DATABASE DESIGN

A. Entity Relationship Model (ERM)

1) ER Diagram

In the ER diagram, we can view the entities- Patient, Doctor, Receptionist, Department, Medicine, Test, OT (Operation Theater), Room, Nurse, Ward_boy, Driver, Ambulance, Carriers, Accountant and Bill. Among these entities, relationships exist which connect all the entities in the diagram. For example, Patient, Doctor and Receptionist are connected via the relationship Appointment. In other words, a receptionist will set up a doctor's appointment for a patient. Similarly, Doctor, Patient and Medicine are connected via the relationship Prescription. Here, a doctor may prescribe one or more medicine to a patient. In a similar way, other entities are connected via relationships in a meaningful way. Cardinality ratios [1] for the entities connected to a relationship are explained in the next section.

2) Cardinalities

Binary Relationship

We can see a binary relationship in the ER diagram.

2. Hospital Management System

Database Management System

Case Studies

Case Study 1

Hospital Management System

Aim: XYZ hospital is a multi specialty hospital that includes a number of departments, rooms, doctors, nurses, compounders, and other staff working in the hospital. Patients having different kinds of ailments come to the hospital and get checkup done from the concerned doctors. If required they are admitted in the hospital and discharged after treatment.

The aim of this case study is to design and develop a database for the hospital to maintain the records of various departments, rooms, and doctors in the hospital. It also maintains records of the regular patients, patients admitted in the hospital, the check up of patients done by the doctors, the patients that have been operated, and patients discharged from the hospital.

Description: In hospital, there are many departments like Orthopedic, Pathology, Emergency, Dental, Gynecology, Anesthetics, I.C.U., Blood Bank, Operation Theater, Laboratory, M.R.I., Neurology, Cardiology, Cancer Department, Corpse, etc. There is an OPD where patients come and get a card (that is, entry card of the patient) for check up from the concerned doctor. After making entry in the card, they go to the concerned doctor's room and the doctor checks up their ailments. According to the ailments, the doctor either prescribes medicine or admits the patient in the concerned department. The patient may choose either private or general room according to his/her need. But before getting admission in the hospital, the patient has to fulfill certain formalities of the hospital like room charges, etc. After the treatment is completed, the doctor discharges the patient. Before discharging from the hospital, the patient again has to complete certain formalities of the hospital like balance charges, test charges, operation charges (if any), blood charges, doctors' charges, etc.

Next we talk about the doctors of the hospital. There are two types of the doctors in the hospital, namely, *regular doctors* and *call on doctors*. Regular doctors are those doctors who come to the hospital daily. Calls on doctors are those doctors who are called by the hospital if the concerned doctor is not available.

Table Description:

Following are the tables along with constraints used in *Hospital Management* database.

3. Database Management Systems: A Case Study of Faculty of Open Education

The Turkish Online Journal of Educational Technology – TOJET January 2004 ISSN: 1303-6521 volume 3 Issue 1 Article 3

Database Management Systems: A Case Study of Faculty of Open Education

Zehra KAMIŞLI, Anadolu University, zkamisli@anadolu.edu.tr

1. INTRODUCTION

We live in the information and the microelectronic age, where technological advancements become a major determinant of our lifestyle. Such advances in technology cannot possibly be made or sustained without concurrent advancement in management systems (5).

The impact of computer technology on organizations and society is increasing as new technologies evolve and existing technologies expand (5). Because of the influences of the innovations in the computer technology, people can also build their own computerized systems with easy-to-use construction tools.

Management is a process by which certain goals are achieved through the use of resources like materials, people, money, time. These resources are considered to be inputs, and the attainment of the goals is viewed as the output of the process. Database systems continue to be a key aspect of Computer Science & Engineering today. Representing knowledge within a computer is one of the central challenges of the field. Database research has focused primarily on this fundamental issue (6).

This paper presents a database management system developed for AOF (Faculty of Open Education) course books. Its rationale is based on the development of a database application that share data and operations through a database. It supports organization's manipulation and retrieval of data. In order to organize large amounts of data (book's names, ISBN numbers, number printed; authors' names, and addresses, etc.) similar user interfaces are developed for different types of users. In addition, they can create special reports. It also encourages users who have no previous programming experience. The detail of the study is presented in the third part of the paper. In the second part of the paper, some knowledge about management information systems and database management systems are given.

2. MANAGEMENT INFORMATION SYSTEMS

Management information systems (MIS) consists of a collection of interrelated data and information structure that organized in such a way that it corresponds to the needs and structure of an organization and can be used by more than one person for more than one application. A (MIS) is a formal, computer-based system, intended to retrieve, extract, and integrate data from various sources in order to provide timely information for routine, structured, and anticipated types of decisions. In addition, it has been successful in acquiring and storing large quantities of detailed data concerning transaction processing (5). MIS has the following characteristics:

1. It supports recording keeping and data processing functions.
2. Same databases can be shared for all functions of the organization
3. Every manager, from the different levels of the organization can reach easily to the data.
4. All data and information can be used only by authorized personnel. Thus, system's security is provided.

Organizations require MIS because of some specific causes. These causes are: (3)

1. Complexity of data transfer and communication problems because of this complexity.
2. Work and force repetitions, creation of same data, and storing them in many different places.
3. Applying many kinds of operations and information flows when obtaining same data.
4. Impossibility of working with data. It's hard to get useful and necessary information because their form and positions aren't proper.
5. Insufficient data source.
6. Limited information support.

2.1. Database Approach

Misplacing information is a problem. Having the right information is not enough; we also have it logically and physically organized so that you can easily access it and make the sense of it. Having the right information and being able to get it quickly will increase productivity. Once getting the information, it must be stored so that people can get it easily and make changes as needed. There are two views of information: physical and logical. The physical view deals with how information is stored on storage devices, while the logical view deals with how you arrange information while you're working with it (1).

4.3.7. Skill Building Exercises/Assignments

Exercise 1 Entity-Relationship Model

Suppose you are a new member of the computer support group of the university library. Your job is to set up the new library information system using a relational database. Your first step to do is set up an Entity-Relationship Model that should roughly represent the following real-world objects:

1. Books. But also periodical journals or books that span several volumes.
2. Books are written by authors or edited by editors.
3. Books may be published in several editions by a publisher.

Explain your Entity-Relationship Model. Find sensible attributes for your entities. Mark primary keys and key candidates. What problems do you encounter? Are there any constraints you can't model with the ER Model?

Exercise 2 Relational Model

The library system shall be realized using a relational database, based on the ER Model you designed. Translate your Entity-Relationship Model into a relational schema. Use primary and foreign keys to all constraints in your ER Model as close as possible. Which of the constraints can you map, which ones cannot be mapped?

4.3.8. Previous Questions (Asked by JNTUK from the concerned Unit)

1. a) Discuss in detail about the main steps in the database design and clearly focus in detail about the goal of each step? In which steps is the ER model mainly used?
b) Draw and explain E-R diagram of an Airline reservation system?
2. a) Describe in detail about conceptual design with ER model.
b) Construct E-R diagram for a banking enterprise
3. What is view, updatable and non-updatable views? Explain the advantages of view in maintaining the security.
4. a) What are nested queries? What is correlation in nested queries? How would you use the operators IN, EXISTS, UNIQUE, ANY and ALL in writing nested queries? Why are they useful? Illustrate your answer by showing how to write the division operator in SQL.
b) Compare constraints and triggers and also give examples for each.

5. a) A university database contains information about professors (identified by social security number, or SSN) and courses (identified by courseid). Professors teach courses; each of the following situations concerns the Teaches relationship set. For each situation, draw an E-R diagram that describes it (assuming that no further constraints hold). Professors can teach the same course in several semesters, and each offering must be recorded. Professors can teach the same course in several semesters, and only the most recent such offering needs to be recorded. Every professor must teach some course. Every professor teaches exactly one course (no more, no less). Every professor teaches exactly one course (no more, no less), and every course must be taught by some professor.
 b) Explain the difference between weak entity and strong entity set? How to represent the strong entity and weak entity set through E-R diagram.
6. What is the difference between a candidate key and the primary key for a given relation? What is a super key?
 b) Discuss in detail about integrity constraints over relations.
7. a) Construct an E-R diagram for a university registrar's office. The office maintains data about each class, including the instructor, the enrollment, and the time and place of the class meetings. For each student-class pair, a grade is recorded. Document all assumptions that you make about the mapping constraints.
 b) Explain the difference between weak entity and strong entity sets? How to represent the strong and weak entity set through E-R diagrams.
8. a) Differentiate specialization and generalization.
 b) What is a view? How views are implemented?
9. a) Discuss in detail about the concepts of E-R model with suitable examples.
 [8M]
 b) What is a group function? List and explain how to use group functions in SQL
 with appropriate examples.
10. a) Explain about various constraints used in ER-model. [8M]
 b) Differentiate between independent and correlated nested queries.

4.3.9. GATE Questions (Where relevant)

1. The rule that a value of a foreign key must appear as a value of some specific table is called a

- | | |
|-----------------------------|----------------------------|
| (A) Referential constraint. | (B) Index. |
| (C) Integrity constraint. | (D) Functional dependency. |

Ans: (A)

2. The clause in SQL that specifies that the query result should be sorted in ascending or descending order based on the values of one or more columns is

- | | |
|--------------|--------------|
| (A) View | (B) Order by |
| (C) Group by | (D) Having |

Ans: (B)

3. It is an abstraction through which relationships are treated as higher level entities

- | | |
|---------------------|---------------------|
| (A) Generalization. | (B) Specialization. |
| (C) Aggregation. | (D) Inheritance. |

Ans: (C)

4. What is data integrity?

- a) It is the data contained in database that is non redundant.
- b) It is the data contained in database that is accurate and consistent.
- c) It is the data contained in database that is secured.
- d) It is the data contained in database that is shared.

Ans: (B)

5. In an E-R diagram double lines indicate

- | | |
|----------------------|-----------------------------|
| Total participation. | (B) Multiple participation. |
| Cardinality N. | (D) None of the above. |

Ans: (A)

6. The operation which is not considered a basic operation of relational algebra is

- | | |
|------------|--------------------|
| (A) Join. | (B) Selection. |
| (C) Union. | (D) Cross product. |

Ans: (A)

7. In SQL the statement **select * from R, S** is equivalent to

- | | |
|-------------------------------------|-----------------------------------|
| (A) Select * from R natural join S. | (B) Select * from R cross join S. |
| (C) Select * from R union join S. | (D) Select * from R inner join S. |

Ans: (B)

8. As per equivalence rules for query transformation, selection operation distributes over

- | | |
|---------------------|-----------------------|
| (A) Union. | (B) Intersection. |
| (C) Set difference. | (D) All of the above. |

Ans: (D)

9. The metadata is created by the

- | | |
|------------------|-----------------------|
| (A) DML compiler | (B) DML pre-processor |
|------------------|-----------------------|

- (C) DDL interpreter (D) Query interpreter

Ans: (C)

10. When an E-R diagram is mapped to tables, the representation is redundant for

- (A) weak entity sets (B) weak relationship sets
(C) strong entity sets (D) strong relationship sets

Ans: (B)

11. In SQL the word 'natural' can be used with

- (A) inner join B) full outer join
(C) right outer join (D) All of the above

Ans: (A)

12. If the closure of an attribute set is the entire relation then the attribute set is a

- (A) superkey (B) candidate key
(C) primary key (D) not a key

Ans: (A)

13. DROP is a _____ statement in SQL.

- (A) Query (B) Embedded SQL
(C) DDL (D) DCL

Ans: (C)

14. If two relations R and S are joined, then the non matching tuples of both R and S are ignored in

- (A) left outer join (B) right outer join
(C) full outer join (D) inner join

Ans: (D)

15. The keyword to eliminate duplicate rows from the query result in SQL is

- (A) DISTINCT (B) NO DUPLICATE
(C) UNIQUE (D) None of the above

Ans: (C)

4.3.10. Interview questions (which are frequently asked in a Technical round - Placements)

1. What is E-R model?

Ans: This data model is based on real world that consists of basic objects called entities and of relationship among these objects. Entities are described in a database by a set of attributes.

2. What is an Entity?

Ans: It is a 'thing' in the real world with an independent existence.

3. What is an Entity type?

Ans: It is a collection (set) of entities that have same attributes.

4. What is an Entity set?

Ans: It is a collection of all entities of particular entity type in the database.

5. What is an Extension of entity type?

Ans: The collections of entities of a particular entity type are grouped together into an entity set.

6. What is Weak Entity set?

Ans: An entity set may not have sufficient attributes to form a primary key, and its primary key compromises of its partial key and primary key of its parent entity, then it is said to be Weak Entity set.

7. What is an attribute?

Ans: It is a particular property, which describes the entity.

8. What is a Relation Schema and a Relation?

Ans: A relation Schema denoted by $R(A_1, A_2, \dots, A_n)$ is made up of the relation name R and the list of attributes A_i that it contains. A relation is defined as a set of tuples. Let r be the relation which contains set tuples $(t_1, t_2, t_3, \dots, t_n)$.

Each tuple is an ordered list of n -values $t=(v_1, v_2, \dots, v_n)$.

9. What is View?

Ans: A simple view can be thought of as a subset of a table. It can be used for retrieving data, as well as updating or deleting rows. Rows updated or deleted in the view are updated or deleted in the table the view was created with. It should also be noted that as data in the original table changes, so does data in the view, as views are the way to look at part of the original table. The results of using a view are not permanently stored in the database. The data accessed through a view is actually constructed using standard T-SQL select command and can come from one to many different base tables or even other views.

10. What's the difference between a primary key and a unique key?

Ans: Both primary key and unique enforce uniqueness of the column on which they are defined. But by default primary key creates a clustered index on the column, where unique creates a nonclustered index by default. Another major difference is that, primary key doesn't allow NULLs, but unique key allows one NULL only.

11. How to implement one-to-one, one-to-many and many-to-many relationships while designing tables?

Ans: One-to-One relationship can be implemented as a single table and rarely as two tables with primary and foreign key relationships.

One-to-Many relationships are implemented by splitting the data into two tables with primary key and foreign key relationships.

Many-to-Many relationships are implemented using a junction table with the keys from both the tables forming the composite primary key of the junction table.

12. What is difference between DELETE & TRUNCATE commands?

Ans: Delete command removes the rows from a table based on the condition that we provide with a WHERE clause. Truncate will actually remove all the rows from a table and there will be no data in the table after we run the truncate command.

TRUNCATE

TRUNCATE is faster and uses fewer system and transaction log resources than DELETE.

TRUNCATE removes the data by deallocating the data pages used to store the table's data, and only the page deallocations are recorded in the transaction log. TRUNCATE removes all rows from a table, but the table structure and its columns, constraints, indexes and so on remain. The counter used by an identity for new rows is reset to the seed for the column.

You cannot use TRUNCATE TABLE on a table referenced by a FOREIGN KEY constraint.

Because TRUNCATE TABLE is not logged, it cannot activate a trigger.

TRUNCATE cannot be Rolled back.

TRUNCATE is DDL Command.

TRUNCATE Resets identity of the table.

DELETE

DELETE removes rows one at a time and records an entry in the transaction log for each deleted row. If you want to retain the identity counter, use DELETE instead. If you want to remove table definition and its data, use the DROP TABLE statement.

DELETE Can be used with or without a WHERE clause

DELETE Activates Triggers.

DELETE can be Rolled back.

DELETE is DML Command.

DELETE does not reset identity of the table.

13. What is the difference between a HAVING CLAUSE and a WHERE CLAUSE?

Ans: Specifies a search condition for a group or an aggregate. HAVING can be used only with the SELECT statement. HAVING is typically used in a GROUP BY clause.

When GROUP BY is not used, HAVING behaves like a WHERE clause. Having Clause is basically used only with the GROUP BY function in a query. WHERE Clause is applied to each row before they are part of the GROUP BY function in a query.

14. What is sub-query? Explain properties of sub-query.

Ans: **Sub-queries** are often referred to as sub-selects, as they allow a SELECT statement to be executed arbitrarily within the body of another SQL statement. A sub-query is executed by enclosing it in a set of parentheses. Sub-queries are generally used to return a single row as an atomic value, though they may be used to compare values against multiple rows with the IN keyword.

A **subquery** is a SELECT statement that is nested within another T-SQL statement. A subquery SELECT statement if executed independently of the T-SQL statement, in which it is nested, will return a result set. Meaning a subquery SELECT statement can standalone and is not depended on the statement in which it is nested. A subquery SELECT statement can return any number of values, and can be found in, the column list of a SELECT statement, a FROM, GROUP BY, HAVING, and/or ORDER BY clauses of a T-SQL statement. A Subquery can also be used as a parameter to a function call. Basically a

subquery can be used anywhere an expression can be used.

Properties of Sub-Query

A subquery must be enclosed in the parenthesis. A subquery must be put in the right hand of the comparison operator, and A subquery cannot contain a ORDER -BY clause. A query can contain more than one sub-queries.

15. What are types of sub-queries?

Ans: Single-row subquery, where the subquery returns only one row. Multiple-row subquery, where the subquery returns multiple rows.,and Multiple column subquery, where the subquery returns multiple columns.

4.3.11. Real-Word (Live) Examples / Case studies wherever applicable

1. Hospital Management System case studies

Database Management System

Case Studies

Case Study 1

Hospital Management System

Aim: XYZ hospital is a multi specialty hospital that includes a number of departments, rooms, doctors, nurses, compounders, and other staff working in the hospital. Patients having different kinds of ailments come to the hospital and get checkup done from the concerned doctors. If required they are admitted in the hospital and discharged after treatment.

The aim of this case study is to design and develop a database for the hospital to maintain the records of various departments, rooms, and doctors in the hospital. It also maintains records of the regular patients, patients admitted in the hospital, the check up of patients done by the doctors, the patients that have been operated, and patients discharged from the hospital.

Description: In hospital, there are many departments like Orthopedic, Pathology, Emergency, Dental, Gynecology, Anesthetics, I.C.U., Blood Bank, Operation Theater, Laboratory, M.R.I., Neurology, Cardiology, Cancer Department, Corpse, etc. There is an OPD where patients come and get a card (that is, entry card of the patient) for check up from the concerned doctor. After making entry in the card, they go to the concerned doctor's room and the doctor checks up their ailments. According to the ailments, the doctor either prescribes medicine or admits the patient in the concerned department. The patient may choose either private or general room according to his/her need. But before getting admission in the hospital, the patient has to fulfill certain formalities of the hospital like room charges, etc. After the treatment is completed, the doctor discharges the patient. Before discharging from the hospital, the patient again has to complete certain formalities of the hospital like balance charges, test charges, operation charges (if any), blood charges, doctors' charges, etc.

Next we talk about the doctors of the hospital. There are two types of the doctors in the hospital, namely, *regular doctors* and *call on doctors*. Regular doctors are those doctors who come to the hospital daily. Calls on doctors are those doctors who are called by the hospital if the concerned doctor is not available.

Table Description:

Following are the tables along with constraints used in *Hospital Management* database.

2. Database Management Systems: A Case Study of Faculty of Open Education

The Turkish Online Journal of Educational Technology – TOJET January 2004 ISSN: 1303-6521 volume 3 Issue 1 Article 3

Database Management Systems: A Case Study of Faculty of Open Education

Zehra KAMIŞLI, Anadolu University, zkamisli@anadolu.edu.tr

1. INTRODUCTION

We live in the information and the microelectronic age, where technological advancements become a major determinant of our lifestyle. Such advances in technology cannot possibly be made or sustained without concurrent advancement in management systems (5).

The impact of computer technology on organizations and society is increasing as new technologies evolve and existing technologies expand (5). Because of the influences of the innovations in the computer technology, people can also build their own computerized systems with easy-to-use construction tools.

Management is a process by which certain goals are achieved through the use of resources like materials, people, money, time. These resources are considered to be inputs, and the attainment of the goals is viewed as the output of the process. Database systems continue to be a key aspect of Computer Science & Engineering today. Representing knowledge within a computer is one of the central challenges of the field. Database research has focused primarily on this fundamental issue (6).

This paper presents a database management system developed for AOF (Faculty of Open Education) course books. Its rationale is based on the development of a database application that share data and operations through a database. It supports organization's manipulation and retrieval of data. In order to organize large amounts of data (book's names, ISBN numbers, number printed; authors' names, and addresses, etc.) similar user interfaces are developed for different types of users. In addition, they can create special reports. It also encourages users who have no previous programming experience. The detail of the study is presented in the third part of the paper. In the second part of the paper, some knowledge about management information systems and database management systems are given.

2. MANAGEMENT INFORMATION SYSTEMS

Management information systems (MIS) consists of a collection of interrelated data and information structure that organized in such a way that it corresponds to the needs and structure of an organization and can be used by more than one person for more than one application. A (MIS) is a formal, computer-based system, intended to retrieve, extract, and integrate data from various sources in order to provide timely information for routine, structured, and anticipated types of decisions. In addition, it has been successful in acquiring and storing large quantities of detailed data concerning transaction processing (5). MIS has the following characteristics:

1. It supports recording keeping and data processing functions.
2. Same databases can be shared for all functions of the organization
3. Every manager, from the different levels of the organization can reach easily to the data.
4. All data and information can be used only by authorized personnel. Thus, system's security is provided.

Organizations require MIS because of some specific causes. These causes are; (3)

1. Complexity of data transfer and communication problems because of this complexity.
2. Work and force repetitions, creation of same data, and storing them in many different places.
3. Applying many kinds of operations and information flows when obtaining same data.
4. Impossibility of working with data. It's hard to get useful and necessary information because their form and positions aren't proper.
5. Insufficient data source.
6. Limited information support.

2.1. Database Approach

Misplacing information is a problem. Having the right information is not enough; we also have it logically and physically organized so that you can easily access it and make the sense of it. Having the right information and being able to get it quickly will increase productivity. Once getting the information, it must be stored so that people can get it easily and make changes as needed. There are two views of information, physical and logical. The physical view deals with how information is stored on storage devices, while the logical view deals with how you arrange information while you're working with it (1).

3. Database Design Case Study East Coast Aquarium

D a t a b a s e D e s i g n C a s e S t u d y E a s t C o a s t A q u a r i u m

Many-to-many relationships are often the bane of the relational data-base designer. Sometimes it is not completely clear that you are dealing with that type of relationship. However, failure to recognize the many-to-many relationship can result in serious data integrity problems.

The organization described in this chapter actually needs two data-bases, the larger of which is replete with many-to-many relationships. In some cases it will be necessary to create additional entities for composite entities to reference merely to ensure data integrity.

Perhaps the biggest challenge with a database design that works for East Coast Aquarium is the lack of complete specifications. As you will see, the people who will be using the application programs to manipulate the aquarium's two new databases have only a general idea of what the programs must do. Unlike Mighty-Mite Motors— which had the history of working from a large collection of existing forms, documents, and procedures— East Coast Aquarium has nothing of that sort.

O r g a n i z a t i o n a l O v e r v i e w

The East Coast Aquarium is a nonprofit organization dedicated to the study and preservation of marine life. Located on the Atlantic Coast in the heart of a major northeastern U.S. city, it provides a wide variety of educational services to the surrounding area. The aquarium is supported by donations, memberships, fees for private functions, gift shop revenues, class fees, and the small admission fee it charges to the public. Research activities are funded by federal and private grants. To help keep costs down, many of the public service jobs (leading tours,

4.3.12. Suggested “Expert Guest Lectures” (both from in and outside of the campus)

NA

4.3.13. Literature references of Relevant NPTEL Videos/Web/You Tube

videos etc.

1. <https://www.youtube.com/watch?v=Wv1c9K4788A>
2. https://www.youtube.com/watch?v=F_xDqBa5w-s
3. <https://www.youtube.com/watch?v=U7uX47J29Jk>

4.3.14. Any Lab requirements; if so link it to Lab Lesson Plan.

NA

4.3.15. Reference Text Books / with Journals Chapters etc.

Text Book:

1. Alexis Leon, Mathews Leon, Fundamentals of Database Management Systems, Tata McGraw Hill Publishers, Second Print, 2010
- Reference Books:
 2. Mark Gillenson, Fundamentals of Database Management Systems, Wiley Publications. Reprint 2008
 3. Raghu Ramakrishnan, Johannes Gehrke , Database Management Systems, McGraw-Hill, 2003
 4. Isrd Group, Introduction to Database Management Systems, Tata McGraw-Hill Education, 2005
 5. Michael V. Mannino , Database Application Development and Design, McGrawHill/Irvin, 2001
 6. Connolly, Database Systems: A Practical Approach to Design, Implementation and Management, 4/E, Pearson Education India, 2008
 7. Patricia Ward, George Dafoulas, Database Management Systems, Cengage Learning EMEA, 2006
 8. Post, Database Management Systems 3e with Cd, Tata McGraw-Hill Education
 9. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, Database System Concepts, Mcgraw-Hill, 2010
 10. Colin Ritchie, Database Principles and Design, Cengage Learning EMEA, 2008
 11. Peter Rob, Carlos Coronel, Database Systems, - 2007

Journal papers:

1. Extracting Entity Relationship Diagram (ERD) from English Sentences

Extracting Entity Relationship Diagram (ERD) from English Sentences

Amani Abdel-Salam Al-Btoush
amanebtoush77@gmail.com

Abstract

Entity Relationship Diagram (ERD) is the first step in database design; it is an important step for the database designers, users, analyst, managers and software engineering. Since English is a universal language, this paper describes a methodology that extracts ERD from English sentences. This methodology is based on a predefined set of heuristic rules that aims to extract the elements of the ERD, then these rules are mapped into a diagram. A diagram generator automatically converts the rules into the ERD according to the rules of generating. The proposed methodology is explained by examples to show how it can provide a mechanism for quickly and easily way in extracting the ERD.

Keywords: Entity Relationship Diagram, entity, relationship, attribute

1. Introduction

The Entity Relationship Diagram (ERD) shows that the real world consists of a collection of entities, the relationships between them, and the attributes that describe them. An entity is the object where we want to store data. A relationship defines the allowed connections between instances of entities [1]. Attribute is a characteristic common to all or most instances of a particular entity. Since the ER approach is easy to understand, a designer can focus on conceptual modeling of an organization, making decisions of what to use from entity sets, relationship sets and constraints [2].

The ER-Diagram tool provides a mechanism for quickly and easily modeling data structures required by a software system. The ERD tool provides all the usual features of a data modeling tool and additionally provides reverse engineering. Thus, the user can create a database system quickly on a number of different target platforms without the need to write any Data Definition Language (DDL) type code.

There are many essential concepts between ERD structure and English grammar structure [3] after analyzing the English sentences, so it is easy to make mapping between them. This paper describes a methodology, which can be able to extract the ERD from a description of the application domain given in English sentences. Using ER-extractor that extracts entities, relationships and attributes according to the heuristic rules it will define, as well as by matching between the structures of both English and ERD structures. After that, ER-generator is depends on the predicate to convert the structure, which next pass the ER-descriptor to start to draw the ERD automatically depending on the rules.

This paper proposed to define a methodology that provide a help to the database designer to automatically extract the ERD from a given English sentences. ERD is the first step in database design, it is also a simple technique described in a graphical way to decide which database fields, relationships and tables will be the base of any database. ERD is a good communication tool between users and who use the system during the identification of the user information requirements process.

2. Extracting Entity Relationship Diagram (ERD) From Relational Database Schema

Extracting Entity Relationship Diagram (ERD) From Relational Database Schema

Hala Khaled Al-Masree

Mutah University, Jordan, Alkarak
Hala83990@gmail.com

Abstract

Database Reverse Engineering (DBRE) is an operation used to extract requirements from any system. The operation is implemented to facilitate the understanding of the system that has a little documentation about design and architecture. DBRE is a very important process used when database designers would like to expand the system or transition to the latest technology in DBRE fields. In the relational database model DBRE try to extract Entity Relationship Diagram (ERD) from relational database schema. Database designers find content of the data for a lot of attributes are not related with their names. In this paper, proposed methodology used to extract ERD from relational database schema with the attributes related with their names, both types of entities regular and weak entity, relationships and keys, which are found in the table that has extracted the relational database schema. The basic inputs of this approach are relational database schema that generated from database. The relational database schema used to extract the information about ERD. After that, obtain information that contain keywords help database designers to extract entities and their attributes semantics related with their names from relational database schemas. Then, Determine primary keys, foreign keys and constraints of the database system. In the final step, the ERD is successfully extracted

Keywords: DBRE, ERD, Relational database schema.

1. Introduction

Since 1976 chen [8] founded the ERD during his researches, the ERD was considered a very good way to do a conceptual model for databases. ERD is graphical way to give information about database in the system. Using ERD diagram database designers can convert this information to build a database tables. Information that has been obtained about database helps to reach to relational database schema. When database designers need to turn from relational database schema to ERD they need a methodology leads them to do that. In many cases, adjustments are made to the database such as writing database code without using ERD, because the people who have modified these adjustments do not saw the importance to use ERD. Therefore it is necessary propose the approach resolve this case.

[5] DBRE also includes these objectives: to provide missing or alternative documentation, to assist with maintenance, to migrate from one hardware or software platform to another, etc. Many companies or specialists in database design need retrieval ERD from relational database schema to achieve the DBRE objectives [16,18].

In this paper, propose methodology in order to provide way to those who need DBRE to extract ERD from relational database schema [10]. The proposed methodology facilitated utilization ERD when the database designers need to develop the databases depending on ERD. Database design is built by the analysis of the problems then extract relational database schema. In addition the database in the most companies maybe needs to modify by different database design specialists. This is should lead to think about approach to reverse engineering [9] from relational database schema to ERD.

(Normalization)

Normalization or schema refinement, concept of functional dependency, normal forms based on functional dependency(1NF, 2NF and 3 NF), concept of surrogate key, Boyce-codd normal form(BCNF), Lossless join and dependency preserving decomposition, Fourth normal form(4NF).

4.4.1. Unit Objectives:

- Identify update, insertion and deletion anomalies
- Identify possible keys given an instance
- Identify possible functional dependencies in a relation
- Determine all keys in a schema
- Decompose a schema into a BCNF schemas

4.4.2. Unit Outcomes:

- Describe functional dependence and use it to determine functional dependencies among table attributes
- Define the terms normal form and normalization
- Explain the need for converting a large table to many smaller tables using the first normal form (1NF), second normal form (2NF), and third normal form (3NF)
- Identify modification anomalies in tables that are not in 1NF, 2NF, and 3NF
- Normalize tables by detecting violations of the normal and apply normalization rules.
- Explain how normalization and ER modeling can be used concurrently to produce good database design
- Explain how some situations require denormalization to generate information efficiently.

4.4.3. Unit Lecture Plan:

| Lecture No. | Topic | Methodology | Quick reference |
|--------------------|---|--------------------|------------------------|
| 1 | Schema Refinement (Normalization): Purpose of Normalization or schema refinement, | Chalk & Board | Text. Book with Page |
| 2 | concept of functional dependency, | Chalk & Board | Text. Book with Page |
| 3 | normal forms based on functional dependency(1NF) | Chalk & Board | Text. Book with Page |
| 4 | normal forms based on functional dependency (2NF) | Chalk & Board | Text. Book with Page |
| 5 | normal forms based on functional dependency (3NF) | Chalk & Board | Text. Book with Page |
| 6 | concept of surrogate key, | Chalk & Board | Text. Book with Page |
| 7 | Boyce-codd normal form(BCNF) | Chalk & Board | Text. Book with Page |
| 8 | Lossless join and dependency preserving decomposition | Chalk & Board | Text. Book with Page |
| 9 | Fourth normal form(4NF) | Chalk & Board | Text. Book with Page |

4.4.4. Teaching Material / Teaching Aids as per above lecture plan.

4.4.4.1. Lecture-1

INTRODUCTION TO SCHEMA REFINEMENT

Conceptual database design gives us a set of relation schemas and integrity constraints (ICs) that can be regarded as a good starting point for the final database design. This initial design must be refined by taking the ICs into account and also by considering performance criteria and typical workloads.

A major aim of relational database design is to minimize data redundancy. The problems associated with data redundancy are illustrated as follows:

Problems caused by redundancy

Storing the same information in more than one place within a database is called **redundancy** and can lead to several problems:

- **Redundant Storage:** Some information is stored repeatedly.
- **Update Anomalies:** If one copy of such repeated data is updated, an inconsistency is created unless all copies are similarly updated.
- **Insertion Anomalies:** It may not be possible to store certain information unless some other, unrelated, information is stored as well.
- **Deletion Anomalies:** It may not be possible to delete certain information without losing some other, unrelated, information as well.

Ex: Consider a relation, Hourly_Emps(ssn, name, lot, rating, hourly_wages, hours_worked)

The key for Hourly_Emps is *ssn*. In addition, suppose that the *hourly_wages* attribute is determined by the *rating* attribute. That is, for a given *rating* value, there is only one permissible *hourly_wages* value. This IC is an example of a *functional dependency*. It leads to possible redundancy in the relation Hourly_Emps, as shown

below:

| ssn | name | lot | rating | hourly_wages | hours_worked |
|-----|------------|-----|--------|--------------|--------------|
| 567 | Adithya | 48 | 8 | 10 | 40 |
| 576 | Devesh | 22 | 8 | 10 | 30 |
| 574 | Ayush Soni | 35 | 5 | 7 | 30 |
| 597 | Rajasekhar | 35 | 5 | 7 | 32 |
| 5c1 | Sunil | 35 | 8 | 10 | 40 |

If the same value appears in the *rating* column of two tuples, the IC tells us that the same value must appear in the *hourly_wages* column as well. This redundancy has the following problems:

- **Redundant Storage:** The rating value 8 corresponds to the hourly_wage 10, and this association is repeated three times.
- **Update Anomalies:** The hourly_wage in the first tuple could be updated without making a similar change in the second tuple.
- **Insertion Anomalies:** We cannot insert a tuple for an employee unless we know the *hourly_wage* for the employee's *rating* value.
- **Deletion Anomalies:** If we delete all tuples with a given rating value (e.g., we delete the tuples for Ayush Soni and Rajasekhar) we lose the association between that *rating* value and its *hourly_wage* value.

4.4.4.2.Lecture-2

Functional Dependency:

An important concept associated with normalization is functional dependency, which describes the relationship between attributes.

FD describes the relationship between attributes in a relation. For example, if A and B are attributes of relation R, B is functionally dependent on A (denoted $A \rightarrow B$), if each value of A is associated with exactly one value of B (A and B may each consist of one or more attributes)

When a functional dependency is present, the dependency is specified as a constraint between the attributes. Consider a relation with attributes A and B where attribute B is functionally dependent on attribute A. i.e. each value of A exactly one value in B. Thus, when two tuples have the same value of A, they also have the same value of B. However, for a given value of B there may be several different values of A. The dependency between attributes A and B can be represented diagrammatically, as shown below fig.

$$A \rightarrow B$$

Determinant Refers to the attribute, or group of attributes, on the left-hand side of the arrow of a functional dependency.

Types of Functional Dependencies:-

1) Full Dependency: -

In a relation, the attribute B is fully functional dependent on A if B is functionally dependent on A, but not on any proper subset of A.

2) Partial Dependency: -

If there is some attribute that can be removed from A and the dependency still holds.

Eg. Staff_No, Sname \rightarrow Branch_No

3) Transitive Dependency: -

In a relation, if attribute(s) A \rightarrow B and B \rightarrow C, then C is transitively dependent on A via B
(provided that A is not functionally dependent on B or C)

Eg. Staff_No \rightarrow Branch_No and Branch_No \rightarrow BAddress

4) Trivial FD: -

X \rightarrow Y where Y is a subset of X

Example: customer_name, loan_number \rightarrow customer_name

5) Nontrivial FD:

X \rightarrow Y where Y is not a subset of X

Example: customer_name, loan_number \rightarrow amount

4.4.4.3.Lecture-3

NORMALIZATION

Definition: Normalization is the process of efficiently organizing data in a database. There are two goals of the normalization process: eliminating redundant data (for example, storing the same data in more than one table) and ensuring data dependencies make sense (only storing related data in a table). Both of these are worthy goals as they reduce the amount of space a database consumes and ensure that data is logically stored. There are several benefits for using Normalization in Database.

Benefits:

- a. Eliminate data redundancy
- b. Improve performance
- c. Query optimization
- d. Faster update due to less number of columns in one table
- e. Index improvement

Different types of Normal Forms

Un-normalized Form: This is a relation which satisfies all the properties of a relation.

1 Normal Form: A Relation is said to be in the first normal form if it is already in un-normalized form and it has no repeating groups.

2 Normal Form: A relation is said to be in the second normal form if it is already in the first normal from and it has no partial dependency.

3 Normal Form: A relation is said to be in the third normal form if it is already in the second normal from and it has no transitive dependency.

Partial Dependency: In a relation having more than one key field, a subset of non-key fields may depend on all the key fields but another subset of non-key fields may depend on only one of the key fields. Such dependency is called partial dependency.

Transitive Dependency: In a relation, there may be dependency among non-key fields. Such dependency is called as transitive dependency.

BCNF: A relation is said to be in Boyce-Codd normal from if it is already in the third normal form and every determinant is a candidate key.

Determinant: - The attribute on the left-hand side of the arrow in a functional dependency is called a determinant. In the below example the combination of SNO, SUBJECT is a determinant.

SNO, SUBJECT → MARKS.

(Or)

A Determinant is any field or combination of filed on which some other filed is fully functionally dependent

Super Key: A Super key is a set if one or more attributes that, taken collectively, allows us to identify uniquely an entity in the entity set.

For example, the social-security attribute of the entity set customer is sufficient to distinguish one customer entity form another. Thus, social-security is a super key. Similarly, the combination of customer-name and social-security is a super key for the entity set customer. The customer-name attribute of customer is not a super key, because several people might have the same name.

Candidate Key: Super key for which no proper subset is a super key. Such minimal super keys are called Candidate keys.

Suppose that a combination of customer-name and customer-street is sufficient to distinguish among member of the customer entity set. Then, both {social-security} and {customer-name, customer-street} are candidate keys. Although the attributes social-security and customer-name together can distinguish customer entities, their combination does not form a candidate key, since the attribute social-security alone is a candidate key.

Fourth Normal Form: A relation is said to be in the fourth normal form if it is already in Boyce-Codd normal form and it has no multi valued dependency.

Multivalued Dependency: MVD represents a dependency between attributes. For example A, B and C in a relation, such that for each value of A there is a set of values for B and a set of values for C. However, the set of values for B and C are independent of each other.

First Normal Form:

A database is in first normal form if it satisfies the following conditions:

- Contains only atomic values
- There are no repeating groups

An atomic value is a value that cannot be divided. For example, in the table shown below, the values in the [Color] column in the first row can be divided into "red" and "green", hence [TABLE_PRODUCT] is not in 1NF.

A repeating group means that a table contains two or more columns that are closely related. For example, a table that records data on a book and its author(s) with the following columns: [Book ID], [Author 1], [Author 2], [Author 3] is not in 1NF because [Author 1], [Author 2], and [Author 3] are all repeating the same attribute.

1st Normal Form Example

How do we bring an unnormalized table into first normal form? Consider the following example:

TABLE_PRODUCT

| Product ID | Color | Price |
|------------|--------------|-------|
| 1 | red, green | 15.99 |
| 2 | yellow | 23.99 |
| 3 | green | 17.50 |
| 4 | yellow, blue | 9.99 |
| 5 | red | 29.99 |

This table is not in first normal form because the [Color] column can contain multiple values. For example, the first row includes values "red" and "green."

To bring this table to first normal form, we split the table into two tables and now we have the resulting tables:

TABLE_PRODUCT_PRICE

| Product ID | Price |
|------------|-------|
| 1 | 15.99 |
| 2 | 23.99 |
| 3 | 17.50 |
| 4 | 9.99 |
| 5 | 29.99 |

TABLE_PRODUCT_COLOR

| Product ID | Color |
|------------|--------|
| 1 | red |
| 1 | green |
| 2 | yellow |
| 3 | green |
| 4 | yellow |
| 4 | blue |
| 5 | red |

Now first normal form is satisfied, as the columns on each table all hold just one value.

4.4.4.4.Lecture-4

2nd Normal Form Example

Consider the following example:

TABLE_PURCHASE_DETAIL

| Customer ID | Store ID | Purchase Location |
|-------------|----------|-------------------|
| 1 | 1 | Los Angeles |
| 1 | 3 | San Francisco |
| 2 | 1 | Los Angeles |
| 3 | 2 | New York |
| 4 | 3 | San Francisco |

This table has a composite primary key [Customer ID, Store ID]. The non-key attribute is [Purchase Location]. In this case, [Purchase Location] only depends on [Store ID], which is only part of the primary key. Therefore, this table does not satisfy second normal form.

To bring this table to second normal form, we break the table into two tables, and now we have the following:

TABLE_PURCHASE

| Customer ID | Store ID |
|-------------|----------|
| 1 | 1 |
| 1 | 3 |
| 2 | 1 |
| 3 | 2 |
| 4 | 3 |

TABLE_STORE

| Store ID | Purchase Location |
|----------|-------------------|
| 1 | Los Angeles |
| 2 | New York |
| 3 | San Francisco |

What we have done is to remove the partial functional dependency that we initially had. Now, in the table [TABLE_STORE], the column [Purchase Location] is fully dependent on the primary key of that table, which is [Store ID].

4.4.4.5.Lecture-5

3rd Normal Form Example

Consider the following example:

TABLE_BOOK_DETAIL

| Book ID | Genre ID | Genre Type | Price |
|---------|----------|------------|-------|
| 1 | 1 | Gardening | 25.99 |
| 2 | 2 | Sports | 14.99 |
| 3 | 1 | Gardening | 10.00 |
| 4 | 3 | Travel | 12.99 |
| 5 | 2 | Sports | 17.99 |

In the table above, [Book ID] determines [Genre ID], and [Genre ID] determines [Genre Type]. Therefore, [Book ID] determines [Genre Type] via [Genre ID] and we have transitive functional dependency, and this structure does not satisfy third normal form.

To bring this table to third normal form, we split the table into two as follows:

TABLE_BOOK

| Book ID | Genre ID | Price |
|---------|----------|-------|
| 1 | 1 | 25.99 |
| 2 | 2 | 14.99 |
| 3 | 1 | 10.00 |
| 4 | 3 | 12.99 |
| 5 | 2 | 17.99 |

TABLE_GENRE

| Genre ID | Genre Type |
|----------|------------|
| 1 | Gardening |
| 2 | Sports |
| 3 | Travel |

Now all non-key attributes are fully functional dependent only on the primary key. In [TABLE_BOOK], both [Genre ID] and [Price] are only dependent on [Book ID]. In [TABLE_GENRE], [Genre Type] is only dependent on [Genre ID].

Fourth Normal Form

Def: -A relation is said to be in the fourth normal form if it is already in Boyce-Codd normal form and it has no multivalued dependency.

For example, if we have two multi-valued attributes in a relation, we have to repeat each value of one of the attributes with every value of the other attribute, to ensure that tuples of the relation are consistent. This type of constraint is referred to as a multi-valued dependency and result in data redundancy. Consider the following

| COURSE | TEACHER | TEXT |
|---------|-------------|----------------------|
| Physics | Prof. Green | Basic Mechanics |
| | Prof. Brown | Principles of Optics |
| | Prof. Black | |
| Math | Prof. White | Modern Algebra |
| | | Projective Geometry |

Fig. Sample tabulation of CTX.

| COURSE | TEACHER | TEXT |
|---------|-------------|----------------------|
| Physics | Prof. Green | Basic Mechanics |
| Physics | Prof. Green | Principles of Optics |
| Physics | Prof. Brown | Basic Mechanics |
| Physics | Prof. Brown | Principles of Optics |

| | | |
|---------|-------------|----------------------|
| Physics | Prof. Black | Basic Mechanics |
| Physics | Prof. Black | Principles of Optics |
| Math | Prof. White | Modern Algebra |
| Math | Prof. White | Projective Geometry |

Fig. Sample tabulation of CTX.

In the above example, teachers Prof. Green, Prof. Brown, Prof. Black belongs to physics course and basic mechanics and principles of optics are physics text books only. However, as there is no direct relationship between members of teacher and text at a given course, we create a tuple for every combination of member of teacher and text to ensure that the relation is consistent. We represent a MVD between attributes are

COURSE ->> TEACHER

COURSE ->> TEXT

So, MVD exist in the above relation, for example, if we want to add a new text for physics we would have to create three new tuples, one for each member of teacher, to ensure that the relation remains consistent. This is an example of an update anomaly caused by the presence of MVD. We decompose the CTX relation into CT, CX relations as given below:

CT

| COURSE | TEACHER |
|---------|-------------|
| Physics | Prof. Green |
| Physics | Prof. Brown |
| Physics | Prof. Black |
| Math | Prof. White |

CX

| COURSE | TEXT |
|---------|----------------------|
| Physics | Basic Mechanics |
| Physics | Principles of Optics |

| | |
|------|---------------------|
| Math | Modern Algebra |
| Math | Projective Geometry |

The above new relations are in 4NF there is no MVD. Note that the 4NF relations do not display data redundancy and the potential for update anomalies is removed. For example, to add a new text for physics, we simple create a single tuple in the CX relation.

4.4.4.6.Lecture-6

Surrogate key

A surrogate key is frequently a sequential number (e.g. a Sybase "identity column") but doesn't have to be. Having the key independent of all other columns insulates the database relationships from changes in data values or database design and guarantees uniqueness.

A unique primary key generated by the RDBMS that is not derived from any data in the database and whose only significance is to act as the primary key.

Some database designers use surrogate keys religiously regardless of the suitability of other candidate keys. However, if a good key already exists, the addition of a surrogate key will merely slow down access, particularly if it is indexed.

There are at least two definitions of a surrogate:

Surrogate (1) – Hall, Owlett and Todd (1976)

A surrogate represents an *entity* in the outside world. The surrogate is internally generated by the system but is nevertheless visible to the user or application.

Surrogate (2) – Wieringa and De Jonge (1991)

A surrogate represents an *object* in the database itself. The surrogate is internally generated by the system and is invisible to the user or application.

A surrogate key should have the following characteristics:

- the value is unique system-wide, hence never reused
- the value is system generated
- the value is not manipulable by the user or application
- the value contains no semantic meaning
- the value is not visible to the user or application
- the value is not composed of several values from different domains.

Advantages:

Immutability

Surrogate keys do not change while the row exists.

Requirement changes

Attributes that uniquely identify an entity might change, which might invalidate the suitability of natural keys.

Performance

Surrogate keys tend to be a compact data type, such as a four-byte integer. This allows the database to query the single key column faster than it could multiple columns.

Compatibility

While using several database application development systems, drivers, and object-relational mapping systems, such as Ruby on Rails or Hibernate

Uniformity

When every table has a uniform surrogate key, some tasks can be easily automated by writing the code in a table-independent way.

4.4.4.7.Lecture-7

Boyce Codd Normal Form

Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with certain type of anomaly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, following conditions must be satisfied:

- are must be in 3rd Normal Form
- and, for each functional dependency ($X \rightarrow Y$), X should be a super Key.

Consider the following relationship : **R (A,B,C,D)**

and following dependencies :

$$A \rightarrow BCD$$

$$BC \rightarrow AD$$

$$D \rightarrow B$$

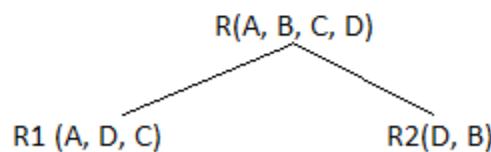
Above relationship is already in 3rd NF. Keys are **A** and **BC**.

Hence, in the functional dependency, **A → BCD**, A is the super key.

in second relation, **BC → AD**, BC is also a key.

but in, **D → B**, D is not a key.

Hence we can break our relationship R into two relationships **R1** and **R2**.



Breaking, table into two tables, one with A, D and C while the other with D and B.

4.4.4.8.Lecture-8

Decomposition

Decomposition in database means breaking tables down into multiple tables.

Two Characteristics of Good Decomposition

- 1) Lossless
- 2) Preserve dependencies

Lossy Decomposition

- Sometimes it's not:

The diagram illustrates a lossy decomposition of a single table into three separate tables. The original table has columns: Name, Price, and Category. The decomposed tables are:

| Name | Price | Category |
|--------|-------|----------|
| Word | 100 | WP |
| Oracle | 1000 | DB |
| Access | 100 | DB |

| Category | Name |
|----------|--------|
| WP | Word |
| DB | Oracle |
| DB | Access |

| Category | Price |
|----------|-------|
| WP | 100 |
| DB | 1000 |
| DB | 100 |

- $(\text{Word}, \text{WP}) + (100, \text{WP}) = (\text{Word}, 100, \text{WP})$
- $(\text{Oracle}, \text{DB}) + (1000, \text{DB}) = (\text{Oracle}, 1000, \text{DB})$
- $(\text{Oracle}, \text{DB}) + (100, \text{DB}) = (\text{Oracle}, \text{100}, \text{DB})$
- $(\text{Access}, \text{DB}) + (1000, \text{DB}) = (\text{Access}, \text{1000}, \text{DB})$
- $(\text{Access}, \text{DB}) + (100, \text{DB}) = (\text{Access}, 100, \text{DB})$

Lossless Decomposition

- Sometimes the same set of data is reproduced:

The diagram illustrates a lossless decomposition of a single table into two separate tables. The original table has columns: Name, Price, and Category. The decomposed tables are:

| Name | Price | Category |
|--------|-------|----------|
| Word | 100 | WP |
| Oracle | 1000 | DB |
| Access | 100 | DB |

| Name | Price |
|--------|-------|
| Word | 100 |
| Oracle | 1000 |
| Access | 100 |

| Name | Category |
|--------|----------|
| Word | WP |
| Oracle | DB |
| Access | DB |

- $(\text{Word}, 100) + (\text{Word}, \text{WP}) \rightarrow (\text{Word}, 100, \text{WP})$
- $(\text{Oracle}, 1000) + (\text{Oracle}, \text{DB}) \rightarrow (\text{Oracle}, 1000, \text{DB})$
- $(\text{Access}, 100) + (\text{Access}, \text{DB}) \rightarrow (\text{Access}, 100, \text{DB})$

Dependency preserving Decomposition

Dependency preserving Decomposition means that decomposition should be such

that every dependency of relation R must be implied in sub-relations or in combination of dependencies implied in sub relation.

Let R be any relation with FD set F, decomposed into sub-relations R_1 and R_2 with FD sets F_1 and F_2 respectively.

Then every dependency of R must be implied in F_1 or F_2 or in combination of these two.

$F_1 \cup F_2 = F$ (dependency must be preserved in decomposition)

If $F_1 \cup F_2 \subset F$ (not dependency preserving decomposition)

$F_1 \cup F_2 \supset F$ (this is not possible)

Example: Given relation R(ABCDE) with following FD's

$A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow E, D \rightarrow B$

decomposed into AB, BC, CD, DE

check whether the decomposition is dependency preserving or not

Solution: In these type of questions dependencies of sub-relations will not be given we have to find them

Find the closure of all attributes of a particular sub-relation from original FD set and then dependencies containing the attributes of that sub-relation will be implied.

For 1st sub-relation

$A^+ = ABCDE$

$B^+ = BCDE$

so $A \rightarrow B$ and $B \rightarrow C$ are dependencies for 1st sub-relation

for 2nd sub-relation

$B^+ = BCDE$

$C^+ = BCDE$

so $B \rightarrow C$ and $C \rightarrow B$ (Redundant) are dependencies for 2nd sub-relation

For 3rd sub-relation

$C \rightarrow D$

$D \rightarrow C$ (redundant)

For 4th sub-relation

$D \rightarrow E$ all subrelations contain all dependencies of original relation except $D \rightarrow B$

which can be derive easily from the dependencies of sub relation

so $F_1 \cup F_2 \cup F_3 \cup F_4 = F$

hence dependency preservation is satisfied

so **dependency preserving decomposition.**

Example: R(ABCD) has following FD's

$F = \{AB \rightarrow CD, D \rightarrow A\}$

decomposed into $D = \{ABC, AD, BCD\}$

check whether dependency preservation is satisfied or not

Solution:

For R_1

$AB^+ = ABCD$ so $AB \rightarrow C$

For R_2

$A^+ = A, D^+ = AD$

$D \rightarrow A$

For R_3

$BD \rightarrow C$ ($AB \rightarrow ABCD$ $D \rightarrow A$ so replace A by D so $BD \rightarrow ABCD$) But we can't derive $AB \rightarrow D$ from dependencies of subrelations
so **Dependency preservation is not satisfied.**

Example: $R(ABCDEF)$ has following FD's

$F = \{A \rightarrow BCDEF, BC \rightarrow ADEF, B \rightarrow F, D \rightarrow E\}$

$D = \{ABCD, BF, DE\}$

check whether decomposition is dependency preserving or not

Solution:

It is not necessary to take closure of all sub attributes every time we can derive dependencies directly for a sub-relation by seeing its attributes and original dependency set

For R_1

Attributes are $ABCD$ now check dependencies in original relation containing attributes $ABCD$

$A \rightarrow BCD$

$BC \rightarrow AD$

For R_2

$B \rightarrow F$

For R_3

$D \rightarrow E$

all above are direct dependencies now check if the remaining dependency from original set can be derived using these direct dependencies

$A \rightarrow E$ ($A \rightarrow D$ $D \rightarrow E$ so $A \rightarrow E$)

$A \rightarrow F$ ($A \rightarrow F$ $B \rightarrow F$ so $A \rightarrow F$)

$BC \rightarrow F$ ($B \rightarrow F$ $BC \rightarrow CF$ so $BC \rightarrow F$)

$BC \rightarrow E$ ($BC \rightarrow D$ $D \rightarrow E$ so $BC \rightarrow E$)

hence all dependencies of original relation are either direct dependencies in sub-relation or can be derived from direct dependencies

so **Dependency preserving Decomposition**

Example: $R(ABCDE)$

$F = \{A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A\}$

$D = \{ABCE, BD\}$ check whether dependency preserving or not

Solution:

For R_1

$A \rightarrow BC, E \rightarrow A$ are direct dependencies

For R_2

$B \rightarrow D$ is direct dependency

using these direct dependencies of sub-relation it is not possible to derive $CD \rightarrow E$ so now we have to find closure of all sub attributes of sub-relation. But for first

table we will have to find 14 closures.

$A^+ = ABCE$ (all attributes of relation R_1 which is not possible to derive using direct dependencies)

If only A determines all attributes then any superset of A also determines all attributes of R_1 so we can skip all supersets of A which will give redundant FD's only

$B^+ = B$

$C^+ = C$

$E^+ = EA$ (A determines all attributes so E will also determine all attributes)

$E^+ = ABCE$ (skip all supersets of E)

$BC^+ = ABCE$ (all supersets of BC can be skipped)

so new dependencies obtained after this step are

$A \rightarrow E, A \rightarrow CE, BC \rightarrow AE$

now after finding all dependencies of R_1 and R_2 using direct, derived as well as new dependencies

$CD \rightarrow E$ still fails so not dependency preserving decomposition.

4.4.5. Test Questions

a) Fill in the blank type of questions:

1. In the _____ normal form, a composite attribute is converted to individual attributes.(first)
2. A relation is considered as _____(two dimensional table)
3. A functional dependency is a relationship between or among _____(attributes)
4. Needing to using more complicated SQL in database applications is a(n) _____ of normalization(disadvantage)
5. Eliminating modification of anomalies is a _____ of normalization(advantage)
6. Multivalued dependencies should _____ be eliminated(always).
7. Creating a read-only database is a task that is _____ assigned to beginning database professionals(commonly).
8. For a number of reasons, normalizations is not often an advantage for a(n) _____ database(read only)
9. Read-only databases are _____ updated.(never)
10. Normalization _____ data duplication(eliminates).

b) Multiple choice questions

1. Needing to use more complicated SQL in database applications is a(n) _____ of normalization.

- A) Advantage
- B) Disadvantage
- C) either an advantage or disadvantage
- D) Neither an advantage nor disadvantage

2. Eliminating modification anomalies is a _____ of normalization

- A) Advantage
- B) Disadvantage
- C) either an advantage or disadvantage
- D) neither an advantage or disadvantage

3. Multivalued dependencies should _____ be eliminated.

- A) Always
- B) Commonly
- C) Seldom
- D) Never

4. When assessing the table structure of an acquired set of tables with data, accessing the validity of possible referential integrity constraints on foreign keys is (part of) the:

- A) first step
 - B) second step
 - C) third step
 - D) fourth step
5. Using the SQL GROUP BY phrase with a SELECT statement can help detect which of the following problems?
- A) The multivalue, multicoloumn problem
 - B) The inconsistent values problem
 - C) The missing values problem
 - D) The general purpose remarks coloumn problem

6. Creating a read-only database is a task that is _____ assigned to beginning database professionals

- A) always
- B) commonly
- C) seldom
- D) never

7. If a table has been normalized so that all determinants are candidate keys, then that table is in:

- A) 1NF
- B) 2NF
- C) 3NF

D) BCNF

8. Each answer below shows example data from a table. Which answer is an example of the multivalue, multicolumn problem?

A) Three columns have the values 534-2435, 534-7867, and 546-2356 in the same row.

B) Three rows have the values Brown Small Chair, Small Chair Brown, and Small Brown Chair in the same column.

C) Three rows have the values Brown, NULL, and Blue in the same column.

D) One row has the value "He is interested in a Silver Porsche from the years 1978-1988" in a column.

9. Every time attribute A appears, it is matched with the same value of attribute B, but not the same value of attribute C. Therefore, it is true that:

A) $A \rightarrow B$.

B) $A \rightarrow C$.

C) $A \rightarrow (B,C)$.

D) $(B,C) \rightarrow A$.

10. The different classes of relations created by the technique for preventing modification anomalies are called:

A) normal forms

B) referential integrity constraints

C) functional dependencies

D) None of the above is correct.

c) True or false questions:

1. When building a database from an existing set of tables, we may safely assume that referential integrity constraints have been enforced on the data we are given.(F)

2. Most of the time, modification anomalies cause problems that are severe enough that a table should be normalized into BCNF.(T)

3. When building a database from an existing set of tables, we still need to consider normalization principles.(T)
4. We have normalized a table into BCNF if all candidate keys are determinants.(B)
5. We use the SQL construct `COLUMNS(*)` to determine the number and type of columns in a table.(F)
6. In all cases, normalization into BCNF is desirable.(F)
7. Denormalized tables are in BCNF.(B)
8. We can eliminate modification anomalies with proper normalization that results in tables in BCNF.(T)
9. Proper normalization eliminates duplicated data.(B)
10. We use the SQL construct `COUNT(*)` to count the number of rows in a table.(T)

4.4.6 Review Questions

d) Objective type of questions (very short notes)

1. What is normalization?

It is a process of analyzing the given relation schemas based on their functional dependencies and primary key to achieve the properties

a)Minimizing redundancy b) Minimizing insertion, deletion and update anomalies.

2. What is functional dependency?

Functional dependency is the starting point of normalization. Functional dependency exists when a relation between two attributes allows you to uniquely determine the corresponding attributes value.

3. What is 1 NF (Normal Form)?

The first normal form or 1NF is the first and the simplest type of normalization that can be implemented in a database. The main aims of 1NF are to:

1. Eliminate duplicative columns from the same table.
2. Create separate tables for each group of related data and identify each row with a unique column (the primary key).

4. What is 2NF?

A relation schema R is in 2NF if it is in 1NF and every non-prime attribute A in R is fully functionally dependent on primary key.

5. What is 3NF?

A relation is in third normal form if it is in Second Normal Form and there are no functional (transitive) dependencies between two (or more) non-primary key attributes.

6. What is BCNF (Boyce-Codd Normal Form)?

A table is in Boyce-Codd normal form (BCNF) if and only if it is in 3NF and every determinant is a candidate key.

7. What is 4NF?

Fourth normal form requires that a table be BCNF and contain no multi-valued dependencies.

8. What is Stored Procedure?

A stored procedure is a named group of SQL statements that have been previously created and stored in the server database.

9. Describe concurrency control?

Concurrency control is the process managing simultaneous operations against a database so that database integrity is not compromised. There are two approaches to concurrency control.

The pessimistic approach involves locking and the optimistic approach involves versioning.

10. What is Fully Functional dependency?

A functional dependency $X \rightarrow Y$ is full functional dependency if removal of any attribute A from X means that the dependency does not hold any more.

e) Essay type Questions <As per requirements>

1. What is functional dependency? Explain briefly?

Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A₁, A₂, ..., A_n, then those two tuples must have to have same values for attributes B₁, B₂, ..., B_n.

Functional dependency is represented by an arrow sign (\rightarrow) that is, $X \rightarrow Y$, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side.

Armstrong's Axioms:

If F is a set of functional dependencies then the closure of F, denoted as F^+ , is the set of all functional dependencies logically implied by F. Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

- **Reflexive rule** – If alpha is a set of attributes and beta is_subset_of alpha, then alpha holds beta.
- **Augmentation rule** – If $a \rightarrow b$ holds and y is attribute set, then $ay \rightarrow b$ also holds. That is adding attributes in dependencies, does not change the basic

dependencies.

- **Transitivity rule** – Same as transitive rule in algebra, if $a \rightarrow b$ holds and $b \rightarrow c$ holds, then $a \rightarrow c$ also holds. $a \rightarrow b$ is called as a functionally that determines b .

Trivial Functional Dependency:

- **Trivial** – If a functional dependency (FD) $X \rightarrow Y$ holds, where Y is a subset of X , then it is called a trivial FD. Trivial FDs always hold.
- **Non-trivial** – If an FD $X \rightarrow Y$ holds, where Y is not a subset of X , then it is called a non-trivial FD.
- **Completely non-trivial** – If an FD $X \rightarrow Y$ holds, where X intersect $Y = \emptyset$, it is said to be a completely non-trivial FD.

2. Define normalization? Explain with an example?

Database Normalisation is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy, if a database design is not perfect, it may contain anomalies, like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables.

Problem Without Normalization

Without Normalization, it becomes difficult to handle and update the database, without facing data loss. Insertion, Updation and Deletion Anomalies are very frequent if Database is not Normalized. To understand these anomalies.

let us take an example of **Student** table.

| S_id | S_Name | S_Address | Subject_opted |
|------|--------|-----------|---------------|
| 401 | Adam | Noida | Bio |
| 402 | Alex | Panipat | Maths |
| 403 | Stuart | Jammu | Maths |

- **Update anomalies** – If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a

few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.

- **Deletion anomalies** – We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.
- **Insert anomalies** – We tried to insert data in a record that does not exist at all.

3. Give a detailed on first normal form with an example?

As per First Normal Form, no two Rows of data must contain repeating group of information i.e each set of column must have a unique value, such that multiple columns cannot be used to fetch the same row. Each table should be organized into rows, and each row should have a primary key that distinguishes it as unique.

The **Primary key** is usually a single column, but sometimes more than one column can be combined to create a single primary key. For example consider a table which is not in First normal form

Student Table :

| Student | Age | Subject |
|---------|-----|----------------|
| Adam | 15 | Biology, Maths |
| Alex | 14 | Maths |
| Stuart | 17 | Maths |

In First Normal Form, any row must not have a column in which more than one value is saved, like separated with commas. Rather than that, we must separate such data into multiple rows.

Student Table following 1NF will be :

| Student | Age | Subject |
|---------|-----|---------|
| Adam | 15 | Biology |

| | | |
|--------|----|-------|
| Adam | 15 | Maths |
| Alex | 14 | Maths |
| Stuart | 17 | Maths |

Using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

4. Explain second normal form with an example?

As per the Second Normal Form there must not be any partial dependency of any column on primary key. It means that for a table that has concatenated primary key, each column in the table that is not part of the primary key must depend upon the entire concatenated key for its existence. If any column depends only on one part of the concatenated key, then the table fails **Second normal form**.

In example of First Normal Form there are two rows for Adam, to include multiple subjects that he has opted for. While this is searchable, and follows First normal form, it is an inefficient use of space. Also in the above Table in First Normal Form, while the candidate key is **{Student, Subject}**, **Age** of Student only depends on Student column, which is incorrect as per Second Normal Form. To achieve second normal form, it would be helpful to split out the subjects into an independent table, and match them up using the student names as foreign keys.

New Student Table following 2NF will be :

| Student | Age |
|---------|-----|
| Adam | 15 |
| Alex | 14 |
| Stuart | 17 |

In Student Table the candidate key will be **Student** column, because all other column i.e **Age** is dependent on it.

New Subject Table introduced for 2NF will be :

| Student | Subject |
|---------|---------|
| | |

| | |
|--------|---------|
| Adam | Biology |
| Adam | Maths |
| Alex | Maths |
| Stuart | Maths |

In Subject Table the candidate key will be {Student, Subject} column. Now, both the above tables qualifies for Second Normal Form and will never suffer from Update Anomalies.

5. Explain Boyce and Codd Normal Form with an example?

Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with certain type of anomaly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, following conditions must be satisfied:

- R must be in 3rd Normal Form
- and, for each functional dependency ($X \rightarrow Y$), X should be a super Key.

Consider the following relationship : **R (A,B,C,D)**

and following dependencies :

$$A \rightarrow BCD$$

$$BC \rightarrow AD$$

$$D \rightarrow B$$

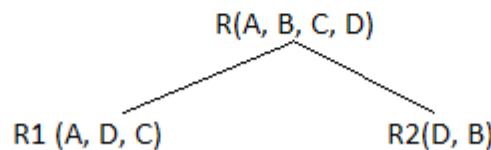
Above relationship is already in 3rd NF. Keys are **A** and **BC**.

Hence, in the functional dependency, **A → BCD**, A is the super key.

in second relation, **BC → AD**, BC is also a key.

but in, **D → B**, D is not a key.

Hence we can break our relationship R into two relationships **R1** and **R2**.



Breaking, table into two tables, one with A, D and C while the other with D and B.

4.4.7. Skill Building Exercises/Assignments

1. What is functional dependency? Explain briefly?
2. Define normalization? Explain various normal forms?
3. Give a detailed note on Boyce codd normal form?
4. Define decomposition? Explain the characteristics of decomposition?
5. Explain dependency preserving decomposition?

4.4.8. Previous Questions (Asked by JNTUK from the concerned

Unit)

4.4.9. GATE Questions (Where relevant)

1. Which normal form is considered adequate for normal relational database design?
(a) 2NF (b) 5NF (c) 4NF (d) 3NF

Ans: option (d)

2. Consider a schema $R(A, B, C, D)$ and functional dependencies $A \rightarrow B$ and $C \rightarrow D$. Then the decomposition of R into $R_1(A, B)$ and $R_2(C, D)$ is

- (a) dependency preserving and lossless join
- (b) lossless join but not dependency preserving
- (c) dependency preserving but not lossless join
- (d) not dependency preserving and not lossless join

Ans: option (c)

3. Relation R with an associated set of functional dependencies, F , is decomposed into BCNF. The redundancy (arising out of functional dependencies) in the resulting set of relations is

- (a) Zero
- (b) More than zero but less than that of an equivalent 3NF decomposition
- (c) Proportional to the size of F^+
- (d) Indeterminate

Ans: option (b)

4. Which one of the following statements about normal forms is FALSE?

- (a) BCNF is stricter than 3NF
- (b) Lossless, dependency-preserving decomposition into 3NF is always possible
- (c) Lossless, dependency-preserving decomposition into BCNF is always possible
- (d) Any relation with two attributes is in BCNF

Ans: option (c)

5. A table has fields F1, F2, F3, F4, and F5, with the following functional dependencies:

F1->F3

F2->F4

(F1,F2)->F5

in terms of normalization, this table is in

- (a) 1NF (b) 2NF (c) 3NF (d) None of these

Ans: option (a)

6. Which of the following is TRUE?

- (a) Every relation in 2NF is also in BCNF
- (b) A relation R is in 3NF if every non-prime attribute of R is fully functionally dependent on every key of R
- (c) Every relation in BCNF is also in 3NF
- (d) No relation can be in both BCNF and 3NF

Ans: option (c)

7. Consider the following functional dependencies in a database.

Date_of_Birth->Age Age->Eligibility

Name->Roll_number Roll_number->Name

Course_number->Course_name Course_number->Instructor

(Roll_number, Course_number)->Grade

The relation (Roll_number, Name, Date_of_birth, Age) is

- (a) in second normal form but not in third normal form
- (b) in third normal form but not in BCNF
- (c) in BCNF
- (d) in none of the above

Ans: option (d)

8. The relation schema Student_Performance (name, courseNo, rollNo, grade) has the following FDs:

name, courseNo \rightarrow grade

rollNo, courseNo \rightarrow grade

name \rightarrow rollNo

rollNo \rightarrow name

The highest normal form of this relation scheme is

- (a) 2NF (b) 3NF (c) BCNF (d) 4NF

Ans: option (b)

9. The relation EMPDT1 is defined with attributes empcode(unique), name, street, city, state, and pincode. For any pincode, there is only one city and state. Also, for any given street, city and state, there is just one pincode. In normalization terms EMPDT1 is a relation in

- (a) 1NF only
(b) 2NF and hence also in 1NF
(c) 3NF and hence also in 2NF and 1NF
(d) BCNF and hence also in 3NF, 2NF and 1NF

Ans: option (b)

10. Which one of the following statements is FALSE?

- (a) Any relation with two attributes is in BCNF
(b) A relation in which every key has only one attribute is in 2NF
(c) A prime attribute can be transitively dependent on a key in a 3 NF relation.
(d) A prime attribute can be transitively dependent on a key in a BCNF relation.

Ans: option (d)

11. Consider the following relational schemes for a library database:

Book (Title, Author, Catalog_no, Publisher, Year, Price)

Collection (Title, Author, Catalog_no)

With the following functional dependencies:

I. Title Author \rightarrow Catalog_no

II. Catalog_no \rightarrow Title Author Publisher Year

III. Publisher Title Year \rightarrow Price

Assume {Author, Title} is the key for both schemes. Which of the following statements is true?

- (a) Both Book and Collection are in BCNF
- (b) Both Book and Collection are in 3NF only
- (c) Book is in 2NF and Collection is in 3NF
- (d) Both Book and Collection are in 2NF only

Ans: option (c)

12. Let R(A,B,C,D,E,P,G) be a relational schema in which the following FDs are known to hold:

AB \rightarrow CD

DE \rightarrow P

C \rightarrow E

P \rightarrow C

B \rightarrow G

The relation schema R is

- (a) in BCNF
- (b) in 3NF, but not in BCNF
- (c) in 2NF, but not in 3NF
- (d) not in 2NF

Ans: option (d)

13. How many candidate keys does the relation R have?

- (a) 3 (b) 4 (c) 5 (d) 6

Ans: option (b)

14. The relation R is

- (a) in 1NF, but not in 2NF.
- (b) in 2NF, but not in 3NF.
- (c) in 3NF, but not in BCNF.
- (d) in BCNF.

Ans: option (a)

4.4.10. Interview questions(which are frequently asked in a technical round – placements)

1. What is normalization?

It is a process of analysing the given relation schemas based on their Functional Dependencies (FDs) and primary key to achieve the properties

(1).Minimizing redundancy, (2). Minimizing insertion, deletion and update anomalies.

2. What is Functional Dependency?

A Functional dependency is denoted by $X \rightarrow Y$ between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuple that can form a relation state r of R. The constraint is for any two tuples t_1 and t_2 in r if $t_1[X] = t_2[X]$ then they have $t_1[Y] = t_2[Y]$. This means the value of X component of a tuple uniquely determines the value of component Y.

3. What is Lossless join property?

It guarantees that the spurious tuple generation does not occur with respect to relation schemas after decomposition.

4. What is 1 NF (Normal Form)?

The domain of attribute must include only atomic (simple, indivisible) values.

5. What is Fully Functional dependency?

It is based on concept of full functional dependency. A functional dependency $X \rightarrow Y$ is full functional dependency if removal of any attribute A from X means that the dependency does not hold any more.

6. What is 2NF?

A relation schema R is in 2NF if it is in 1NF and every non-prime attribute A in R is fully functionally dependent on primary key.

7. What is 3NF?

A relation schema R is in 3NF if it is in 2NF and for every FD $X \rightarrow A$ either of the following is true

X is a Super-key of R.

A is a prime attribute of R.

In other words, if every non prime attribute is non-transitively dependent on primary key.

8. What is BCNF (Boyce-Codd Normal Form)?

A relation schema R is in BCNF if it is in 3NF and satisfies an additional constraint that for every FD $X \rightarrow A$, X must be a candidate key.

4.4.11. Real-Word (Live) Examples / Case studies wherever applicable –

Not applicable in this unit

4.4.12. Suggested "Expert Guest Lectures" (both from in and outside of the campus)

Not applicable in this unit

4.4.13. Literature references of Relevant NPTEL Videos/Web/YouTube videos etc.

- c. <https://www.youtube.com/watch?v=1057YmExS-I>
- d. <https://www.youtube.com/watch?v=FR4QleZaPeM>

4.4.14. Any Lab requirements; if so link it to Lab Lesson Plan.

Not applicable in this unit

4.4.15. Reference Text Books / with Journals Chapters etc.

Textbooks:

1. Database Management Systems, 3/e Raghuram Krishnan, Johannes Gehrke, TMH
2. Database Management System, 6/e Ramez Elmasri, Shamkant B. Navathe, PEA
3. Database Principles Fundamentals of Design Implementation and Management, Corlos Coronel, Steven Morris, Peter Robb, Cengage Learning

4.5. Transaction Management and Concurrency Control:

4.5.1. Unit Objectives:

- Describe the different problems in managing the distributed

Transaction, properties of transactions, transaction log, and transaction management with SQL using commit rollback and savepoint.

Concurrency control for lost updates, uncommitted data, inconsistent retrievals and the Scheduler. Concurrency control with locking methods : lock granularity, lock types, two phase locking for ensuring serializability, deadlocks, Concurrency control with time stamp ordering : Wait/Die and Wound/Wait Schemes, Database Recovery management : Transaction recovery.

SQL constructs that grant access or revoke access from user or user groups. Basic transactions.

- Discuss about recovery of distributed transactions
- Explain a popular algorithm called 2-phase commit protocol
- Discuss the aspects of concurrency control in distributed transactions

4.5.2. Unit Objectives:

- Understanding the purpose of concurrency control
- Explain what a transaction is, its properties and reason for designing databases around transactions
- Analyze the problems of data management in a concurrency environment
- Critically compare the relative strengths of different concurrency control approaches

4.5.3. Unit Lecture Plan:

| Lecture No. | Topic | Methodology | Quick reference |
|--------------------|--|--------------------|------------------------|
| 1 | Transaction, properties of transactions, transaction log | Chalk & Board | Text. Book with Page |
| 2 | transaction management with SQL using commit rollback and savepoint. | Chalk & Board | Text. Book with Page |
| 3 | Concurrency control for lost updates | Chalk & Board | Text. Book with Page |
| 4 | uncommitted data, inconsistent retrievals and the Scheduler | Chalk & Board | Text. Book with Page |
| 5 | Concurrency control with locking methods | Chalk & Board | Text. Book with Page |
| 6 | lock granularity, lock types | Chalk & Board | Text. Book with Page |
| 7 | two phase locking for ensuring serializability | Chalk & Board | Text. Book with Page |
| 8 | Deadlocks | Chalk & Board | Text. Book with |

| | | | Page |
|----|--|---------------|----------------------|
| 9 | Concurrency control with time stamp ordering : Wait/Die and Wound/Wait Schemes | Chalk & Board | Text. Book with Page |
| 10 | Database Recovery management : Transaction recovery. | Chalk & Board | Text. Book with Page |
| 11 | SQL constructs that grant access | Chalk & Board | Text. Book with Page |
| 12 | revoke access from user or user groups | Chalk & Board | Text. Book with Page |

4.5.4 Teaching Material / Teaching Aids as per above lecture plan.

4.5.4.1.1.1 LECTURE- 1

Transaction:

A transaction is an event which occurs on the database. Collection of operations that form a single logical unit of work are called transaction.

➤ **Transaction access data using two operations :**

Generally a transaction **reads** a value from the database or **writes** a value to the database. A read operation does not change the image of the database in any way. But a write operation, whether performed with the intention of inserting, updating or deleting data from the database, changes the image of the database.

➤ Say for example, we have two accounts A and B, each containing Rs 1000/-. We now start a transaction to deposit Rs 100/- from account A to Account B.

```
T1: Read A;  
A = A - 100;  
Write A;  
Read B;  
B = B + 100;  
Write B;
```

- **The Four Properties** of Transactions Every transaction, for whatever purpose it is being used, has the following four properties. Taking the initial letters of these four properties we collectively call them the **ACID Properties**.
 - i) Atomicity= all changes are made (commit), or none (rollback).
 - ii) Consistency = transaction won't violate declared system integrity constraints
 - iii) Isolation= results independent of concurrent transactions.
 - iv) Durability= committed changes survive various classes of hardware failure

4.5.4.2 LECTURE -2

PROPERTIES OF TRANSACTION

1) ATOMICITY :This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.

Eg: Suppose before execution of transaction T_i the values of accounts A & B are 1000\$ and 2000\$ respectively. Now suppose during execution of transaction T_i , a failure occurs. Consider a failure occurs after write(A) operation but before write(B), then the values reflected in the database are 950\$ and 2000\$. Thus, the sum A+B no longer preserved. To avoid this, atomicity should be ensured. To ensure atomicity database system keeps track of the old values of any data on which a transaction performs a write. If the transaction does not complete its execution, the database restores the old values. Atomicity is handled by transaction management component.

2) CONSISTENCY:The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.

Eg: Transaction T_1 transfers \$100 from Account A to Account B. Both Account A and Account B contains \$500 each before the transaction.

Transaction T_1

Read (A)

A=A-100

Write (A)

Read (B)

B=B+100

Write (B)

Consistency Constraint :

Before Transaction execution Sum = A + B Sum = 500 + 500 Sum = 1000

After Transaction execution Sum = A + B Sum = 400 + 600 Sum = 1000

Before the execution of transaction and after the execution of transaction SUM must be equal.

3) ISOLATION: In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

Eg: The database is said to be temporarily inconsistent while the transaction of transfer funds from A to B is executing and at the same time second concurrently running transaction reads at this intermediate point and computes A+B it will observe an inconsistent value which results in inconsistent state even after both transactions have completed. To avoid this problem of concurrent execution, transactions should be executed in isolation. The isolation property of a transaction ensures that the concurrent execution of transaction results in a system state that could have been obtained if transactions are executed serially i.e one after the other.

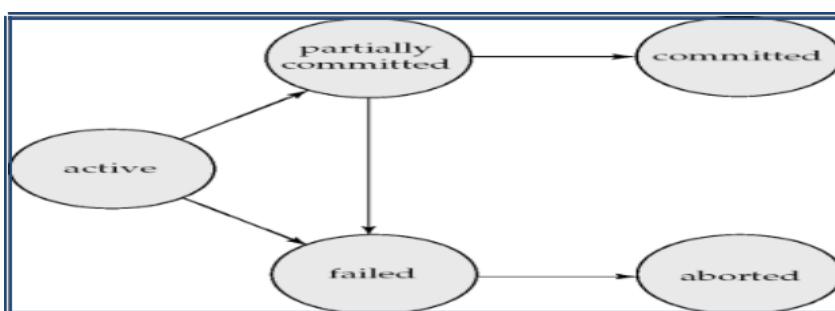
4) DURABILITY: Durability ensures that any transaction committed to the database will not be lost. Durability is ensured through the use of database backups and transaction logs that facilitate the restoration of committed transactions in spite of any subsequent software or hardware failures. The database should be durable enough to hold all its latest updates even if the system fails or restarts.

Eg: If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.

TRANSACTION STATE DIAGRAM :

The following are the different states in transaction processing in a Database System.

1. Active: This is the initial state. The transaction stay in this state while it is executing.



2. Partially Committed: This is the state after the final statement of the transaction is executed.

3. Failed: After the discovery that normal execution can no longer proceed.

4. Aborted: The state after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.

5. Committed: The state after successful completion of the transaction. We cannot abort or rollback a committed transaction.

4.5.4.3 Lecture- 3

TRANSACTION LOG

- **Log** is the most widely used structure for recording database modifications.
The log is a sequence of log records, recording all the update activities in the database. It has following **four fields** :

1) Transaction identifier : is the unique identifier of the transaction that performs the write operation.

2) Data item identifier: is the unique identifier of the data item written.

3) Old value : is the value of the data item prior to the write.

4) New value : is the value of the data item that it will have after the write.

It is represented as :

<Ti start> : Transaction Ti is started

<Ti, Xj, V1, V2> : Transaction Ti has performed a write on data item Xj, Xj had value V1 before the write and will have value V2 after the write.

<Ti commit> : Transaction Ti has committed

<Ti abort> : Transaction Ti has aborted

4.5.4.4 Lecture -4

TRANSACTION MANAGEMENT WITH SQL USING COMMIT, ROLLBACK &SAVEPOINT

Transaction control commands manage changes made by DML commands. The following describes the nature of transactions:

- All transactions have a beginning and an end.
- A transaction can be saved or undone.
- If a transaction fails in the middle, no part of the transaction can be saved to the database.

Transactional control is the ability to manage various transactions that may occur within a relational database management system. There are **three commands used**

to control transactions :

- i) Commit ii) Rollback iii) Savepoint

- 1) **COMMIT** : The commit command saves all transactions to the database since the last COMMIT or ROLLBACK command happens.

Syntax : commit[work];

The keyword commit is the only mandatory part of the syntax. Keyword work is optional.

Eg: SQL> delete from emp

Where emp_age>60;

This deletes the record of those employee whose age is above 60years.

Though the changes are reflected on database they are not actually saved in database, rather they are stored in temporary area. To **store changes** permanently in the database commit command is used.

SQL>commit work;

The above command will make changes permanently in database, since last commit or rollback command was issued.

- 2) **ROLLBACK**: The rollback command is the transactional control command used to undo transactions that have not already been saved to the database. The rollback command can only be used to undone the transactions since the last COMMIT or ROLLBACK command was issued.

Syntax : rollback[work];

The keyword rollback is the only mandatory part of the syntax. Keyword work is optional.

Eg: SQL> delete from emp

Where emp_age>60;

This deletes the record of those employee whose age is above 60years. Though the changes are reflected on database they are not actually saved in database, rather they are stored in temporary area. To **discard changes** permanently in the database commit command is used.

SQL>rollback work;

The above command will discard changes made on database, since last commit or rollback command was issued.

- 3) **SAVEPOINT**: Savepoints offer a mechanism to rollback portions of transactions.

General eg : Consider that a person walking and after passing some distance the road got split into two tracks. The person is not sure to choose which track, so before randomly selecting one track he will make a signal flag, so that if the track was not the right one he can rollback to signal flag and select the right track. In this example the signal flag becomes the savepoint.

Syntax : savepointname_of_savepoint;

Eg: Before deleting the records of employee we should create a savepoint and then use delete command to delete employee records.

SQL>savepointDeleteEmp;

SQL>delete from emp

Where emp_age>60;

After some time, if manager orders to not delete employer's record the changes can undone by using the rollback to savepoint command.

SQL>rollback to DeleteEmp;

It will rollback the changes made to Emp table.

4.5.4.5 Lecture - 5

CONCURRENCY CONTROL FOR LOST UPDATES, UNCOMMITTED DATA, INCONSISTENT RETRIEVALS AND THE SCHEDULER :

Concurrent execution have following problems:

- Lost updates
 - Uncommitted data/ Dirty read
 - Inconsistent retrieval/ Unrepeatable read
- 1) **LOST UPDATES:** The update of one transaction is overwritten by another transaction.

Eg: Suppose T1 credits 100\$ to account A and T2 debits 50\$ from account A. The initial value of A=500\$. If credit and debit happens correctly the final value is 550\$. But, if we run T1 & T2 concurrently as follows:

| T1(Credit) | T2(Debit) |
|------------------|------------------|
| Read(A) {A=500} | Read(A) {A=500} |
| A:A+100 {A=600} | A:A-50 {A=450} |
| Write(A) {A=600} | Write(A) {A=450} |
| | Commit |

Final value of A=450. The credit of T1 is missing (lost update) from the account.

- 2) **UNCOMMITTED DATA/DIRTY READ:** Reading of a non-existent value of A by T2. If T1 updates A which is after read by T2, then if at sudden T1 aborts then the value which is read by T2 will never be existed.

Eg: Let T1 modified A=600. But T1 failed and the update value 600 is removed from the database then A will get stored to its initial old value 500. Hence the value read by T2 i.e A=600 is an non-existent value(reading dirty data and no commit command).

| T1(Credit) | T2(Debit) |
|------------|-----------|
|------------|-----------|

| | |
|---|--|
| Read(A) {A=500} A:A+100 {A=600} Write(A) {A=600} T1 failed to complete | Read(A) {A=500} A:A+100 {A=600} Write(A) {A=600} |
|---|--|

- 3) **UNREPEATABLE READ/ INCONSISTENT RETRIEVAL:** If T2 reads A, which is then altered by T1 and T1 commits now when T2 re-read A it will find a different value of A in the second read.

| T1(Credit) | T2(Debit) |
|--|---|
| Read(A) {A=500} A:A+100 {A=600} Write(A) {A=600} | Read(A) {A=500} A:A-50 {A=450} Commit Write(A) {A=600} |

Eg: Here T1 and T2 reads A=500. Now, T1 modifies A to 600 when, T2 reads

it gets value of 600 this should not be the case because T2 in the same execution should get only one value of A (500 or 600 but not both) which is happening after the commit.

SCHEDULAR:

When multiple transactions are executing concurrently, then the order of execution of operations from the various transactions is known as schedule. 1)Serial Schedule2)Non-Serial Schedule

Serial Schedule:Transactions are executed one by one without any interleaved operations from other transactions.

Non-Serial Schedule:A schedule where the operations from a set of concurrent transactions are interleaved.

SERIALIZABILITY:

What is Serializability?

A given non serial schedule of n transactions is serializable,if it is equivalent to some serial schedule.

i.e. this non serial schedule produce the same result as of the serial schedule. Then the given non serial schedule is said to be serializable. A schedule that is not serializable is called a non-serializable.

Non-Serial Schedule Classification:

Serializable,NotSerializable,Recoverable,NonRecoverable.

Serializable Schedule Classification:ConflictSerializable,ViewSerializable.

i)ConflictSerializableSchedule:If a schedule S can be transformed into a schedule S' by a series of swaps of non conflicting instruction then we say that S and S' are conflict equivalent. A schedule S is called conflict serializable if it is conflict equivalent to a serial schedule.

ii)ViewSerializableSchedule:All conflict serializable schedule are view serializable. But there are view serializable schedule that are not conflict serializable. A schedule S is a view serializable if it is view equivalent to a serial schedule.

iii) Recoverable Schedule Classification:Cascade and Cascadeless.

To recover from the failure of a transaction Ti, we may have to rollback several transactions.This phenomenon in which a single transaction failure leads to a series of transaction is called cascading roll back. Avoid cascading roll back by not allowing reading uncommitted data.But this lead to a serial schedule.

4.5.4.6 Lecture - 6

CONCURRENCY CONTROL WITH LOCKING METHODS :

In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions. We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions.

Different types of protocols/schemes used to control concurrent execution of

transactions are:

- Lock based protocol
- Timestamp based protocol
- Validation based protocol
- Multiple granularity

1) LOCK BASED PROTOCOL:

Database systems equipped with lock-based protocols by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds:

- **Binary Locks** : A lock on a data item can be in two states; it is either locked or unlocked.
- **Shared/exclusive Locks**: This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write as well as read operation, it is an exclusive lock.

Read locks are shared because no data value is being changed.

There are four types of lock protocols available:

i)**Simplistic Lock Protocol** :Simplistic lock-based protocols allow transactions to obtain a lock on every object before a 'write' operation is performed. Transactions may unlock the data item after completing the 'write' operation.

ii)**Pre-claiming Lock Protocol**:Pre-claiming protocols evaluate their operations and create a list of data items on which they need locks. Before initiating an execution, the transaction requests the system for all the locks it needs beforehand. If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.

iii)**Two-Phase Locking – 2PL**: This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.

Eg:lock X(B);

```
    Read B;  
    B: = B – 100;  
    Write B;  
    lock X(A);  
    Read A;  
    A: = A + 100;  
    Write A;  
    unlock(B);
```

unlock(A);

Two-phase locking has two phases: one is **growing**, where all the locks are being acquired by the transaction; and the second phase is **shrinking**, where the locks held by the transaction are being released. To claim an exclusive (write) lock, a transaction must first acquire a shared (read) lock and then upgrade it to an exclusive lock.

iv) Strict Two-Phase Locking: The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time. Strict-2PL does not have cascading abort as 2PL does.

2) TIMESTAMP BASED PROTOCOL :

The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp. Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one. In addition, every data item is given the latest read and write-timestamp. This lets the system know when the last 'read and write' operation was performed on the data item.

Timestamp Ordering Protocol :

The timestamp-ordering protocol ensures serializability among transactions in their conflicting read and write operations. This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.

- The timestamp of transaction T_i is denoted as $TS(T_i)$.
- Read timestamp of data-item X is denoted by $R\text{-timestamp}(X)$.
- Write timestamp of data-item X is denoted by $W\text{-timestamp}(X)$.

Timestamp ordering protocol works as follows:

i) If a transaction T_i issues a $\text{read}(X)$ operation:

- If $TS(T_i) < W\text{-timestamp}(X)$ Operation rejected.
- If $TS(T_i) \geq W\text{-timestamp}(X)$ Operation executed.

ii) If a transaction T_i issues a $\text{write}(X)$ operation:

- If $TS(T_i) < R\text{-timestamp}(X)$ Operation rejected.
- If $TS(T_i) < W\text{-timestamp}(X)$ Operation rejected and T_i rolled back.

Thomas' Write Rule: This rule states if $TS(T_i) < W\text{-timestamp}(X)$, then the operation is rejected and T_i is rolled back. Timestamp ordering rules can be modified to make the schedule view serializable. Instead of making T_i rolled back, the 'write' operation itself is ignored.

3) VALIDATION BASED PROTOCOL:

Execution of transaction T_i is done in three phases.

1. **Read and execution phase:** Transaction T_i writes only to temporary local variables
2. **Validation phase:** Transaction T_i performs a ``validation test'' to determine if local variables can be written without violating serializability.
3. **Write phase:** If T_i is validated, the updates are applied to the database; otherwise, T_i is rolled back.

Also called as **optimistic concurrency control** since transaction executes fully in the hope that all will go well during validation.

Each transaction T_i has 3 timestamps

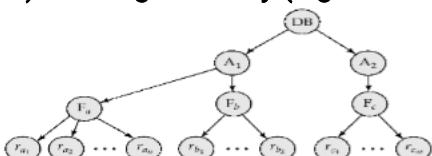
- $\text{Start}(T_i)$: the time when T_i started its execution
- $\text{Validation}(T_i)$: the time when T_i entered its validation phase
- $\text{Finish}(T_i)$: the time when T_i finished its write phase

4) MULTIPLE GRANULARITY:

Allow data items to be of various sizes and define a hierarchy of data granularities, where the small granularities are nested within larger ones. Can be represented graphically as a tree (but don't confuse with tree-locking protocol). When a transaction locks a node in the tree *explicitly*, it *implicitly* locks all the node's descendants in the same mode.

Granularity of locking (level in tree where locking is done):

- i) **fine granularity (lower in tree):** high concurrency, high locking overhead.
- ii) **coarse granularity (higher in tree):** low locking overhead, low concurrency



The levels, starting from the coarsest (top) level are *database, area, file and record*.

4.5.4.7. Lecture - 7

DEADLOCKS:

In a multi-process system, deadlock is an unwanted situation that arises in a shared resource environment, where a process indefinitely waits for a resource that is held by another process.

For example, assume a set of transactions $\{T_0, T_1, T_2, \dots, T_n\}$. T_0 needs a resource X to complete its task. Resource X is held by T_1 , and T_1 is waiting for a resource Y, which is held by T_2 . T_2 is waiting for resource Z, which is held by T_0 . Thus, all the processes wait for each other to release resources. In this situation, none of the processes can finish their task. This situation is known as a deadlock. Deadlocks are not healthy for a system. In case a system is stuck in a deadlock, the transactions involved in the deadlock are either rolled back or restarted.

4.5.4.8 Lecture - 8

CONCURRENCY CONTROL WITH TIME STAMP ORDERING:

Deadlock Prevention:

To prevent any deadlock situation in the system, the DBMS aggressively inspects all the operations, where transactions are about to execute. The DBMS inspects the operations and analyzes if they can create a deadlock situation. If it finds that a deadlock situation might occur, then that transaction is never allowed to be executed.

There are deadlock prevention schemes that use timestamp ordering mechanism of transactions in order to predetermine a deadlock situation.

i) Wait-Die Scheme: In this scheme, if a transaction requests to lock a resource (data item), which is already held with a conflicting lock by another transaction, then one of the two possibilities may occur:

- If $TS(T_i) < TS(T_j)$ – that is T_i , which is requesting a conflicting lock, is older than T_j – then T_i is allowed to wait until the data-item is available.
- If $TS(T_i) > TS(T_j)$ – that is T_i is younger than T_j – then T_i dies. T_i is restarted later with a random delay but with the same timestamp. This scheme allows the older transaction to wait but kills the younger one.

ii) Wound-Wait Scheme: In this scheme, if a transaction requests to lock a resource (data item), which is already held with conflicting lock by another transaction, one of the two possibilities may occur:

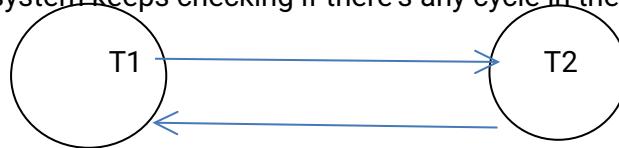
- If $TS(T_i) < TS(T_j)$, then T_i forces T_j to be rolled back – that is T_i wounds T_j . T_j is restarted later with a random delay but with the same timestamp.

- If $TS(T_i) > TS(T_j)$, then T_i is forced to wait until the resource is available. This scheme allows the younger transaction to wait; but when an older transaction requests an item held by a younger one, the older transaction forces the younger one to abort and release the item. In both the cases, the transaction that enters the system at a later stage is aborted.

Deadlock Avoidance:

Aborting a transaction is not always a practical approach. Instead, deadlock avoidance mechanisms can be used to detect any deadlock situation in advance. Methods like "wait-for graph" are available but they are suitable for only those systems where transactions are lightweight having fewer instances of resource. In a bulky system, deadlock prevention techniques may work well.

Wait-for Graph: This is a simple method available to track if any deadlock situation may arise. For each transaction entering into the system, a node is created. When a transaction T_i requests for a lock on an item, say X , which is held by some other transaction T_j , a directed edge is created from T_i to T_j . If T_j releases item X , the edge between them is dropped and T_i locks the data item. The system maintains this wait-for graph for every transaction waiting for some data items held by others. The system keeps checking if there's any cycle in the graph.



Here, we can use any of the two following approaches:

- First, do not allow any request for an item, which is already locked by another transaction. This is not always feasible and may cause starvation, where a transaction indefinitely waits for a data item and can never acquire it.
- The second option is to roll back one of the transactions. It is not always feasible to roll back the younger transaction, as it may be important than the older one. With the help of some relative algorithm, a transaction is chosen, which is to be aborted. This transaction is known as the victim and the process is known as victim selection.

Transaction Management

4.5.4.9. Lecture 9

DATABASE RECOVERY MANAGEMENT: TRANSACTION RECOVERY :

FAILURE CLASSIFICATION:

1) Transaction failure :

- Logical errors: transaction cannot complete due to some internal error condition.
- System errors: the database system must terminate an active transaction due to an error condition (e.g., deadlock)

2) System crash: a power failure or other hardware or software failure causes the system to crash.

Fail-stop assumption: non-volatile storage contents are assumed to not be corrupted by system crash. Database systems have numerous integrity checks to prevent corruption of disk data

3) Disk failure: a head crash or similar disk failure destroys all or part of disk storage

Destruction is assumed to be detectable: disk drives use checksums to detect

failures

To ensure atomicity despite failures, we first output information describing the modifications to stable storage without modifying the database itself. We study two approaches:

1) log-based recovery 2) shadow-paging

1) LOG BASED RECOVERY :

We assume (initially) that transactions run serially, that is, one after the other. A log is kept on stable storage. The log is a sequence of log records, and maintains a record of update activities on the database. Each and every log record is given a unique id called log sequence number.

A log record is written for each of the following actions:

- Updating a page
- Commit
- Abort
- End
- Undoing an update

Two approaches using logs :

i) Deferred database modification

ii) Immediate database modification

- **The deferred database modification** scheme records all modifications to the log, but defers all the writes to after partial commit.
Assume that transactions execute serially. Transaction starts by writing $\langle T_i \text{ start} \rangle$ record to log.
 - i) A write(X) operation results in a log record $\langle T_i, X, V \rangle$ being written, where V is the new value for X
 - I Note: old value is not needed for this scheme
 - ii) The write is not performed on X at this time, but is deferred.
 - iii) When T_i partially commits, $\langle T_i \text{ commit} \rangle$ is written to the log
 - iv) Finally, the log records are read and used to actually execute the previously deferred writes.
 - **The immediate database modification** scheme allows database updates of an uncommitted transaction to be made as the writes are issued. Since undoing may be needed, update logs must have both old value and new value.
- i) Update log record must be written *before* database item is written
 - a. We assume that the log record is output directly to stable storage

- b. Can be extended to postpone log record output, so long as prior to execution of an output(B) operation for a data block B , all log records corresponding to items B must be flushed to stable storage

- ii) Output of updated blocks can take place at any time before or after transaction commit
- iii) Order in which blocks are output can be different from the order in which they are written.

2) SHADOW PAGING :

Shadow paging is an alternative to log-based recovery; this scheme is useful if transactions execute serially

Idea: maintain *two* page tables during the lifetime of a transaction –the current page table, and the shadow page table. Store the shadow page table in nonvolatile storage, such that state of the database prior to transaction execution may be recovered. Shadow page table is never modified during execution

To start with, both the page tables are identical. Only current page table is used for data item accesses during execution of the transaction. Whenever any page is about to be written for the first time

- A copy of this page is made onto an unused page.
- The current page table is then made to point to the copy
- The update is performed on the copy

Advantages of shadow-paging over log-based schemes:

- no overhead of writing log records
- recovery is trivial

CHECKPOINT:

Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system. As time passes, the log file may grow too big to be handled at all. Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

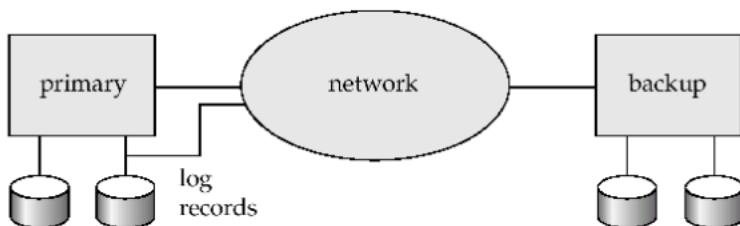
When a system with concurrent transactions crashes and recovers, it behaves in the following manner:

- The recovery system reads the logs backwards from the end to the last checkpoint.
- It maintains two lists, an undo-list and a redo-list.

- If the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>, it puts the transaction in the redo-list.
- If the recovery system sees a log with <Tn, Start> but no commit or abort log found, it puts the transaction in the undo-list. All the transactions in the undo-list are then undone and their logs are removed. All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.

REMOTE BACKUP:

Remote backup provides a sense of security in case the primary location where the database is located gets destroyed. Remote backup can be offline or realtime or online. In case it is offline, it is maintained manually.



Online backup systems are more real-time and lifesavers for database administrators and investors. An online backup system is a mechanism where every bit of the real-time data is backed up simultaneously at two distant places. One of them is directly connected to the system and the other one is kept at a remote place as backup.

As soon as the primary database storage fails, the backup system senses the failure and switches the user system to the remote storage. Sometimes this is so instant that the users can't even realize a failure.

ARIES:

- ARIES is a state of the art recovery method
 - i)Incorporates numerous optimizations to reduce overheads during normal processing and to speed up recovery
 - ii)Unlike the advanced recovery algorithm, ARIES
 - Uses log sequence number (LSN) to identify log records
 - Stores LSNs in pages to identify what updates have already been applied to a database page.
 - Physiological redo.
 - Dirty page table to avoid unnecessary redos during recovery

ARIES RECOVERY ALGORITHM:

ARIES recovery involves three passes

1)Analysis pass: Determines

- Which transactions to undo
- Which pages were dirty (disk version not up to date) at time of crash

- RedoLSN: LSN from which redo should start

2)Redo pass:

- Repeats history, redoing all actions from RedoLSN
- RecLSN and PageLSNs are used to avoid redoing actions already reflected on page

3)Undo pass:

- Rolls back all incomplete transactions.
- Transactions whose abort was complete earlier are not undone
- Key idea: no need to undo these transactions: earlier undo actions were logged, and are redone as required.

4.5.4.10. Lecture - 10

SQL CONSTRUCTS THAT GRANT ACCESS OR REVOKE ACCESS FROM USER OR USER GROUPS:

Data Control Language (DCL) statements is used to create roles, permissions, and referential integrity as well it is used to control access to database by securing it.

DCL Commands: Grant, Revoke

GRANT - gives user's access privileges to database

REVOKE - withdraw access privileges given with the GRANT command.

CREATING A USER

```
SQL>CONNECT SYSTEM/MANAGER;
SQL>CREATE USER "USERNAME" IDENTIFIED BY "PASSWORD"
SQL>GRANT DBA TO "USERNAME"
SQL>CONNECT "USERNAME"/"PASSWORD";
```

EXAMPLE:

CREATING A USER

```
SQL>CONNECT SYSTEM/MANAGER;
SQL>CREATE USER CSE2 IDENTIFIED BY CSECSE;
SQL>GRANT DBA TO CSE2;
SQL>CONNECT CSE2/CSECSE;
SQL>REVOKE DBA FROM CSE2;
```

4.5.4.11. Lecture - 11

PL/SQL PROCEDURES:

- PL/SQL stands for PROCEDURAL Language Extensions to SQL.
- PL/SQL extends SQL by adding programming structures and subroutines available in any high level language.
- PL/SQL can be used for both server-side and Client side Development.
- PL/SQL has syntax and rules that determine how programming statements work together.
- PL/SQL is not a stand alone Programming Language.

- PL/SQL is a part of the ORACLE RDBMS and hence can reside in two environments, the CLIENT and the SERVER.

Any MODULE that is developed using PL/SQL can be moved easily between SERVER SIDE and CLIENT SIDE applications.

- When PL/SQL Engine is located upon the SERVER, the whole PL/SQL block is passed to the PL/SQL Engine on the ORACLE SERVER.
- When the PL/SQL Engine is located upon the CLIENT, the PL/SQL processing is done on the CLIENT SIDE. All SQL statements that are embedded within the PL/SQL block, are sent to the ORACLE SERVER for further processing. If the PL/SQL block does not contain any SQL statements, the entire block is executed on the CLIENT SIDE.

PL/SQL BLOCK

DECLARE

—Declarations of memory variables, constants, cursorsetc., in PL/SQL

BEGIN

—SQL executable statements

—PL/SQL executable statements

EXCEPTION

/*SQL or PL/SQL code to handle errors that may arise during the execution of the code block between BEGIN and EXCEPTION section

END;

SYNTAX's of CONTROL STATEMENTS in PL/SQL

1. BRANCHING

2. SELECTION

3. LOOPING

1.BRANCHING STATEMENTS

i). Simple IF

ii). ELSIF

iii). ELSE IF

i) SIMPLE IF

```
IF condition THEN
```

```
    statement1;
```

```
    statement2;
```

```
END IF;
```

ii) IF-THEN-ELSE STATEMENT

```
IF condition THEN
```

```
    statement1;
```

```
ELSE
```

```
    statement2;
```

```
END IF;
```

iii) ELSIF STATEMENTS

```
IF condition1 THEN
```

```
    statement1;
```

```
ELSIF condition2 THEN
```

```
    statement2;
```

```
ELSIF condition3 THEN
```

```
    statement3;
```

```
ELSE
```

```
    statementn;
```

```
END IF;
```

iv) NESTED IF

```
IF condition THEN
```

```
    statement1;
```

```
ELSE
```

```
    IF condition THEN
```

```
        statement2;
```

```
    ELSE
```

```
        statement3;
```

```
    END IF;
```

```
END IF;
```

```
ELSE
```

```
    statement3;
```

```
END IF;
```

2) SELECTION IN PL/SQL

i) SIMPLE CASE

CASE SELECTOR

```
WHEN Expr1 THEN statement1;
```

```
WHEN Expr2 THEN statement2;
```

```
:
```

```
:
```

```

    :
ELSE
statementn;

END CASE;
ii)SEARCHED CASE

CASE
WHEN searchcondition1 THEN statement1;
WHEN searchcondition2 THEN statement2;
    :
    :
    :
ELSE
    statementn;
END CASE;

```

3)ITERATIONS IN PL/SQL

i)SIMPLE LOOP

```

LOOP
    statement1;
EXIT [ WHEN Condition];
END LOOP;
ii)WHILE LOOP

WHILE condition LOOP

    statement1;
    statement2;
END LOOP;

```

iii)FOR LOOP

```

FOR counter IN [REVERSE]
LowerBound..UpperBound
LOOP
    statement1;
    statement2;
END LOOP;

```

4.5.4.12. Lecture 12

FUNCTIONS AND TRIGGERS:

FUNCTIONS:

```

CREATE OR REPLACE FUNCTION FUNCTIONNAME (ARGUMENT {IN} RETURN
DATATYPE } {IS,AS}
VARIABLE DECLARATION;
CONSTANT DECLARATION;
BEGIN
PL/SQL SUBPROGRAM BODY;
EXCEPTION
EXCEPTION PL/SQL BLOCK;
END;

STORED PROCEDURES:
CREATE OR REPLACE PROCEDURE PROCEDURENAME (ARGUMENT {IN,OUT,INOUT}
DATATYPE } {IS,AS}
VARIABLE DECLARATION;
CONSTANT DECLARATION;
BEGIN
PL/SQL SUBPROGRAM BODY;
EXCEPTION
EXCEPTION PL/SQL BLOCK;
END;

```

TRIGGERS:

A trigger is an operation that is executed when some kind of event occurs to the database. It can be a data or object change.

Creation of Triggers:

- Triggers are created with the CREATE TRIGGER statement.
- This statement specifies that the In which table trigger is defined and on which events trigger will be invoked.
- To drop Trigger one can use DROP TRIGGER statement.

Syntax: CREATE TRIGGER [owner.]trigger_name ON[owner.] table_name
FOR[INSERT/UPDATE/DELETE] AS IF UPDATE(column_name) [{AND/OR}
UPDATE(COLUMN_NAME)...] { sql_statements }

Triggers Types:

- a) Row level Triggers b) Statement Level Triggers

Row Level triggers : A row level trigger is fired each time the table is affected by the triggering statement.

- Example: If an UPDATE statement updates multiple rows of a table, a row trigger s fired once for each row affected by the update statement.
- A row trigger will not run, if a triggering statement affects no rows.
- If FOR EACH ROW clause is written that means trigger is row level trigger.

Statement Level Triggers: A statement level trigger is fired once on behalf of the triggering statement, regardless of the number of rows in the table

that the triggering statement affects, even if no rows are affected.

- Example: If a DELETE statement deletes several rows from a table, a statement level DELETE trigger is fired only once.
- Default when FOR EACH ROW clause is not written in trigger that means trigger is statement level trigger

Rules of Triggers:

- Triggers cannot create or modify Database objects using triggers For example, cannot perform "CREATE TABLE..." or ALTER TABLE" sqlstatements under the triggers.
- It cannot perform any administrative tasks o For example, cannot perform "BACKUP DATABASE..." task under the triggers It cannot pass any kind of parameters.
- It cannot directly call triggers WRITETEXT statements do not allow a trigger

Advantages of Triggers:

Triggers are useful for auditing data changes or auditing database as well as managing business rules. Below are some examples:

- 1) Triggers can be used to enforce referential integrity (For example you may not be able to apply foreign keys)
- 2) Can access both new values and old values in the database when going to do any insert, update or delete

Disadvantages of Triggers

- Triggers hide database operations.
- For example when debugging a stored procedure, it's possible to not be aware that a trigger is on a table being checked for data changes
- Executing triggers can affect the performance of a bulk import operation .

4.5.5 Test Questions

c) Fill in the blanks type of questions :

1) _____ is collection of operations that performs a single logical function in database application.

2) Duty of Database manager is to enforce integrity and _____ checks.

3) Transaction management ensures _____ and _____ properties.
A) Atomicity and Abstraction

- B)Atomicity and Intigrity
- C)None of these
- D)Atomicity and Durability

4) _____ is an interface between low level database and application program.

5)Granting and Authorization for data access is provided by _____.

6)Database Manager is also known as _____.

7)User which interact with the system using database query language is called as _____.

8)Schema Definition is written by _____.

9)The property of transaction that persists all the crashes is

10) _____ states that only valid data will be written to the database.

d) Multiple choice questions

1)Who detects the failure of the system and restore the database to consistent state ?

- A)Application Programmer
- B)Database Administrator
- C)Storage Manager
- D)Naive User

2)Database Manager Performs following query related operation !

- A)Creating Schema
- B)Retrieving Record
- C)Creating View
- D)Drop Table

3)Schema Definition is written by _____.

- A)Database Manager
- B)Database Administrator

- C)Storage Manager
- D)Application Developer

4) Which of the user write program in host language and embed the DML statements into it ?

- A)Sophisticated User
- B)Specialized User
- C)Application Programmer
- D)Naive User

5) Which of the following has “all-or-none” property ?

- a) Atomicity
- b) Durability
- c) Isolation
- d) All of the mentioned

6) A sophisticated locking mechanism known as 2-phase locking which includes Growing phase and

- A. Shrinking Phase
- B. Release phase
- C. Commit phase
- D. Acquire Phase

7) A sophisticated locking mechanism known as 2-phase locking which includes Growing phase and

- A. Shrinking Phase
- B. Release phase
- C. Commit phase
- D. Acquire Phase

8) Commit and rollback are related to

- A. data integrity
- B. data consistency
- C. data sharing
- D. data security

9. The Oracle RDBMS uses the ____ statement to declare a new transaction start and its properties.

- a) BEGIN
- b) SET TRANSACTION
- c) BEGIN TRANSACTION
- d) COMMIT

e) True or False questions

1) Integrity Constraints are enforced by Naive Users while DBA write those integrity constraints.

- A) True B) False

2) The "C" in the ACID properties of transactions means that Concurrency control is required for simultaneous transactions.

- A) True
B) False

3) Durability of a transaction means that once a transaction is successfully completed, changes resulting from it are stored in the database on permanent storage forever.

- A) True
B) False

4) Both uncommitted dependency and inconsistent retrieval problems can occur in situations where only one transaction is writing to part of the database.

- A) True
B) False

5) Locks can be placed on a record or a table, but not the full database.

- A) True
B) False

6) An operating system failure affects all active transactions, while a device failure affects all active transactions plus all committed transactions that have been recorded on the disk that failed.

- A) True
B) False

7) The main objective of checkpoints is to reduce the number and duration of database backups.

- A) True
B) False

8) Since constraint checking must occur before the end of a transaction, if a larger transaction is divided into a number of smaller transactions it may be difficult to check some important constraints.

- A) True
- B) False

9) Balancing concurrency control overhead with potential interference problems can be achieved by selecting an appropriate isolation level for transactions.

- A) True
- B) False

10) By default, constraints are enforced immediately after each INSERT, UPDATE, and DELETE statement, but this may be overridden by the DBA with a constraint timing clause.

- A) True
- B) False

11) Database Manager is having control over the system.

- A) True
- B) False

4.5.6 Review Questions

1. Enumerate the properties of transaction.

Ans) PROPERTIES OF TRANSACTION

1) ATOMICITY: This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.

Eg: Suppose before execution of transaction T_i the values of accounts A & B are 1000\$ and 2000\$ respectively. Now suppose during execution of transaction T_i , a failure occurs. Consider a failure occurs after write(A) operation but before write(B), then the values reflected in the database are 950\$ and 2000\$. Thus, the sum A+B no longer preserved. To avoid this, atomicity should be ensured. To ensure atomicity database system keeps track of the old values of any data on which a transaction performs a write. If the transaction does not complete its execution, the

database restores the old values. Atomicity is handled by transaction management component.

2) CONSISTENCY:The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.

Eg: Transaction T1 transfers \$100 from Account A to Account B. Both Account A and Account B contains \$500 each before the transaction.

Transaction T1

Read (A)

A=A-100

Write (A)

Read (B)

B=B+100

Write (B)

Consistency Constraint :

Before Transaction execution Sum = A + B Sum = 500 + 500 Sum = 1000

After Transaction execution Sum = A + B Sum = 400 + 600 Sum = 1000

Before the execution of transaction and after the execution of transaction SUM must be equal.

3) ISOLATION: In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

Eg: The database is said to be temporarily inconsistent while the transaction of transfer funds from A to B is executing and at the same time second concurrently running transaction reads at this intermediate point and computes A+B it will observe an inconsistent value which results in inconsistent state even after both transactions have completed. To avoid this problem of concurrent execution, transactions should be executed in isolation. The isolation property of a transaction ensures that the concurrent execution of transaction results in a system state that could have been obtained if transactions are executed serially i.e one after the other.

4) DURABILITY:Durability ensures that any transaction committed to the database will not be lost.Durability is ensured through the use of database backups and transaction logs that facilitate the restoration of committed transactions in spite of any subsequent software or hardware failures. The database should be durable enough to hold all its latest updates even if the system fails or restarts.

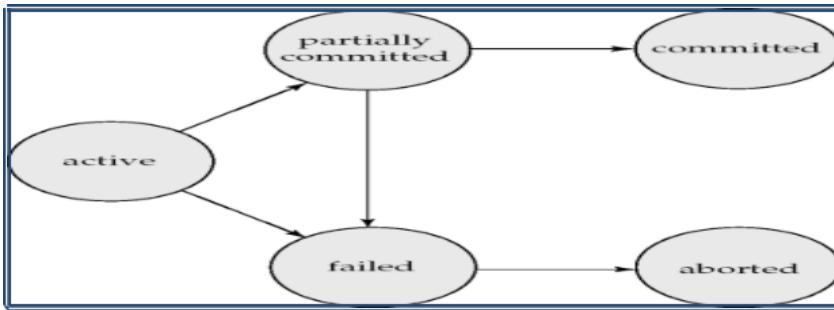
Eg:If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once

the system springs back into action.

TRANSACTION STATE DIAGRAM :

The following are the different states in transaction processing in a Database System.

1. Active: This is the initial state. The transaction stay in this state while it is executing.



2. Partially Committed:

Committed: This is the state after the final statement of the transaction is executed.

Failed: After the discovery that normal execution can no longer proceed.

Aborted: The state after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.

Committed: The state after successful completion of the transaction. We cannot abort or rollback a committed transaction.

2. Emphasize on concurrency control with locking methods.

Ans: CONCURRENCY CONTROL WITH LOCKING METHODS :

In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions. We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions.

Different types of protocols/schemes used to control concurrent execution of transactions are:

- Lock based protocol
- Timestamp based protocol
- Validation based protocol
- Multiple granularity

4) LOCK BASED PROTOCOL:

Database systems equipped with lock-based protocols by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds:

- **Binary Locks** : A lock on a data item can be in two states; it is either locked or unlocked.
- **Shared/exclusive Locks**: This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write as well as read operation, it is an exclusive lock.

Read locks are shared because no data value is being changed.

There are four types of lock protocols available:

i)**Simplistic Lock Protocol** :Simplistic lock-based protocols allow transactions to obtain a lock on every object before a 'write' operation is performed. Transactions may unlock the data item after completing the 'write' operation.

ii)**Pre-claiming Lock Protocol**:Pre-claiming protocols evaluate their operations and create a list of data items on which they need locks. Before initiating an execution, the transaction requests the system for all the locks it needs beforehand. If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.

iii)**Two-Phase Locking – 2PL**: This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.

Eg:lock X(B);

```
    Read B;  
    B: = B – 100;  
    Write B;
```

lock X(A);

```
    Read A;  
    A: = A + 100;  
    Write A;
```

unlock(B);

unlock(A);

Two-phase locking has two phases: one is **growing**, where all the locks are being acquired by the transaction; and the second phase is **shrinking**, where the locks held by the transaction are being released. To claim an exclusive (write) lock, a transaction must first acquire a shared (read) lock and then upgrade it to an exclusive lock.

iv)**Strict Two-Phase Locking**:The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to

execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time. Strict-2PL does not have cascading abort as 2PL does.

5) TIMESTAMP BASED PROTOCOL :

The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp. Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one. In addition, every data item is given the latest read and write-timestamp. This lets the system know when the last 'read and write' operation was performed on the data item.

Timestamp Ordering Protocol :

The timestamp-ordering protocol ensures serializability among transactions in their conflicting read and write operations. This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.

- The timestamp of transaction T_i is denoted as $TS(T_i)$.
- Read timestamp of data-item X is denoted by $R\text{-timestamp}(X)$.
- Write timestamp of data-item X is denoted by $W\text{-timestamp}(X)$.

Timestamp ordering protocol works as follows:

i) If a transaction T_i issues a $\text{read}(X)$ operation:

- If $TS(T_i) < W\text{-timestamp}(X)$ Operation rejected.
- If $TS(T_i) \geq W\text{-timestamp}(X)$ Operation executed.

ii) If a transaction T_i issues a $\text{write}(X)$ operation:

- If $TS(T_i) < R\text{-timestamp}(X)$ Operation rejected.
- If $TS(T_i) < W\text{-timestamp}(X)$ Operation rejected and T_i rolled back.

Thomas' Write Rule: This rule states if $TS(T_i) < W\text{-timestamp}(X)$, then the operation is rejected and T_i is rolled back. Timestamp ordering rules can be modified to make the schedule view serializable. Instead of making T_i rolled back, the 'write' operation itself is ignored.

6) VALIDATION BASED PROTOCOL:

Execution of transaction T_i is done in three phases.

1. **Read and execution phase:** Transaction T_i writes only to temporary local variables
2. **Validation phase:** Transaction T_i performs a ``validation test'' to determine if local variables can be written without violating serializability.
3. **Write phase:** If T_i is validated, the updates are applied to the database; otherwise, T_i is rolled back.

Also called as **optimistic concurrency control** since transaction executes fully in the hope that all will go well during validation.

Each transaction T_i has 3 timestamps

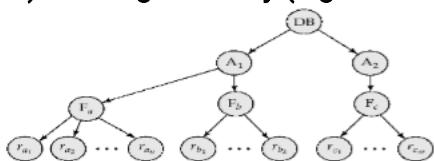
- $\text{Start}(T_i)$: the time when T_i started its execution
- $\text{Validation}(T_i)$: the time when T_i entered its validation phase
- $\text{Finish}(T_i)$: the time when T_i finished its write phase

4) MULTIPLE GRANULARITY:

Allow data items to be of various sizes and define a hierarchy of data granularities, where the small granularities are nested within larger ones. Can be represented graphically as a tree (but don't confuse with tree-locking protocol). When a transaction locks a node in the tree *explicitly*, it *implicitly* locks all the node's descendants in the same mode.

Granularity of locking (level in tree where locking is done):

- i)**fine granularity (lower in tree)**: high concurrency, high locking overhead.
- ii)**coarse granularity (higher in tree)**: low locking overhead, low concurrency



The levels, starting from the coarsest (top) level are *database, area, file and record*.

- 3) Define Deadlock. Explain how Concurrency control is done with Time Stamp Ordering.

DEADLOCKS:

In a multi-process system, deadlock is an unwanted situation that arises in a shared resource environment, where a process indefinitely waits for a resource that is held by another process.

For example, assume a set of transactions $\{T_0, T_1, T_2, \dots, T_n\}$. T_0 needs a resource X to complete its task. Resource X is held by T_1 , and T_1 is waiting for a resource Y, which is held by T_2 . T_2 is waiting for resource Z, which is held by T_0 . Thus, all the processes wait for each other to release resources. In this situation, none of the processes can finish their task. This situation is known as a deadlock. Deadlocks are not healthy for a system. In case a system is stuck in a deadlock, the transactions involved in the deadlock are either rolled back or restarted.

CONCURRENCY CONTROL WITH TIME STAMP ORDERING:

Deadlock Prevention:

To prevent any deadlock situation in the system, the DBMS aggressively inspects all the operations, where transactions are about to execute. The DBMS inspects the operations and analyzes if they can create a deadlock situation. If it finds that a deadlock situation might occur, then that transaction is never allowed to be executed.

There are deadlock prevention schemes that use timestamp ordering mechanism of transactions in order to predetermine a deadlock situation.

i)**Wait-Die Scheme:** In this scheme, if a transaction requests to lock a resource (data item), which is already held with a conflicting lock by another transaction, then one of the two possibilities may occur:

- If $TS(T_i) < TS(T_j)$ – that is T_i , which is requesting a conflicting lock, is older than T_j – then T_i is allowed to wait until the data-item is available.
- If $TS(T_i) > TS(T_j)$ – that is T_i is younger than T_j – then T_i dies. T_i is restarted later with a random delay but with the same timestamp. This scheme allows the older transaction to wait but kills the younger one.

ii)**Wound-Wait Scheme:** In this scheme, if a transaction requests to lock a resource (data item), which is already held with conflicting lock by another transaction, one of the two possibilities may occur:

- If $TS(T_i) < TS(T_j)$, then T_i forces T_j to be rolled back – that is T_i wounds T_j . T_j is restarted later with a random delay but with the same timestamp.
- If $TS(T_i) > TS(T_j)$, then T_i is forced to wait until the resource is available. This scheme allows the younger transaction to wait; but when an older transaction requests an item held by a younger one, the older transaction forces the younger one to abort and release the item. In both the cases, the transaction that enters the system at a later stage is aborted.

- 4) Discuss about database recovery management and transaction recovery.

Ans: DATABASE RECOVERY MANAGEMENT: TRANSACTION RECOVERY :

FAILURE CLASSIFICATION:

1) Transaction failure :

- Logical errors: transaction cannot complete due to some internal error condition.
- System errors: the database system must terminate an active transaction due to an error condition (e.g., deadlock)

2) System crash: a power failure or other hardware or software failure causes the system to crash.

Fail-stop assumption: non-volatile storage contents are assumed to not be corrupted by system crash. Database systems have numerous integrity checks to prevent corruption of disk data

3) Disk failure: a head crash or similar disk failure destroys all or part of disk storage

Destruction is assumed to be detectable: disk drives use checksums to detect failures

To ensure atomicity despite failures, we first output information describing the modifications to stable storage without modifying the database itself. We study two approaches:

1) log-based recovery 2) shadow-paging

4) LOG BASED RECOVERY :

We assume (initially) that transactions run serially, that is, one after the other. A log is kept on stable storage. The log is a sequence of log records, and maintains a record of update activities on the database. Each and every log record is given a unique id called log sequence number.

A log record is written for each of the following actions:

- Updating a page
- Commit
- Abort
- End
- Undoing an update

Two approaches using logs :

i) Deferred database modification

ii) Immediate database modification

- The **deferred database modification** scheme records all modifications to the log, but defers all the writes to after partial commit.
Assume that transactions execute serially. Transaction starts by writing $\langle T, start \rangle$ record to log.
- v) A write(X) operation results in a log record $\langle T, X, V \rangle$ being written, where V is the new value for X
 - I Note: old value is not needed for this scheme
- vi) The write is not performed on X at this time, but is deferred.
- vii) When T partially commits, $\langle T, commit \rangle$ is written to the log
- viii) Finally, the log records are read and used to actually execute the previously deferred writes.
- The **immediate database modification** scheme allows database updates of an uncommitted transaction to be made as the writes are issued. Since undoing may be needed, update logs must have both old value and new value.

i) Update log record must be written *before* database item is written

- a. We assume that the log record is output directly to stable storage
- b. Can be extended to postpone log record output, so long as prior to execution of an output(B) operation for a data block B , all log records corresponding to items B must be flushed to stable storage
- iv) Output of updated blocks can take place at any time before or after transaction commit
- v) Order in which blocks are output can be different from the order in which they are written.

5) SHADOW PAGING :

Shadow paging is an alternative to log-based recovery; this scheme is useful if transactions execute serially

Idea: maintain *two* page tables during the lifetime of a transaction – the current page table, and the shadow page table. Store the shadow page table in nonvolatile storage, such that state of the database prior to transaction execution may be recovered. Shadow page table is never modified during execution

To start with, both the page tables are identical. Only current page table is used for data item accesses during execution of the transaction. Whenever any page is about to be written for the first time

- A copy of this page is made onto an unused page.
- The current page table is then made to point to the copy
- The update is performed on the copy

Advantages of shadow-paging over log-based schemes:

- o no overhead of writing log records
- o recovery is trivial

CHECKPOINT:

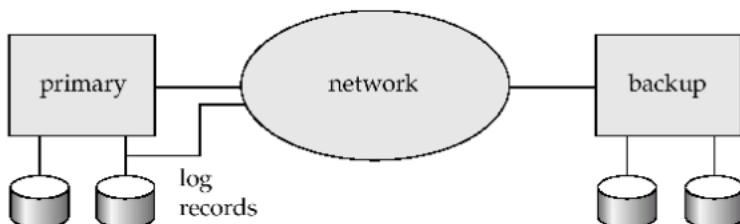
Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system. As time passes, the log file may grow too big to be handled at all. Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

When a system with concurrent transactions crashes and recovers, it behaves in the following manner:

- The recovery system reads the logs backwards from the end to the last checkpoint.
- It maintains two lists, an undo-list and a redo-list.
- If the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>, it puts the transaction in the redo-list.
- If the recovery system sees a log with <Tn, Start> but no commit or abort log found, it puts the transaction in the undo-list. All the transactions in the undo-list are then undone and their logs are removed. All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.

REMOTE BACKUP:

Remote backup provides a sense of security in case the primary location where the database is located gets destroyed. Remote backup can be offline or realtime or online. In case it is offline, it is maintained manually.



Online backup systems are more real-time and lifesavers for database administrators and investors. An online backup system is a mechanism where every bit of the real-time data is backed up simultaneously at two distant places. One of them is directly connected to the system and the other one is kept at a remote place as backup.

As soon as the primary database storage fails, the backup system senses the failure and switches the user system to the remote storage. Sometimes this is so instant that the users can't even realize a failure.

ARIES:

- ARIES is a state of the art recovery method
 - i)Incorporates numerous optimizations to reduce overheads during normal processing and to speed up recovery
 - ii)Unlike the advanced recovery algorithm, ARIES
 - Uses log sequence number (LSN) to identify log records
 - Stores LSNs in pages to identify what updates have already been applied to a database page.
 - Physiological redo.
 - Dirty page table to avoid unnecessary redos during recovery

ARIES RECOVERY ALGORITHM:

ARIES recovery involves three passes

1)Analysis pass: Determines

- Which transactions to undo
- Which pages were dirty (disk version not up to date) at time of crash
- RedoLSN: LSN from which redo should start

2)Redo pass:

- Repeats history, redoing all actions from RedoLSN
- RecLSN and PageLSNs are used to avoid redoing actions already reflected on page

3)Undo pass:

- Rolls back all incomplete transactions.
- Transactions whose abort was complete earlier are not undone
- Key idea: no need to undo these transactions: earlier undo actions were logged, and are redone as required.

4.5.7 Assignments: 1

1. Enumerate the properties of transaction.
2. Discuss about database recovery management and transaction recovery.
3. Emphasize on concurrency control with locking methods.

4.5.8 Previous Questions (Asked by JNTUK from the concerned Unit)

1. a)What is a serializable schedule? What is a recoverable schedule?

What is a schedule that avoids cascading aborts? What is a strict schedule?

b) Discuss in detail about the phases the recovery manager proceeds when the system is restarted after a crash.

2. a) Is every conflict serializable schedule is serializable? Explain.
b) Explain different types of failures that arise due to loss of non-volatile storage.
3. a) What overheads are associated with lock-based concurrency control?
Discuss blocking and aborting overheads specifically?
Define these terms: atomicity, consistency, isolation, durability, schedule, blind write, dirty read, unrepeatable read, serializable schedule, recoverable schedule, avoids-cascading- aborts schedule.
4. a) Explain various types of lock based concurrency control with a neat sketch and examples.
b) Describe in detail about shadow paging recovery technique. Under what circumstances does it not require a log?

4.5.9 GATE Questions (Where relevant) – No Relevant Questions

1. Consider the following transactions with data items P and Q initialized to zero:

```
T1: read (P) ;  
    read (Q) ;  
    if P = 0 then Q := Q + 1 ;  
    write (Q) ;  
  
T2: read (Q) ;  
    read (P) ;  
    if Q = 0 then P := P + 1 ;  
    write (P) ;
```

Any non-serial interleaving of T1 and T2 for concurrent execution leads to

- a) A serializable schedule
 - b) A schedule that is not conflict serializable
 - c) A conflict serializable schedule
 - d) A schedule for which a precedence graph cannot be drawn
2. Which of the following concurrency control protocols ensure both conflict serializability and freedom from deadlock? I. 2-phase locking II. Time-stamp ordering

- A) I only
- B) II only
- C) Both I and II
- D) neither I nor II

3. Consider the transactions T1, T2, and T3 and the schedules S1 and S2 given below.

```
T1: r1(X); r1(Z); w1(X); w1(Z)  
T2: r2(Y); r2(Z); w2(Z)  
T3: r3(Y); r3(X); w3(Y)  
S1: r1(X); r3(Y); r3(X); r2(Y); r2(Z);  
    w3(Y); w2(Z); r1(Z); w1(X); w1(Z)  
S2: r1(X); r3(Y); r2(Y); r3(X); r1(Z);  
    r2(Z); w3(Y); w1(X); w2(Z); w1(Z)
```

Which one of the following statements about the schedules is TRUE?

- A) Only S1 is conflict-serializable.
- B) Only S2 is conflict-serializable.
- C) Both S1 and S2 are conflict-serializable.
- D) Both S1 and S2 are conflict-serializable.

4. Consider the following log sequence of two transactions on a bank account, with initial balance 12000, that transfer 2000 to a mortgage payment and then apply a 5% interest.

1. T1 start
2. T1 B old=12000 new=10000
3. T1 M old=0 new=2000
4. T1 commit
5. T2 start
6. T2 B old=10000 new=10500
7. T2 commit

Suppose the database system crashes just before log record 7 is written. When the system is restarted, which one statement is true of the recovery procedure?

- A) We must redo log record 6 to set B to 10500
- B) We must undo log record 6 to set B to 10000 and then redo log records 2 and 3
- C) We need not redo log records 2 and 3 because transaction T1 has committed

D) We can apply redo and undo operations in arbitrary order because they are idempotent

5. Consider the following transaction involving two bank accounts x and y.

```
read(x); x := x - 50; write(x); read(y); y := y + 50; write(y)
```

The constraint that the sum of the accounts x and y should remain constant is that of

- A) Atomicity
- B) Consistency
- C) Isolation
- D) Durability

6. Which one of the following is NOT a part of the ACID properties of database transactions?

- A) Atomicity
- B) Consistency
- C) Isolation
- D) Deadlock-freedom

7. Consider the following two phase locking protocol. Suppose a transaction T accesses (for read or write operations), a certain set of objects {O₁, ..., O_k}. This is done in the following manner: **Step 1**. T acquires exclusive locks to O₁, ..., O_k in increasing order of their addresses. **Step 2**. The required operations are performed. **Step 3**. All locks are released. This protocol will

- A) guarantee serializability and deadlock-freedom
- B) guarantee neither serializability nor deadlock-freedom
- C) guarantee serializability but not deadlock-freedom
- D) guarantee deadlock-freedom but not serializability

8. Suppose a database schedule S involves transactions T₁, ..., T_n. Construct the precedence graph of S with vertices representing the transactions and edges representing the conflicts. If S is serializable, which one of the following orderings of the vertices of the precedence graph is guaranteed to yield a serial schedule?

- A) Topological order
- B) Topological order
- C) Breadth-first order
- D) Ascending order of transaction indices

9. Consider the following three schedules of transactions T1, T2 and T3.
[Notation: In the following NYO represents the action Y (R for read, W for write) performed by transaction N on object O.]

(S1) 2RA 2WA 3RC 2WB 3WA 3WC 1RA 1RB 1WA 1WB

(S2) 3RC 2RA 2WA 2WB 3WA 1RA 1RB 1WA 1WB 3WC

(S3) 2RZ 3RC 3WA 2WA 2WB 3WC 1RA 1RB 1WA 1WB

Which of the following statements is TRUE?

- A) S1, S2 and S3 are all conflict equivalent to each other
- B) No two of S1, S2 and S3 are conflict equivalent to each other
- C) S2 is conflict equivalent to S3, but not to S1
- D) S1 is conflict equivalent to S2, but not to S3

4.5.10 Interview questions (which are frequently asked in a Technical round - Placements)

1. What is Transaction? Give an example in DBMS.
2. Explain Atomicity with real-time example.
3. Name the different states of transaction. How are they related to each other?
4. Explain the significance of Rollback and Savepoint.
5. How do you ensure atomicity despite failures?
6. What are the methods used for deadlock prevention?

7. Spotlight the significance of Checkpoint.
8. COMMIT takes more time than ROLLBACK .

4.5.11 Real-Word (Live) Examples / Case studies wherever applicable

Not Required in this chapter

4.5.12 Suggested "Expert Guest Lectures"

Ms.P.P.Priyanka (Asst Prof)Department of CSE, entitled " Locking Mechanisms"

4.5.13 Literature references of Relevant NPTEL Videos/Web/YouTube videos etc.

http://www.tutorialspoint.com/dbms/dbms_transaction.htm

<http://nptel.ac.in/video.php?subjectId=106106093>

4.5.14 Any Lab requirements; if so link it to Lab Lesson Plan.

Not Required in this chapter

4.5.15 Reference Text Books / with Journals Chapters etc.

TRANSACTION MANAGEMENT TECHNIQUES AND PRACTICES IN CURRENT CLOUD COMPUTING ENVIRONMENTS : A SURVEY

Ahmad Waqas^{1,2}, Abdul Waheed Mahessar¹, Nadeem Mahmood^{1,3},
Zeeshan

Bhatti¹, Mostafa Karbasi¹, Asadullah Shah¹

¹Department of Computer Science, Kulliyah of Information and Communication
Technology, International Islamic University Malaysia
²Department of Computer Science, Sukkur Institute of Business Administration, Pakistan
³Department of Computer Science, University of Karachi, Pakistan

ABSTRACT

Distributed database management systems (DDBMs) have limited capacity to manage overhead of distributed transactions due to certain failures. These types of failures don't guarantee the enforcement of transactional properties and which reduces the scalability and availability of these systems. Research in this area proved that scaling out while providing consistency of transactions and guaranteeing availability in the presence of different failures in distributed computing environment is not feasible. As a result, the database community have used vertical scaling rather than horizontal scaling. Moreover changes in data access patterns resulting from a new generation of web applications with high scalability and availability guarantees weaker consistency. In this paper we have analyse that how different systems such as ElasTras, Mstore, Sinfonia, ecStore and Gstore provide scalable transactional data access in cloud computing environment. We have also discussed transaction management (TM) in Gstore in detail and compared with other techniques and protocols used in current cloud based systems.

KEYWORDS

Distributed Database Management Systems, Transaction Management, Cloud Computing, Transactional Data Access, Data Access Patterns

1. INTRODUCTION

Transaction is a set of query instructions to be executed atomically on a single consistent view of a database [1]. The main challenge is to provide complete transaction management (TM) support in a cloud computing environment [2]. This is achieved by implementing ACID (Atomicity, Consistency, Isolation and Durability) [1] properties without compromising the scalability properties of the cloud. However, the underlying data storage services provide mostly eventual consistency. The objective of this study is to analyse, investigate and review the key concepts and techniques with respect to TM. Research efforts in historical perspective reveal specific trends and developments in the area of TM [3][4]. Therefore, we have surveyed important research literature of the current cloud based systems and TM in DDBMSs. Our work provides a

comprehensive and structured overview of TM techniques in cloud computing environment [5][6]. We have discussed following products in this paper:

- i. G-Store: [7] A scalable data store for transactional multi key access in the cloud
- ii. Megastore: [8] Providing Scalable, highly available storage for interactive services
- iii. Scalable Transactions for web applications in the cloud [9]
- iv. ElasTranS: [10] An elastic transactional data store in the cloud
- v. ecStore: [11] Towards elastic transactional cloud storage with range query support

- vi. Sinfonia: [12] a new paradigm for building scalable distributed systems
- vii. PNUTS: [13] Yahoo's hosted data serving platform
- viii. Dynamo: [14] Amazon's highly available key-value store
- ix. Bigtable: [15] A distributed storage system for structured data

Researchers have realized that supporting distributed transactions does not allow scalable and available designs [4]. Therefore to satisfy the scalability requirements web applications, designers have scarified the ability to support distributed transactions. This resulted in the design of simpler data stores based on the key-value schema, where tables are viewed as a huge collection of key-value entries. Key-value stores such as Bigtable, PNUTS, Dynamo, ecStore and their open source analogous, have been the preferred data stores for applications in the cloud. The property common to all systems is the key-value abstraction where data is viewed as key-value pairs and atomic access is supported only at the granularity of single keys. These systems limit access granularity to single key accesses, while providing minimal consistency and atomicity guarantees on multi-key accesses. While this property works well for current applications, it is insufficient for the next generation web applications which emphasize collaboration. Since collaboration by definition requires consistent access to groups of keys, scalable and consistent multi key access is critical for such applications [9].

Further the concept of key-value stores and accesses at the granularity of single keys was put forward as the sole means to attain high scalability and availability in such systems. Based on these principles, a number of key value stores also called row stores like Bigtable, PNUTS, Dynamo, ecStore and Hbase were designed and successfully implemented. Single key atomic access semantics naturally allows efficient horizontal data partitioning, and provide the basis for scalability and availability in these systems. However, all these key value stores although highly scalable, stop short of providing transactional guarantees even on a single row. Therefore to satisfy the scalability requirements of web applications, designers have scarified the ability to support distributed transactions.

As we know relational database technologies have been extremely successful in the traditional enterprise setting but they have limitations in providing additional key features and multi-key value support in cloud data management [16]. Also deployment of modern application ideas which were not technically and economically feasible in the traditional enterprise setting, have become possible due to the cloud computing paradigm.

In literature, there are important contributions in the area of database integration in cloud environment. Das [4] presented a good analysis and survey of scalable and elastic transactional data in Cloud. Sakr [17] contributed a very good and detail survey of large scale data management techniques and approaches in current cloud system. Abadi [18] highlighted important issues and challenges in managing big data in cloud computing platform. Other important publications in this regard are [16][19][9][20].

The paper is divided into six sections, after introduction in section 2 we have discussed the important concepts related to transaction management followed by section 3 which describes the cloud computing environment. Section 4 elaborates the distributed database applications followed by section 5 which describes the transaction management techniques used in distributed systems. Section 6 presents the comparison of different transaction management techniques used in distributed and cloud computing environment followed by conclusion.

2. OVERVIEW OF TRANSACTION MANAGEMENT IN TRADITIONAL DISTRIBUTED DBMS

2.1. Transaction Management

Transaction Management deals with the problems of always keeping the database in consistent state even when concurrent accesses and failures occur (Figure 1). A transaction is a collection of

4
2

actions that make consistent transformations of system states while preserving system consistency [1]. In contrast a distributed transaction is a transaction that updates data on two or more networked computer systems. Distributed transactions extend the benefits of transactions to applications that must update distributed data. The consistency and reliability aspects of transactions are due to four following properties called ACID (Atomicity, Consistency, Isolation, and Durability) [1]. Atomicity ensures complete transaction execution, consistency in transaction correctness, isolation ensures transaction independence in parallel executions and durability refers to transaction persistence in database. Therefore to ensure integrity of the data, we require that the database system maintain the ACID properties of the transactions.

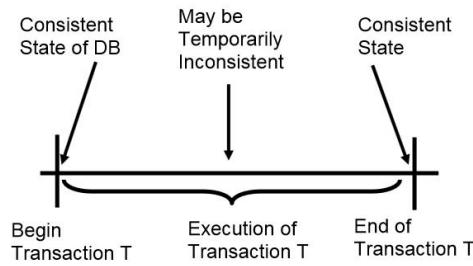


Figure 1: A transactional model

2.2. Types of Transactions

Transactions have been classified in different ways. One criterion is the duration of transactions other is the type of transaction. Gray [21] classified the transactions as online (short-life) and batch (long-life). Online transactions have a very short execution/response times and relatively affects the small portion of database. This class of transactions probably covers a large majority of current transaction applications. Banking transactions and airline reservation transactions are very good examples of online transactions. In contrast, batch transactions take longer time to execute and have access a large portion of the database. Statistical applications, report generations and image processing are the examples of batch transactions.

Transactions are also classified in terms of transaction structure [1]. Flat transaction consists of a sequence of primitives embraced between a "begin" and "end" markers and nested transactions have transactions within the main transaction. The operations of a transaction may themselves be transactions. Transactions are also categorized in terms of its read and write actions. If the transactions are restricted so that all the read actions are performed before any write actions, the transaction is called a two-step transaction. Similarly, if the transaction is restricted so that a data item has to be read before it can be updated (written), the transaction is called restricted (or read-before-write). If a transaction is both two-step and restricted it is called restricted two-step transaction.

2.3. Transaction Failures

There are different types of transaction failures. Failure can be due to an error in the transaction caused by incorrect input data as well as the detection of a present or potential deadlock [21]. Furthermore, some concurrency control algorithms do not permit a transaction to proceed or even to wait if the data that they attempt to access are currently being accessed by another transaction. The usual approach to take in cases of transaction failure is to abort the transaction, thus resetting the database to its state prior to the start of this transaction.

2.4. Database Log

Each update transaction not only changes the database but the change is also recorded in the database log. The log contains information necessary to recover the database state following a failure.

2.5. Write-Ahead Logging (WAL) Protocol

WAL protocol [22] is considered to be an important protocol for maintaining logs, whether the log is written synchronously or asynchronously. Consider a case where the updates to the database are written into the stable storage before the log is modified in stable storage to reflect the update. If a failure occurs before the log is written, the database will remain in updated form, but the log will not indicate the update that makes it impossible to recover the database to a consistent and up-to-date state. Therefore, the stable log is always updated prior to the stable database update.

We can say precisely as

- 1) Before a stable database is updated (perhaps due to actions of a yet uncommitted transaction), the before images should be stored in the stable log. This facilitates "undo".
- 2) When a transaction commits, the after images have to be stored in the stable log prior to the updating of the stable database. This facilitates "redo".

2.6. Check-point

In most of the local recovery manager (LRM) implementation strategies, the execution of the recovery action requires searching the entire log [21]. This is a significant overhead because the LRM is trying to find all the transactions that need to be undone and redone. The overhead can be reduced if it is possible to build a wall which signifies that the database at that point is up-to-date and consistent. In that case, the redo has to start from that point on and the undo only has to go back to that point. This process of building the wall is called check pointing. It is achieved in three steps:

- 1) Write a begin_checkpoint record into the log.
- 2) Collect the checkpoint data into the stable storage.
- 3) Write an end_checkpoint record into the log.

2.7. Distributed Reliability Protocols

To understand distributed reliability protocols (DRP) [1] we assume that at the originating site of a transaction there is a coordinator process and at each site where the transaction executes there are participant processes. Thus, the DRP are implemented between the coordinator and the participants. The reliability techniques in traditional DDBMS consist of commit, termination and recovery protocols. The primary requirement of commit protocols is that they maintain the atomicity of distributed transactions. This means that even though the execution of the distributed transaction involves multiple sites, some of which might fail while executing, the effects of the transaction on the distributed database is either all or nothing. This is called atomic commitment.

Also the termination protocols preferably are non-blocking which means, it permits a transaction to terminate at the operational sites without waiting for recovery of the failed site. That would significantly improve the response time performance of transactions. Moreover the DRP must be independent i.e. they have the ability to terminate a transaction that was executing at the time of a failure without having to consult any other site.

2.8. Two-Phase Commit

Two-phase commit (2PC) [21] is a very simple and elegant protocol that ensures the atomic commitment of distributed transactions. Two rules govern the decision of commit and abort, which together are called the global commit rule.

- 1) If even one participant votes to abort the transaction, the coordinator has to reach a global abort decision.
- 2) If all the participants vote to commit the transaction, the coordinator has to reach a global commit decision.

2PC permits a participant to unilateral abort a transaction until it has decided to register an affirmative vote. Once participant votes to commit or abort a transaction, it cannot change its vote. While a participant is in the ready state, it can move either to abort the transaction or to commit it, depending on the nature of the message from the coordinator. The global termination decision is taken by the coordinator according to the global commit rule. The coordinator and participant processes enter certain states where they have to wait for messages from one another. To guarantee that they can exit from these states and terminate, timers are used.

2.9. Distributed 2PL Protocol

Distributed 2PL [21] requires the availability of lock managers at each site. The distributed 2PL TM protocol is similar to the C2PL-TM [21] with two major modifications. The messages that are sent to the central site lock manager in C2PL-TM are sent to the lock managers at all participating sites in D2PL-TM. The second difference is that the operations are not passed to the data processors by the coordinating transaction manager, but by the participating lock managers. This means that the coordinating transaction manager does not wait for a lock request granted message. Also the participating data processors send the end of operation messages to the coordinating TM or each data processor can send it to its own lock manager who can then release the locks and inform the coordinating TM (Figure 2).

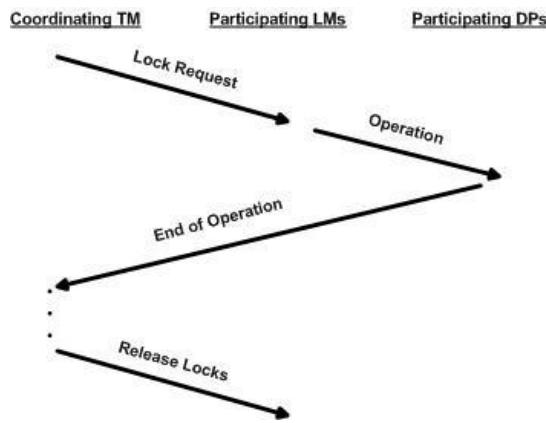


Figure 2: Communication Structure of D2PL

2.10. Optimistic Concurrency Control (OCC) Protocol

In relational databases the, OCC protocol [23] is a type of concurrency control method which assumes that multiple transactions can complete without affecting each other. Therefore transactions can proceed without locking the data resources that they affect. Before committing, each transaction verifies that no other transaction has modified its data. If the check reveals conflicting modifications, the committing transaction rolls

back.

OCC is generally used in environments with low data contention. When conflicts are rare, transactions can complete without the expense of managing locks and without having transactions wait for other transactions' locks to clear, leading to higher throughput than other concurrency control methods. However, if conflicts happen often, the cost of repeatedly restarting transactions hurts performance significantly; other concurrency control methods have better performance under these conditions.

More specifically, OCC transactions involve these phases:

- Begin: Record a timestamp marking the transaction's beginning.
- Modify: Read and write database values.
- Validate: To check whether other transactions have modified data that this transaction has used (read or write).
- Commit/Rollback: If there is no conflict, make all changes to the actual state of the database if there is no conflict else resolve (abort transaction).

2.11. Mini-transactions

Mini transactions allow the applications to atomically access and conditionally modify data at multiple memory nodes. Mini transactions are also useful for improving performance, in many ways. First, mini-transactions allow users to batch together updates, which eliminate multiple network round-trips. Second, because of their limited scope, mini-transactions can be executed within the commit protocol. For example Sinfonia can start, execute, and commit a mini-transaction with two network round-trips. In contrast, database transactions, being more powerful and higher level, require two round-trips just to commit and additional round-trips to start and execute. Third, mini-transactions can execute in parallel with a replication scheme, providing availability with a replication scheme, providing availability with little extra latency.

For example, consider a cluster file system, one of the applications built with Sinfonia. With a mini-transaction, a host can atomically populate an inode stored in one memory node, and link this inode to a directory entry stored in another memory node, and these updates can be conditional on the inode being free (to avoid races). Like database transactions, mini-transactions hide the complexities that arise from concurrent execution and failures. We can say mini-transaction allow an application to update data in multiple memory nodes while ensuring ACID properties.

3. CLOUD COMPUTING AND DATA MANAGEMENT

Cloud computing eases the provision of on-demand computing resources by means of internet with surety of scalability, reliability and availability [24][31]. In accordance to National Institute of Standards and Technology (NIST), "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction" [NIST]. The cloud delivers resources to clients exploiting three fundamental services models that are Infrastructure-as-a-Service, Platform-as-a-Service, and Software-as-a-Services [25][26]. These service models are deployed as Public, Private, Community and Hybrid Cloud.

Cloud computing is a good business for organizations having large data centres to rent their resources. It provides efficient solutions for handling and analyzing big data. Cloud based databases predictably run and managed in a cloud computing environment that are deployed either as virtual machine images or Database-as-a-Service which follows SQL or NoSQL data models (Figure 3). The most common cloud based storage systems include

OracleDB, IBM DB2, Ingres, NuoDB, BitCan, Hadoop, MongoDB, PNUTS, ecStore, Bigtable, GStore, MStore and

4
6

DynamoDB. Cloud based database management system ideally ensures few desired properties for efficient data analysis. These properties include efficiency, fault tolerance, ability to run in a heterogeneous environment, ability to operate on encrypted data and ability to interface with business intelligence products [18].

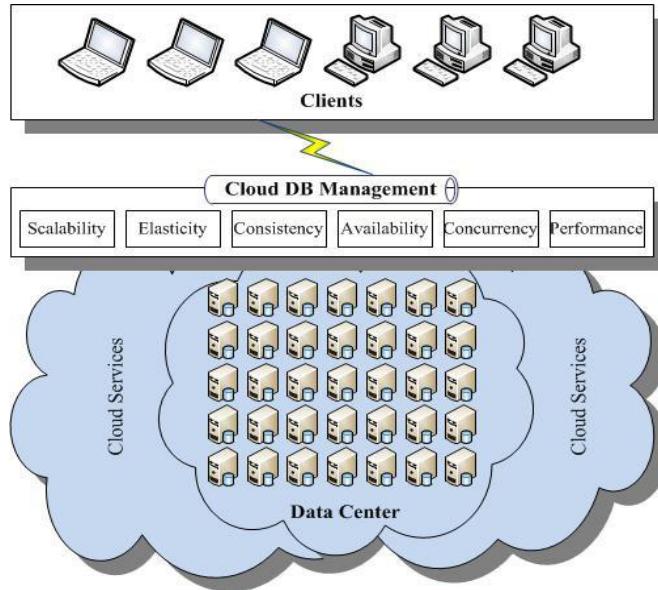


Figure 3: Communication Structure of D2PL

4. INTRODUCTION OF DISTRIBUTED STORAGE SYSTEMS

4.1. Bigtable: A distributed storage system for structured data

Bigtable is a distributed storage system for managing structured data that is designed to scale of a very large size (petabytes) of data across thousands of commodity servers. Many projects at Google store data in Big table, including Google Earth, web indexing, Orkut, and Google Finance. In many ways, Bigtable resembles a database, it shares many implementation strategies with databases and parallel databases and achieved scalability and high performance but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model instead it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Also clients can control the locality of their data through careful choices in their schemas.

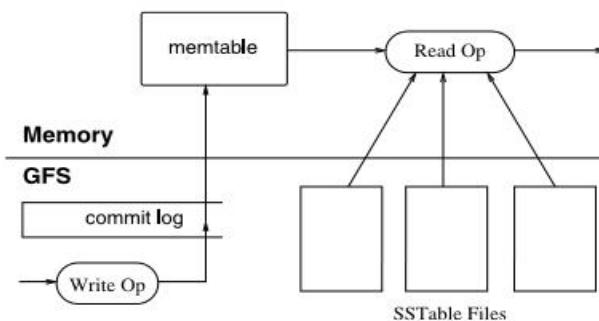


Figure 4: Bigtable Tablet

Figure 4 represents the Bigtable tablet. Big table schema parameters let clients dynamically control whether to serve data out of memory or from disk. Bigtable can be used with MapReduce [27], a frame work for running large-scale parallel computations developed at Google.

4.2. PNUTS: Yahoo!'s Hosted Data Serving Platform

PNUTS [13] is a parallel and geographically distributed database system for Yahoo's web applications. It provides data storage organized as hashed or ordered tables, low latency for large number of concurrent requests including updates and queries, and guarantees novel per-record consistency. It is a hosted, centrally managed and geographically distributed service, and utilizes automated load-balancing and fails over to reduce operational complexity.

Traditional database systems provided us a model for reasoning about consistency in the presence of concurrent operations, called serializable transactions. Generally supporting general serializable transactions over a globally replicated and distributed system is very expensive. Further, serializability of general transactions is inefficient and often unnecessary, many distributed replicated systems go to the extreme of providing only eventual consistency. In this a client can update any replica of an object and all updates to an object will eventually be applied, but potentially in different orders at different replicas.

PNUTS exposes a simple relational model to users, and supports single-table scans with predicates. Additional features include scatter-gather operations, a facility for asynchronous notification of clients and a facility for bulk loading. To meet response-times goals, PNUTS cannot use write-all replication protocols that are employed by systems deployed in localized clusters like bigtable. However, not every read of the data necessarily needs to see the most current version. Moreover it employs redundancy at multiple levels and leverages consistency model to support high-available reads and writes even after a failure or partition. PNUTS provide architecture based on record-level, asynchronous geographic replication and uses guaranteed message delivery service instead of a persistent log. PNUTS provides a consistency model that offers applications transactional features with partial serializability.

4.3. ElasTras: An Elastic Transactional Data Store in the Cloud

ElasTras [10] is an elastic transactional data store, which was designed with the goal of scalable and elastic TM in the cloud computing environment, while providing transactional guarantees. ElasTras is analogous to partitioned databases which are common in enterprise systems, while adding features and components critical towards elasticity of the data store (Figure 5).

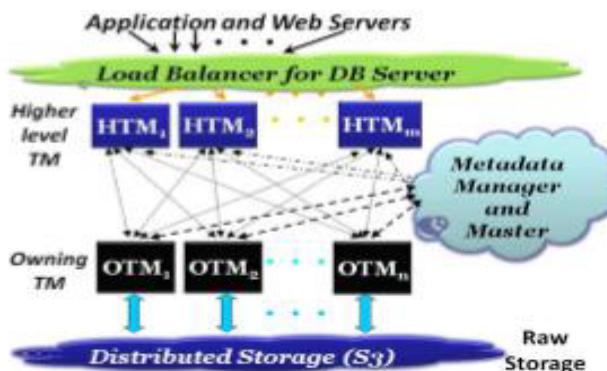


Figure 5: Overview of the ElasTraS system

It has been designed with the goal of scalable and elastic TM in the cloud. It uses a key-value based design similar to Bigtable and designed with the intent to provide transactional guarantees in a scalable manner, rather than retrofitting these features into an existing system. For providing transactional guarantees it uses two level hierarchy of TM, which are responsible for providing transactional guarantees, while also providing elastic scalability with increase in demand. ElasTras implements proven database techniques for dealing with concurrency control, isolation, and recovery, while using design principles of scalable systems (Bigtable) to overcome the limitations of DDBMS.

4.4. G-Store: A scalable Data Store for Transactional Multi key Access in the Cloud

G-Store [7] is a scalable data store providing transactional multi key access which guarantees dynamic, non-overlapping groups of keys using a key-value store as an underlying substrate. It inherits the important features of scalability, high availability and fault-tolerance. The basic innovation that allows scalable multi key access is the key group abstraction which defines a granule of on-demand transactional access. The key grouping protocol uses the key group abstraction to transfer exclusive read/write access for all keys in a group to a single node which then efficiently executes the operations on the key group. The key group abstraction allows applications to select members of group from any set of keys in the data store and dynamically create and dissolve groups, while allowing the data store to provide efficient, scalable and transactional access to these groups of keys (Figure 6). At any instant of time, a given key can participate in a single group, but during its life time, a key can be a member of multiple groups. In comparison M-Store although, supports key grouping but once a key is created as part of a group, it has to be in the group for the rest of its life time. M-Store uses the entity group abstraction for providing transactional guarantees over the collocated keys which form a contiguous part of the key range. But the static natures of entity groups as well as the contiguity requirement are often insufficient.

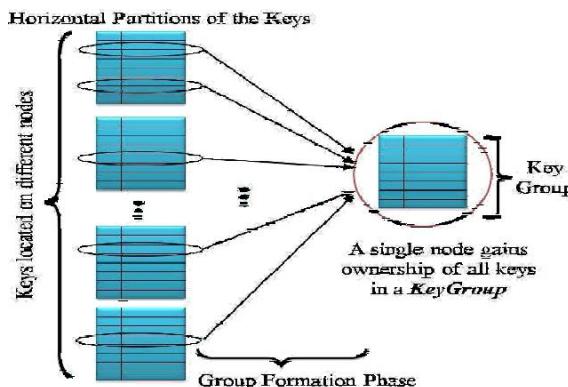


Figure 6: The key group abstraction in G-store [7]

There are no ordering restrictions of keys in key groups therefore the members of a group can be on different nodes. Thus, using a key-value store would require distributed synchronization such as 2PC to provide atomic and consistent accesses to these keys. G-Store inherits the data model as well as the set of operations from the underlying key-value store, the only addition being that the notions of atomicity and consistency are executed from a single key to a group of keys. Moreover, it is targeted to applications whose data has an inherent key-value schema, and requires scalable and transactional access to group of keys which are formed dynamically by the applications.

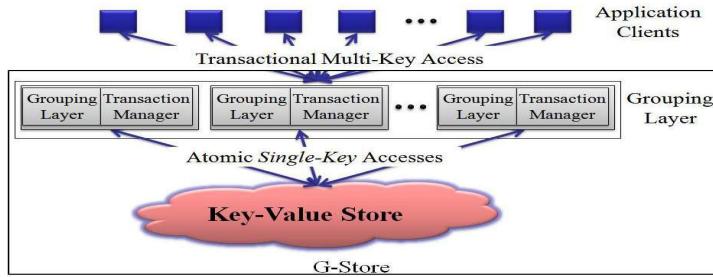


Figure 7: G-Store: Client based Implementation [7]

4.5. Scalable Transactions for Web Applications in the Cloud

It supports transaction in a scalable fashion in cloud computing environment [28]. This approach splits the transaction manager into any number of local transaction managers (LTMs) and partitions the application data and the load of transaction processing across LTMs. Moreover, to maintain the ACID properties even in the case of server failures for transactional system they replicate data items and transaction states to multiple LTMs, and periodically checkpoint consistent data snapshots to the cloud storage service. It uses Bigtable data model, so that transactions are allowed to access any number of data items by primary-keys accessed by the transaction must be given before executing the transaction.

4.6. EcStore: Towards Elastic Transactional Cloud Storage with Range Query support

The ecStore [11] is a highly elastic distributed storage system with efficient range-query and transactional support that can be dynamically deployed in the cloud cluster. It is an elastic cloud storage system that supports automated data partitioning and replication, load balancing, efficient range query, and transactional access. In ecStore, data objects are distributed and replicated in a cluster of commodity computer nodes located in the cloud. Users can access data via transactions which bundle read and write operations on multiple data items. The storage system consists of three main layers: a distributed storage layer, a replication layer, and a TM layer.

4.7. Dynamo: Amazon's Highly Available Key-Value Store

Dynamo [14] is highly available and scalable distributed data store built for Amazon's platform. It is used to manage the state of services that have very high reliability requirements and need tight control over the trade-offs between availability, consistency, cost-effectiveness and performance. It provides a simple primary key only interface to meet the requirements of these applications. In Dynamo data is partitioned and replicated using consistent hashing, and consistency is facilitated by object versioning. The consistency among replicas during updates is maintained by a quorum-like technique and a decentralized replica synchronization protocol. It employs a gossip-based distributed failure detection and membership protocol. Dynamo is a completely decentralized system with minimal need for manual administration. Storage nodes can be added and removed from Dynamo without requiring any manual partitioning or redistribution. Dynamo targets applications that require only key/value access with primary focus on high availability where updates are not rejected even in the wake of network partitions or server failures. Dynamo is targeted mainly at applications that need an always 'writeable' data store where no updates are rejected due to failures or concurrent writes. Also Dynamo is built for an infrastructure within a single administrative domain where all nodes are assumed to be trusted. Moreover, Dynamo doesn't require support for hierarchical namespaces or complex relational schema and

it is built for latency sensitive applications.

5
0

4.8. Megastore: Providing Scalable, High Available Storage for Interactive Services

Megastore (M-store) [8] is a storage system developed to meet the storage requirements of modern interactive online services. Most of the commercial database systems based on relational data model have the inbuilt capacity for designing and building applications. However, they limited capacity to manage large scale applications where billions of users are involved. NoSQL [19] data stores such as Google's Bigtable [15], HBase [29] are highly scalable, but their limited API and loose consistency models complicate application development. Replicating data across distant data centres while providing low latency is challenging and it is novel in that it blends the scalability of a NoSQL data store with the convenience of relational database. It uses synchronous replication to achieve high availability and a consistent view. It provides fully serializable ACID semantics over distant replicas with low enough latencies to support interactive applications. M-store use Paxos [30] to build a highly available system that provides reasonable latencies for interactive applications while synchronously replicating writes across geographically distributed data centres. Further, M-store takes a middle ground approach in RDBMS and NoSQL design space, by partitioning the data store and replicate each partition separately, providing full ACID semantics within partitions. M-store has been widely deployed within Google for several years. It handles more than 3 billion write and 20 billion read transactions daily and stores nearly a petabyte of primary data across many global data centres.

4.9. Sinfonia: a new paradigm for building scalable distributed systems

Sinfonia [12] is a novel mini-transaction primitive that enables efficient and consistent access to data, while hiding the complexities that arise from concurrency and failures. It does not require dealing with message-passing protocols, a major complication in existing distributed systems. Moreover, protocols for handling distributed state include protocols for replication, file data and metadata management, cache consistency, and group membership. These protocols are highly non-trivial to develop. With Sinfonia developers do not have to deal with message-passing protocols. Instead, developers just design and manipulate data structures within the service Sinfonia. It targets particularly data centre infrastructure applications, such as cluster file systems, lock managers, and group communication services. These applications must be fault-tolerant and scalable, and must provide consistency and reasonable performance. In Sinfonia's mini-transactions are short-lived and streamlined to scale well in a data centre. In a nutshell, Sinfonia is a service that allows hosts to share application data in a fault-tolerant, scalable, and consistent manner.

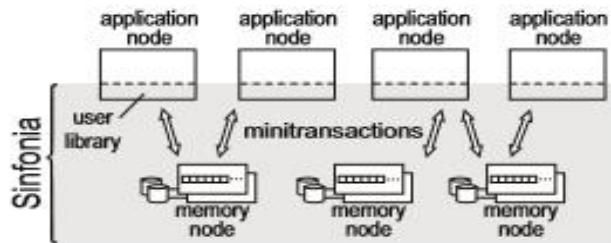


Figure 8: Sinfonia: Sharing of data in application nodes

5. TRANSACTION MANAGEMENT TECHNIQUES IN DISTRIBUTED SYSTEMS

5.1. The Grouping Protocol

The major goal of the key grouping protocol [22] is to transfer key ownership safely from the follower to the leader during group formation, and from the leader to the followers during group deletion by satisfying the correctness and liveness properties. Group creation

is initiated by an application client by sending group create request. Group formation can either be atomic or best effort. The key grouping protocol involves the leader of the group, which acts as the coordinator,

and the followers, which are the owners of the keys in the group. The leader key can either be specified by the client, or is automatically selected by the system. The group create request is routed to the node which owns the leader key. The leader logs the member list, and sends a join request to all followers. Once the group creation phase completes, the client can issue operations on the group.

5.2. G-Store

G-store [7] is based on key-group abstraction which provides facility for applications to select members of a group dynamically in the data store as well as dynamically break the groups. It maintains the efficiency, scalability and transactional access to groups of keys. The key group don't have any ordering restrictions (group members can be on different nodes) compare to Mstore [8]. For TM G-Store also use the concept of key grouping protocol to ensure that the ownership of the members of a group remain within a single node. In this way implementing TM in a group does not require any distributed synchronization and is similar to TM in single node relational databases. This allows G-Store to use matured and proven techniques of database research for TM on a group. Also transactions are guaranteed to be limited to the boundaries of a single node. This design approach allows G-Store to provide efficient and scalable transactional access within a group.

In implementation of G-Store an optimistic concurrency control (OCC) is used for guaranteeing serializable transactions. Atomicity and durability properties of transactions are achieved through WAL similar to that used in databases. Also, as leader owns access to the group members, the updates are asynchronously applied to the underlying key-value store. Moreover, the key Group abstraction does not define a relationship between two groups. Groups are independent of each other and the transaction on a group generates consistency only within the confines of a group.

The key Group abstraction allows efficient execution of ACID transactions on a group while allowing the system to scale efficiently through horizontal partitioning, a design which forms the core of scalable key-value stores. G -Store is similar to RDBMSs which rely on atomic disk reads and writes at the granularity of pages to implement transactions spanning higher level logical entities such as tables, while G-Store uses atomic access at a single key granularity provided by the underlying key-value store. The only difference in G-Store is transactions are limited to smaller logical entities called groups.

During the lifetime of a group, all accesses to the members of the groups are guaranteed to be through the leader, and the leader has exclusive access to the members of the group. The leader therefore caches the contents of the group members as well as the updates made as a result of the transactions on the group. This obviates the need for expensive distributed commitment protocols like 2PC when a transaction on a group is being executed, even though the keys in a group can be physically located on multiple nodes. This results in efficient transaction execution and reduction in execution latency.

All updates to a group are appended to a WAL which is similar to a commit log in database. The leader executes the group operation, logs them, and updates its local cache which stores the data corresponding to the members of the group. The WAL is maintained as a part of the protocol state, and is separate from any log used by the underlying key-value store.

5.3. Megastore

Mstore [8] is based on entity group unlike key-groups in G-Store. In Mstore entities groups are static in nature, although it takes a step beyond single key access patterns by supporting transactional access for groups of keys, which are formed using keys with a specific hierarchical

structure [8]. Since, keys cannot be updated in place, once a key is created as a part of group, it has to be in the group for the rest of its lifetime. Thus entity groups have static nature.

Entities within an entity group are mutated with single phase ACID transactions for which the commit record is replicated via paxos. In Mstore operations across entity groups could rely on expensive 2-phase commits and provide efficient asynchronous messaging. A transaction in a sending entity group places one or more messages in a queue, transactions in receiving entity groups atomically consume these messages and apply ensuing mutations. In Mstore cross entities group transactions supported via 2PC.

5.4. Scalable Transactions for Web Applications in the Cloud

In this approach the clients issues HTTP requests to a web application, which in turn issues transactions to a transaction processing system (TPS) [9]. The TPS is composed of any number of LTM, each of which is responsible for a subset of all data items. The web application can submit a transaction to any LTM that is responsible for one of the accessed data items. This LTM then acts as the coordinator of the transaction across all LTMs in charge of the data items accessed by the transaction. The LTMs operate on an in-memory copy of the data items loaded from the cloud storage service. Data updates resulting from transactions are kept in memory of the LTMs and periodically check pointed back to the cloud storage service.

It works with the 2PL. In the first phase, the coordinator requests all involved LTMs and asks them to check that the operation can indeed been executed correctly. If all LTMs vote favourably, then the second phase actually commits the transaction, otherwise transaction is aborted.

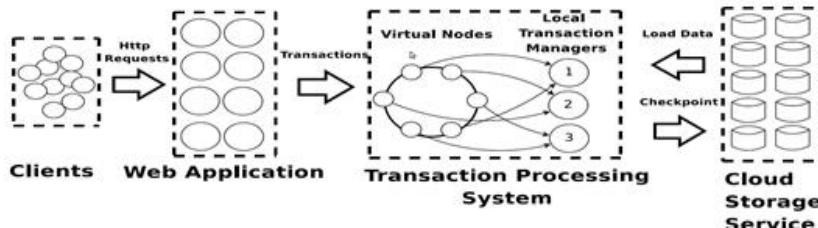


Figure 9: System Model

They assign data items to LTMs using consistent hashing. To achieve a balanced assignment, they first cluster data items into virtual nodes, and then assign virtual nodes to LTMs (Figure 9). To avoid LTM failures, virtual nodes and transaction states are replicated to one or more LTMs. After an LTM server failure, the latest updates can then be recovered and affected transactions can continue execution while satisfying ACID properties.

5.5. ElasTraS

ElasTraS [10] provides a two level hierarchy of transaction managers which are responsible for providing transactional guarantees, while providing elastic scalability with increase in demand. ElasTraS uses two types of transactional managers HTM (High level Transaction Manager) and OTM (Owning Transaction Manager).The OTM are the entities responsible for the executions on the partitions of the databases, and have exclusive access rights to the partitions they own. These are analogous to the tablet servers in Bigtable. An OTM is responsible for all the concurrency control and recovery functionality for the partitions it owns. At the top of the stack are the application and web servers that interact with the

database. Requests to the database are handled through the load balancer. When a transaction request arrives, the load balancer forwards it to a HTM. The HTM decides whether it can execute the transaction locally or route it to the appropriate OTM which owns exclusive access rights to the data accessed by the transaction.

In ElasTraS the database tables are partitioned and can be configured for both static and dynamic partitioning. Static partitioning is same like database partitioning, the database designer partitions the database, and ElasTraS is responsible for mapping partitions to specific OTM and reassigning partitions with changing load characteristics to ensure scalability and elasticity. In this configuration the application is aware of the partitions, and hence can be designed to limit transactions to single partitions. In such a configuration, ElasTraS can provide ACID transactional guarantees for transactions to single partitions. In dynamic partitioning configuration, in addition to managing partition mapping, is also responsible for database partitioning using range or hash based partitioning schemes. To ensure scalability in dynamic partitioning set up, and avoid distributed transaction, it only supports mini-transactions. More precisely, TM in a statically partitioned setup, applications can limit transactions to single partitions. ElasTranS guarantees consistency only within a partition of the databases and there is no notion of consistency across partitions or global serializability.

5.6. ecStore

The TM in ecStore [11] use two facts or characteristics in cloud storage. First, it is usually sufficient to perform operations on a recent snapshot of data rather than on up-to-second most recent data. Second, the locality of data accessed transactions, because in web applications users are more likely to operate on their own data, which forms an entity group or a key group as characterized in Mstore and Gstore. By using both facts of cloud data, ecStore use a hybrid scheme of multi-version and optimistic concurrency control. The beauty of this approach is that multiple versions of data can benefit the read only transactions, while the optimistic method protects the system from the locking overhead of update transactions.

In this hybrid scheme, each transaction has a start-up timestamp, which is assigned when the transaction starts, and commits timestamps, which is set up during the commit process. In addition, each data object also maintains the commit timestamp of its most recent update transaction. When a transaction accesses a data object, the most recent version of the data with a timestamp less than transaction's start-up timestamp is returned. Thus no locking overhead is incurred by the read requests. However, there are two main differences when we combine with the multiversion method. First, read-only transactions run against a consistent snapshot of the database, hence they can commit without validation phase. Second, the validation phase of update transactions uses the version number of data objects to check for write-write and write-read conflicts among concurrent transactions [17]. In particular, an update transaction is allowed to commit only if the version of any data object observed by this transaction during the read phase is still the same when the transaction is validated, meaning that these data objects have not been updated by other concurrent transactions. By using version-based validation, there is no need to store old write-sets of committed transactions just for the purpose of validating read-set/write-set instructions.

5.7. Sinfonia

Mini-transactions allow an application to update in multiple memory nodes while ensuring atomicity, consistency, isolation, and durability. Mini-transactions have compare items, read items, and write items. Compare items are locations to be tested for equality against supplied data, if any test fails, the mini-transaction aborts. If the mini-transaction commits, read items are locations to be read and returned, while writes items are locations to be written (Figure 10). Application nodes execute and commit mini-transactions using the two-phase protocol. Phase 1 executes and prepares the mini-transaction, while phase 2 commits it. More precisely, in phase 1, the

coordinator (application node) generates a new transaction id (tid) and sends the mini-transaction to the participants (memory nodes). First, each participant then tries to lock the address of its items in the mini-transaction (without blocking) then executes the comparisons in the compare items and, if all comparisons succeed, performs the reads in the read items and

5
4

buffer the write items. Finally it decides on a vote, if all locations could be locked and all compare items succeeded, the vote is for committing, otherwise it is for aborting. In phase 2, the coordinator tells participants to commit if all votes are committing. If commit, participants applies the write items, otherwise it discards them. In either case, the participants releases all locks acquired by the mini-transaction.

The coordinator never logs any information, unlike in standard two-phase commit. If the mini-transaction aborts because some locks were busy, the coordinator retries the mini-transaction after a while using a new tid. Participants log mini-transactions in the redo-log in the first phase (if logging is enabled); logging occurs only if the participant votes to commit. Only write items are logged, not compare or read items, to save space. The redo-log in Sinfonia also serves as a write-ahead log to improve performance. Participants keep an in-doubt list of undecided transaction tids, a forced-abort list of tids that must be voted to abort, and a decided list of finished tids and their outcome.

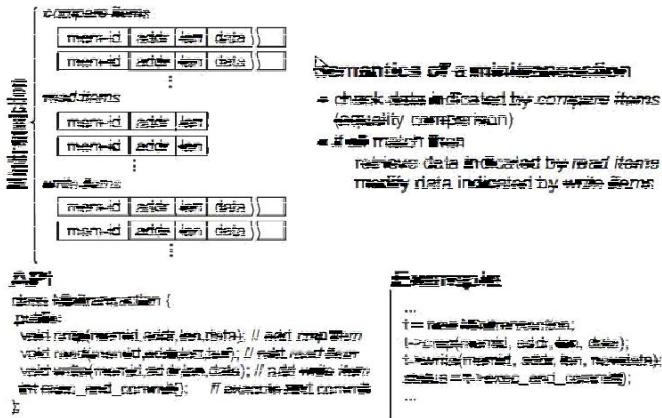


Figure 10: Sinfonia: MiniTransaction

5.8. PNUTS

PNUTS [13] uses OCC protocol for TM. In PNUTS we have test-and-set write instead of read-modify-write, where a call performs the requested write to the record if and only if the present version of the record is the same as required version. This call can be used to implement transactions that first read a record, and then do a write to the record based on the read. The test-and-set write ensures that two such concurrent increment transactions are properly serialized. The API of PNUTS with comparison to that of SQL, may be criticized for revealing too many implementation details such as sequence numbers. However, revealing these details does allow the application to indicate cases where it can do with some relaxed consistency for higher performance such as read-critical. Similarly, a test-and-set write allows us to implement single-row transactions without any locks, a highly desirable property in distributed systems. According to the need, the API can be packaged into the traditional begin transaction and commit for single-row transactions at the cost of losing expressiveness. In particular, PNUTS make no guarantees as to consistency for multi-record transactions. PNUTS model can provide serializability on a per-record basis. In particular, if an application reads or writes the same record multiple times in the same transaction, the application must use record versions to validate its own reads and writes to ensure serializability for the transaction.

6. COMPARISON OF TM IN G-STORE WITH OTHER PRODUCTS

The comparison of transaction management in G-store with other products is represented

in table 1. It is very difficult to run existing shared memory applications transparently on a distributed

5
5

system. The loosely coupled nature of distributed systems created hard efficiency and fault tolerance problems. Further, the G-Store depends on the underlying key-value store for consistency (table 1). If BigTable is used as underlying key-value store it provides synchronous or strong consistency and if PNUTS and Dynamo are used then it provides asynchronous consistency. Generally Mstore provides synchronous consistency but in the case of group to group communications it offers asynchronous consistency (table 1). On the other hands most of the systems use multi-version or eventual consistency.

Table 1: Comparison of Transaction Management in G-Store with other Products:

| | G-Store | MStore | ecStore | Sinfonia | Scalable Transaction for Web Applications in the Cloud | Elastrans |
|---|--|------------------------------|--|-------------|--|---|
| Key-Access | M-Key | Entity Groups | Multiple | Single | Single | Single |
| Key-Grouping | Dynamic | Static | Static | N/A | N/A | N/A |
| Order of Keys | NO | YES/Contiguous | YES | N/A | N/A | N/A |
| Concurrency Control Protocol in TM | OCC Protocol | 2PC | OCC and Multiversion / Hybrid Approach | 2PC | 2PC | Protocols used by Traditional Distributed Databases |
| Partitioning | Range & Hash | Range | Range | ----- | Hash | Range or Hash in Dynamic Mode |
| Replication | Synchronous (if Bigtable) | Synchronous (Paxos rep algo) | Asynchronous | Synchronous | Synchronous | Synchronous |
| Consistency | Strong (if Bigtable depends on kv store, Weak if Dy and PNUTS) | Synchronous | Multiversion / Eventual | Synchronous | Multiversion / Eventual | Multiversion |
| Distributed Transactions | YES | YES | YES | YES | YES | YES |
| Appropriate for Collaborative Applications | YES | NO | NO | NO | NO | NO |
| Minitransactions | NO | NO | NO | YES | NO | YES in Dynamic Configuration |
| Atomicity and Durability | WAL | WAL | WAL | WAL | WAL | WAL |

For concurrency control the G-Store uses an optimistic concurrency control protocol to guarantee the serializable transactions. In comparison, Mstore uses the expensive 2PC protocol for concurrency in transactions and ecStore use a hybrid approach of multi-version and optimistic concurrency control. The advantage of ecStore approach is the handling of multiple versions of data which benefits the read only transactions, while the optimistic method protects the system from the locking overhead of update transactions. G-Store uses the concept of key grouping protocol in TM to ensure that the ownership of the members of a group remain within a single node. In this way implementing TM in a group does not require any distributed synchronization. Moreover, G-Store is suitable for collaborative applications compared to other applications such as Sinfonia which uses distributed share memory system. The comparison of cloud based storage systems is represented in table 2.

Table 2: Comparison of Features of Cloud Based Storage Systems:

| Features | Partitioning | Load Balancing | Replication | | Distributed Transaction Support |
|----------|--------------|----------------------------------|-------------|--------------------------|---------------------------------|
| | | | Syn/Asy | Consistency | |
| PNUTS | Hash/Range | Data Migration | Asyn | Time line + Eventual | No |
| ecStore | Range | Data Migration | Asyn | Multi-version + Eventual | Yes |
| Bigtable | Range | Other | Sync | Multi-version | No |
| Dynamo | Hash | Multi virtual nodes on a machine | Asyn | Eventual | No |

7. CONCLUSIONS

There is no single database product or technology which provides the complete solution for data management challenges in cloud computing environment. There are different products (implementations) are being used to target specific database design and implementation issues in cloud environment. There is always a trade-off in using such technologies in terms of performance, availability, consistency, concurrency, scalability and elasticity. There are some important areas of further research where we still need more attention. It includes the consistency at different scales in a cloud, elastic TM techniques for elastic cloud architectures, performance management, federated databases for federated clouds, and data security in cloud environment,

We have analysed various TM techniques used in current cloud based systems along with the techniques and protocols used by traditional distributed database systems. We have focused on G-Store approach of TM and compared with other existing approaches. G-store can inherit data model from underlying key value store, so its TM technique can also be used on PNUTS, BigTable and Dynamo. The major advantage of G-Store for TM in cloud computing environment is dynamic key grouping which is the requirement of current and future collaborative web application that needs dynamic grouping without prior requirement of specific key orders. Although, Mstore also have grouping feature but that is static in nature and the key grouping technique where entity groups have specific key ordering is not appropriate for future collaborative web applications. In comparison, all other systems are based on single key access semantics. It is observed that most of the system use WAL as a preferred approach for atomicity and durability requirement of transaction.

REFERENCES

- [1] P. V. Tamer Ozsu, Principles of Distributed Database Systems, Springer. 2011.
- [2] B. Hayes, "Cloud computing," Commun. ACM, vol. 51, no. 7, p. 9, Jul. 2008.
- [3] R. G. Tiwari, S. B. Navathe, and G. J. Kulkarni, "Towards Transactional Data Management over the Cloud," 2010 Second Int. Symp. Data, Privacy, E-Commerce, pp. 100–106, Sep. 2010.
- [4] S. Das, "Scalable and Elastic Transactional Data Stores for Cloud Computing Platforms," 2011.
- [5] A. El Abbadi, "Big Data and Cloud Computing : Current State and Future Opportunities," in *EDBT 2011*, Uppsala, Sweden, 2011, pp. 0–3.
- [6] D. Kossmann, T. Kraska, and S. Loesing, "An evaluation of alternative architectures for transaction processing in the cloud," in Proceedings of the 2010 international conference on Management of data - SIGMOD'10, 2010, pp. 579–590.
- [7] S. Das and A. El Abbadi, "G-Store : A Scalable Data Store for Transactional Multi key Access

in the Cloud," in *SoCC'2010*, Indianapolis, Indiana, USA, 2010, pp. 163–174.

- [8] J. Baker, C. Bond, J. C. Corbett, J. J. Furman, A. Khorlin, J. Larson, L. Jean-michel, Y. Li, A. Lloyd, and V. Yushprakh, "Megastore : Providing Scalable , Highly Available Storage for Interactive Services," in *5th Biennial Conference on Innovative Data Systems Research*, 2011, pp. 223–234.
- [9] Z. Wei, G. Pierre, and C.-H. Chi, "CloudTPS: Scalable Transactions for Web Applications in the Cloud," *IEEE Trans. Serv. Comput.*, vol. 5, no. 4, pp. 525–539, 2012.
- [10] S. Das, A. El Abbadi, C. Science, and U. C. S. Barbara, "ElastraS : An Elastic Transactional Data Store in the Cloud," in *USENIX HotCloud'2*, 2009.
- [11] H. T. Vo, C. Chen, and B. C. Ooi, "Towards elastic transactional cloud storage with range query support," *Proc. VLDB Endow.*, vol. 3, no. 1–2, pp. 506–514, Sep. 2010.
- [12] M. K. Aguilera, A. Merchant, A. Veitch, and C. Karamanolis, "Sinfonia : A New Paradigm for Building Scalable Distributed Systems," in *SOSP 2007*, Stevenson, Washington, USA, 2007, pp. 159–174.
- [13] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "PNUTS : Yahoo !'s Hosted Data Serving Platform," in *VLDB 2008*, Auckland, New Zealand, 2008.
- [14] G. Decandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo : Amazon 's Highly Available Key-value Store," *ACM SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, 2007.
- [15] F. A. Y. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable : A Distributed Storage System for Structured Data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, 2008.
- [16] D. Agrawal, A. El Abbadi, S. Antony, and S. Das, "Data Management Challenges in Cloud Computing Infrastructures," in *DNIS 2010, LNCS*, 5999, 2010, pp. 1–10.
- [17] S. Sakr, A. Liu, D. M. Batista, and M. Alomari, "A Survey of Large Scale Data Management Approaches in Cloud Environments," *IEEE Commun. Surv. Tutorials*, vol. 13, no. 3, pp. 311–336, 2011.
- [18] D. J. Abadi, "Data Management in the Cloud : Limitations and Opportunities," in *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 2009, pp. 1–10.
- [19] R. Cattell, "Scalable SQL and NoSQL Data Stores," in *ACM SIGMOD Record*, 2010, vol. 39, no. 4, pp. 12–27.
- [20] P. Romano and L. Rodrigues, "Cloud-TM : Harnessing the Cloud with Distributed Transactional Memories," pp. 1–6.
- [21] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann. 1993.
- [22] Y. Challal and H. Seba, "Group Key Management Protocols : A Novel Taxonomy," *Int. J. Inf. Technol.*, vol. 2, no. 1, pp. 105–118, 2005.
- [23] X. C. Song, I. C. Society, and J. W. S. Liu, "Maintaining Temporal Consistency : Pessimistic vs . Optimistic Concurrency Control," *IEEE Trans. Knowl. Data Eng.*, vol. 7, no. 5, pp. 786–796, 1995.
- [24] A. Waqas, Z. M. Yusof, A. Shah and M. A. Khan, "Architecture for Resources Sharing Between Clouds," in *2014 Conference on Information Assurance and Cyber Security (CIACS2014)*, pp.23,28, 12-13 June 2014. doi: 10.1109/CIACS.2014.6861326
- [25] A. Waqas, Zulkefli.Muhammed, Asadullah Shah, "A security -based survey and classification of Cloud Architectures, State of Art and Future Directions," in *2013 International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*, pp.284,289, 23-24 Dec. 2013. doi: 10.1109/ACSAT.2013.63
- [26] Tinghuai Ma, Chu Ya, Licheng Zhao and Otgonbayar Ankhbayar , "security -based survey and classification in Cloud Computing: policy and Algorithm," *IETE Tech. Rev.*, vol. 31, no. 1, pp. 4-16, 2014.
- [27] J. Dean and S. Ghemawat, "MapReduce : Simplified Data Processing on Large Clusters," in *6th Symposium on Operating Systems Design and Implementation*, 2004, pp. 137–149.
- [28] Z. Wei, G. Pierre, and C. Chi, "Scalable Transactions for Web Applications in the Cloud," in *Euro-Par 2009, Parallel Processing*, 2009, pp. 442–453.
- [29] M. N. Vora, "Hadoop-HBase for large-scale data," in *Proceedings of 2011 International Conference on Computer Science and Network Technology*, 2011, pp. 601–605.
- [30] L. Lamport, "Paxos Made Simple," *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, 2001.
- [31] A. Waqas, Zulkefli.Muhammed, Asadullah Shah, "Fault Tolerant Cloud Auditing," in *2013 5th International Conference on Information and Communication Technology for the Muslim*

Database file organization, file organization on disk, heap files and sorted files, hashing, single and multi-level indexes, dynamic multilevel indexing using B-Tree and B+ tree, index on multiple keys.

4.6. Storage and Indexing:

4.6.1. Unit Objectives:

- Provide a solid introduction to the topic of file structure design
- Discuss, in detail, the data structures necessary for achieving its efficiency objectives.
- Introducing techniques for organization and manipulation of data in *secondary storage* including the low level aspects of file manipulation which include basic file operations, secondary storage devices and system software.
- Introducing the most important high-level file structures tools which include indexing, sequential processing, B trees, Hashing.

4.6.2. Unit Objectives:

- Explain the importance of file structures in the Data Storage and Manipulation
- Show how various kind of secondary storage devices to store data.
- Show how the File Structure approach differs from the data base approach.
- Know the low level aspects of file manipulation.

- Know the importance of data compression.
- Know some of the high-level file structures tools and recognize the difference between various indexing techniques.
- Implement some of the learned techniques and concepts using C++ for solving various file management problems.

4.6.3. Unit Lecture Plan:

| Lecture No. | Topic | Methodology | Quick reference |
|--------------------|--|--------------------|-------------------------|
| 1 | Storage and Indexing: Database file organization, | Chalk & Board | Text. Book with Page |
| 2 | file organization on disk | Chalk & Board | Text. Book with Page |
| 3 | heap files and sorted files | Chalk & Board | Text. Book with Page |
| 4 | Hashing | Chalk & Board | Text. Book with Page |
| 5 | single and multi-level indexes | Chalk & Board | Text. Book with Page |
| 6 | dynamic multilevel indexing using B-Tree | Chalk & Board | Text. Book with Page |
| 7 | dynamic multilevel indexing using B+ tree | Chalk & Board | Text. Book with Page |
| 8 | index on multiple keys | Chalk & Board | Text. Book with Page |

4.6.4 Teaching Material / Teaching Aids as per above lecture plan.

4.6.4.1 Lecture-1

DATABASE FILE ORGANIZATION

The basic abstraction of data in a DBMS is a collection of records, or a *file*, and each file consists of one or more pages. The *files and access methods* software layer organizes data carefully to support fast access to desired subsets of records. Understanding how records are organized is essential to using a database system effectively.

A file organization is a method of arranging the records in a file when the file is stored on disk. Each file organization makes certain operations efficient but other operations expensive.

Consider a file of employee records, each containing *age*, *name*, and *sal* fields, which we use as a running example in this chapter. If we want to retrieve employee records in order of increasing age, sorting the file by age is a good file organization, but the sort order is expensive to maintain if the file is frequently modified. Further, we are often interested in supporting more than one operation on a given collection of records. In our example, we may also want to retrieve all employees who make more than \$5000. We have to scan the entire file to find such employee records.

DATA ON EXTERNAL STORAGE

DBMS stores vast quantities of data, and the data must persist across program executions. Therefore, data is stored on external storage devices such as disks and tapes, and fetched into main memory as needed for processing. The unit of information read from or written to disk is a *page*. The size of a page is a DBMS parameter, and typical values are 4KB or 8KB.

Disks are the most important external storage devices. They allow us to retrieve any page at a (more or less) fixed cost per page. However, if we read several pages in the order that they are stored physically, the cost can be much less than the cost of reading the same pages in a random order.

Tapes are sequential access devices and force us to read data one page after the other. They are mostly used to archive data that is not needed on a regular basis.

Data is read into memory for processing, and written to disk for persistent storage, by a layer of software called the *buffer manager*. When the *files and access methods* layer (which we often refer to as just the file layer) needs to process a page, it asks the buffer manager to fetch the page, specifying the page's rid. The buffer manager fetches the page from disk if it is not already in memory.

FILE ORGANIZATIONS AND INDEXING

An index is a data structure that organizes data records on disk to optimize certain kinds of retrieval operations. An index allows us to efficiently retrieve all records that

satisfy search conditions on the search key fields of the index. We can also create additional indexes on a given collection of data records, each with a different search key, to speed up search operations that are not efficiently supported by the file organization used to store the data records.

Consider our example of employee records. We can store the records in a file organized as an index on employee age; this is an alternative to sorting the file by age. Additionally, we can create an auxiliary index file based on salary, to speed up queries involving salary. The first file contains employee records, and the second contains records that allow us to locate employee records satisfying a query on salary.

We use the term data entry to refer to the records stored in an index file. A data entry with search key value k , denoted as k^* , contains enough information to locate (one or more) data records with search key value k . We can efficiently search an index to find the desired data entries, and then use these to obtain data records (if these are distinct from data entries).

There are three main alternatives for what to store as a data entry in an index:

1. A data entry h is an actual data record (with search key value k).
2. A data entry is a (k, rid) pair, where rid is the record id of a data record with search key value k .
3. A data entry is a $(k, rid-list)$ pair, where $rid-list$ is a list of record ids of data records with search key value k .

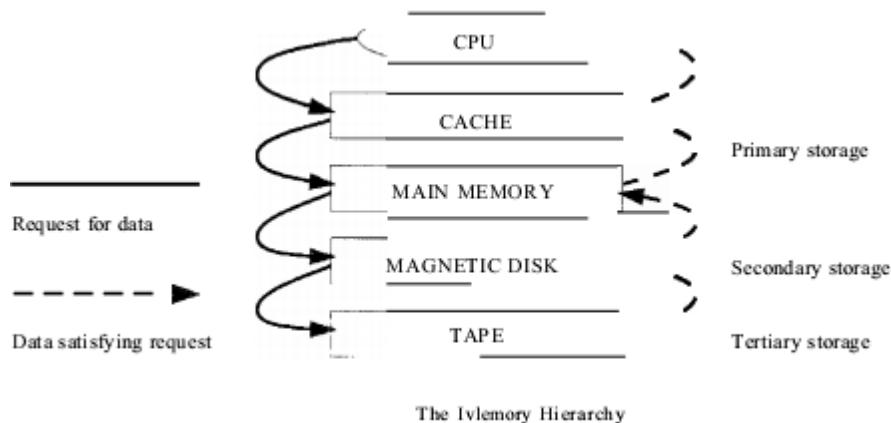
4.6.4.2 Lecture-2

FILE ORGANIZATION ON DISK:

STORING DATA: DISKS AND FILES

THE MEMORY HIERARCHY

Memory in a computer system is arranged in a hierarchy, as shown in Figure 9.1. At the top, we have primary storage, which consists of cache and main memory and provides very fast access to data. Then comes secondary storage, which consists of slower devices, such as magnetic disks. Tertiary storage is the slowest class of storage devices; for example, optical disks and tapes. Currently, the cost of a given amount of main memory is about 100 times



the cost of the same amount of disk space, and tapes are even less expensive than disks. Slower storage devices such as tapes and disks play an important role in database systems because the amount of data is typically very large. Since buying enough main memory to store all data is prohibitively expensive, we must store data on tapes and disks and build database systems that can retrieve data from lower levels of the memory hierarchy into main memory as needed for processing.

There are reasons other than cost for storing data on secondary and tertiary storage. On systems with 32-bit addressing, only 2^{32} bytes can be directly referenced in main memory; the number of data objects may exceed this number! Further, data must be maintained across program executions. This requires storage devices that retain information when the computer is restarted (after a shutdown or a crash); we call such storage nonvolatile. Primary storage is usually volatile (although it is possible to make it nonvolatile by adding a battery backup feature), whereas secondary and tertiary storage are nonvolatile.

Tapes are relatively inexpensive and can store very large amounts of data. They are a good choice for *archival* storage, that is, when we need to maintain data for a long period but do not expect to access it very often. A Quantum DLT 4000 drive is a typical tape device; it stores 20 GB of data and can store about twice as much by compressing the data. It records data on 128 *tape tracks*, which can be thought of as a linear sequence of adjacent bytes, and supports a sustained transfer rate of 1.5 MB/sec with uncompressed data (typically 3.0 MB/sec with compressed data). A single DLT 4000 tape drive can be used to access up to seven tapes in a stacked configuration, for a maximum compressed data capacity of about 280 GB.

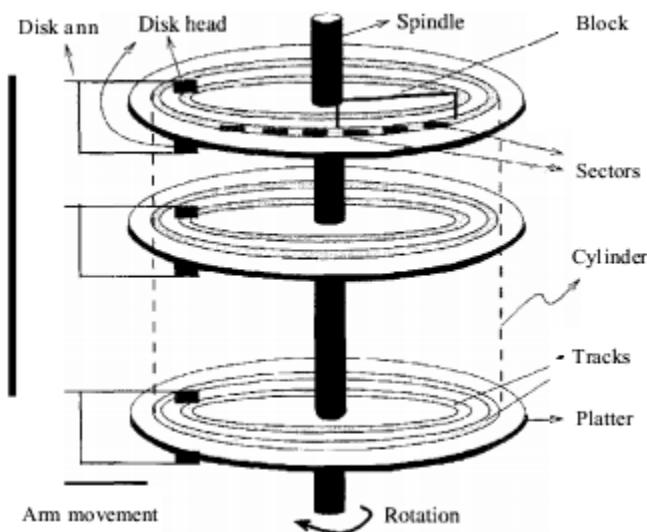
The main drawback of tapes is that they are sequential access devices. We must essentially step through all the data in order and cannot directly access a given location on tape. For example, to access the last byte on a tape, we would have to wind through the entire tape first. This makes tapes unsuitable for storing *operational data*, or data that is frequently accessed. Tapes are mostly used to back up operational data periodically.

Magnetic Disks

Magnetic disks support direct access to a desired location and are widely used for

database applications. A DBMS provides seamless access to data on disk; applications need not worry about whether data is in main memory or disk. To understand how disks work, consider Figure , which shows the structure of a disk in simplified form.

Data is stored on disk in units called disk blocks. A disk block is a contiguous sequence of bytes and is the unit in which data is written to a disk and read from a disk. Blocks are arranged in concentric rings called tracks, on one or more platters. Tracks can be recorded on one or both surfaces of a platter; we refer to platters as single-sided or double-sided, accordingly. The set of all tracks with the same diameter is called a cylinder, because the space occupied by these tracks is shaped like a cylinder; a cylinder contains one track per platter surface. Each track is divided into arcs, called sectors, whose size is a



characteristic of the disk and cannot be changed. The size of a disk block can be set when the disk is initialized as a multiple of the sector size.

An array of disk heads, one per recorded surface, is moved as a unit; when one head is positioned over a block, the other heads are in identical positions with respect to their platters. To read or write a block, a disk head must be positioned on top of the block.

Current systems typically allow at most one disk head to read or write at any one time. All the disk heads cannot read or write in parallel~this technique would increase data transfer rates by a factor equal to the number of disk heads and considerably speed up sequential scans. The reason they cannot is that it is very difficult to ensure that all the heads are perfectly aligned on the corresponding tracks. Current approaches are both expensive and more prone to faults than disks with a single active In practice, very few commercial products support this capability and then only in a limited way; for example, two disk heads may be able to operate in parallel

A disk controller interfaces a disk drive to the computer. It implements commands to read or write a sector by moving the arm assembly and transferring data to and from the disk surfaces. A checksum is computed for when data is written to a sector and

stored with the sector. The checksum is computed again when the data on the sector is read back. If the sector is corrupted or the

REDUNDANT ARRAYS OF INDEPENDENT DISKS

Disk are potential bottlenecks for system performance and storage system reliability. Even though disk performance has been improving continuously, microprocessor performance has advanced much more rapidly. The performance of microprocessors has improved at about 50 percent or more per year, but disk access times have improved at a rate of about 10 percent per year and disk transfer rates at a rate of about 20 percent per year. In addition, since disks contain mechanical elements, they have much higher failure rates than electronic parts of a computer system. If a disk fails, all the data stored on it is lost.

A **disk array** is an arrangement of several disks, organized to increase performance and improve reliability of the resulting storage system. Performance is increased through data striping. Data striping distributes data over several disks to give the impression of having a single large, very fast disk. Reliability is improved through **redundancy**. Instead of having a single copy of the data, redundant information is maintained.

Redundancy

While having more disks increases storage system performance, it also lowers overall storage system reliability. Assume that the mean-time-to-failure (MTTF), of a single disk is 50,000 hours (about 5.7 years). Then, the MTTF of an array of 100 disks is only $50,000/100 = 500$ hours or about 21 days, assuming that failures occur independently and the failure probability of a disk does not change over time. (Actually, disks have a higher failure probability early and late in their lifetimes. Early failures are often due to undetected manufacturing defects; late failures occur since the disk wears out. Failures do not occur independently either: consider a fire in the building, an earthquake, or purchase of a set of disks that come from a 'bad' manufacturing batch.)

Reliability of a disk array can be increased by storing redundant information. If a disk fails, the redundant information is used to reconstruct the data on the failed disk. Redundancy can immensely increase the MTTF of a disk array. When incorporating redundancy into a disk array design, we have to make two choices. First, we have to decide where to store the redundant information. We can either store the redundant information on a small number of check disks or distribute the redundant information uniformly over all disks.

The second choice we have to make is how to compute the redundant information. Most disk arrays store parity information: In the parity scheme, an extra check disk contains information that can be used to recover from failure of anyone disk in the array. Assume that we have a disk array with D disks and consider the first bit on each data disk. Suppose that i of the D data bits are 1. The first bit on the check disk is set to 1 if i is odd; otherwise, it is set to 0. This bit on the check disk is called the parity of the data bits. The check disk contains parity information for each set of corresponding D data bits.

For example, with an additional 10 disks with redundant information, the MTTF of our example storage system with 100 data disks can be increased to more than 250 years! "What is more important, a large MTTF implies a small failure probability during the actual usage time of the storage system, which is usually much smaller than the reported lifetime or the MTTF. (Who actually uses 10-year-old disks?)

In a RAID system, the disk array is partitioned into reliability groups, where a reliability group consists of a set of *data disks* and a set of *check disks*. A common *7'cdundancy scheme* (see box) is applied to each group. The number of check disks depends on the RAID level chosen. In the remainder of this section, we assume for ease of explanation that there is only one reliability group. The reader should keep in mind that actual RAID implementations consist of several reliability groups, and the number of groups plays a role in the overall reliability of the resulting storage system.

Choice of RAID Levels

If data loss is not an issue, RAID Level 0 improves overall system performance at the lowest cost. RAID Level 0+1 is superior to RAID Level 1. The main application areas for RAID Level 0+1 systems are small storage subsystems where the cost of mirroring is moderate. Sometimes, RAID Level 0+1 is used for applications that have a high percentage of writes in their workload, since RAID Level 0+1 provides the best write performance. RAID Levels 2 and 4 are always inferior to RAID Levels 3 and 5, respectively. RAID Level 3 is appropriate for workloads consisting mainly of large transfer requests of several contiguous blocks. The performance of a RAID Level 3 system is bad for workloads with many small requests of a single disk block. RAID Level 5 is a good general-purpose solution. It provides high performance for large as well as small requests. RAID Level 6 is appropriate if a higher level of reliability is required.

DISK SPACE MANAGEMENT

The lowest level of software in the DB. the disk space manager, manages space on disk. Abstractly, the disk space manager supports the concept of a page as a unit of data and provides to allocate or de-allocate a page and read or write a page. The size of a page is chosen to be the size of a disk block and pages are stored as disk blocks so that reading or writing a page can be done in one disk I/O.

It is often useful to allocate a sequence of pages (IS a *contiguous* sequence of blocks to hold data frequently accessed in sequential order. This capability is essential for exploiting the advantages of sequentially accessing disk blocks, which we discussed earlier in this chapter. Such a capability, if desired, must be provided by the disk space manager to higher level layers of the DBMS.

The disk space manager hides details of the underlying hardware (and possibly the operating system) and allows higher levels of the software to think of the data a collection of pages.

BUFFER MANAGER

To understand the role of the buffer manager, consider a simple example. Suppose that the database contains 1 million pages, but only 1000 pages of main memory are available for holding data. Consider a query that requires a scan of the entire file. Because all the data cannot be brought into main memory at one time, the DBMS must bring pages into main memory as they are needed and, in the process, decide what existing page in main memory to replace to make space for the new page. The policy used to decide which page to replace is called the replacement policy.

In terms of the DBMS architecture presented in Section 1.8, the buffer manager is the software layer responsible for bringing pages from disk to main memory as needed. The buffer manager manages the available main memory by partitioning it into a collection of pages, which we collectively refer to as the buffer pool. The main memory pages in the buffer pool are called frames; it is convenient to think of them as slots that can hold a page (which usually resides on disk or other secondary storage media).

Higher levels of the DBMS code can be written without worrying about whether data pages are in memory or not; they ask the buffer manager for the page, and it is brought into a frame in the buffer pool if it is not already there. Of course, the higher-level code that requests a page must also release the page when it is no longer needed, by informing the buffer manager, so that the frame containing the page can be reused. The higher-level code must also inform the buffer manager if it modifies the requested page; the buffer manager then makes sure that the change is propagated to the copy of the page on disk.

In addition to the buffer pool itself, the buffer manager maintains some book-keeping information and two variables for each frame in the pool: *pinCount* and *dirty*. The number of times that the page currently in a given frame has been requested but not released—the number of current users of the page—is recorded in the *pin_count* variable for that frame. The Boolean variable *dirty* indicates whether the page has been modified since it was brought into the buffer pool from disk.

4.6.4.3 Lecture -3

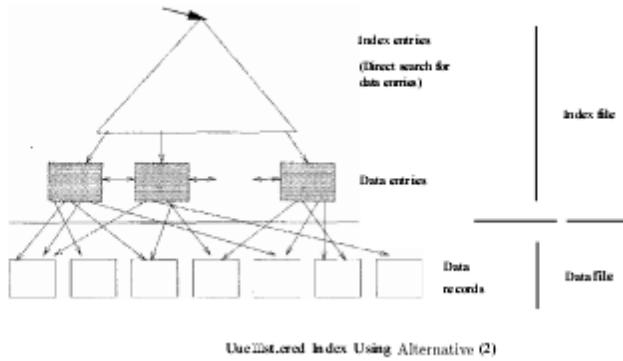
HEAP FILES AND SORTED FILES:

Clustered Indexes

When a file is organized so that the ordering of data records is the same as or close to the ordering of data entries in some index, we say that the index is clustered; otherwise, it is un-clustered. An index that uses Alternative (1) is clustered, by definition. An index that uses Alternative (2) or (3) can be a clustered index only if the data records are sorted on the search key field. Otherwise, the order of the data records is random, defined purely by their physical order, and there is no reasonable way to arrange the data entries in the index in the same order.

The cost of using an index to answer a range search query can vary tremendously based on whether the index is clustered. If the index is clustered, i.e., we are using the search key of a clustered file, the rids in qualifying data entries point to a contiguous

collection of records, and we need to retrieve only a few data pages. If the index is unclustered, each qualifying data entry could contain a rid that points to a distinct data page, leading to as many data page I/Os as the number of data entries that match the range



Primary and Secondary Indexes

Two data entries are said to be duplicates if they have the same value for the search key field associated with the index. A primary index is guaranteed not to contain duplicates, but an index on other (collections of) fields can contain duplicates. In general, a secondary index contains duplicates. If we know that no duplicates exist, that is, we know that the search key contains some candidate key, we call the index a **unique** index.

INDEX DATA STRUCTURES

One way to organize data entries is to hash data entries on the search key. Another way to organize data entries is to build a tree-like data structure that directs a search for data entries. We note that the choice of hash or tree indexing techniques can be combined with any of the three alternatives for data entries.

Hash-Based Indexing

In this approach, the records in a file are grouped in buckets, where a bucket consists of a primary page and, possibly, additional pages linked in a chain. The bucket to which a record belongs can be determined by applying a special function, called a hash function, to the search key. Given a bucket number, a hash-based index structure allows us to retrieve the primary page for the bucket in one or two disk I/Os.

On inserts, the record is inserted into the appropriate bucket, with 'overflow' pages allocated as necessary. To search for a record with a given search key value, we apply the hash function to identify the bucket to which such records belong and look at all pages in that bucket. If we do not have the search key value for the record, for example, the index is based on *sal* and we want records with a given age value, we have to scan all pages in the file.

Hash indexing is illustrated in Figure , where the data is stored in a file that is hashed on *age*; the data entries in this first index file are the actual data records. Applying the hash function to the *age* field identifies the page that the record belongs to. The hash function h for this example is quite simple; it converts the search key value to its binary representation and uses the two least significant bits as the bucket identifier.

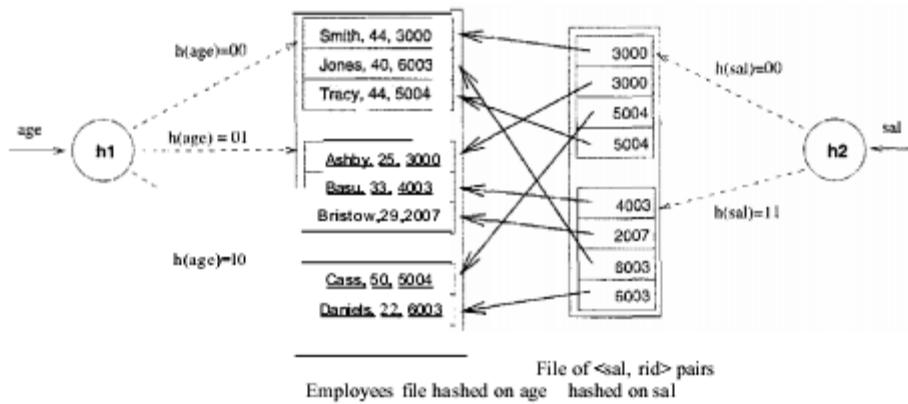


Figure also shows an index with search key *sal* that contains (sal, rid) pairs as data entries. The *tid* (short for *record id*) component of a data entry in this second index is a pointer to a record with search key value *sal* (and is shown in the figure as an arrow pointing to the data record).

The file of employee records is hashed on *age*, and Alternative (1) is used for data entries. The second index, on *sal*, also uses hashing to locate data entries, which are now $(\text{sal}, \text{rid}$ of employee record $)^\sim$ pairs; that is, Alternative (2) is used for data entries.

Note that the search key for an index can be any sequence of one or more fields, and it need not uniquely identify records. For example, in the salary index, two data entries have the same search key value 6003. (There is an unfortunate overloading of the term *key* in the database literature. A *primary key* or *candidate key*-fields that uniquely identify a record

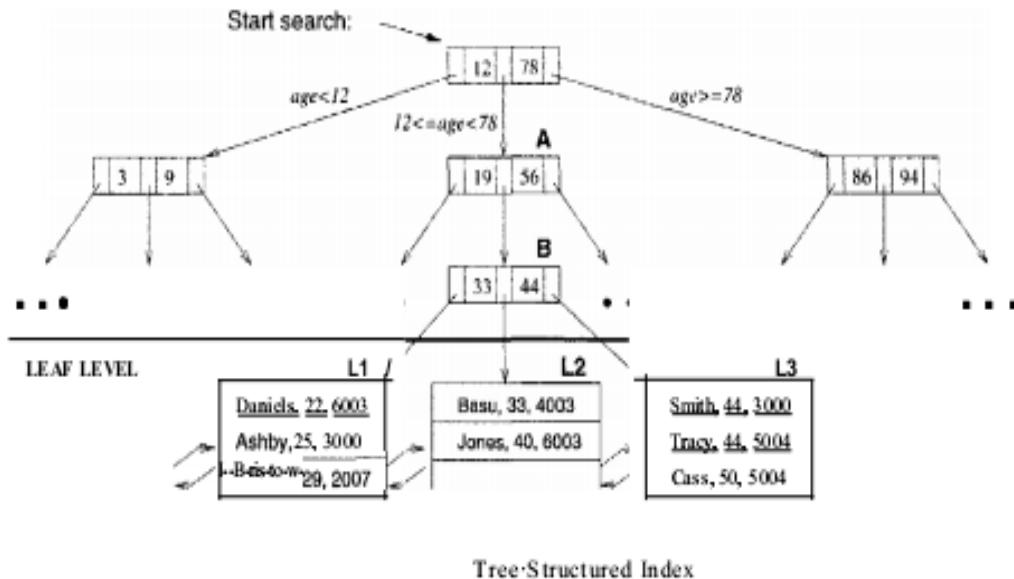
Tree-Based Indexing

An alternative to hash-based indexing is to organize records using a tree-like data structure. The data entries are arranged in sorted order by search key value, and a hierarchical search data structure is maintained that directs searches to the correct page of data entries.

In a tree-structured index with search *keyage*. Each node in this figure (e.g., nodes labeled A, B, L1, L2) is a physical page, and retrieving a node involves a disk I/O.

The lowest level of the tree, called the **leaf level**, contains the data entries; in our example, these are employee records. To illustrate the ideas better, as if there were additional employee records, some with age less than 22 and some with age greater than EiO (the lowest and highest age values that appear in Figure 8.2). Additional records with age less than 22 would appear in leaf pages to the left page L1, and

records with age greater than 50 would appear in leaf pages to the right of page L~~.



This structure allows us to efficiently locate all data entries with search key values in a desired range. All searches begin at the topmost node, called the root, and the contents of pages in non-leaf levels direct searches to the correct leaf page. Non-leaf pages contain node pointers separated by search key values. The node pointer to the left of a key value k points to a subtree that contains only data entries less than k . The node pointer to the right of a key value k points to a subtree that contains only data entries greater than or equal to k .

Thus, the number of disk I/Os incurred during a search is equal to the length of a path from the root to a leaf, plus the number of leaf pages with qualifying data entries. The B+ tree is an index structure that ensures that all paths from the root to a leaf in a given tree are of the same length, that is, the structure is always balanced in height. Finding the correct leaf page is faster

The operations we consider are these:

Scan: Fetch all records in the file. The pages in the file must be fetched from disk into the buffer pool. There is also a CPU overhead per record for locating the record on the page (in the pool).

Search with Equality Selection: Fetch all records that satisfy an equality selection; for example, "Find the employee record for the employee with *age* 23 and *sal* 50." Pages that contain qualifying records must be fetched from disk, and qualifying records must be located within retrieved pages.

Search with Range Selection: Fetch all records that satisfy a range selection; for example, "Find all employee records with *age* greater than 35."

Insert a Record: Insert a given record into the file. We must identify the page in the file into which the new record must be inserted, fetch that page from disk, modify it to include the new record, and then write back the modified page. Depending on the file organization, we may have to fetch, modify, and write back other pages as well.

Delete a Record: Delete a record that is specified using its rid. We must identify the page that contains the record, fetch it from disk, modify it, and write it back. Depending on the file organization, we may have to fetch, modify, and write back other pages as well.

Cost Model

In our comparison of file organizations, and in later chapters, we use a simple cost model that allows us to estimate the cost (in terms of execution time) of different database operations. We use B to denote the number of data pages when records are packed onto pages with no wasted space, and R to denote the number of records per page. The average time to read or write a disk page is D , and the average time to process a record (e.g., to compare a field value to a selection constant) is C . In the hashed file organization, we use a function, called a *hash function*, to map a record into a range of numbers; the time required to apply the hash function to a record is H . For tree indexes, we will use F to denote the fan-out, which typically is at least 100.

Typical values today are $D = 15$ milliseconds, $C = 100$ nanoseconds; we therefore expect the cost of I/O to dominate. I/O is often (even typically) the dominant component of the cost of database operations, and so considering I/O costs gives us a good first approximation to the true costs. Further, CPU speeds are steadily rising, whereas disk speeds are not increasing at a similar pace. (On the other hand, as main memory sizes increase, a much larger fraction of the needed pages are likely to fit in memory.)

Heap Files

Scan: The cost is $B(D + RC)$ because we must retrieve each of B pages taking time D per page, and for each page, process R records taking time C per record.

Search with Equality Selection: Suppose that we know in advance that exactly one record matches the desired equality selection, that is, the selection is specified on a candidate key. On average, we must scan half the file, assuming that the record exists and the distribution of values in the search field is uniform. For each retrieved data page, we must check all records on the page to see if it is the desired record. The cost is $0.5B(D + RC)$. If no record satisfies the selection, however, we must scan the entire file to verify this.

If the selection is not on a candidate key field (e.g., "Find employees aged 18"), we always have to scan the entire file because records with $age = 18$ could be dispersed all over the file, and we have no idea how many such records exist.

Search with Range Selection: The entire file must be scanned because qualifying records could appear anywhere in the file, and we do not know how many qualifying

records exist. The cost is $B(D + RC)$.

Insert: we assume that records are always inserted at the end of the file. we must fetch the last page in the file, add the record, and write the page back. The cost is $2D + C$.

Delete: We must find the record, remove the record from the page, and write the modified page back. we assume that no attempt is made to compact the file to reclaim the free space created by deletions, for simplicity. The cost is the cost of searching plus $C + D$.

Sorted Files

Scan: The cost is $B(D + RC)$ because all pages must be examined. Note that this case is no better or worse than the case of unordered files. However, the order in which records are retrieved corresponds to the sort order, that is, all records in *age* order, and for a given age, by *sal* order.

Search with Equality Selection: We assume that the equality selection matches the sort order (*age*, *sal*). In other words, we assume that a selection condition is specified on at least the first field in the composite key (e.g., *age* = 30). If not (e.g., selection *sal* = t50 or *department* = "Toy"), the sort order does not help us and the cost is identical to that for a heap file.

We can locate the first page containing the desired record or records, should any qualifying records exist, with a binary search in $\log_2 B$ steps. (This analysis assumes that the pages in the sorted file are stored sequentially, and we can retrieve the *i*th page on the file directly in one disk I/O.) Each step requires a disk I/O and two comparisons. Once the page is known, the first qualifying record can again be located by a binary search of the page at a cost of $C \log_2 R$. The cost is $D \log_2 B + C \log_2 R$, which is a significant improvement over searching heap files.

Search with Range Selection: Again assuming that the range selection matches the composite key, the first record that satisfies the selection is located as for search with equality. Subsequently, data pages are sequentially retrieved until a record is found that does not satisfy the range selection; this is similar to an equality search with many qualifying records.

The cost is the cost of search plus the cost of retrieving the set of records that satisfy the search. The cost of the search includes the cost of fetching the first page containing qualifying, or matching, records. For small range selections, all qualifying records appear on this page. For larger range selections, we have to fetch additional pages containing matching records.

Insert: To insert a record while preserving the sort order, we must first find the correct position in the file, add the record, and then fetch and rewrite all subsequent pages (because all the old records are shifted by one slot, assuming that the file has no empty slots). On average, we can assume that the inserted record belongs in the middle of the file. Therefore, we must read the latter half of the file and then write it back after adding the new record. The cost is that of searching to find the position of the new record plus $2 \cdot (0.5B(D + RC))$, that is, search cost plus $B(D + RC)$.

Delete: We must search for the record, remove the record from the page, and write the

modified page back. We must also read and write all subsequent pages because all records that follow the deleted record must be moved up to compact the free space.² The cost is the same as for an insert, that is, search cost plus $B(D + RC)$. Given the rid of the record to delete, we can fetch the page containing the record directly.

Clustered Files

In a clustered file, extensive empirical study has shown that pages are usually at about 67 percent occupancy. Thus, the number of physical data pages is about $1.5B$, and we use this observation in the following analysis.

Scan: The cost of a scan is $1.5B(D + RC)$ because all data pages must be examined; this is similar to sorted files, with the obvious adjustment for the increased number of data pages. Note that our cost metric does not capture potential differences in cost due to sequential I/O. We would expect sorted files to be superior in this regard, although a clustered file using ISAM (rather than B+ trees) would be close.

\ Search with Equality Selection: We assume that the equality selection matches the search key (*age, sal*). We can locate the first page containing the desired record or records, should any qualifying records exist, in $\log F 1.5B$ steps, that is, by fetching all pages from the root to the appropriate leaf. In practice, the root page is likely to be in the buffer pool and we save an I/O, but we ignore this in our simplified analysis. Each step requires a disk I/O and two comparisons. Once the page is known, the first qualifying record can again be located by a binary search of the page at a cost of $Clog_2 R$. The cost is $D\log F 1.5B + Clog_2 R$, which is a significant improvement over searching even sorted files.

If several records qualify (e.g., "Find all employees aged 18"), they are guaranteed to be adjacent to each other due to the sorting on *age*, and so the cost of retrieving all such records is the cost of locating the first such record ($D\log F 1.5B + Clog_2 R$) plus the cost of reading all the qualifying records in sequential order.

Search with Range Selection: Again assuming that the range selection matches the composite key, the first record that satisfies the selection is located as it is for search with equality. Subsequently, data pages are sequentially retrieved (using the next and previous links at the leaf level) until a record is found that does not satisfy the range selection; this is similar to an equality search with many qualifying records.

Insert: To insert a record, we must first find the correct leaf page in the index, reading every page from root to leaf. Then, we must add the new record. Most of the time, the leaf page has sufficient space for the new record, and all we need to do is to write out the modified leaf page. Occasionally, the leaf is full and we need to retrieve and modify other pages, but this is sufficiently rare

Heap File with Un-clustered Tree Index

The number of leaf pages in an index depends on the size of a data entry. we will assume data entry in the index is a tenth the size of an employee data record which is typical. The number of leaf pages in the index is $0.1(L5B) = 0.15B$, if we take into account the 67 percent occupancy of index pages. Similarly, the number of data entries on a page $10(0.67R) = 6.7R$, taking into account the relative size and occupancy.

Scan: Consider Figure 8.1, which illustrates an unclustered index. To do a full scan of

the file of employee records, we can scan the leaf level of the index and for each data entry, fetch the corresponding data record from the underlying file, obtaining data records in the sort order (*age, sal*).

We can read all data entries at a cost of $0.15B(D + 6.7RC)$ I/Os. Now comes the expensive part: We have to fetch the employee record for each data entry in the index. The cost of fetching the employee records is one I/O per record, since the index is unclustered and each data entry on a leaf page of the index could point to a different page in the employee file. The cost of this step is $BR(D + C)$, which is prohibitively high. If we want the employee records in sorted order, we would be better off ignoring the index and scanning the employee file directly, and then sorting it. A simple rule of thumb is that a file can be sorted by a two-Pass algorithm in which each pass requires reading and writing the entire file. Thus, the I/O cost of sorting a file with B pages is $4B$, which is much less than the cost of using an un-clustered index.

Search with Range Selection: Again assuming that the range selection matches the composite key, the first record that satisfies the selection is located as it is for search with equality. Subsequently, data entries are sequentially retrieved (using the next and previous links at the leaf level of the index) until a data entry is found that does not satisfy the range selection. For each qualifying data entry, we incur one I/O to fetch the corresponding employee records. The cost can quickly become prohibitive as the number of records that satisfy the range selection increases. As a rule of thumb, if 10 percent of data records satisfy the selection condition, we are better off retrieving all employee records, sorting them, and then retaining those that satisfy the selection.

Insert: We must first insert the record in the employee heap file, at a cost of $2D + C$. In addition, we must insert the corresponding data entry in the index. Finding the right leaf page costs $D/0.15B + C/0.7R$, and writing it out after adding the new data entry costs another D .

Delete: We need to locate the data record in the employee file and the data entry in the index, and this search step costs $D/0.15B + C/0.7R + D$. Now, we need to write out the modified pages in the index and the data file, at a cost of $2D$.

Heap File With Un-clustered Hash Index

Scan: As for an unclustered tree index, all data entries can be retrieved in-expensively, at a cost of $0.125B(D + 8RC)$ I/Os. However, for each entry, we incur the additional cost of one I/O to fetch the corresponding data record; the cost of this step is $BR(D + C)$. This is prohibitively expensive, and further, results are unordered. So no one ever scans a hash index.

Search with Equality Selection: This operation is supported very efficiently for matching selections, that is, equality conditions are specified for each field in the composite search key (*age, sal*). The cost of identifying the page that contains qualifying data entries is H . Assuming that this bucket consists of just one page (i.e., no overflow pages), retrieving it costs D . If we assume that we find the data entry after scanning half the records on the page, the cost of scanning the page is $0.5(8R)C = 4RC$. Finally, we have to fetch the data record from the employee file, which is another D . The total cost is therefore $H + 2D + 4RC$, which is even lower than the cost for a tree index.

If several records qualify, they are *not* guaranteed to be adjacent to each other. The cost of retrieving all such records is the cost of locating the first qualifying data entry

$(H+D+4RC)$ plus one I/O per qualifying record. The cost of using an un-clustered index therefore depends heavily on the number of qualifying records.

Search with Range Selection: The hash structure offers no help, and the entire heap file of employee records must be scanned at a cost of $B(D+RC)$.

Insert: We must first insert the record in the employee heap file, at a cost of $2D+C$. In addition, the appropriate page in the index must be located, modified to insert a new data entry, and then written back. The additional cost is $H+2D+C$.

Delete: We need to locate the data record in the employee file and the data entry in the index; this search step costs $H+2D+4RC$. Now, we need to write out the modified pages in the index and the data file, at a cost of $2D$.

Clustered Index Organization

Design Examples Illustrating Clustered Indexes

To illustrate the use of a clustered index on age for a range query, consider the following example:

```
SELECT E.dno  
FROM Employees E  
WHERE E.age > 40
```

If we have a H+ tree index on age , we can use it to retrieve only tuples that satisfy the selection $E.age > 40$. Whether such an index is worthwhile depends first of all on the selectivity of the condition: what fraction of the employees are older than 40? If virtually everyone is older than 40, we gain little by using an index on age ; a sequential scan of the relation would do almost as well. However, suppose that only 10 percent of the employees are older than 40. Now, is an index useful? The answer depends on whether the index is clustered. If the index is un-clustered, we could have one page I/O per qualifying employee, and this could be more expensive than a sequential scan, even if only 10 percent of the employees qualify! On the other hand, a clustered B+ tree index on age requires only 10 percent of the I/Os for a sequential scan (ignoring the few I/Os needed to traverse from the root to the first retrieved leaf page and the I/Os for the relevant index leaf pages).

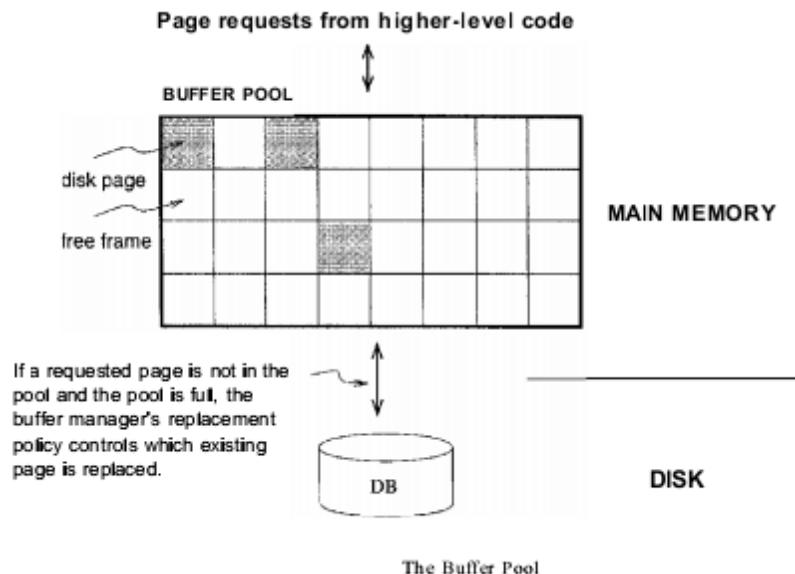
Let us consider whether an index on dna might suit our purposes better. We could use the index to retrieve all tuples, grouped by dna , and for each dna count the number of tuples with $age > 10$. (This strategy can be used with both hash and B+ tree indexes; we only require the tuples to be *grouped*, not necessarily *sorted*, by dna .) Again, the efficiency depends crucially on whether the index is clustered. If it is, this plan is likely to be the best if the condition on age is not very selective. (Even if we have a clustered index on age , if the condition on age is not selective, the cost of sorting qualifying

tuples on *dna* is likely to be high.) If the index is not clustered, we could perform one page I/O per tuple in Employees, and this plan would be terrible. Indeed, if the index is not clustered, the optimizer will choose the straightforward plan based on sorting on *dna*. Therefore, this query suggests that we build a clustered index on *dna* if the condition on *age* is not very selective. If the condition is very selective, we should consider building an index (not necessarily clustered) on *age* instead.

Clustering is also important for an index on a search key that does not include a candidate key, that is, an index in which several data entries can have the same key value. To illustrate this point, we present the following query:

```
SELECT      E.dno FROM Employees E WHERE      E.hobby='Stamps'
```

If many people collect stamps, retrieving tuples through an unclustered index on *hobby* can be very inefficient. It may be cheaper to simply scan the relation to retrieve all tuples and to apply the selection on-the-fly to the retrieved tuples. Therefore, if such a query is important, we should consider making the index on *hobby* a clustered index. On the other hand, if we assume that *eid* is a key for Employees, and replace the condition *E.hobby= 'Stamps'* by *E.eid=552*, we know that at most one Employees tuple will satisfy this selection condition. In this case, there is no advantage to making the index clustered.



Initially, the *pin_count* for every frame is set to 0, and the *dirty* bits are turned off. When a page is requested the buffer manager does the following:

1. Checks the buffer pool to see if some frame contains the requested page and, if so, increments the *pin_count* of that frame. If the page is not in the pool, the buffer

manager brings it in as follows:

Chooses a frame for replacement, using the replacement policy, and increments its *pin_count*.

If the *dirty* bit for the replacement frame is on, writes the page it contains to disk (that is, the disk copy of the page is overwritten with the contents of the frame).

Reads the requested page into the replacement frame.

2. Returns the (main memory) address of the frame containing the requested page to the requestor.

Incrementing *pirLco'llnt* is often called **pinning** the requested page in its frame. When the code that calls the buffer manager and requests the page subsequently calls the buffer manager and releases the page, the *pin_count* of the frame containing the requested page is decremented. This is called **unpinning** the page. If the requestor has modified the page, it also informs the buffer manager of this at the time that it unpins the page, and the *dirty* bit for the frame is set.

The best-known replacement policy is least recently used (LRU). This can be implemented in the buffer manager using a queue of pointers to frames with *pin_count* 0. A frame is added to the end of the queue when it becomes a candidate for replacement (that is, when the *p'irLco'unt* goes to 0). The page chosen for replacement is the one in the frame at the head of the queue.

A variant of LRU, called clock replacement, has similar behavior but less overhead. The idea is to choose a page for replacement using a *current* variable that takes on values 1 through *N*, where *N* is the number of buffer frames, in circular order. One can think of the frames being arranged in a circle, like a clock's face, and *current* as a clock hand moving across the face. To approximate LRU behavior, each frame also has an associated *referenced* bit, which is turned on when the page *pln~count* goes to 0.

The LRU and clock policies are not always the best replacement strategies for a database system, particularly if many user requests require sequential scans of the data. Consider the following illustrative situation. Suppose the buffer pool has 4 frames, and the file to be scanned has 10 or fewer pages. Assuming, for simplicity, that there are no competing requests for pages, only the first scan of the file does any I/O. Page requests in subsequent scans always find the desired page in the buffer pool. On the other hand, suppose that the file to be scanned has 11 pages (which is one more than the number of available pages in the buffer pool). Using LRU, every scan of the file will result in reading every page of the file! In this situation, called sequential flooding, LRU is the *worst* possible replacement strategy.

Other replacement policies include first in first out (FIFO) and most recently used (MRU), which also entail overhead similar to LRU, and random, among others. The details of these policies should be evident from their names and the preceding discussion of LRU and clock.

One approach to mapping the database to files is to use several files and to store records of only one fixed length in any given file. An alternative is to structure our files so that we can accommodate multiple lengths for records; however, files of fixed length records are easier to implement than are files of variable length records. Many of the techniques used for the former can be applied to the variable length case. Thus we begin by considering a file of fixed length records.

Fixed Length Records

As an example, let us consider a file of *account* records for our *bank* database.
.Each record of this file is defined (in pseudo code) as:

```
type deposit = record  
    Account _number char (10);  
    Branch_name char (22);  
    Balance numeric (12, 2);  
end
```

If we assume that each character occupies 1 byte and that numeric (12, 2) occupies 8 bytes, our account record is 40 bytes long. A simple approach is to use the first 40 bytes for the first record, the next 40 bytes for the second record and so on.

However there are two main problems with this simple approach.

It is difficult to delete a record from this structure. The space occupied by the record to be deleted must be filled with some other record of the file, or we must have a way of marking deleted records so that they can be ignored.

Unless the block size happens to be a multiple of 40(which is unlikely), some records will cross block boundaries. That is, part of the record will be stored in one block and part in another. It would thus require two block accesses to read or write such a record.

When a record is deleted, we could move the record that came after it into the space formerly occupied by the deleted record, and so on, until every record following the deleted record has been moved ahead. Such an approach requires moving a large

number of records. It might be easier simply to move the final record of the file into the space occupied by the deleted record.

It is undesirable to move records to occupy the space freed by the deleted record, since doing so requires additional block accesses. Since insertions tend to be more frequent than deletions, it is acceptable to leave open the space occupied by the deleted record, and to wait for a subsequent insertion before reusing the space. A simple marker on the deleted record is not sufficient, since it is hard to find this available space when an insertion is being done. Thus we need to introduce an additional structure.

At the beginning of the file, we allocate a certain number of bytes as a file header. The header will contain a variety of information about the file.

For now, all we need to store there is the address of the first record whose contents are deleted. We use this first record to store the address of the second available record, and so on. Intuitively we can think of these stored addresses as pointers, since they point to the location of a record. The deleted records thus form a linked list, which is often referred to as a free list.

On insertion of a new record, we use the record pointed to by the header. We change the header pointer to point to the next available record. If no space is available, we add the new record to the end of the file.

Insertion and deletion for files of fixed length records are simple to implement, because the space made available by a deleted record is exactly the space needed to insert a record. If we allow records of variable length in a file, this match no longer holds. An inserted record may not fit in the space left free by a deleted record, or it may fill only part of that space.

Variable Length Records

Variable length records arise in the database systems in several ways.

- Storage of multiple record types in a file
 - Record types that allow variable lengths for one or more fields.
 - Record types that allow repeating fields, such as arrays or multisets.
- Different techniques for implementing variable length records exist.

The **slotted page structure** is commonly used for organizing records within a

block. There is a header at the beginning of each block, containing the following information.

The number of record entries in the header.

The end of free space in the block

An array whose entries contain the location and size of each record.

The actual records are allocated contiguously in the block, starting from the end of the block. The free space in the block is contiguous, between the final entry in the header array, and the first record. If a record is inserted, space is allocated for it at the end of free space, and an entry containing its size and location is added to the header.

If a record is deleted, the space that it occupies is freed, and its entry is set to deleted (its size is set to -1, for example). Further the records in the block before the deleted records are moved, so that the free space created by the deletion gets occupied, and all free space is again between the final entry in the header array and the first record. The end of free space pointer in the header is appropriately updated as well. Records can be grown or shrunk by similar techniques, as long as there is space in the block. The cost of moving the records is not too high, since the size of a block is limited: a typical value is 4 kilobytes.

The slotted page structure requires that there be no pointers that point directly to records. Instead, pointers must point to the entry in the header that contains the actual location of the record. This level of indirection allows records to be moved to prevent fragmentation of space inside a block, while supporting indirect pointers to the record.

Databases often store data that can be much larger than a disk block .For instance, an image or an audio recording may be multiple megabytes in size, while a video object may be multiple gigabytes in size. Recall that SQL supports the types blob and clob, which store binary and character large objects.

Most relational databases restrict the size of a record to be no larger than the size of a block, to simplify buffer management and free space management. Large objects are often stored in a special file (or collection of files) instead of being stored with the other (short) attributes of records in which they occur. Large objects are often represented using B+ - tree file organizations.

Organization of Records in a File

So far, we have studied how records are represented in a file structure. A relation

is a set of records. Given a set of records; the next question is how to organize them in a file. Several of the possible ways of organizing records in files are:

Heap File Organization

Any record can be placed anywhere in the file where there is space for a record. There is no ordering of records. Typically, there is a single file for each relation.

Sequential File Organization

Records are stored in sequential order, according to the value of a “search key” of each record.

Hashing File Organization

A hash function is computed on some other attribute of each record. The result of the hash function specifies in which block of the file the record should be placed.

4.6.4.4 Lecture-4

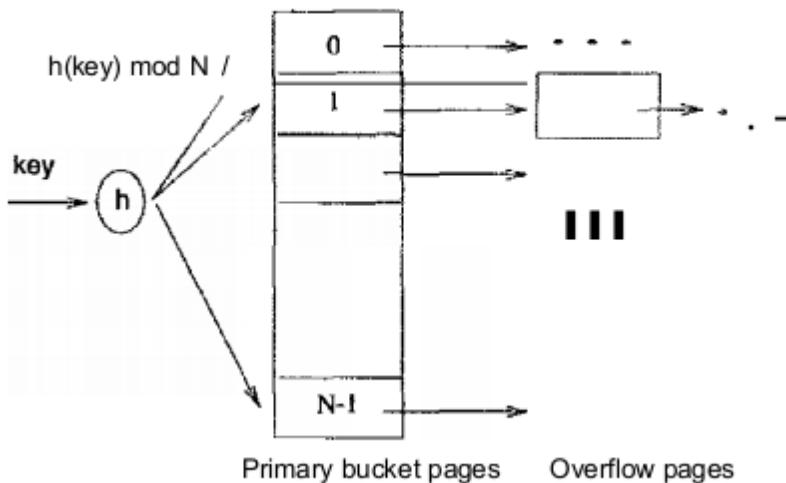
HASH-BASED INDEXING

STATIC HASHING

The Static Hashing scheme is the pages containing the data can be viewed as a collection of buckets, with one primary page and possibly additional overflow pages per bucket. A file consists of buckets a through $N - 1$, with one primary page per bucket initially. Buckets contain *data entries*,

To search for a data entry, we apply a hash function h to identify the bucket to which it belongs and then search this bucket. To speed the search of a bucket, we can maintain data entries in sorted order by search key value; in this chapter, we do not sort entries, and the order of entries within a bucket has no significance. To insert a data entry, we use the hash function to identify the correct bucket and then put the data entry there. If there is no space for this data entry, we allocate a new *overflow* page, put the data entry on this page, and add the page to the overflow chain of the bucket. To delete a data entry, we use the hashing function to identify the correct bucket, locate the data entry by searching the bucket, and then remove it. If this data entry is the last in an overflow page, the overflow page is removed from the overflow chain of the bucket and added to a list of *free pages*.

The hash function is an important component of the hashing approach. It must distribute values in the domain of the search field uniformly over the collection of buckets. If we have N buckets, numbered 0 through $N-1$, a hash function h of the form $h(\text{value}) = (a * \text{value} + b)$ works well in practice. (The bucket identified is $h(\text{value}) \bmod N$.) The constants a and b can be chosen to 'tune' the hash function.

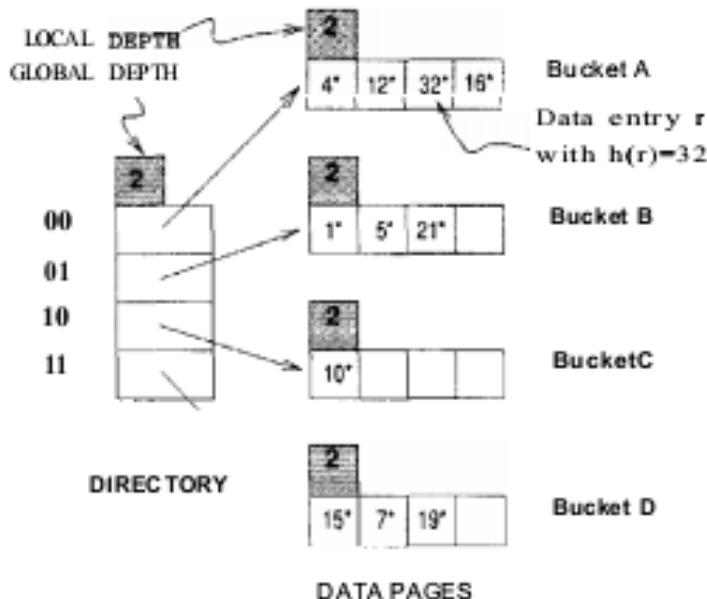


EXTENDIBLE HASHING

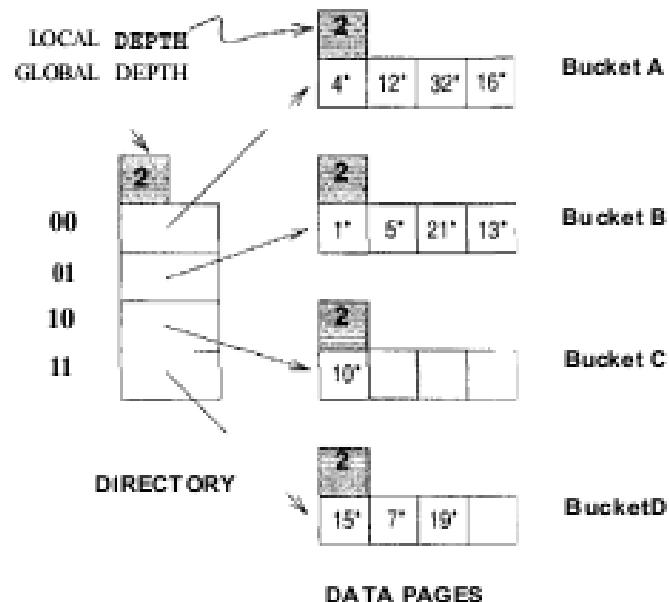
To understand Extendible Hashing, let us begin by considering a Static Hashing file. If we have to insert a new data entry into a full bucket, we need to add an overflow page. If we do not want to add overflow pages, one solution is to reorganize the file at this point by doubling the number of buckets and redistributing the entries across the new set of buckets. This solution suffers from one major defect—the entire file has to be read, and twice (h') many pages have to be written to achieve the reorganization. This problem, however, can be overcome by a simple idea: Use a **directory** of pointers to buckets, and double the size of the number of buckets by doubling just the directory and splitting *only* the bucket that overflowed.

To understand the idea, consider the sample file shown in Figure 11.2. The directory consists of an array of size 4, with each element being a pointer to a bucket.. (The *global depth* and *local depth* fields are discussed shortly, ignore them for now.) To locate a data entry, we apply a hash function to the search field and take the last 2 bits of its binary representation to get a number between 0 and 3. The pointer in this array position gives us the desired bucket.; we assume that each bucket can hold four data entries. Therefore, to locate a data entry with hash value 5 (binary 101), we look at directory element 01 and follow the pointer to the data page (bucket B in the figure).

To insert a data entry, we search to find the appropriate bucket.. For example, to insert a data entry with hash value 13 (denoted as 13*), we examine directory element 01 and go to the page containing data entries 1*, 5*, and 21 *. Since



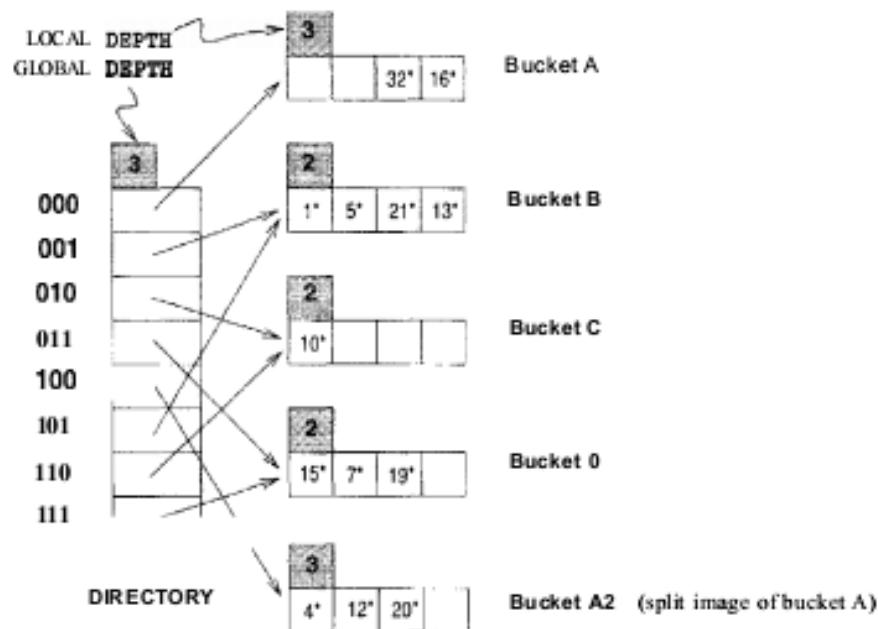
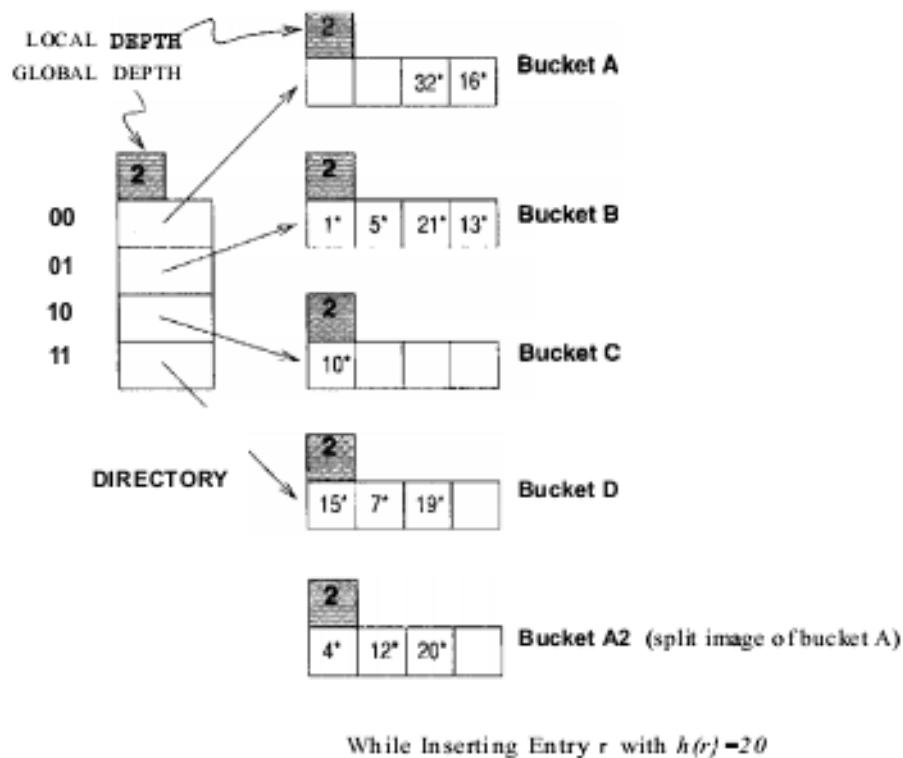
Example of an Extendible Hashed File



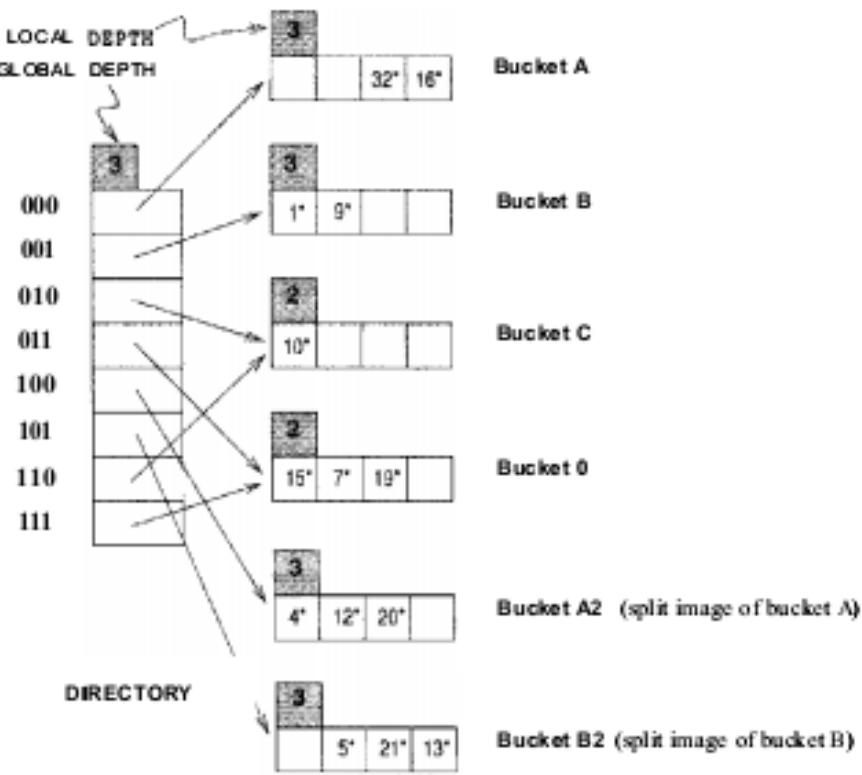
After Inserting Entry T with $h(T) = 13$

Next, let us consider insertion of a data entry into a full bucket. The essence of the Extendible Hashing idea lies in how we deal with this case. Consider the insertion of data entry 20* (binary 10100). Looking at directory element 00, we are led to bucket A, which is already full. We first **split** the bucket by allocating a new bucket¹ and redistributing the contents (including the new entry to be inserted) across the old

bucket and its 'split image.' To redistribute entries across the old bucket and its split image, we consider the last *three* bits of $h(T)$; the last two bits are 00, indicating a data entry that belongs to one of these two buckets, and the third bit discriminates between these buckets. The redistribution of entries is illustrated in Figure



After Inserting Entry r with $h(r) = 20$



After Inserting Entry r with $h(r) = 9$

We observe that the basic technique used in Extendible Hashing is to treat the result of applying a hash function h a "a binary number and interpret the last d bits, where d depends on the size of the directory, as an offset into the directory. In our example, d is originally 2 because we only have four buckets; after the split, d becomes 3 because we now have eight buckets. A corollary is that, when distributing entries across a bucket and its split image, we should do so on the basis of the d th bit. (Note how entries are redistributed in our example; see Figure 11.5.) The number d , called the **global depth** of the hashed file, is kept as part of the header of the file. It is used every time we need to locate a data entry.

An important point that arises is whether splitting a bucket necessitates a directory doubling. Consider our example, as shown in Figure 11.5. If we now insert 9^* , it belongs in bucket B; this bucket is already full. We can deal with this situation by splitting the bucket and using directory elements 001 and 101 to point to the bucket and its split image. Hence, a bucket split does not necessarily require a directory doubling. However, if either bucket A or A2 grows full and an insert then forces a bucket split, we are forced to double the directory again.

To differentiate between these cases and determine whether a directory doubling is needed, we maintain a **local depth** for each bucket. If a bucket whose local depth is equal to the global depth is split, the directory must be doubled. Going back to the example, when we inserted 9^* into the index shown in Figure 11.5, it belonged to bucket B with local depth 2, whereas the global depth was 3. Even though the bucket was split, the directory did not have to be doubled. Buckets A and A2, on the other hand, have local depth equal to the global depth, and, if they grow full and are split, the directory

must then be doubled.

Initially, all local depths are equal to the global depth (which is the number of bits needed to express the total number of buckets). We increment the global depth by 1 each time the directory doubles, of course. Also, whenever a bucket is split (whether or not the split leads to a directory doubling), we increment by 1 the local depth of the split bucket and assign this same (incremented) local depth to its (newly created) split image. Intuitively, if a bucket has local depth l , the hash values of data entries in it agree on the last l bits; further, no data entry in any other bucket of the file has a hash value with the same last l bits. A total of 2^{dl} directory elements point to a bucket with local depth l ; if $d = l$, exactly one directory element points to the bucket and splitting such a bucket requires directory doubling.

4.6.4.5 Lecture-5

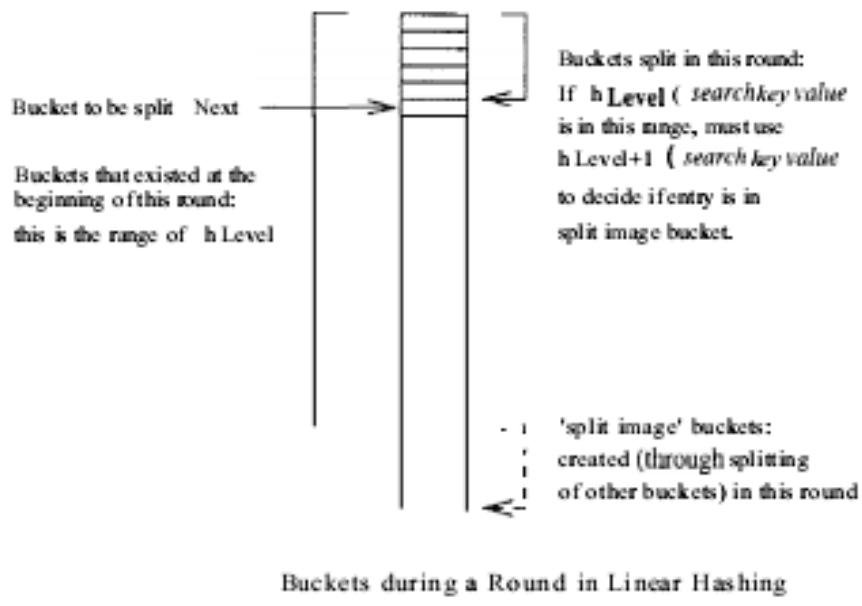
SINGLE AND MULTI-LEVEL INDEXES

LINEAR HASHING

Linear Hashing is a dynamic hashing technique, like Extendible Hashing, adjusting gracefully to inserts and deletes. In contrast to Extendible Hashing, it does not require a directory, deals naturally with collisions, and offers a lot of flexibility with respect to the timing of bucket splits (allowing us to trade off slightly greater overflow chains for higher average space utilization). If the data distribution is very skewed, however, overflow chains could cause Linear Hashing performance to be worse than that of Extendible Hashing.

The scheme utilizes a *family* of hash functions h_a, h_1, h_2, \dots , with the property that each function's range is twice that of its predecessor. That is, if h_i maps a data entry into one of M buckets, h_{i+1} maps a data entry into one of $2M$ buckets. Such a family is typically obtained by choosing a hash function h and an initial number N of buckets, and defining $h_i(\text{value}) = h(\text{value}) \bmod (2^i N)$. If N is chosen to be a power of 2, then we apply h and look at the last d_i bits; d_0 is the number of bits needed to represent N , and $d_i = d_0 + i$. Typically we choose h to be a function that maps a data entry to some integer. Suppose that we set the initial number N of buckets to be 32. In this case d_0 is 5, and h_a is therefore $h \bmod 32$, that is, a number in the range 0 to 31. The value of h_1 is $d_0 + 1 = 6$, and h_1 is $h \bmod (2 * 32)$, that is, a number in the range 0 to 127. Then h_2 yields a number in the range 0 to 127, and so on.

The idea is best understood in terms of **rounds** of splitting. During round number *Level*, only hash functions h_{Level} and $h_{\text{Level}+1}$ are in use. The buckets in the file at the beginning of the round are split, one by one from the first to the last bucket, thereby doubling the number of buckets. At any given point within a round, therefore, we have buckets that have been split, buckets that are yet to be split, and buckets created by splits in this round.

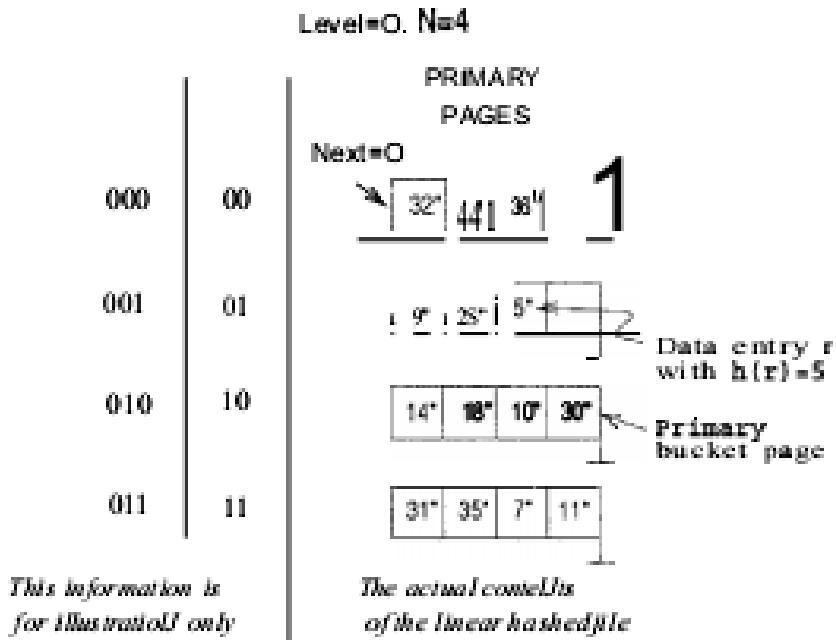


Unlike Extendible Hashing, when an insert triggers a split, the bucket into which the data entry is inserted is not necessarily the bucket that is split. An overflow page is added to store the newly inserted data entry (which triggered the split), as in Static Hashing. However, since the bucket to split is chosen in round-robin fashion, eventually all buckets are split, thereby redistributing the data entries in overflow chains before the chains get to be more than one or two pages long.

We now describe Linear Hashing in more detail. A counter $Level$ is used to indicate the current round number and is initialized to 0. The bucket to split is denoted by $Next$ and is initially bucket 0 (the first bucket). We denote the number of buckets in the file at the beginning of round $Level$ by N_{Level} . We can easily verify that $N_{Level} = N * 2^{Level}$. Let the number of buckets at the beginning of round 0, denoted by No , be N . We show a small linear hashed file in Figure 11.8. Each bucket can hold four data entries, and the file initially contains four buckets, as shown in the figure.

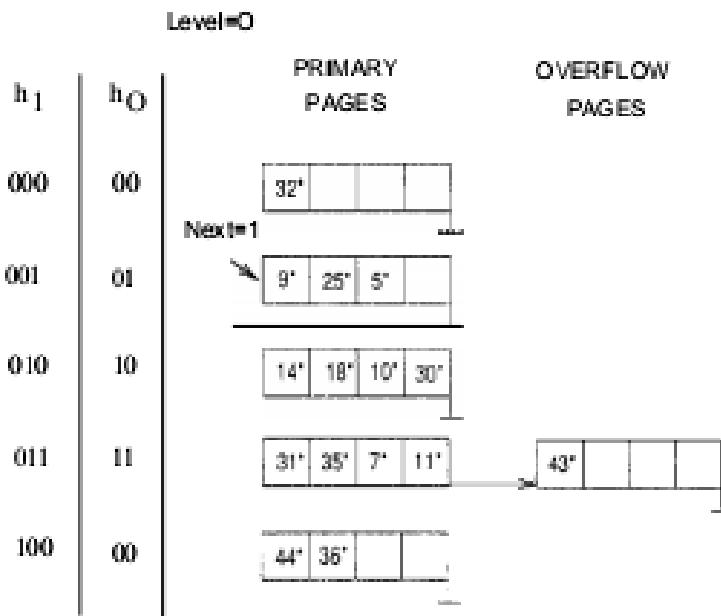
We have considerable flexibility in how to trigger a split, thanks to the use of overflow pages. We can split whenever a new overflow page is added, or we can impose additional conditions based all conditions such as space utilization. For our examples, a split is 'triggered' when inserting a new data entry causes the creation of an Overflow page.

Whenever a split is triggered the $Next$ bucket is split, and hash function $hLevel+1$ redistributes entries between this bucket (say bucket number b) and its split image; the split image is therefore bucket number $b + NLevel$. After splitting a bucket, the value of $Next$ is incremented by 1. In the example file, insertion of data entry 43* triggers a split. The file after completing the insertion.

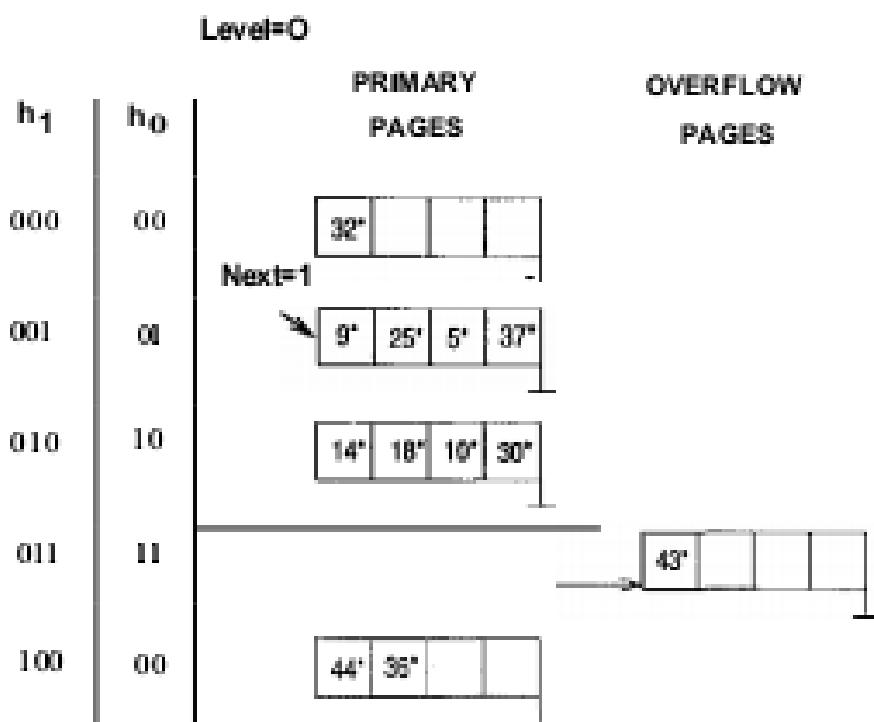


Example of a Linear Hashed File

At any time in the middle of a round *Level*, all buckets above bucket *Next* have been split, and the file contains buckets that are their split images, as illustrated in Figure 11.7. Buckets *Next* through *NLevl* have not yet been split. If we use *hLevel* on a data entry and obtain a number *b* in the range *Next* through *NLevel*, the data entry belongs to bucket *b*. For example, $h_0(18)$ is 2 (binary 10); since this value is between the current values of *Ne:r:t* (= 1) and *N₁* (=:-: 4), this bucket has not been split. However, if we obtain a number *b* in the range 0 through *Next*, the data entry may be in this bucket or in its split image (which is bucket number *b+NLevel*); we have to use *hLevel+1* to determine to which of these two buckets the data entry belongs. In other words, we have to look at one more bit of the data entry's hash value. For example, $h_0(32)$ and $h_0(44)$ are both a (binary 00). Since *Next* is currently equal to 1, which indicates a bucket that has been split, we have to apply *h1*. We have $h_1(32) = 0$ (binary 000) and $h_1(44) = 4$ (binary 100). Therefore, 32 belongs in bucket A and 44 belongs in its split image, bucket A2.



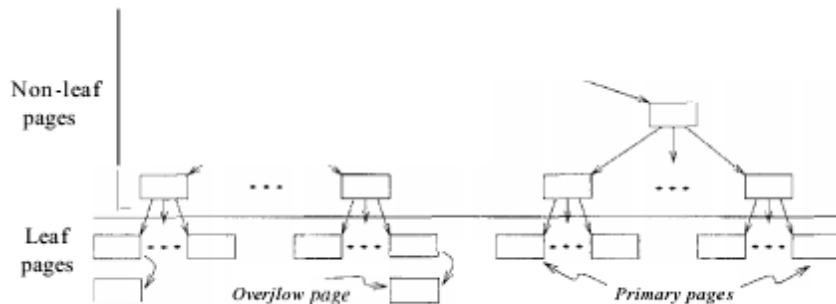
After Inserting Record r with $h(r) = 43$



After Inserting Record r with $h(r) = 37$

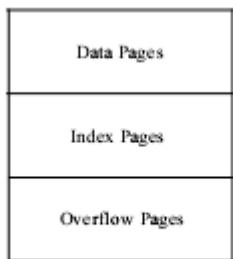
INDEXED SEQUENTIAL ACCESS METHOD (ISAM)

The data entries of the ISAM index are in the leaf pages of the tree and additional *overflow* pages chained to some leaf page. Database systems carefully organize the layout of pages so that page boundaries correspond closely to the physical characteristics of the underlying storage device. The ISAM structure is completely static (except for the overflow pages, of which it is hoped, there will be few) and facilitates such low-level optimizations.



Each tree node is a disk page, and all the data resides in the leaf pages. This corresponds to an index that uses Alternative (1) for data entries, we can create an index with Alternative

(2) by storing the data records in a separate file and storing (key, rid) pairs in the leaf pages of the ISAM index. When the file is created, all leaf pages are allocated sequentially and sorted on the search key value. (If Alternative (2) or (3) is used, the data records are created and sorted before allocating the leaf pages of the ISAM index.) The non-leaf level pages are then allocated. If there are several inserts to the file subsequently, so that more entries are inserted into a leaf than will fit onto a single page, additional pages are needed because the index structure is static. These additional pages are allocated from an overflow area. The allocation of pages



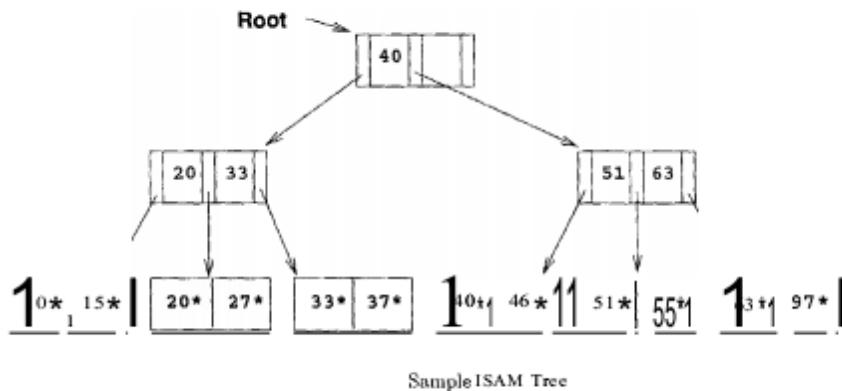
Page Allocation in ISAM

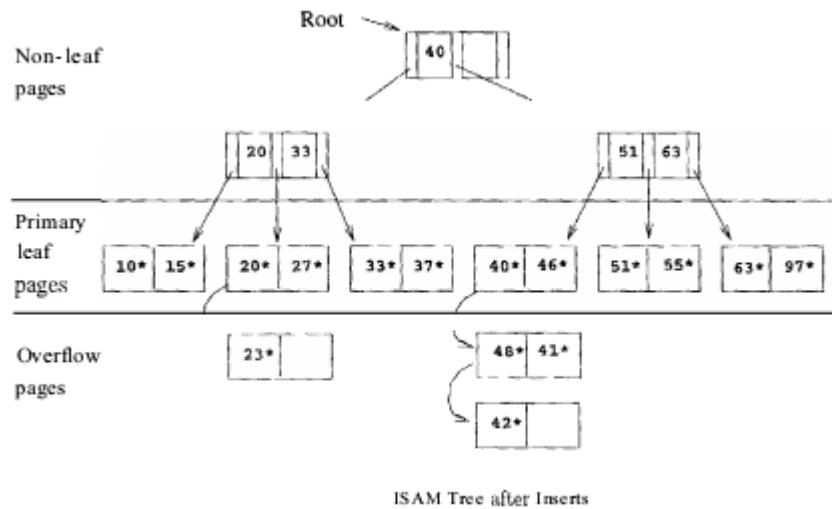
The basic operations of insertion, deletion, and search are all quite straight forward. In an equality selection search, we start at the root node and determine which subtree to search by comparing the value in the search field of the given record with the key values in the node. (The search algorithm is identical to that for a B+ tree; we present this algorithm in more detail later.) For a range query, the starting point in the data (or leaf) level is determined similarly, and data pages are then retrieved sequentially. For

inserts and deletes, the appropriate page is determined as for a search, and the record is inserted or deleted with overflow pages added if necessary.

The following example illustrates the ISAM index structure. Consider the tree shown in Figure 10.5. All searches begin at the root. For example, to locate a record with the key value 27, we start at the root and follow the left pointer, since $27 < 40$. We then follow the middle pointer, since $20 \leq 27 < 33$. For a range search, we find the first qualifying data entry as for an equality selection and then retrieve primary leaf pages sequentially (also retrieving overflow pages as needed by following pointers from the primary pages). The primary leaf pages are assumed to be allocated sequentially this assumption is reasonable because the number of such pages is known when the tree is created and does not change subsequently under inserts and deletes-and so no 'next leaf page' pointers are needed.

We assume that each leaf page can contain two entries. If we now insert a record with key value 23, the entry 23^* belongs in the second data page, which already contains 20^* and 27^* and has no more space. We deal with this situation by adding an *overflow* page and putting 23^* in the overflow page. Chains of overflow pages can easily develop. For instance, inserting 48^* , 41^* , and 42^* leads to an overflow chain of two pages. The tree of Figure 10.5 with all these insertions is shown in Figure





Overflow Pages, Locking Considerations

Note that, once the ISAM file is created, inserts and deletes affect only the contents of leaf pages. A consequence of this design is that long overflow chains could develop if a number of inserts are made to the same leaf. These chains can significantly affect the time to retrieve a record because the overflow chain has to be searched as well when the search gets to this leaf. (Although data in the overflow chain can be kept sorted, it usually is not, to make inserts fast.) To alleviate this problem, the tree is initially created so that about 20 percent of each page is free. However, once the free space is filled in with inserted records, unless space is freed again through deletes, overflow chains can be eliminated only by a complete reorganization of the file.

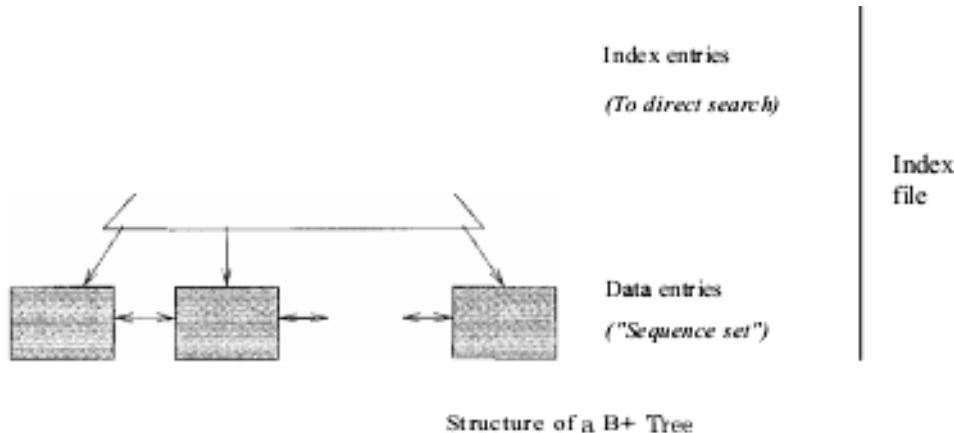
The fact that only leaf pages are modified also has an important advantage with respect to concurrent access. When a page is accessed, it is typically 'locked' by the requestor to ensure that it is not concurrently modified by other users of the page. To modify a page, it must be locked in 'exclusive' mode, which is permitted only when no one else holds a lock on the page. Locking can lead to queues of users (*transactions*, to be more precise) waiting to get access to a page. Queues can be a significant performance bottleneck, especially for heavily accessed pages near the root of an index structure. In the ISAM structure, since we know that index-level pages are never modified, we can safely omit the locking step. Not locking index-level pages is an important advantage of ISAM over a dynamic structure like a B+ tree. If the data distribution and size are relatively static, which means overflow chains are rare, ISAM might be preferable to B+ trees due to this advantage.

4.6.4.6 Lecture-6

DYNAMIC MULTILEVEL INDEXING USING B-TREE

B+ TREES: A DYNAMIC INDEX STRUCTURE

A static structure such as the ISAM index suffers from the problem that long overflow chains can develop as the file grows, leading to poor performance. This problem motivated the development of more flexible, dynamic structures that adjust gracefully to inserts and deletes. The B+ tree search structure, which is widely, is a balanced tree in which the internal nodes direct the search and the leaf nodes contain the data entries. Since the tree structure grows and shrinks dynamically, it is not feasible to allocate the leaf pages sequentially as in ISAM, where the set of primary leaf pages was static. To retrieve all leaf pages efficiently, we have to link them using page pointers. By organizing them into a doubly linked list, we can easily traverse the sequence of leaf pages (sometimes called the sequence set) in either direction.



The following are some of the main characteristics of a B+ tree:

Operations (insert, delete) on the tree keep it balanced.

A minimum occupancy of 50^{the} percent is guaranteed for each node except the root if the deletion algorithm discussed in Section 10.6 is implemented. However, deletion is often implemented by simply locating the data entry and removing it, without adjusting the tree needed to guarantee the 50^{often} percent occupancy, because files typically grow rather than shrink.

Searching for a record requires just a traversal from the root to the appropriate leaf. Vie refer to the length of a path from the root to a leaf any leaf, because the tree is balanced as the height of the tree. For example, a tree with only a leaf level and a single index level, such as the tree shown in Figure 10.9, has height 1, and a tree that h&'3 only the root node has height 0. Because of high fan-out, the height of a B+ tree is rarely more than 3 or 4.

SEARCH

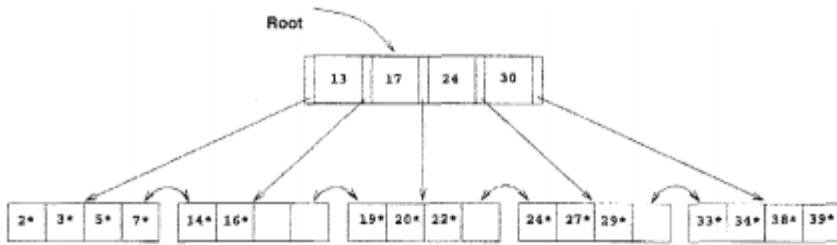
```

fune find (search key value K) returns nodepointer
  // Given a search key value, finds its leaf node
  return tree_search(root, K);                                // searches from root
endfune

fune tree_search (nodepointer, search key value K) returns nodepointer
  // Searches tree for entry
  if *nodepointer is a leaf, return nodepointer;
  else,
    if  $K < K_1$  then return tree_search(P0, K);
    else,
      if  $K \geq K_m$  then return tree_search(Pm, K); // m = # entries
      else,
        find i such that  $K_i \leq K < K_{i+1}$ 
        return tree_search(Pi, K)
endfune

```

Algorithm for Search



Example of a B+ Tree, Order d=2

4.6.4.7 Lecture-7

INDEX ON MULTIPLE KEYS

INSERT

The algorithm for insertion takes an entry, finds the leaf node where it belongs, and inserts it there. Pseudo code for the B+ tree insertion algorithm is given in Figure HUG. The basic idea behind the algorithm is that we recursively insert the entry by calling the insert algorithm on the appropriate child node. Usually, this procedure results in going down to the leaf node where the entry belongs, placing the entry there, and returning all the way back to the root node. Occasionally a node is full and it must be split. When the node is split, an entry pointing to the node created by the split must be inserted into its parent; this entry is pointed to by the pointer variable *newchildentry*. If the (old) root is split, a new root node is created and the height of the tree increase by 1

To illustrate insertion, let us continue with the sample tree shown in Figure . If we insert entry 8*, it belongs in the left-most leaf, which is already full. This insertion causes a split of the leaf page; the split pages are shown in Figure . The tree must now be adjusted to take the new leaf page into account, so we insert an entry consisting of the pair (5, *pointer to new page*) into the parent node. Note how the key 5, which discriminates between the split leaf page and its newly created sibling, is 'copied up.'

//*T*e cannot just 'push up' 5, because every data entry must appear in a leaf page.

Since the parent node is also full, another split occurs. In general we have to split a non-leaf node when it is full, containing $2d$ keys and $2d + 1$ pointers, and we have to add another index entry to account for a child split. We now have $2d+1$ keys and $2d+2$ pointers, yielding two minimally full non-leaf nodes, each containing d keys and $d + 1$ pointers, and an extra key, which we choose to be the 'middle' key. This key and a pointer to the second non-leaf node constitute an index entry that must be inserted into the parent of the split non-leaf node. The middle key is thus 'pushed up' the tree, in contrast to the case for a split of a leaf page.

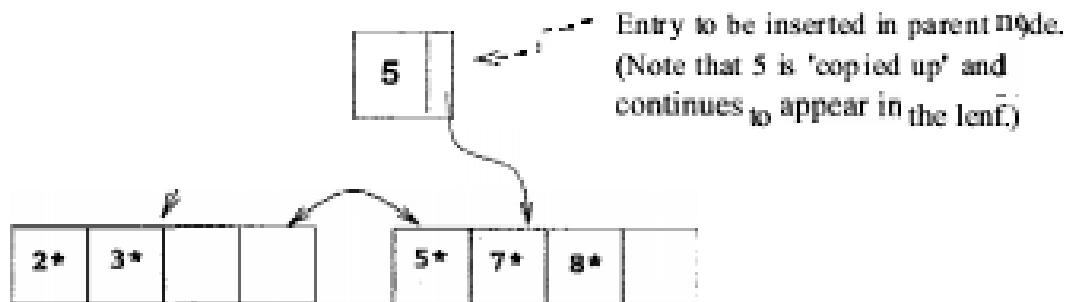
```

proc insetT (nodepointer1, entry, newchildentry)
// Insert entry into subtree with Root '*nodepointer'; degree is d;
//'newchildentry' null initially, and null on return unless child is split

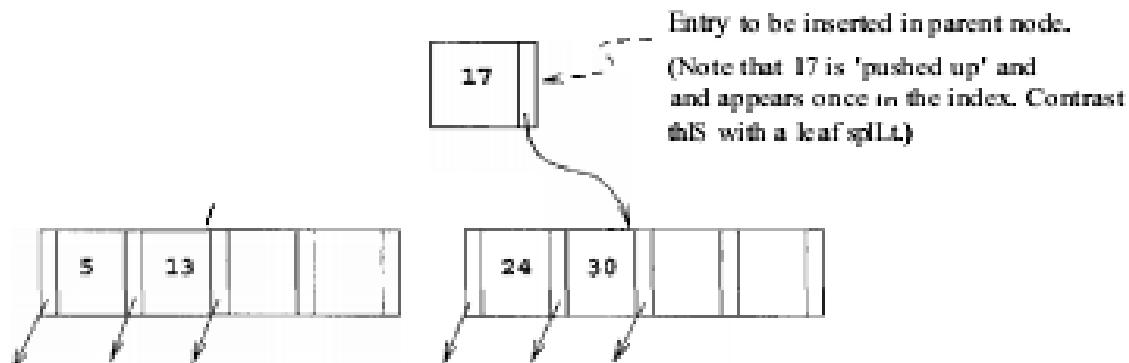
if *nodepointer is a non-leaf node, say N,
    find i such that  $K_i \leq$  entry's key value  $< K_{i+1}$ ; // choose subtree
    insert(Pi, entry, newchildentry); // recursively, insert entry
    if newchildentry is null, return; // usual case; didn't split child
    else, // we split child, must insert *newchildentry in N
        if N has space, // usual case
            put *newchildentry on it, set newchildentry to null, return;
        else, // note difference wrt splitting of leaf page!
            split N; //  $2d + 1$  key values and  $2d + 2$  nodepointers
            first d key values and d + 1 nodepointers stay,
            last d keys and d + 1 pointers move to new node, N2;
            // *newchildentry set to guide searches between N and N2
            newchildentry = & ((smallest key value on N2,
                pointer to N2));
            if N is the root, // root node was just split
                create new node with (pointer to N, *newchildentry);
                make the tree's root-node pointer point to the new node;
                return;

if *nodepointer is a leaf node, say L,
    if L has space, // usual case
        put entry on it, set newchildentry to null, and return;
    else, // once in a while, the leaf is full
        split L; first d entries stay, rest move to brand new node L2;
        newchildentry = & ((smallest key value on L2, pointer to L2));
        set sibling pointers in L and L2;
        return;
endproc

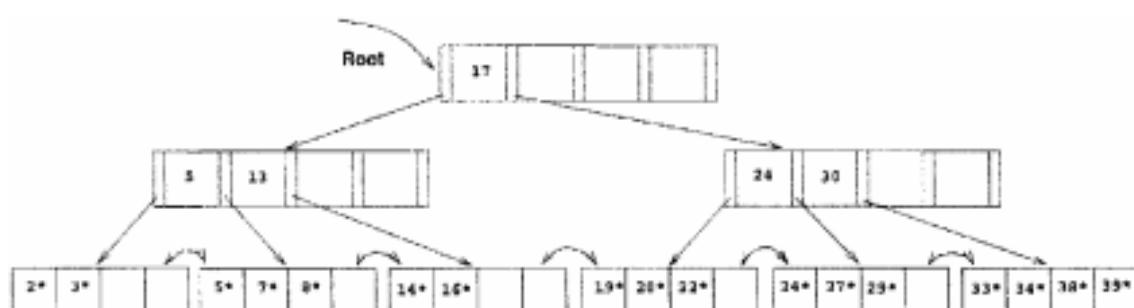
```



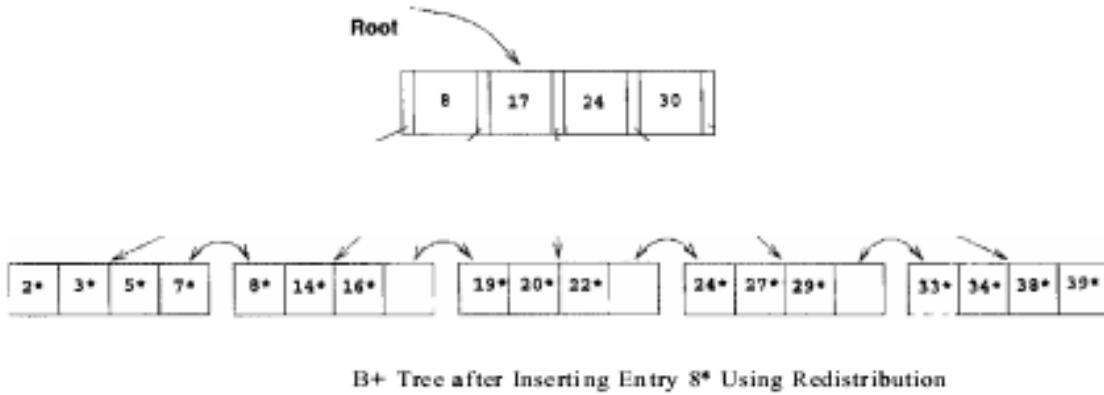
Split Leaf Pages during Insert of Entry 8^*



Split Index Pages during Insert of Entry 8^*



B+ Tree after Inserting Entry 8^*



DELETE

The algorithm for deletion takes an entry, finds the leaf node where it belongs, and deletes it. Pseudo code for the B+ tree deletion algorithm is given in Figure 10.15. The basic idea behind the algorithm is that we recursively delete the entry by calling the delete algorithm on the appropriate child node. We usually go down to the leaf node where the entry belongs, remove the entry from there, and return all the way back to the root node. Occasionally a node is at minimum occupancy before the deletion, and the deletion causes it to go below the occupancy threshold. When this happens, we must either redistribute entries from an adjacent sibling or merge the node with a sibling to maintain minimum occupancy. If entries are redistributed between two nodes, their parent node must be updated to reflect this; the key value in the index entry pointing to the second node must be changed to be the lowest search key in the second node. If two nodes are merged, their parent must be updated to reflect this by deleting the index entry for the second node; this index entry is pointed to by the pointer variable *oldchildentry* when the delete call returns to the parent node. If the last entry in the root node is deleted in this manner because one of its children was deleted, the height of the tree decreases by 1.

To illustrate deletion, let us consider the sample tree shown in Figure 10.13. To delete entry 19*, we simply remove it from the leaf page on which it appears, and we are done because the leaf still contains two entries. If we subsequently delete 20*, however, the leaf contains only one entry after the deletion. The (only) sibling of the leaf node that contained 20* has three entries, and we can therefore deal with the situation by redistribution; we move entry 24* to the leaf page that contained 20* and copy up the new splitting key (27, which is the new low key value of the leaf from which we borrowed 24*) into the parent. This process is illustrated in Figure

Suppose that we now delete entry 24*. The affected leaf contains only one entry (22*) after the deletion, and the (only) sibling contains just two entries (27* and 29*). Therefore, we cannot redistribute entries. However, these two leaf nodes together contain only three entries and can be merged. While merging, we can 'tos::' the entry ((27, *pointer* to second leaf page)) in the parent, which pointed to the second leaf page, because the second leaf page is empty after the merge and can be discarded. The right subtree of Figure 10.16 after this step in the deletion of entry 2!1 * is shown in Figure

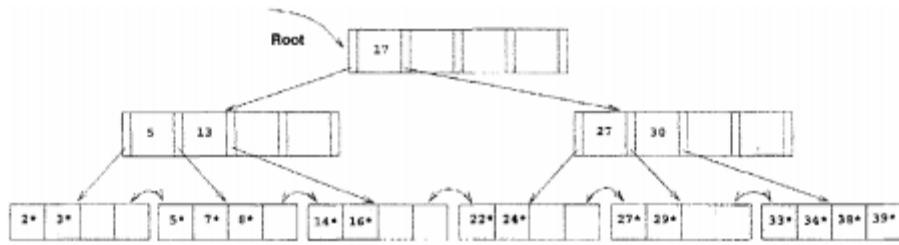
```

proc delete (parentpointer, nodepointer, entry, oldchildentry)
// Deletes entry from subtree with root *nodepointer'; degree is d;
// 'oldchildentry' null initially, and null upon return unless child deleted
if *nodepointer is a non-leaf node, say N,
    find i such that  $K_i \leq$  entry's key value <  $K_{i+1}$ ; // choose subtree
    delete(nodepointer, Pi, entry, oldchildentry); // recursive delete
    if oldchildentry is null, return; // usual case: child not deleted
    else, // we discarded child node (see discussion)
        remove *oldchildentry from N, // next, check for underflow
        if N has entries to spare, // usual case
            set oldchildentry to null, return; // delete doesn't go further
        else, // note difference wrt merging of leaf pages!
            get a sibling S of N; // parentpointer arg used to find S
            if S has extra entries,
                redistribute evenly between N and S through parent;
                set oldchildentry to null, return;
            else, merge N and S // call node on rhs M
                oldchildentry = & (current entry in parent for M);
                pull splitting key from parent down into node on left;
                move all entries from M to node on left;
                discard empty node M, return;

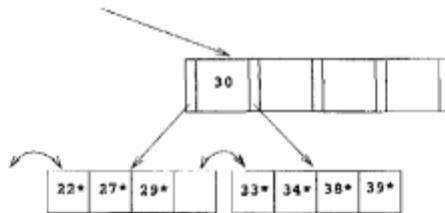
if *nodepointer is a leaf node, say L,
    if L has entries to spare, // usual case
        remove entry, set oldchildentry to null, and return;
    else, // once in a while, the leaf becomes underfull
        get a sibling S of L; // parentpointer used to find S
        if S has extra entries,
            redistribute evenly between L and S;
            find entry in parent for node on right; // call it M
            replace key value in parent entry by new low-key value in M;
            set oldchildentry to null, return;
        else, merge L and S // call node on rhs M
            oldchildentry = & (current entry in parent for M);
            move all entries from M to node on left;
            discard empty node M, adjust sibling pointers, return;
endproc

```

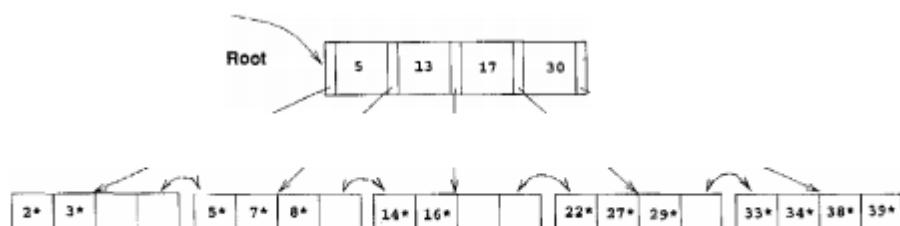
Algorithm for Deletion from B+ Tree of Order r!



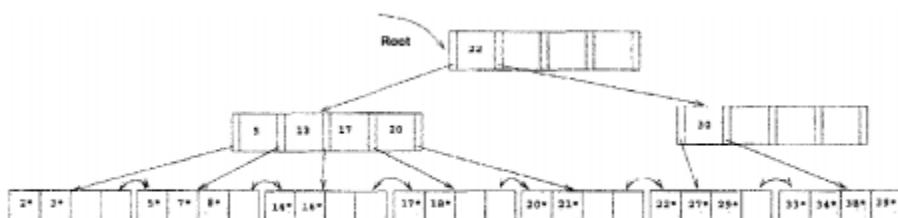
B+ Tree after Deleting Entries 19* and 20*



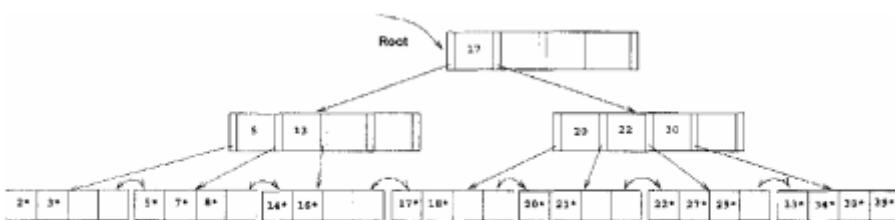
Partial B+ Tree during Deletion of Entry 24*



B+ Tree after Deleting Entry 24*



A B+ Tree during a Deletion



B+ Tree after Deletion

Test Questions

a.Fill in the blanks type of questions

1. _____ is used for backup and archival data.
2. Jukebox is a kind of _____ storage.
3. _____ storage loses its contents when the power to the device is removed.
4. _____ is the fastest and costliest form of storage.
5. _____ file organization is used to store large objects.
6. In _____ file organization, mixing of tuples from more than one relation is done, but it allows efficient processing of the join.
7. _____ of memory is available for storage of copies of disk blocks.
8. CR-R is the short form of _____
9. _____ stands for write once, read many.
10. Sequential file organization is also known as _____

b. Multiple choice questions

1. The method in which records are physically stored in a specified order according to a key field in each record is
 1. hash. (B) direct. (C) sequential. (D) all of the above.

Ans: A

2. The method of access which uses key transformation is known as
(A) direct. (B) hash. (C) random. (D) sequential.

Ans: B

3. The file organization that provides very fast access to any arbitrary record of a file is
1. Ordered file (B) Unordered file (C) Hashed file (D) B-tree

Ans: C

4. Which of the following database object does not physically exist?
1. base table (B) index (C) view (D) none of the above

Ans: C

5. The physical location of a record is determined by a mathematical formula that transforms a file key into a record location is :
1. B-Tree File (B) Hashed File (C) Indexed File (D) Sequential file.

Ans: B

6. Hierarchical model is also called
1. Tree structure (B) Plex Structure (C) Normalize Structure (D) Table Structure

Ans: A

7. A B-tree of order m has maximum of _____ children
1. m (B) m+1 (C) m-1 (D) m/2

Ans: A

8. The method of access which uses key transformation is known as
1. Direct (B) Hash (C) Random (D) Sequential

Ans: B

c). True or False questions

1. Insertion and deletion of fixed length records are simple to implement than that in variable length records.
2. The slotted page structure requires that there be no pointers that point directly to records.
3. Insertion is easy in heap file organization.
4. Deletion is difficult in sequential file organization.
5. Overflow blocks are used during insertion in sequential file organization.
6. Auxiliary storage devices are also useful for transferring data from one computer to another.
7. It is more economical to store data on secondary storage devices than in primary storage devices.
8. In case of sequential file organization, records are stored in some predetermined sequence, one after another.
9. A file could be made of records which are of different sizes. These records are called variable length records.
10. Permanent storage of data is done on the main memory.

Review Questions

Objective type of questions(Very short notes)

1. How data differs from information?
2. What are the different types of cache memory?
3. List at least two devices where flash memory is used.
4. What is the significance of different colors for CDs/DVDs?
5. Which storage media belongs to volatile category?
6. Due to which reasons, variable length records arise in database system?
7. How seek time and latency time differs from each other?
8. What details about an index for a relation is stored in data dictionary?
9. List any four information stored in data dictionary.
10. If any of the file organization has inconsistent data-dictionary, Will you be able to access required information? Justify by giving brief justification.

b) Analytical type questions <Minimum of ten>

c) Essay type Questions

1. What are the Constituents of file? Explain all the possible file operations.
2. Explain variable length records.
3. How insertion and deletion are managed in fixed length records?
4. How insertion and deletion are managed in variable length records?

5. Explain sequential file organization.
6. Explain multi table clustering file organization.
7. What type of information is stored in data dictionary?
8. Using example, explain the insertion and deletion of records in heap file organization.
9. Explain any two non-volatile storage media.
10. Explain any two auxiliary storage devices in detail.

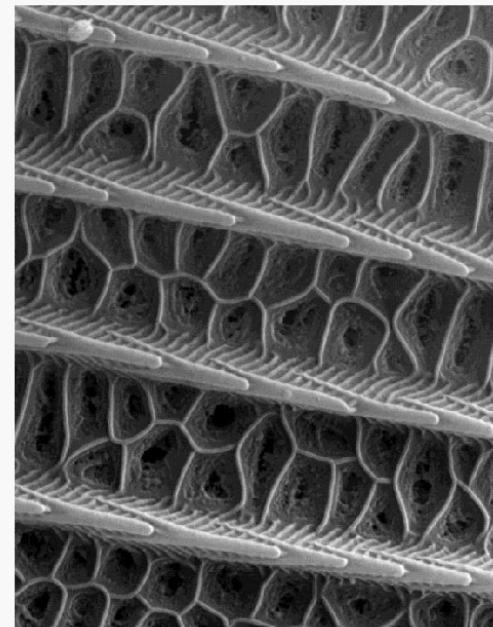
d) Problems <As per required Number>

1. Assume that initially the tuples with keys $m*i$, $i=1,\dots,n$ and the corresponding index entries are stored in consecutive blocks, no overflow is needed and space utilization is 66%. Display the state of the indexes and the relation assuming $p=6$, $r=44$, $k=20$, $d=160$, $m=1000$, $n=30$.
2. Consider a linear hash structure where buckets can hold up to two records. Initially the structure is empty. Assume that the utilization threshold value is 100%. The keys and their hash values are given above. Show the structure after all values have been inserted.

e) Case study <As per required Number>

File Organization

A Case Study: Butterfly Wing



i. Skill Building Exercises/Assignments

1. Explain (a) Heap file (b) Sorted file. Also discuss their advantages and disadvantages.

Ans: Heap File is an unordered set of records, stored on a set of pages. This class provides basic support for inserting, selecting, updating, and deleting records. Temporary heap files are used for external sorting and in other relational operators. A sequential scan of a heap file (via the Scan class) is the most basic access method. Sorted file The sort utility shall perform one of the following functions:

1. Sort lines of all the named files together and write the result to the specified output.
2. Merge lines of all the named (presorted) files together and write the result to the specified output.
3. Check that a single input file is correctly presorted. Comparisons shall be based on one or more sort keys extracted from each line of input (or, if no sort keys are specified, the entire line up to, but not including, the terminating), and shall be performed using the collating sequence of the current locale.
2. Describe a method for direct search? Explain how data is stored in a file so that direct searching can be performed.

Ans: For a file of unordered fixed length records using unspanned blocks and contiguous allocation, it is straight forward to access any record by its position in the file. If the file records are numbered 0,1,2,---,r-1 and the records in each block are numbered 0,1,---bfr-1; where bfr is the blocking factor, then ith record of the file is located in block $[(i/bfr)]$ and is the $(i \bmod bfr)$ th record in that block. Such a file is often called a relative or direct file because records can easily be accessed directly by their relative positions. Accessing a record based on a search condition; however, it facilitates the construction of access paths on the file, such as the indexes.

3. List any two significant differences between a file processing system and a DBMS.

Ans: File Processing System vs. DBMS Data Independence - Data independence is the capacity to change the schema at one level of a database system without having to change the schema at the next level. In file processing systems the data and applications are generally interdependent, but DBMS provides the feature of data independence. Data Redundancy – Data redundancy means unnecessary duplication of data. In file processing systems there is redundancy of data, but in DBMS we can reduce data redundancy by means of normalization process without

affecting the original data. If we do so in file processing system, it becomes too complex.

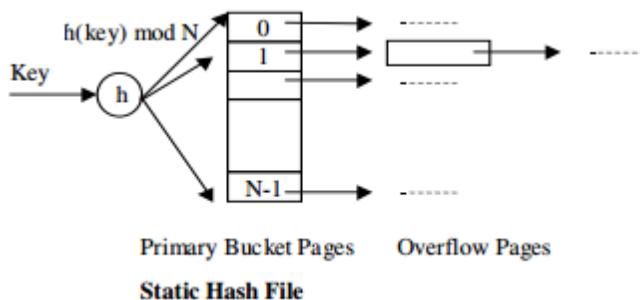
4. What is an INDEX as defined in ORACLE? Write the syntax of creating an INDEX. Create an index for the table Client, field CLIENT_NO of Q.

Ans: Indexes in Oracle – Index is typically a listing of keywords accompanied by the location of information on a subject. In other words, An index can be viewed as an auxiliary table which contains two fields: the key and the location of the record of that key. Indexes are used to improve the performance of the search operation. Indexes are not strictly necessary to running Oracle, they do speed the process. Syntax of Creating an Index: CREATE [BITMAP] [UNIQUE] INDEX ON ([,] . . .); Command: CREATE INDEX client_client_no ON client(client_no);

5. Describe the static hash file with buckets and chaining and show how insertion, deletion and modification of a record can be performed.

Ans: In static hash file organization, the term bucket is used to denote a unit storage that can store one or more records. A file consists of buckets 0 through N-1, with one primary page per bucket initially and additional overflow pages chained with bucket, if required later. Buckets contain data entries (or data records). In hashing scheme, a hash function, h , is performed on the key of the record to identify the bucket to which data record belongs to. The hash function is an important component of the hashing approach. The main problem with static hash file is that the number of buckets is fixed.

Insertion of a record – To insert a data entry, the hash function is used to identify the



correct bucket and then put the data entry there. If there is no space for this data entry, a new overflow page will be allocated, put the data entry on this page, and the page to the overflow chain of the bucket. Deletion of a record – To delete a data entry, the hash function is used to identify the correct bucket, locate the data entry by searching the bucket, and then remove it. If the data entry is the last in an overflow page, the overflow page is removed from the overflow chain of the bucket and

added to a list of free pages. Modification of a record – To modify a data entry, the hash function is used to identify the correct bucket, locate the data entry by searching the bucket and get it, modify the data entry, and then rewrite the modified data entry on it.

6. What are the reasons for having variable length records? What types of separator characters are needed for each?

Ans: Variable-Length Records – Variable-length records are those records, which are of different sizes. A file may contain variable-length records in any of the following situations:

- Records having variable length fields – In this case, the end-of-field symbol along with end-of-record symbol can be used as the separator characters.
- Records having repeating fields – In this case, the number of repetitions of repeating fields can be used with end-of-record symbol as the separator characters.
- Records having optional fields – In this case, the end-of-field symbol along with end-of-record symbol can be used as the separator characters.
- File containing records of different record types - In this case, a special field (or control field) can be prefixed with each record and end-of-record symbol can be used as the separator characters. In each of the above situations either we can use end-of-record symbol or the length of each record can be prefixed at the beginning of the record for the separation of the variable-length records.

7. What is the difference between a primary index and a secondary index? What are the advantages of using an index and what are its disadvantages.

Ans: Primary Index: A primary index is an ordered file whose records are of fixed length with two fields. The first field is the ordering key field-called primary key-of the data file, and the second field is a pointer to a disk block. There is one index entry in the index file for each block in the data file. Each index entry has the value of the primary key field for the first record in a block and a pointer to that block as its two field values. A major problem with a primary index is insertion and deletion of records. If we attempt to insert a record in it's correct positioning the data file, we have to not only move records to make space for the new record but also change some index entries. Secondary Index: A secondary index is also an ordered file with two fields. The first field is non-ordering field of the data file that is an indexing field. The second field is either a block pointer or a record pointer. A secondary index on a candidate key looks just like a dense primary index, except that the records pointed to by successive values in the index are not stored sequentially. In contrast, if the search key of a secondary index is not a candidate key, it is not enough to point to just the first record with each search-key value. The

remaining records with the same search – key value could be anywhere in the file, since the records are ordered by the search key of the primary index, rather than by the search key if the secondary index. Therefore, a secondary index must contain pointers to all the records. Secondary indices improve the performance of queries that use keys other than the search key of the primary index. However, they impose a significant overhead on modification of the database. The designer of a database decides which secondary indices are desirable on an estimate of the relative frequency of query's and modifications. Some of the advantages of using an index are: (i) Indexes speed up search on the indexed attributes(s). Without an index either a sequential search or some sort of binary search would be needed. (ii) Indexes can also speed up sequential processing of the file when the file is not stored as a sequential file. Some of the disadvantages of using an index are : (i) An index requires additional storage. This additional storage can be significant when a number of indexes are being used on a file. (ii) Insertion, deletion and updates on a file with indexes takes more time than on a file without any indexes

8. What are the causes of bucket overflow in a hash file organization? What can be done to reduce the occurrence of bucket overflow?

Ans: When a record is inserted, the bucket to which it is mapped has space to store the record. If the bucket does not have enough space, a bucket overflow is said to occur. Bucket overflow can occur for several reasons: Insufficient buckets : The number of buckets, which we denote nb, must be chosen such than $nb > nr/fr$, where nr , denotes the total number of records that will be stored, and fr , denotes the number of records that will fit in a bucket. This designation, of course, assumes that the total number of records is known when the hash function is chosen. Skew : Some buckets are assigned more records than are others, so a bucket may overflow even when other buckets still have space. This situation is called bucket skew. Skew can occur for two reasons:

1. Multiple records may have the same search key.
2. The chosen hash function may result in non-uniform distribution of search keys. So, that the probability of bucket overflow is reduced, the number of buckets is chosen to be $(nr/fr)*(1+d)$, where d is a fudge factor typically around 0.2. Some space is wasted: About 20 percent o the space in the buckets will be empty. But the benefit is that the probability of overflow is reduced.

Despite allocation of a few more buckets than required, bucket overflow can still occur. We handle bucket overflow by using overflow buckets. If a

record must be inserted into a bucket b, and b is already full, the system provides an overflow bucket for b, and inserts the record into the overflow bucket, and so on. All the overflow buckets of a given bucket are chained together in a linked list. Overflow handling using such linked list is called overflow chaining.

9. Discuss the techniques for a hash file to expand and shrink dynamically. What are the advantages and disadvantages of each?

Ans: The hashing techniques that allow dynamic file expansion are: (i) Extendible hashing (ii) Linear hashing The main advantage of extendible hashing that makes it attractive is that performance of the file does not degrade as the file grows. Also, no space is allocated in extendible hashing for future growth, but additional buckets can be allocated dynamically as needed. A disadvantage is that the directory must be searched before accessing the buckets themselves, resulting in two blocks accesses instead of one in static hashing.

10. Discuss the mechanism to read data from and write to a disk.

Ans: Disk read/write heads are mechanisms that read data from or write data to disk drives. The heads have gone through a number of changes over the years. In a hard drive, the heads 'fly' above the disk surface with clearance of as little as 3 nanometers. The "flying height" is constantly decreasing to enable higher area density. The flying height of the head is controlled by the design of an air-bearing etched onto the disk-facing surface of the slider. The role of the air bearing is to maintain the flying height constant as the head moves over the surface of the disk. If the head hits the disk's surface, a catastrophic head crash can result. The heads themselves started out similar to the heads in tape recorders—simple devices made out of a tiny C-shaped piece of highly magnetizable material called ferrite wrapped in a fine wire coil. When writing, the coil is energized, a strong magnetic field forms in the gap of the C, and the recording surface adjacent to the gap is magnetized. When reading, the magnetized material rotates past the heads, the ferrite core concentrates the field, and a current is generated in the coil. The gap where the field is very strong is quite narrow. That gap is roughly equal to the thickness of the magnetic media on the recording surface. The gap determines the minimum size of a recorded area on the disk. Ferrite heads are large, and write fairly large features.

ii.

Previous Questions (Asked by JNTUK from the concerned Unit)

1. A) Describe in detail about algorithms for updating single level indices.
B) Give comparison of different file organizations.
2. Describe a B+ tree for the following set of key values:(2,3,5,7,11,17,19,23,29,31)
Assume that the tree is initially empty and values are added in ascending order.
 - i) Construct B+ tree for the case where the number of pointer that will fit in one node is four.
 - ii) Show the step involved to find records with a search-key value of 11.
3. a) What is an index? Discuss important properties of an index that affect the efficiency of searches using the index. b) Describe in detail about different RAID levels.
4. . a) What are the main differences between ISAM and B+ tree indexes?
b) Describe a B+ tree for the following set of key values:(2,3,5,7,11,17,19,23,29,31)
Assume that the tree is initially empty and values are added in ascending order.
Construct B+ tree for the case where the number of pointer that will fit in one node is six.
5. a) What are the causes of bucket overflow in a hash file organization? What can be done to reduce the occurrence of bucket overflows?
b) Discuss about multilevel indices in detail.
6. A) Explain main characteristics of a B+ tree in detail. Discuss operations on B+ trees.
B) Describe a B+ tree for the following set of key values:(2,3,5,7,11,17,19,23,29,31)
Assume that the tree is initially empty and values are added in ascending order. Construct B+ tree for the case where the number of pointer that will fit in one node is six.
7. a) Explain the distinction between closed and open hashing. Discuss the relative merits of each technique in database applications.
b) On what factors techniques for indexing and hashing must be evaluated?
Explain.
8. Explain all the operations on B+ tree by taking a sample example.

iii. GATE Questions (Where relevant)

Q.1 A collection of field is called a record with respect of DBMS, a record corresponds to

- (a) Tuple
- (b) Relation
- (c) File
- (d) Attribute

www.edugrabs.com

Q.2 Commonly, a database consists of a process that provides data from secondary storage to one or more _____ processes that are application using the data.

- (a) slave, master
- (b) master, slave
- (c) client, server
- (d) server, client

www.edugrabs.com

Q.3 RAID stands for

- (a) Rapid Action In Disaster
- (b) Random Access Internal Disks
- (c) Rapid Application Interface Design
- (d) Redundant Arrays of Independent Disks

www.edugrabs.com

Q.4 The linked list of deleted records is often called as

- (a) useless
- (b) overflow
- (c) pointers
- (d) free list

www.edugrabs.com

- Q.5** Two disks are called mirrors of each other if
- (a) they share the same address space in the main memory
 - (b) the second disk is the continuation of the first
 - (c) they hold identical copies of data
 - (d) they contain blocks of a single file

www.edugrabs.com

- Q.6** At the root of a B-tree there are at least
- (a) 5 used pointers
 - (b) 4 used pointers
 - (c) 3 used pointers
 - (d) 2 used pointers

www.edugrabs.com

- Q.7** Each block of B-tree will have space for
- (a) cannot be determined
 - (b) n search-key values, and $n + 1$ pointers
 - (c) $n + 1$ search-key values and n pointers
 - (d) n search-key values, and n pointers

www.edugrabs.com

- Q.8** Match the following:
- | | |
|---------------------|--------------------------|
| (i) Ordered indices | (A) Sorted ordering |
| (ii) Hash indices | (B) Exponential function |
| | (C) Uniform distribution |
- (a) (i) – (C), (ii) – (A)
 - (b) (i) – (B), (ii) – (C)
 - (c) (i) – (A), (ii) – (C)
 - (d) (i) – (A), (ii) – (B)

www.edugrabs.com

Q.9 What is true for variable length record

- I. Storage of multiple record types in a file
 - II. Record types that allow variable lengths for one or more field
 - III. Record types that allow repeating fields
 - IV. It is difficult to delete a record from this structure. The space occupied by the record to be deleted must be filled with some other record of file.
- (a) All of the above
 - (b) Only I and IV
 - (c) I, II , III
 - (d) I and II

www.edugrabs.com

Q.10 The smallest individual unit of a magnetic disk is

- (a) Platter
- (b) Cylinder
- (c) Sector
- (d) Track

www.edugrabs.com

iv. Interview questions (which are frequently asked in a Technical round - Placements)

v. Real-Word (Live) Examples / Case studies wherever applicable



community workshop series

University of North Carolina at Chapel Hill Libraries
Carboro Cybrary | Chapel Hill Public Library | Durham County Public Library

FILE ORGANIZATION

| | |
|--|---------|
| GETTING STARTED <i>Prerequisites</i> <i>What You Will Learn</i> | PAGE 02 |
| PRINCIPLES OF FILE ORGANIZATION <i>Organization Trees</i> <i>Creating Categories</i> | PAGE 03 |
| FILES AND FOLDERS <i>Creating Folders</i> <i>Saving Files in Folders</i> <i>Moving vs. Copying</i> <i>Deleting Files and Folders</i> | PAGE 05 |
| DIFFERENT VIEWS | PAGE 10 |
| CREATING SHORTCUTS | PAGE 10 |
| KEEPING THINGS ORGANIZED | PAGE 11 |

To view our full schedule, handouts, and additional tutorials, visit our website:
www.library.unc.edu/cws

Last Updated July 2015

outside of the campus)

vii. Literature references of Relevant NPTEL Videos/Web/You Tube videos etc.

1. <https://www.youtube.com/watch?v=Mj-6nNJBI7c&list=PLyvBGMFYV3auVdxQ1-88ivNFpmUEy-U3M&index=22>
2. <https://www.youtube.com/watch?v=S93ECU5FbHc&list=PLyvBGMFYV3auVdxQ1-88ivNFpmUEy-U3M&index=23>
3. <https://www.youtube.com/watch?v=ZzaSVf-fFrY&list=PLyvBGMFYV3auVdxQ1-88ivNFpmUEy-U3M&index=24>
4. <https://www.youtube.com/watch?v=aVMSNLnZc-Q&list=PLyvBGMFYV3auVdxQ1-88ivNFpmUEy-U3M&index=25>
5. <https://www.youtube.com/watch?v=Dj3p3kfFiuE&list=PLyvBGMFYV3auVdxQ1-88ivNFpmUEy-U3M&index=26>

i. Any Lab requirements; if so link it to Lab Lesson Plan.

ii. Reference Text Books / with Journals Chapters etc.

iDistance: An Adaptive B⁺-Tree Based Indexing Method for Nearest Neighbor Search

H. V. JAGADISH

University of Michigan

BENG CHIN OOI and KIAN-LEE TAN

National University of Singapore

CUI YU

Monmouth University

and

RUI ZHANG

National University of Singapore

In this article, we present an efficient B⁺-tree based indexing method, called iDistance, for K-nearest neighbor (KNN) search in a high-dimensional metric space. iDistance partitions the data based on a space- or data-partitioning strategy, and selects a reference point for each partition. The data points in each partition are transformed into a single dimensional value based on their similarity with respect to the reference point. This allows the points to be indexed using a B⁺-tree structure and KNN search to be performed using one-dimensional range search. The choice of partition and reference point adapts the index structure to the data distribution.

We conducted extensive experiments to evaluate the iDistance technique, and report results demonstrating its effectiveness. We also present a cost model for iDistance KNN search, which can be exploited in query optimization.

Categories and Subject Descriptors: H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Indexing, KNN, nearest neighbor queries

Authors' addresses: H. V. Jagadish, Department of Computer Science, University of Michigan, 1301 Beal Avenue, Ann Arbor, MI 48109; email: jag@cs.umich.edu; B. C. Ooi, K.-L. Tan, and R. Zhang, Department of Computer Science, National University of Singapore Kent Ridge, Singapore 117543; email: {ooibc,tankl,zhangrui}@comp.nus.edu.sg; C. Yu, Department of Computer Science, Monmouth University, 400 Cedar Avenue, West Long Branch, NJ 07764-1898; email: cyu@monmouth.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.
© 2005 ACM 0362-5915/05/0600-0364 \$5.00

ACM Transactions on Database Systems, Vol. 30, No. 2, June 2005, Pages 364–397.

Fractal Prefetching B⁺-Trees: Optimizing Both Cache and Disk Performance

Shimin Chen, Phillip B. Gibbons[†], Todd C. Mowry, and Gary Valentin[‡]

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
{chen,sm,tcm}@cs.cmu.edu

[†]Information Sciences Research Center
Bell Laboratories
Murray Hill, NJ 07974

[‡]DB2 UDB Development Team
IBM Toronto Lab
Markham, Ontario, Canada L6G 1C7
valentin@ca.ibm.com

ABSTRACT

B⁺ Trees have been traditionally optimized for I/O performance with disk pages as tree nodes. Recently, researchers have proposed new types of B⁺ Trees optimized for CPU cache performance in main memory environments, where the tree node sizes are one or a few cache lines. Unfortunately, due primarily to this large discrepancy in optimal node sizes, existing disk optimized B⁺ Trees suffer from poor cache performance while cache optimized B⁺ Trees exhibit poor disk performance. In this paper, we propose *fractal prefetching B⁺ Trees* (fpB⁺ Trees), which embed "cache optimized" trees within "disk optimized" trees, in order to optimize both cache and I/O performance. We design and evaluate two approaches to breaking disk pages into cache optimized nodes: *disk first* and *cache first*. These approaches are somewhat biased in favor of maximizing disk and cache performance, respectively, as demonstrated by our results. Both implementations of fpB⁺ Trees achieve dramatically better cache performance than disk optimized B⁺ Trees: a factor of 1.1-1.8 improvement for search, up to a factor of 4.2 improvement for range scans, and up to a 20 fold improvement for updates, all without significant degradation of I/O performance. In addition, fpB⁺ Trees accelerate I/O performance for range scans by using jump pointer arrays to prefetch leaf pages, thereby achieving a speed up of 2.5-5 on IBM's DB2 Universal Database.

1. INTRODUCTION

The B⁺ Tree is a ubiquitous structure for indexing disk resident data. It provides basic index operations such as search, range scan, insertion and deletion, while minimizing the number of disk accesses. To optimize I/O performance, traditional "disk optimized" B⁺ Trees are composed of nodes the size of a disk page—i.e., the natural transfer size for reading or writing to disk. Recently, several studies [5, 6, 19] have considered B⁺ Tree variants for indexing memory resident data. These studies present new types of B⁺ Trees—*cache sensitive B⁺ Trees* [19], *partial key B⁺ Trees* [5], and *prefetching B⁺ Trees* [6]—that optimize for

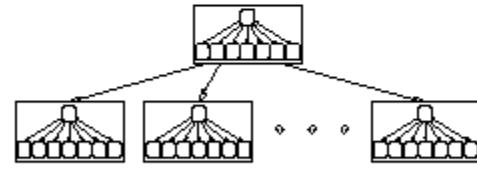


Figure 1: Self-similar "tree within a tree" structure

CPU cache performance by minimizing the impact of cache misses. These "cache optimized" B⁺ Trees are composed of nodes the size of a cache line¹—i.e., the natural transfer size for reading or writing to main memory.

Unfortunately, B⁺ Trees optimized for disk suffer from poor CPU cache performance, and B⁺ Trees optimized for cache suffer from poor I/O performance. This is primarily because of the large discrepancy in node sizes: disk pages are typically 4KB-64KB while cache lines are often 32B-128B, depending on the system. Thus existing disk optimized B⁺ Trees suffer an excessive number of cache misses to search in a (large) node, wasting time and forcing the eviction of useful data from the cache. Likewise, existing cache optimized B⁺ Trees, in searching from the root to the desired leaf, may fetch a distinct page for each node on this path. This is a significant performance penalty, for the smaller nodes of cache optimized B⁺ Trees imply much deeper trees than in the disk optimized cases (e.g., twice as deep). The I/O penalty for range scans on nonclustered indexes of cache-optimized trees is even worse: a distinct page may be fetched for each leaf node in the range, increasing the number of disk accesses by the ratio of the node sizes (e.g., a factor of 500).

1.1 Our Approach: Fractal Prefetching B⁺-Trees

In this paper, we propose and evaluate *Fractal Prefetching B⁺ Trees* (fpB⁺ Trees), which are a new type of B⁺ Tree that optimizes both cache and I/O performance. In a nutshell, an fpB⁺ Tree is a single index structure that can be viewed at two different granularities: at a coarse granularity, it contains disk optimized nodes that are roughly the size of a disk page, and at a fine granularity, it contains cache optimized nodes that are roughly the size of a cache line. We refer to a fpB⁺ Tree as being "fractal" because of its self similar "tree within a tree" structure, as illustrated in Figure 1. The cache optimized aspect is modeled after the *prefetching B⁺ Trees* that we proposed earlier [6], which

¹Current affiliation: Intel Research Pittsburgh, 417 South Craig St., Pittsburgh, PA 15213 phillip.b.gibbons@intel.com

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM SIGMOD '2002 June 4-6, Madison, Wisconsin, USA
Copyright 2002 ACM 1-58113-497-5/02/06 ...\$5.00.

¹In the case of prefetching B⁺ Trees [6], the nodes are several cache lines wide.

6. Reference text books/web material etc.,

<Please provide a detailed list of textbooks and reference materials relevant for the entire subject. Note: Please do not copy all the lecture level references. You are requested to avoid any references which may only be relevant to 1 or 2 lectures>

Textbooks:

- 1. Database Management Systems, 3/e Raghuram Krishnan, Johannes Gehrke, TMH**
- 2. Database Management System, 6/e Ramez Elmasri, Shamkant B. Navathe, PEA**
- 3. Database Principles Fundamentals of Design Implementation and Management, Corlos Coronel, Steven Morris, Peter Robb, Cengage Learning**

Web-links (including videos):

- 1. <https://www.youtube.com/watch?v=1057YmExS-I>**
- 2. <https://www.youtube.com/watch?v=o1HLZRtFwxk>**
- 3. <https://www.youtube.com/watch?v=Wv1c9K4788A>**

Please add if you have more...

Reference Journals:

- 1.**
- 2.**

Please add if you have more...

7. Mid Question Paper + Schemes of Evaluation.

<Please include atleast three mid 1 and three mid 2 papers with scheme of evaluation for each of them. Note: All of the papers are to be included here in this section as softcopy. Please do not attach scanned hardcopies at the end.>

8. Fast track material for Back-Log students.

Please include the following:

- *A quick reference document (a maximum of 4 pages) to study the basic concepts related to the subject*
- *A minimum of 3 unique question papers with 8 questions each along with detailed solutions. These question papers need NOT have complex questions. The idea is to make the student clear the exam. Hence, please include only basic and average complexity questions after studying which a student can clear the exam*
- *Each student is expected to write each of the papers a minimum of 10 times*
- *Please include all the papers (softcopies) and solutions in this section. Please do NOT attach handwritten sheets at the end.*

DBMS - Quick Guide

DBMS - Overview

Database is a collection of related data and data is a collection of facts and figures that can be processed to produce information.

A **database management system** stores data in such a way that it becomes easier to retrieve, manipulate, and produce information.

Characteristics

A modern DBMS has the following characteristics –

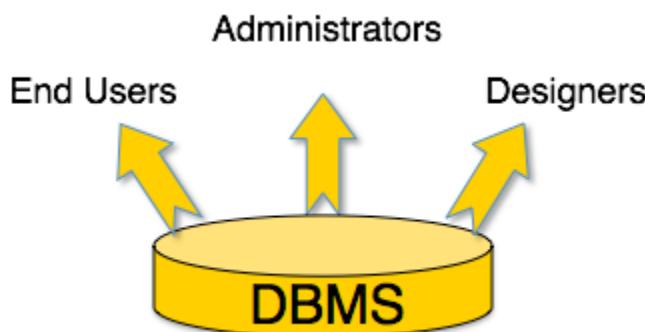
- **Real-world entity** – A modern DBMS is more realistic and uses real-world entities to design its architecture. It uses the behavior and attributes too. For example, a school database may use students as an entity and their age as an attribute.
- **Relation-based tables** – DBMS allows entities and relations among them to form tables. A user can understand the architecture of a database just by looking at the table names.
- **Isolation of data and application** – A database system is entirely different than its data. A database is an active entity, whereas data is said to be passive, on which the database works and organizes. DBMS also stores metadata, which is data about data, to ease its own process.
- **Less redundancy** – DBMS follows the rules of normalization, which splits a relation when any of its attributes is having redundancy in values. Normalization is a mathematically rich and scientific process that reduces data redundancy.
- **Consistency** – Consistency is a state where every relation in a database remains consistent. There exist methods and techniques, which can detect attempt of

leaving database in inconsistent state. A DBMS can provide greater consistency as compared to earlier forms of data storing applications like file-processing systems.

- **Query Language** – DBMS is equipped with query language, which makes it more efficient to retrieve and manipulate data. A user can apply as many and as different filtering options as required to retrieve a set of data. Traditionally it was not possible where file-processing system was used.
- **ACID Properties** – DBMS follows the concepts of Atomicity, Consistency, Isolation, and Durability (normally shortened as ACID). These concepts are applied on transactions, which manipulate data in a database. ACID properties help the database stay healthy in multi-transactional environments and in case of failure.
- **Multiuser and Concurrent Access** – DBMS supports multi-user environment and allows them to access and manipulate data in parallel. Though there are restrictions on transactions when users attempt to handle the same data item, but users are always unaware of them.
- **Multiple views** – DBMS offers multiple views for different users. A user who is in the Sales department will have a different view of database than a person working in the Production department. This feature enables the users to have a concentrate view of the database according to their requirements.
- **Security** – Features like multiple views offer security to some extent where users are unable to access data of other users and departments. DBMS offers methods to impose constraints while entering data into the database and retrieving the same at a later stage. DBMS offers many different levels of security features, which enables multiple users to have different views with different features.

Users

A typical DBMS has users with different rights and permissions who use it for different purposes. Some users retrieve data and some back it up. The users of a DBMS can be broadly categorized as follows –



- **Administrators** – Administrators maintain the DBMS and are responsible for administrating the database. They are responsible to look after its usage and by whom it should be used. They create access profiles for users and apply limitations to maintain isolation and force security. Administrators also look after DBMS resources like system license, required tools, and other software and

hardware related maintenance.

- **Designers** – Designers are the group of people who actually work on the designing part of the database. They keep a close watch on what data should be kept and in what format. They identify and design the whole set of entities, relations, constraints, and views.
- **End Users** – End users are those who actually reap the benefits of having a DBMS. End users can range from simple viewers who pay attention to the logs or market rates to sophisticated users such as business analysts.

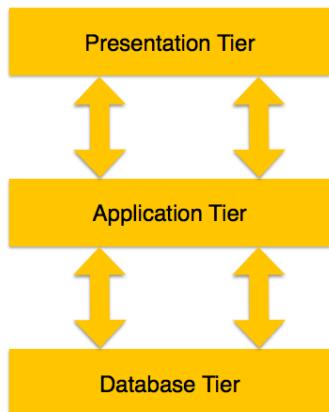
DBMS - Architecture

In 1-tier architecture, the DBMS is the only entity where the user directly sits on the DBMS and uses it. Any changes done here will directly be done on the DBMS itself. It does not provide handy tools for end-users. Database designers and programmers normally prefer to use single-tier architecture.

If the architecture of DBMS is 2-tier, then it must have an application through which the DBMS can be accessed. Programmers use 2-tier architecture where they access the DBMS by means of an application. Here the application tier is entirely independent of the database in terms of operation, design, and programming.

3-tier Architecture

A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.



- **Database (Data) Tier** – At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.
- **Application (Middle) Tier** – At this tier reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not aware of any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database.
- **User (Presentation) Tier** – End-users operate on this tier and they know nothing

about any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier.

Multiple-tier database architecture is highly modifiable, as almost all its components are independent and can be changed independently.

DBMS - Data Models

Data models define how the logical structure of a database is modeled. Data Models are fundamental entities to introduce abstraction in a DBMS. Data models define how data is connected to each other and how they are processed and stored inside the system.

The very first data model could be flat data-models, where all the data used are to be kept in the same plane. Earlier data models were not so scientific, hence they were prone to introduce lots of duplication and update anomalies.

Entity-Relationship Model

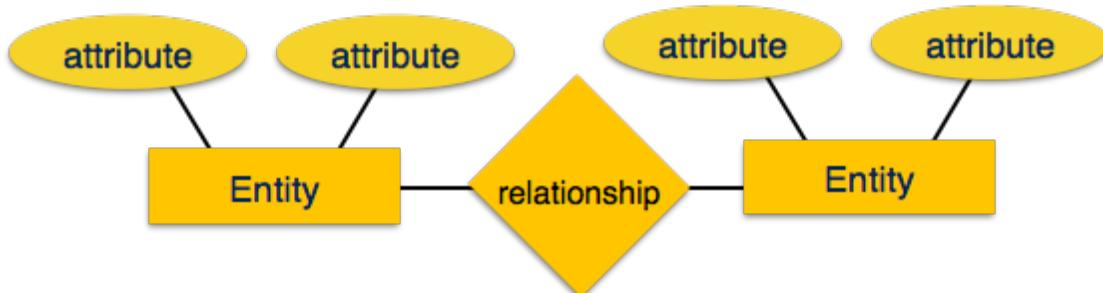
Entity-Relationship (ER) Model is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the ER Model creates entity set, relationship set, general attributes and constraints.

ER Model is best used for the conceptual design of a database.

ER Model is based on –

- **Entities** and their *attributes*.
- **Relationships** among entities.

These concepts are explained below.

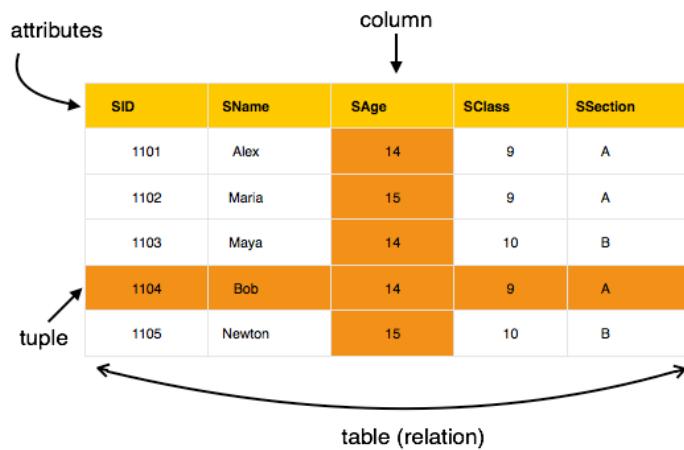


- **Entity** – An entity in an ER Model is a real-world entity having properties called **attributes**. Every **attribute** is defined by its set of values called **domain**. For example, in a school database, a student is considered as an entity. Student has various attributes like name, age, class, etc.
- **Relationship** – The logical association among entities is called **relationship**. Relationships are mapped with entities in various ways. Mapping cardinalities

define the number of association between two entities.

Relational Model

The most popular data model in DBMS is the Relational Model. It is more scientific a model than others. This model is based on first-order predicate logic and defines a table as an **n-ary relation**.



The main highlights of this model are -

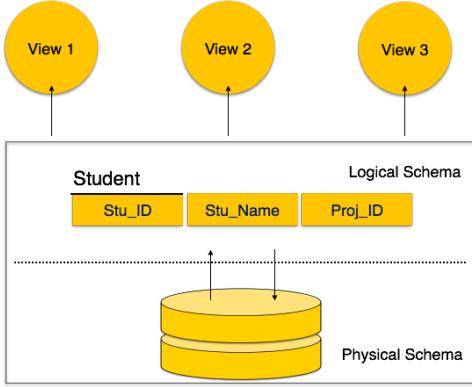
- Data is stored in tables called **relations**.
- Relations can be normalized.
- In normalized relations, values saved are atomic values.
- Each row in a relation contains a unique value.
- Each column in a relation contains values from a same domain.

DBMS - Data Schemas

Database Schema

A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It's the database designers who design the schema to help programmers understand the database and make it useful.



A database schema can be divided broadly into two categories –

- **Physical Database Schema** – This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.
- **Logical Database Schema** – This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.

Database Instance

It is important that we distinguish these two terms individually. Database schema is the skeleton of database. It is designed when the database doesn't exist at all. Once the database is operational, it is very difficult to make any changes to it. A database schema does not contain any data or information.

A database instance is a state of operational database with data at any given time. It contains a snapshot of the database. Database instances tend to change with time. A DBMS ensures that its every instance (state) is in a valid state, by diligently following all the validations, constraints, and conditions that the database designers have imposed.

ER Model - Basic Concepts

The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases.

Entity

An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.

An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.

Attributes

Entities are represented by means of their properties, called **attributes**. All attributes have values. For example, a student entity may have name, class, and age as attributes.

There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Types of Attributes

- **Simple attribute** – Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.
- **Composite attribute** – Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.
- **Derived attribute** – Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth.
- **Single-value attribute** – Single-value attributes contain single value. For example – Social_Security_Number.
- **Multi-value attribute** – Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.

These attribute types can come together in a way like –

- simple single-valued attributes
- simple multi-valued attributes
- composite single-valued attributes
- composite multi-valued attributes

Entity-Set and Keys

Key is an attribute or collection of attributes that uniquely identifies an entity among entity set.

For example, the roll_number of a student makes him/her identifiable among students.

- **Super Key** – A set of attributes (one or more) that collectively identifies an entity in an entity set.
- **Candidate Key** – A minimal super key is called a candidate key. An entity set may have more than one candidate key.
- **Primary Key** – A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.

Relationship

The association among entities is called a relationship. For example, an employee **works_at** a department, a student **enrolls** in a course. Here, Works at and Enrolls are called relationships.

Relationship Set

A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called **descriptive attributes**.

ER Diagram Representation

Let us now learn how the ER Model is represented by means of an ER diagram. Any object, for example, entities, attributes of an entity, relationship sets, and attributes of relationship sets, can be represented with the help of an ER diagram.

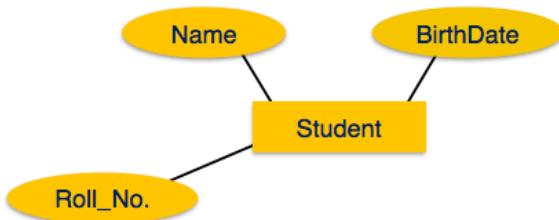
Entity

Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.

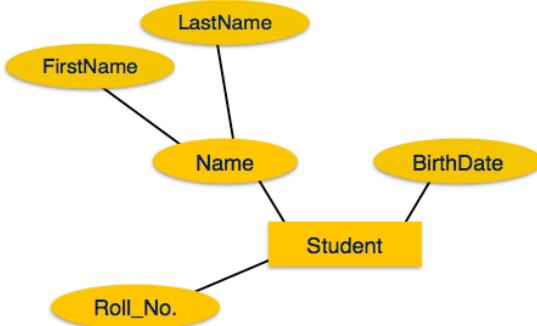


Attributes

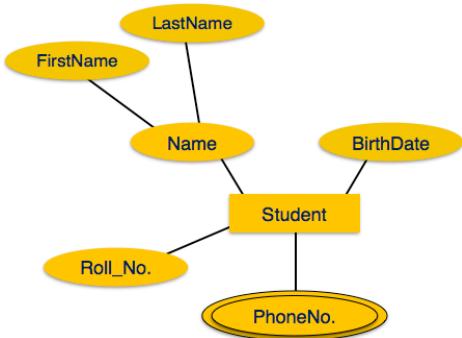
Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).



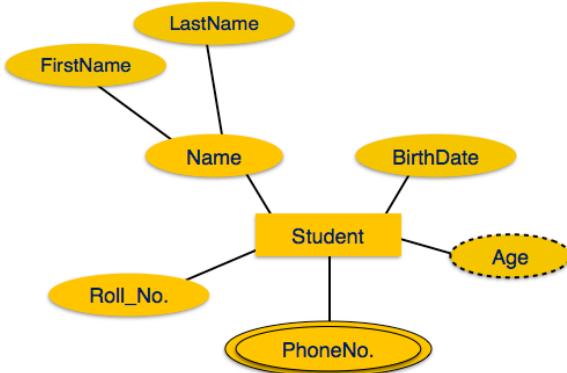
If the attributes are **composite**, they are further divided in a tree like structure. Every node is then connected to its attribute. That is, composite attributes are represented by ellipses that are connected with an ellipse.



Multivalued attributes are depicted by double ellipse.



Derived attributes are depicted by dashed ellipse.



Relationship

Relationships are represented by diamond-shaped box. Name of the relationship is written inside the diamond-box. All the entities (rectangles) participating in a relationship, are connected to it by a line.

Relation Data Model

Relational data model is the primary data model, which is used widely around the world for data storage and processing. This model is simple and it has all the properties and capabilities required to process data with storage efficiency.

Concepts

Tables – In relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represent records and columns represent the attributes.

Tuple – A single row of a table, which contains a single record for that relation is called a tuple.

Relation instance – A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.

Relation schema – A relation schema describes the relation name (table name), attributes, and their names.

Relation key – Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.

Attribute domain – Every attribute has some pre-defined value scope, known as attribute domain.

Constraints

Every relation has some conditions that must hold for it to be a valid relation. These conditions are called **Relational Integrity Constraints**. There are three main integrity constraints –

- Key constraints
- Domain constraints
- Referential integrity constraints

Key Constraints

There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called **key** for that relation. If there are more than one such minimal subsets, these are called **candidate keys**.

Key constraints force that –

- in a relation with a key attribute, no two tuples can have identical values for key attributes.
- a key attribute can not have NULL values.

Key constraints are also referred to as Entity Constraints.

Domain Constraints

Attributes have specific values in real-world scenario. For example, age can only be a positive integer. The same constraints have been tried to employ on the attributes of a relation. Every attribute is bound to have a specific range of values. For example, age cannot be less than zero and telephone numbers cannot contain a digit outside 0-9.

Referential integrity Constraints

Referential integrity constraints work on the concept of Foreign Keys. A foreign key is a key attribute of a relation that can be referred in other relation.

Referential integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exist.

.SQL Overview

SQL is a programming language for Relational Databases. It is designed over relational algebra and tuple relational calculus. SQL comes as a package with all major distributions of RDBMS.

SQL comprises both data definition and data manipulation languages. Using the data definition properties of SQL, one can design and modify database schema, whereas data manipulation properties allows SQL to store and retrieve data from database.

Data Definition Language

SQL uses the following set of commands to define database schema -

CREATE

Creates new databases, tables and views from RDBMS.

For example -

```
Create database tutorialspoint;  
Create table article;  
Create view for_students;
```

DROP

Drops commands, views, tables, and databases from RDBMS.

For example-

```
Drop object_type object_name;  
Drop database tutorialspoint;  
Drop table article;  
Drop view for_students;
```

ALTER

Modifies database schema.

```
Alter object_type object_name parameters;
```

For example-

```
Alter table article add subject varchar;
```

This command adds an attribute in the relation **article** with the name **subject** of string type.

Data Manipulation Language

SQL is equipped with data manipulation language (DML). DML modifies the database instance by inserting, updating and deleting its data. DML is responsible for all forms data modification in a database. SQL contains the following set of commands in its DML section –

- SELECT/FROM/WHERE
- INSERT INTO/VALUES
- UPDATE/SET/WHERE
- DELETE FROM/WHERE

These basic constructs allow database programmers and users to enter data and information into the database and retrieve efficiently using a number of filter options.

SELECT/FROM/WHERE

- **SELECT** – This is one of the fundamental query command of SQL. It is similar to the projection operation of relational algebra. It selects the attributes based on the condition described by WHERE clause.
- **FROM** – This clause takes a relation name as an argument from which attributes are to be selected/projected. In case more than one relation names are given, this clause corresponds to Cartesian product.
- **WHERE** – This clause defines predicate or conditions, which must match in order to qualify the attributes to be projected.

For example –

```
Select author_name  
From book_author  
Where age > 50;
```

This command will yield the names of authors from the relation **book_author** whose age is greater than 50.

INSERT INTO/VALUES

This command is used for inserting values into the rows of a table (relation).

Syntax-

```
INSERT INTO table (column1 [, column2, column3 ... ]) VALUES (value1 [, value2, value3 ... ])
```

Or

```
INSERT INTO table VALUES (value1, [value2, ... ])
```

For example –

```
INSERT INTO tutorialspoint (Author, Subject) VALUES ("anonymous", "computers");
```

UPDATE/SET/WHERE

This command is used for updating or modifying the values of columns in a table (relation).

Syntax –

```
UPDATE table_name SET column_name = value [, column_name = value ...] [WHERE condition]
```

For example –

```
UPDATE tutorialspoint SET Author="webmaster" WHERE Author="anonymous";
```

DELETE/FROM/WHERE

This command is used for removing one or more rows from a table (relation).

Syntax –

```
DELETE FROM table_name [WHERE condition];
```

For example –

```
DELETE FROM tutorialspoints  
WHERE Author="unknown";
```

DBMS - Normalization

Functional Dependency

Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A1, A2,..., An, then those two tuples must have to have same values for attributes B1, B2, ..., Bn.

Functional dependency is represented by an arrow sign (\rightarrow) that is, $X \rightarrow Y$, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side.

Normalization

If a database design is not perfect, it may contain anomalies, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.

- **Update anomalies** – If data items are scattered and are not linked to each other

properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.

- **Deletion anomalies** – We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.
- **Insert anomalies** – We tried to insert data in a record that does not exist at all.

Normalization is a method to remove all these anomalies and bring the database to a consistent state.

First Normal Form

First Normal Form is defined in the definition of relations (tables) itself. This rule defines that all the attributes in a relation must have atomic domains. The values in an atomic domain are indivisible units.

| Course | Content |
|-------------|----------------|
| Programming | Java, c++ |
| Web | HTML, PHP, ASP |

We re-arrange the relation (table) as below, to convert it to First Normal Form.

| Course | Content |
|-------------|---------|
| Programming | Java |
| Programming | c++ |
| Web | HTML |
| Web | PHP |
| Web | ASP |

Each attribute must contain only a single value from its pre-defined domain.

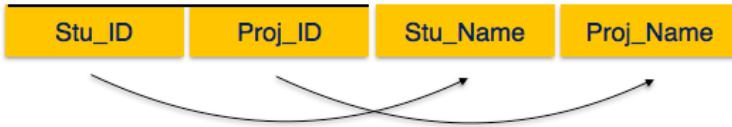
Second Normal Form

Before we learn about the second normal form, we need to understand the following –

- **Prime attribute** – An attribute, which is a part of the prime-key, is known as a prime attribute.
- **Non-prime attribute** – An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.

If we follow second normal form, then every non-prime attribute should be fully functionally dependent on prime key attribute. That is, if $X \rightarrow A$ holds, then there should not be any proper subset Y of X, for which $Y \rightarrow A$ also holds true.

Student_Project



We see here in Student_Project relation that the prime key attributes are Stu_ID and Proj_ID. According to the rule, non-key attributes, i.e. Stu_Name and Proj_Name must be dependent upon both and not on any of the prime key attribute individually. But we find that Stu_Name can be identified by Stu_ID and Proj_Name can be identified by Proj_ID independently. This is called **partial dependency**, which is not allowed in Second Normal Form.

Student

| | | |
|--------|----------|---------|
| Stu_ID | Stu_Name | Proj_ID |
|--------|----------|---------|

Project

| | |
|---------|-----------|
| Proj_ID | Proj_Name |
|---------|-----------|

We broke the relation in two as depicted in the above picture. So there exists no partial dependency.

Third Normal Form

For a relation to be in Third Normal Form, it must be in Second Normal form and the following must satisfy -

- No non-prime attribute is transitively dependent on prime key attribute.
- For any non-trivial functional dependency, $X \rightarrow A$, then either -
 - X is a superkey or,
 - A is prime attribute.

Student_Detail

| | | | |
|--------|----------|------|-----|
| Stu_ID | Stu_Name | City | Zip |
|--------|----------|------|-----|

The diagram shows a horizontal line connecting Stu_ID and Zip at the top to City at the bottom. There is a curved arrow from Stu_ID to City, and another curved arrow from Zip to City.

We find that in the above Student_detail relation, Stu_ID is the key and only prime key attribute. We find that City can be identified by Stu_ID as well as Zip itself. Neither Zip is a superkey nor is City a prime attribute. Additionally, $\text{Stu_ID} \rightarrow \text{Zip} \rightarrow \text{City}$, so there exists **transitive dependency**.

To bring this relation into third normal form, we break the relation into two relations as follows -

| Student_Detail | | |
|----------------|----------|-----|
| Stu_ID | Stu_Name | Zip |

| ZipCodes | |
|----------|------|
| Zip | City |

Boyce-Codd Normal Form

Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. BCNF states that –

- For any non-trivial functional dependency, $X \rightarrow A$, X must be a super-key.

In the above image, Stu_ID is the super-key in the relation Student_Detail and Zip is the super-key in the relation ZipCodes. So,

$\text{Stu_ID} \rightarrow \text{Stu_Name}, \text{Zip}$

and

$\text{Zip} \rightarrow \text{City}$

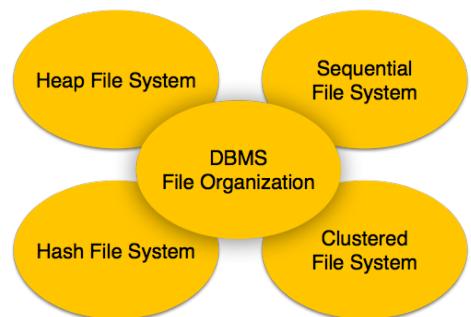
Which confirms that both the relations are in BCNF.

DBMS - File Structure

Relative data and information is stored collectively in file formats. A file is a sequence of records stored in binary format. A disk drive is formatted into several blocks that can store records. File records are mapped onto those disk blocks.

File Organization

File Organization defines how file records are mapped onto disk blocks. We have four types of File Organization to organize file records –



Heap File Organization

When a file is created using Heap File Organization, the Operating System allocates memory area to that file without any further accounting details. File records can be placed anywhere in that memory area. It is the responsibility of the software to manage the records. Heap File does not support any ordering, sequencing, or indexing on its own.

Sequential File Organization

Every file record contains a data field (attribute) to uniquely identify that record. In sequential file organization, records are placed in the file in some sequential order based on the unique key field or search key. Practically, it is not possible to store all the records sequentially in physical form.

Hash File Organization

Hash File Organization uses Hash function computation on some fields of the records. The output of the hash function determines the location of disk block where the records are to be placed.

File Operations

Operations on database files can be broadly classified into two categories –

- **Update Operations**
- **Retrieval Operations**

Update operations change the data values by insertion, deletion, or update. Retrieval operations, on the other hand, do not alter the data but retrieve them after optional conditional filtering. In both types of operations, selection plays a significant role. Other than creation and deletion of a file, there could be several operations, which can be done on files.

- **Open** – A file can be opened in one of the two modes, **read mode** or **write mode**. In read mode, the operating system does not allow anyone to alter data. In other words, data is read only. Files opened in read mode can be shared among several entities. Write mode allows data modification. Files opened in write mode can be read but cannot be shared.
- **Locate** – Every file has a file pointer, which tells the current position where the data is to be read or written. This pointer can be adjusted accordingly. Using find (seek) operation, it can be moved forward or backward.
- **Read** – By default, when files are opened in read mode, the file pointer points to the beginning of the file. There are options where the user can tell the operating system where to locate the file pointer at the time of opening a file. The very next data to the file pointer is read.
- **Write** – User can select to open a file in write mode, which enables them to edit its contents. It can be deletion, insertion, or modification. The file pointer can be located at the time of opening or can be dynamically changed if the operating system allows to do so.
- **Close** – This is the most important operation from the operating system's point of view. When a request to close a file is generated, the operating system
 - removes all the locks (if in shared mode),

- o saves the data (if altered) to the secondary storage media, and
- o releases all the buffers and file handlers associated with the file.

The organization of data inside a file plays a major role here. The process to locate the file pointer to a desired record inside a file varies based on whether the records are arranged sequentially or clustered.

DBMS - Indexing

We know that data is stored in the form of records. Every record has a key field, which helps it to be recognized uniquely.

Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done. Indexing in database systems is similar to what we see in books.

Indexing is defined based on its indexing attributes. Indexing can be of the following types –

- **Primary Index** – Primary index is defined on an ordered data file. The data file is ordered on a **key field**. The key field is generally the primary key of the relation.
- **Secondary Index** – Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
- **Clustering Index** – Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.

Ordered Indexing is of two types –

- Dense Index
- Sparse Index

Dense Index

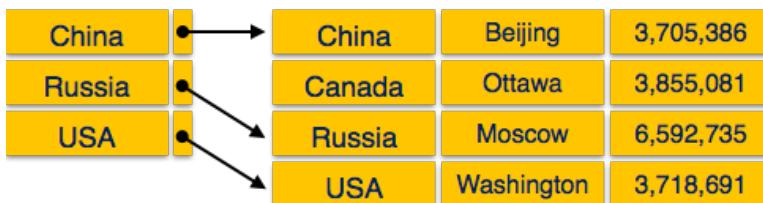
In dense index, there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain search key value and a pointer to the actual record on the disk.

| | | | | |
|--------|---|--------|------------|-----------|
| China | → | China | Beijing | 3,705,386 |
| Canada | → | Canada | Ottawa | 3,855,081 |
| Russia | → | Russia | Moscow | 6,592,735 |
| USA | → | USA | Washington | 3,718,691 |

Sparse Index

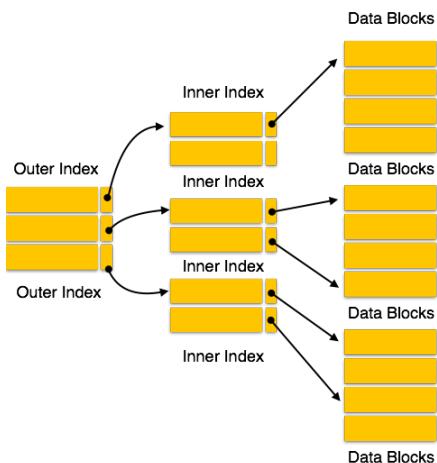
In sparse index, index records are not created for every search key. An index record here contains a search key and an actual pointer to the data on the disk. To search a record, we first proceed by index record and reach at the actual location of the data. If the data we are looking for is not where we directly reach by following the index, then the system

starts sequential search until the desired data is found.



Multilevel Index

Index records comprise search-key values and data pointers. Multilevel index is stored on the disk along with the actual database files. As the size of the database grows, so does the size of the indices. There is an immense need to keep the index records in the main memory so as to speed up the search operations. If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses.



Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.

DBMS - Transaction

A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.

ACID Properties

A transaction is a very small unit of a program and it may contain several low level tasks. A transaction in a database system must maintain Atomicity, Consistency, Isolation, and Durability – commonly known as ACID properties – in order to ensure accuracy, completeness, and data integrity.

- **Atomicity** – This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should

be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.

- **Consistency** - The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
- **Durability** - The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.
- **Isolation** - In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

Serializability

When multiple transactions are being executed by the operating system in a multiprogramming environment, there are possibilities that instructions of one transaction are interleaved with some other transaction.

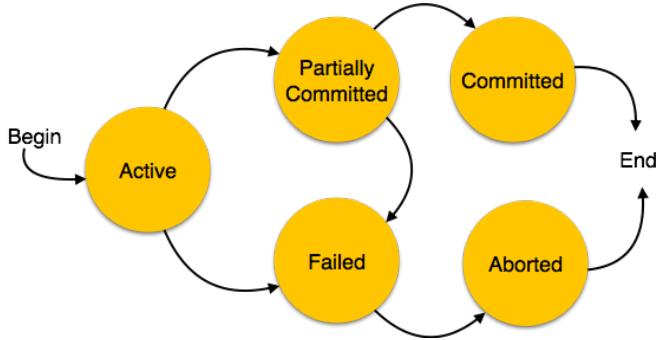
- **Schedule** - A chronological execution sequence of a transaction is called a schedule. A schedule can have many transactions in it, each comprising of a number of instructions/tasks.
- **Serial Schedule** - It is a schedule in which transactions are aligned in such a way that one transaction is executed first. When the first transaction completes its cycle, then the next transaction is executed. Transactions are ordered one after the other. This type of schedule is called a serial schedule, as transactions are executed in a serial manner.

In a multi-transaction environment, serial schedules are considered as a benchmark. The execution sequence of an instruction in a transaction cannot be changed, but two transactions can have their instructions executed in a random fashion. This execution does no harm if two transactions are mutually independent and working on different segments of data; but in case these two transactions are working on the same data, then the results may vary. This ever-varying result may bring the database to an inconsistent state.

To resolve this problem, we allow parallel execution of a transaction schedule, if its transactions are either serializable or have some equivalence relation among them.

States of Transactions

A transaction in a database can be in one of the following states –



- **Active** – In this state, the transaction is being executed. This is the initial state of every transaction.
- **Partially Committed** – When a transaction executes its final operation, it is said to be in a partially committed state.
- **Failed** – A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.
- **Aborted** – If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction aborts
 - Re-start the transaction
 - Kill the transaction
- **Committed** – If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.

DBMS - Concurrency Control

In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions. We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions. Concurrency control protocols can be broadly divided into two categories –

- Lock based protocols
- Time stamp based protocols

Lock-based Protocols

Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds –

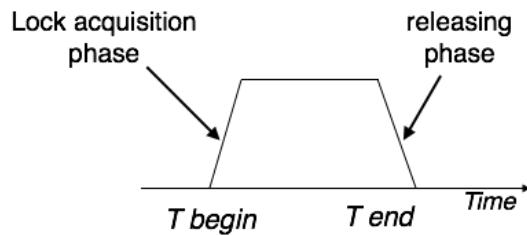
- **Binary Locks** – A lock on a data item can be in two states; it is either locked or unlocked.
- **Shared/exclusive** – This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write

operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

There are two types of lock protocols available –

Two-Phase Locking 2PL

This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.

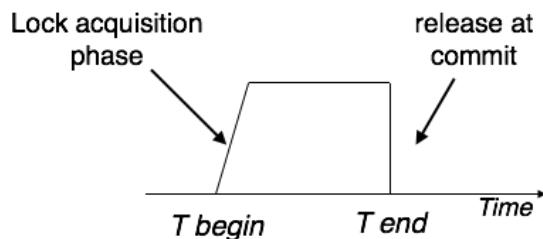


Two-phase locking has two phases, one is **growing**, where all the locks are being acquired by the transaction; and the second phase is shrinking, where the locks held by the transaction are being released.

To claim an exclusive (write) lock, a transaction must first acquire a shared (read) lock and then upgrade it to an exclusive lock.

Strict Two-Phase Locking

The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.



Strict-2PL does not have cascading abort as 2PL does.

Timestamp-based Protocols

The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp.

Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.

Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one.

In addition, every data item is given the latest read and write-timestamp. This lets the system know when the last 'read and write' operation was performed on the data item.

DBMS - Deadlock

In a multi-process system, deadlock is an unwanted situation that arises in a shared resource environment, where a process indefinitely waits for a resource that is held by another process.

For example, assume a set of transactions $\{T_0, T_1, T_2, \dots, T_n\}$. T_0 needs a resource X to complete its task. Resource X is held by T_1 , and T_1 is waiting for a resource Y, which is held by T_2 . T_2 is waiting for resource Z, which is held by T_0 . Thus, all the processes wait for each other to release resources. In this situation, none of the processes can finish their task. This situation is known as a deadlock.

Deadlocks are not healthy for a system. In case a system is stuck in a deadlock, the transactions involved in the deadlock are either rolled back or restarted.

DBMS - Data Recovery

Crash Recovery

DBMS is a highly complex system with hundreds of transactions being executed every second. The durability and robustness of a DBMS depends on its complex architecture and its underlying hardware and system software. If it fails or crashes amid transactions, it is expected that the system would follow some sort of algorithm or techniques to recover lost data.

Failure Classification

To see where the problem has occurred, we generalize a failure into various categories, as follows –

Transaction failure

A transaction has to abort when it fails to execute or when it reaches a point from where it can't go any further. This is called transaction failure where only a few transactions or processes are hurt.

Reasons for a transaction failure could be –

- **Logical errors** – Where a transaction cannot complete because it has some code error or any internal error condition.
- **System errors** – Where the database system itself terminates an active transaction because the DBMS is not able to execute it, or it has to stop because of some system condition. For example, in case of deadlock or resource unavailability, the system aborts an active transaction.

System Crash

There are problems – external to the system – that may cause the system to stop abruptly and cause the system to crash. For example, interruptions in power supply may cause the failure of underlying hardware or software failure.

Examples may include operating system errors.

Disk Failure

In early days of technology evolution, it was a common problem where hard-disk drives or storage drives used to fail frequently.

Disk failures include formation of bad sectors, unreachability to the disk, disk head crash or any other failure, which destroys all or a part of disk storage.

Recovery with Concurrent Transactions

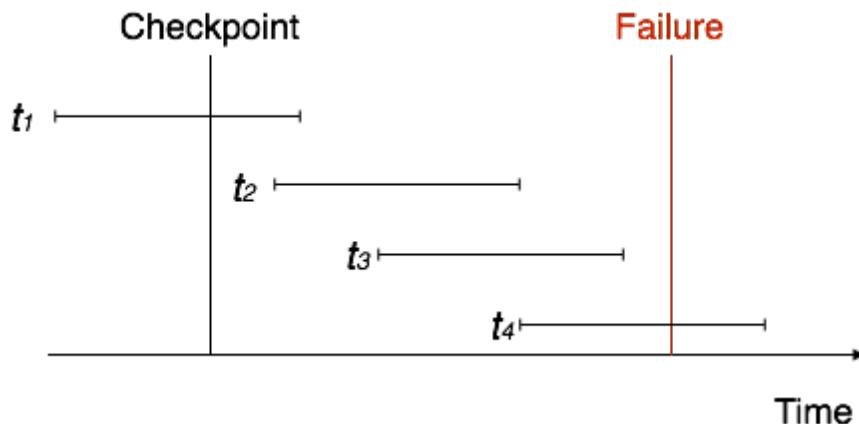
When more than one transaction are being executed in parallel, the logs are interleaved. At the time of recovery, it would become hard for the recovery system to backtrack all logs, and then start recovering. To ease this situation, most modern DBMS use the concept of 'checkpoints'.

Checkpoint

Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system. As time passes, the log file may grow too big to be handled at all. Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

Recovery

When a system with concurrent transactions crashes and recovers, it behaves in the following manner –



- The recovery system reads the logs backwards from the end to the last checkpoint.
- It maintains two lists, an undo-list and a redo-list.
- If the recovery system sees a log with $\langle T_n, \text{Start} \rangle$ and $\langle T_n, \text{Commit} \rangle$ or just $\langle T_n, \text{Commit} \rangle$, it puts the transaction in the redo-list.
- If the recovery system sees a log with $\langle T_n, \text{Start} \rangle$ but no commit or abort log found, it puts the transaction in undo-list.

All the transactions in the undo-list are then undone and their logs are removed. All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.

4. Reference text books/web material etc.,

<Please provide a detailed list of textbooks and reference materials relevant for the entire subject. Note: Please do not copy all the lecture level references. You are requested to avoid any references which may only be relevant to 1 or 2 lectures>

Textbooks:

1. *Database Management Systems, 3/e Raghuram Krishnan, Johannes Gehrke, TMH*
2. *Database Management System, 6/e Ramez Elmasri, Shamkant B. Navathe, PEA*
3. *Database Principles Fundamentals of Design Implementation and Management, Carlos Coronel, Steven Morris, Peter Robb, Cengage Learning*

Web-links (including videos):

1. <https://www.youtube.com/watch?v=1057YmExS-I>
2. <https://www.youtube.com/watch?v=o1HLZRtFwxk>
3. <https://www.youtube.com/watch?v=Wv1c9K4788A>

Please add if you have more...

Reference Journals:

1.

2.

Please add if you have more...

5. Mid Question Paper + Schemes of Evaluation.

<Please include atleast three mid 1 and three mid 2 papers with scheme of evaluation for each of them. Note: All of the papers are to be included here in this section as softcopy. Please do not attach scanned hardcopies at the end.>

6. Fast track material for Back-Log students.

Please include the following:

- A quick reference document (a maximum of 4 pages) to study the basic concepts related to the subject
- A minimum of 3 unique question papers with 8 questions each along with detailed solutions. These question papers need NOT have complex questions. The idea is to make the student clear the exam. Hence, please include only basic and average complexity questions after studying which a student can clear the exam
- Each student is expected to write each of the papers a minimum of 10 times
- Please include all the papers (softcopies) and solutions in this section. Please do NOT attach handwritten sheets at the end.

7. Sample Question Papers with solutions (Minimum 3)

- *A minimum of 3 unique question papers with 8 questions each along with detailed solutions. These question papers should follow bloom's taxonomy. They should also include basic, average and complex questions so that all the students including the toppers get benefitted by practicing with these papers. The basic objective is to help students get top marks so that they can end up in merit list!*

8. Virtual Labs if required

- Please provide the details of any virtual labs if applicable. Please provide the links to the videos etc. Please do NOT embed videos as it will increase the size of the documents.

9. *Mapping of Assignments / Question Papers with course objective learning outcomes.*

| <i>Course objective</i> | <i>Assignment 1</i> | <i>Assignment 2</i> | <i>Assignment 3</i> | <i>Assignment 4</i> | <i>Assignment 5</i> | <i>Add more columns if there are extra assignment s</i> |
|--|---------------------|---------------------|---------------------|---------------------|---------------------|---|
| <i>Please write Objective 1 from section 1</i> | <i>Yes / No</i> |
| <i>Please write Objective 2 from section 1</i> | <i>Yes / No</i> |
| <i>So on....</i> | | | | | | |

So on....

| | | | | | |
|------------------|--|--|--|--|--|
| <i>So on....</i> | | | | | |
|------------------|--|--|--|--|--|

10. Bloom's Taxonomy checklist

| Bloom's Digital Taxonomy Checklist | | | | |
|--|---|---|---|---|
| Check the skills that apply to determine the level at which the student is working | | | | |
| Remembering | Understanding | Applying | Analyzing | Evaluating |
| bookmarking choosing defining finding labeling listing matching recalling | comparing diagramming explaining interpreting outlining relating summarizing tagging | building constructing developing editing experimenting with modeling playing solving | classifying dissecting distinguishing examining inspecting linking mashing simplifying | appraising blogging critiquing defending disproving influencing interpreting measuring |