Database Management System

UNIT I – INTRODUCTION TO DBMS – 3 HRS

DBMS Overview

- A **database management system (DBMS)** is a software package designed to define, manipulate, retrieve and manage data in a database.
- A DBMS generally manipulates the data itself, the data format, field names, record structure and file structure. It also defines rules to validate and manipulate this data.
- A DBMS relieves users of framing programs for data maintenance. Fourth-generation query languages, such as SQL, are used along with the DBMS package to interact with a database.
- Some other DBMS examples include: MySQL, SQL Server, Oracle, dBASE, FoxPro, MS-Access etc.

Database

- A database is a collection of information that is organized so that it can be easily accessed, managed and updated. Data is organized into rows, columns and tables, and it is indexed to make it easier to find relevant information.

Characteristics of DBMS

- Provides security and removes redundancy (duplication)
- Insulation between programs and data abstraction
- Support of multiple views of the data
- Sharing of data and multiuser transaction processing
- DBMS allows entities and relations among them to form tables.
- It follows the ACID concept (Atomicity, Consistency, Isolation, and Durability).
- DBMS supports multi-user environment that allows users to access and manipulate data in parallel.

DBMS vs. Flat File

SN	DBMS	Flat File Management System
1	Multi-user access	It does not support multi-user access
2	Design to fulfill the need for small and large businesses	It is only limited to smaller DBMS system.
3	Remove redundancy and Integrity	Redundancy and Integrity issues
4	Expensive. But in the long term Total Cost of Ownership is cheap	It's cheaper
5	Easy to implement complicated transactions	No support for complicated transactions

Objective of DBMS

1. Eliminate redundant data.
2. Make access to the data easy for the user.
3. Provide for mass storage of relevant data.
4. Protect the data from physical harm and un-authorized systems.
5. Allow for growth in the data base system.
6. Make the latest modifications to the data base available immediately.
7. Allow for multiple users to be active at one time.
8. Provide prompt response to user requests for data.

Importance of DBMS

- A database management system is important because it manages data efficiently and allows users to perform multiple tasks with ease.
- A database management system stores, organizes and manages a large amount of information within a single software application. Use of this system increases efficiency of business operations and reduces overall costs.
- Database management systems are important to businesses and organizations because they provide a highly efficient method for handling multiple types of data.
- Some of the data that are easily managed with this type of system include: employee records, student information, payroll, accounting, project management, inventory and library books. These systems are built to be extremely versatile.
- Without database management, tasks have to be done manually and take more time. Data can be categorized and structured to suit the needs of the company or organization.
- Data is entered into the system and accessed on a routine basis by assigned users. Each user may have an assigned password to gain access to their part of the system. Multiple users can use the system at the same time in different ways.

Advantages of DBMS

- DBMS offers a variety of techniques to store & retrieve data
- DBMS serves as an efficient handler to balance the needs of multiple applications using the same data
- Application programmers never exposed to details of data representation and storage.
- A DBMS uses various powerful functions to store and retrieve data efficiently.
- Offers Data Integrity and Security
- The DBMS implies integrity constraints to get a high level of protection against prohibited access to data.
- A DBMS schedules concurrent access to the data in such a manner that only one user can access the same data at a time
- Reduced Application Development Time

Disadvantages of DBMS

- Cost of Hardware and Software of a DBMS is quite high which increases the budget of your organization.
- Most database management systems are often complex systems, so the training for users to use the DBMS is required.
- In some organizations, all data is integrated into a single database which can be damaged because of electric failure or database is corrupted on the storage media
- Use of the same program at a time by many users sometimes lead to the loss of some data.
- DBMS can't perform sophisticated calculations

Application of DBMS

SN	Sector	Use of DBMS
1	Banking	For customer information, account activities, payments, deposits, loans, etc.
2	Airlines	For reservations and schedule information.
3	Universities	For student information, course registrations, colleges and grades.
4	Telecommunication	It helps to keep call records, monthly bills, maintaining balances, etc.
5	Finance	For storing information about stock, sales, and purchases of financial instruments like stocks and bonds.
6	Sales	Use for storing customer, product & sales information.
7	Manufacturing	It is used for the management of supply chain and for tracking production of items. Inventories status in warehouses.
8	HR Management	For information about employees, salaries, payroll, deduction, generation of paychecks, etc.

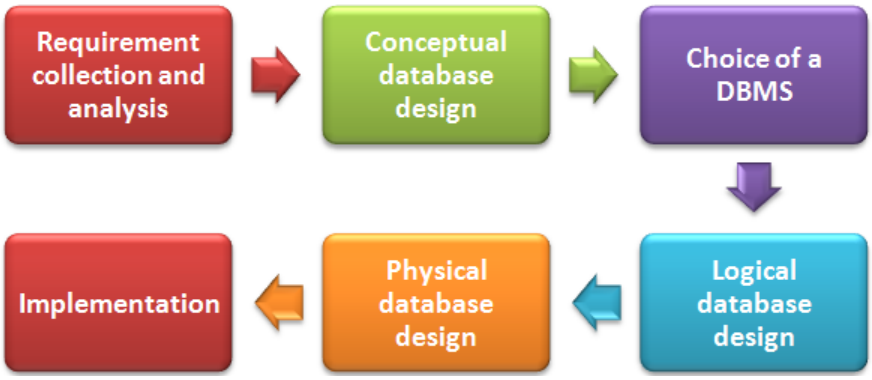
UNIT II – Database Design, Architecture & Model – 6 HRS

Overview of Database Design Process

Database systems are designed to manage large bodies of information. Database design mainly involves the design of the database schema. The design of a complete database application environment that meets the needs of the enterprise being modeled requires attention to a broader set of issues.

In this text, we focus initially on the writing of database queries and the design of database schemas. Database Design is a collection of processes that facilitate the designing, development, implementation and maintenance of database systems. It helps produce database systems

1. That meet the requirements of the users
2. Have high performance.



- A high-level data model provides the database designer with a conceptual framework in which to specify the data requirements of the database users, and how the database will be structured to fulfill these requirements.
- The **initial phase** of database design, then, is to characterize fully the data needs of the prospective database users. The database designer needs to interact extensively with domain experts and users to carry out this task. The **outcome** of this phase is a **specification of user requirements**.
- Next, the designer chooses a data model, and by applying the concepts of the chosen data model, translates these requirements into a **conceptual schema of the database**.
- The schema developed at this **conceptual-design phase** provides a detailed overview of the enterprise. The designer reviews the schema to confirm that all data requirements are indeed satisfied and are not in conflict with one another. The designer can also examine the design to remove any redundant features. The focus at this point is on describing the data and their relationships, rather than on specifying physical storage details.
- In terms of the relational model, the conceptual-design process involves decisions on what attributes we want to capture in the database and how to group these attributes to form the various tables.

- A fully developed conceptual schema indicates the functional requirements of the enterprise. In a specification of functional requirements, users describe the kinds of operations (or transactions) that will be performed on the data.
- Example operations include modifying or updating data, searching for and retrieving specific data, and deleting data. At this stage of conceptual design, the designer can review the schema to ensure it meets functional requirements.
- The process of moving from an abstract data model to the implementation of the database proceeds in **two final design phases**.
- In the **logical-design phase**, the designer maps the high-level conceptual schema onto the implementation data model of the database system that will be used.
- The designer uses the resulting system-specific database schema in the subsequent **physical-design phase**, in which the physical features of the database are specified.

Database Design for a University Organization

To illustrate the design process, let us examine how a database for a university could be designed. The initial specification of user requirements may be based on interviews with the database users, and on the designer's own analysis of the organization.

The description that arises from this design phase serves as the basis for specifying the conceptual structure of the database. Here are the **major characteristics** of the university.

- ✓ The university is organized into departments. Each department is identified by a unique name (dept name), is located in a particular building, and has a budget.
- ✓ Each department has a list of courses it offers. Each course has associated with it a course id, title, dept name, and credits.
- ✓ Instructors are identified by their unique ID. Each instructor has name, associated department (dept name), and salary.
- ✓ Students are identified by their unique ID. Each student has a name, an associated major department (dept name), and tot cred (total credit hours the student earned thus far).
- ✓ The university maintains a list of classrooms, specifying the name of the building, room number, and room capacity.
- ✓ The university maintains a list of all classes (sections) taught. Each section is identified by a course id, sec id, year, and semester, and has associated with it a semester, year, building, room number, and time slot id (the time slot when the class meets).
- ✓ The department has a list of teaching assignments specifying, for each instructor, the sections the instructor is teaching.
- ✓ The university has a list of all student course registrations, specifying, for each student, the courses and the associated sections that the student has taken (registered for).

View of Data - Data Abstraction

A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data. A major purpose of a database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained.

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database.
Since many database-system users are not computer trained, developers hide the complexity from users through **several levels of abstraction**, to simplify users' interactions with the system:

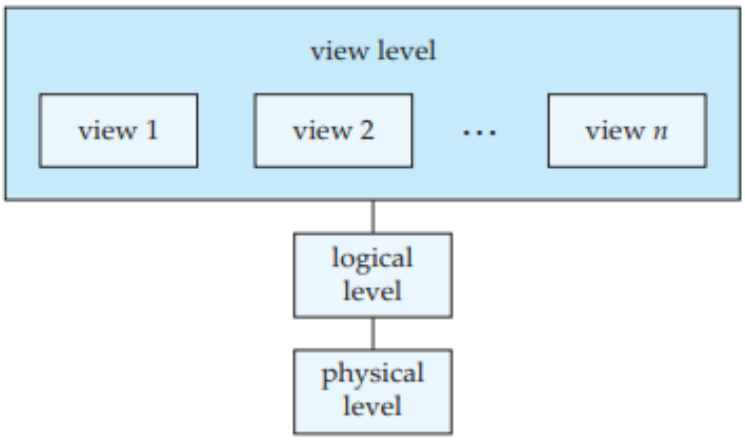


Figure 1.1 The three levels of data abstraction.

- ✓ **Physical level.** The lowest level of abstraction describes how the data are actually stored. The physical level describes complex low-level data structures in detail.

- ✓ **Logical level.** The next-higher level of abstraction describes what data are stored in the database, and what relationships exist among those data. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.

- ✓ **View level.** The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.

Data Models

Data Model a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. A data model provides a way to describe the design of a database at the physical, logical, and view levels.
The data models can be classified into four different categories:

Relational Model

- ✓ The relational model uses a collection of tables to represent both data and the relationships among those data.
- ✓ Each table has multiple columns, and each column has a unique name. Tables are also known as relations. Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes.
- ✓ The columns of the table correspond to the attributes of the record type. The relational data model is the most widely used data model, and a vast majority of current database systems are based on the relational model.

Entity-Relationship Model.

The entity-relationship (E-R) data model uses a collection of basic objects, called entities, and relationships among these objects. An entity is a “thing” or “object” in the real world that is distinguishable from other objects. The entity-relationship model is widely used in database design.

Object-Based Data Model.

Object-oriented programming (especially in Java, C++, or C#) has become the dominant software-development methodology. This led to the development of an object-oriented data model that can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity. The object-relational data model combines feature of the object-oriented data model and relational data model.

Semistructured Data Model.

The semistructured data model permits the specification of data where individual data items of the same type may have different sets of attributes. The Extensible Markup Language (XML) is widely used to represent semistructured data.

Historically, the network data model and the hierarchical data model preceded the relational data model. These models were tied closely to the underlying implementation, and complicated the task of modeling data. As a result, they are used little now, except in old database code that is still in service in some places.

Database Languages

A database system provides a **data-definition language** to specify the database schema and a **data-manipulation language** to express database queries and updates.

Data-Manipulation Language(DML)

A **data-manipulation language (DML)** is a language that enables users to access or manipulate data as organized by the appropriate data model.

Examples of DML:

- **SELECT** – is used to retrieve data from the a database.
- **INSERT** – is used to insert data into a table.
- **UPDATE** – is used to update existing data within a table.
- **DELETE** – is used to delete records from a database table.

There are basically two types:

- ✓ **Procedural DMLs** require a user to specify what data are needed and how to get those data.
- ✓ **Declarative DMLs (also referred to as nonprocedural DMLs)** require a user to specify what data are needed without specifying how to get those data.

Data-Definition Language (DDL)

DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema.

It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in database.

Examples of DDL commands:

- **CREATE** – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
- **DROP** – is used to delete objects from the database.
- **ALTER**-is used to alter the structure of the database.
- **TRUNCATE**–is used to remove all records from a table, including all spaces allocated for the records are removed.
- **COMMENT** –is used to add comments to the data dictionary.
- **RENAME** –is used to rename an object existing in the database.

Data Control Language (DCL)

DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

Examples of DCL commands:

- **GRANT**-gives user's access privileges to database.
- **REVOKE**-withdraw user's access privileges given by using the GRANT command.

Transaction Control Language (TCL)

TCL commands deals with the transaction within the database.

Examples of TCL commands:

- **COMMIT**– commits a Transaction.
- **ROLLBACK**– rollbacks a transaction in case of any error occurs.

- **SAVEPOINT**– sets a save point within a transaction.
- **SET TRANSACTION**– specify characteristics for the transaction.

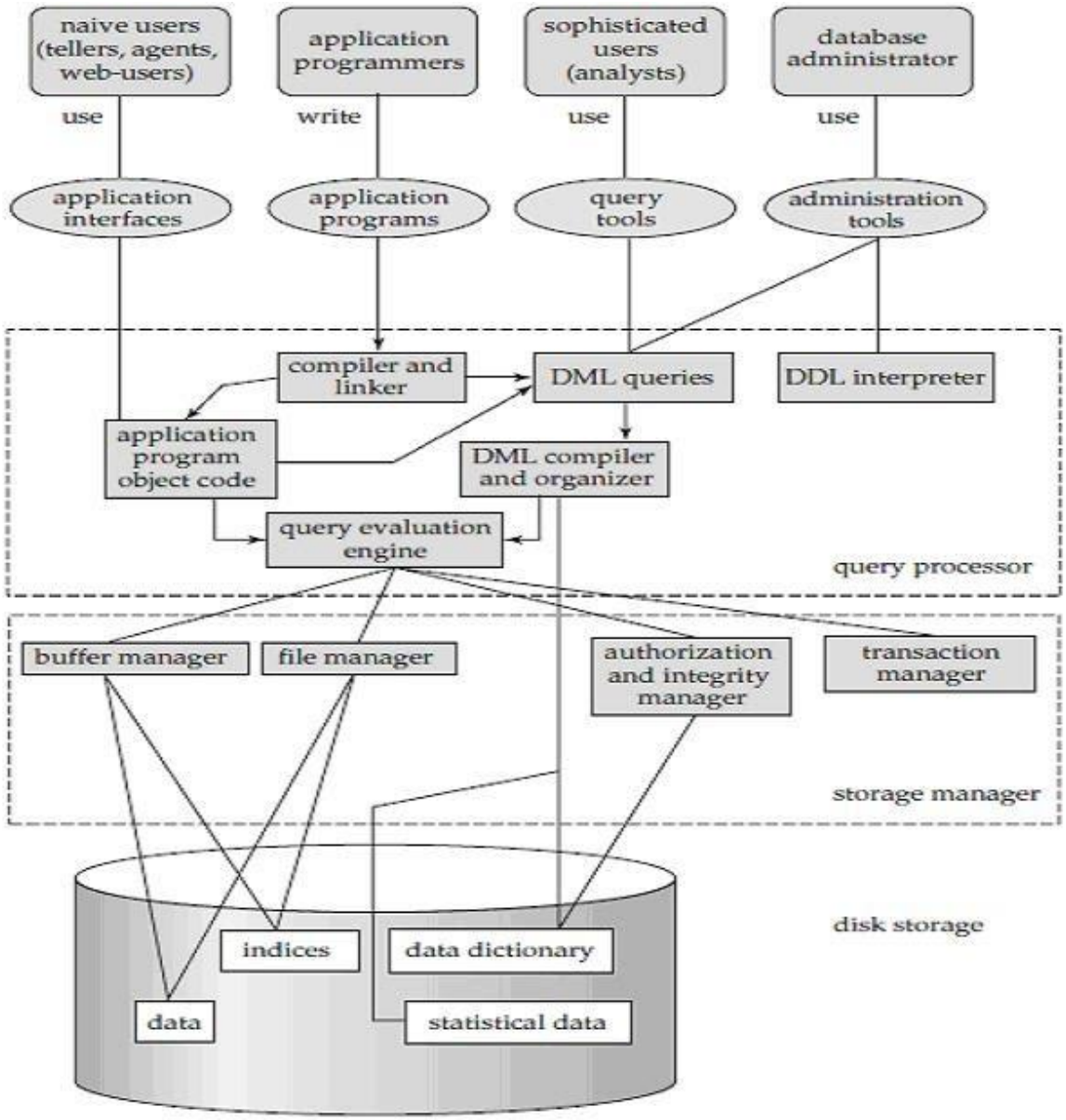
Query by Example (QBE)

- ✓ **Query by example** is a query language used in relational databases that allows users to search for information in tables and fields by providing a simple user interface where the user will be able to input an example of the data that he or she wants to access.
- ✓ The principle of QBE is that it is merely an abstraction between the user and the real query that the database system will receive. In the background, the user's query is transformed into a database manipulation language form such as SQL, and it is this SQL statement that will be executed in the background.
- ✓ (QBE) is a method of query creation that allows the user to search for documents based on an example in the form of a selected text string or in the form of a document name or a list of documents.
- ✓ Because the QBE system formulates the actual query, QBE is easier to learn than formal query languages, such as the standard Structured Query Language (SQL), while still enabling powerful searches.
- ✓ In terms of database management system, QBE can be thought of as a "fill-in-the blanks" method of query creation. The Microsoft Access Query Design Grid is an example. To conduct a search for field data matching particular conditions, the user enters criteria into the form, creating search conditions for as many fields as desired. A query is automatically generated to search the database for matching data.

DBMS Structure

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.
- ✓ The **query processor** is important because it helps the database system simplify and facilitate access to data.



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.

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.
- ✓ It 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.

Transaction Manager

- ✓ A transaction is a collection of operations that performs a single logical function in a database application.
- ✓ Each transaction is a unit of both atomicity and consistency. Thus, we require that transactions do not violate any database-consistency constraints.
- ✓ That is, if the database was consistent when a transaction started, the database must be consistent when the transaction successfully terminates.
- ✓ Transaction - manager ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.

DBMS Architecture

DBMS architecture helps in design, development, implementation, and maintenance of a database. A database stores critical information for a business. Selecting the correct Database Architecture helps in quick and secure access to this data.

1 Tier Architecture

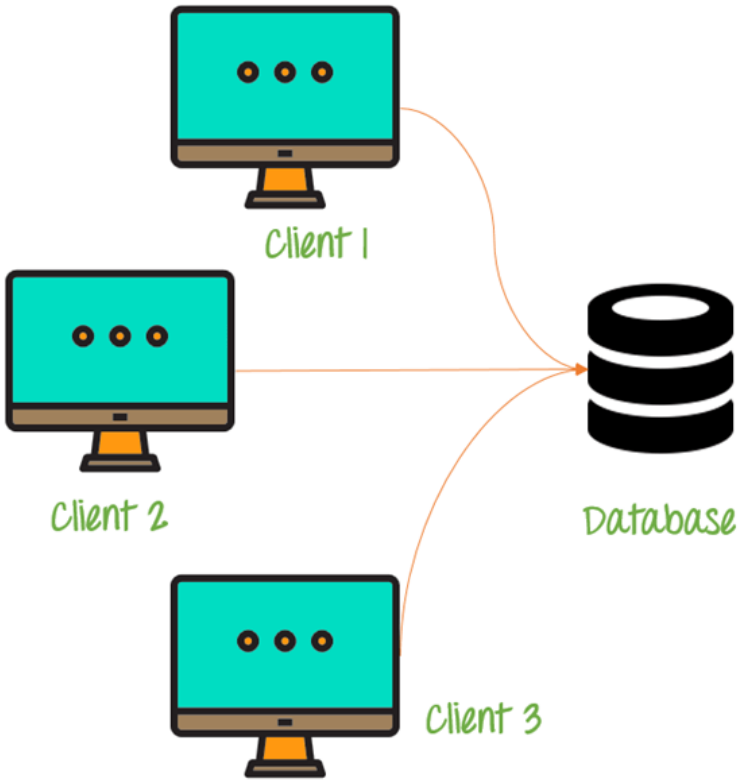
Single Tier Architecture

- ✓ The simplest of Database Architecture are 1 tier where the Client, Server, and Database all reside on the same machine.
- ✓ Anytime you install a DB in your system and access it to practise SQL queries it is 1 tier architecture. But such architecture is rarely used in production.

2 Tier Architecture

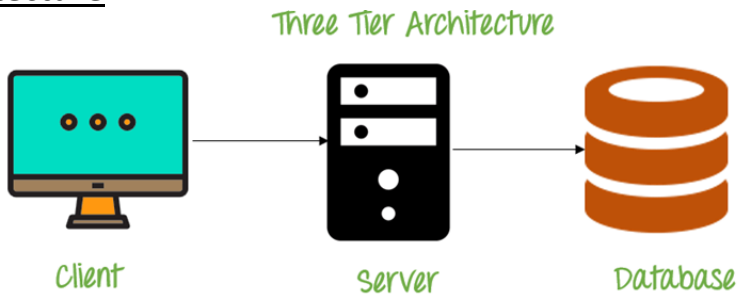
A two-tier architecture is a database architecture where:

1. Presentation layer runs on a client (PC, Mobile, Tablet, etc.)
2. Data is stored on a Server.



- ✓ An application interface which is called ODBC (Open Database Connectivity) an API which allows the client-side program to call the DBMS.
- ✓ Today most of the DBMS offers ODBC drivers for their DBMS. 2 tier architecture provides added security to the DBMS as it is not exposed to the end user directly.

3 Tier Architecture



3-tier schema is an extension of the 2-tier architecture. 3-tier architecture has following layers:

1. Presentation layer (your PC, Tablet, Mobile, etc.)
2. Application layer (server)
3. Database Server

This DBMS architecture contains an Application layer between the user and the DBMS, which is responsible for communicating the user's request to the DBMS system and send the response from the DBMS to the user.
The application layer (business logic layer) also processes functional logic, constraint, and rules before passing data to the user or down to the DBMS Three tier architecture is the most popular DBMS architecture.

The goal of Three-tier architecture is:

- To separate the user applications and physical database
- Proposed to support DBMS characteristics
- Program-data independence
- Support of multiple views of the data

Example of Three-tier Architecture is any large website on the internet.

Data Independence

The main purpose of **data abstraction** is achieving **data independence** in order to save time and cost required when the database is modified or altered.
Data independence is the ability of to make changes to **data** characteristics without have to make changes to the programs that access the **data**. It's **important** because of the savings in time and potential errors caused by reducing modifications to **data** access software.

Physical level data independence:

It refers to the characteristic of being able to modify the physical schema without any alterations to the conceptual or logical schema.
These alterations or modifications to the physical structure may include:

- Utilising new storage devices.
- Modifying data structures used for storage.
- Altering indexes or using alternative file organisation techniques etc.

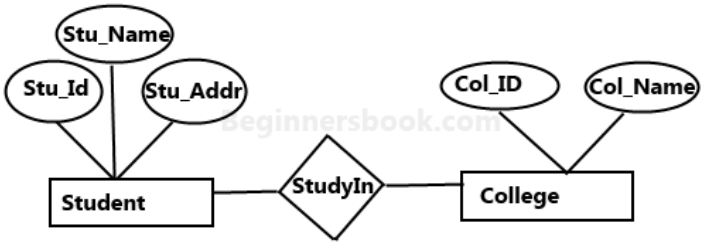
Logical level data independence:

It refers characteristic of being able to modify the logical schema without affecting the external schema or application program. The user view of the data would not be affected by any changes to the conceptual view of the data.
These changes may include insertion or deletion of attributes, altering table structures entities or relationships to the logical schema etc.

ER Model - Basic Concepts

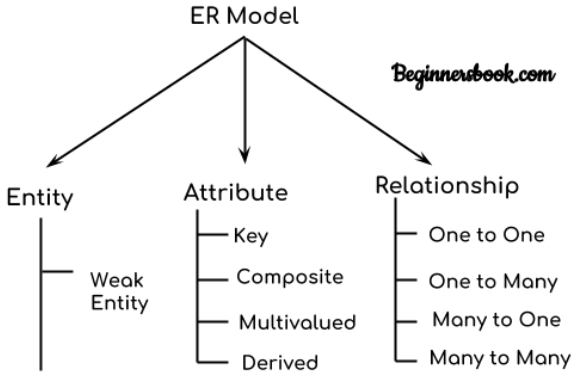
The **ER model** defines the conceptual view of a database. It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases.

Entity Relationship Diagram, also known as **ERD**, **ER Diagram** or **ER** model, is a type of structural **diagram** for use in database design. An **ERD** contains different symbols and connectors that visualize two important information: The major entities within the system scope, and the inter-relationships among these entities.



Sample E-R Diagram

Components of a ER Diagram

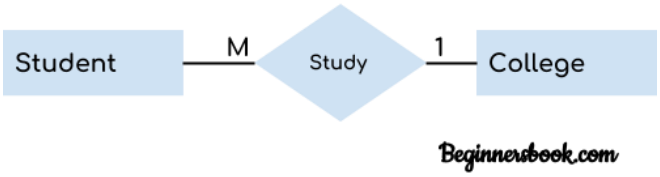


Components of ER Diagram

- ✓ **Rectangle**: Represents Entity sets.
- ✓ **Ellipses**: Attributes
- ✓ **Diamonds**: Relationship Set
- ✓ **Lines**: They link attributes to Entity Sets and Entity sets to Relationship Set
- ✓ **Double Ellipses**: Multivalued Attributes
- ✓ **Dashed Ellipses**: Derived Attributes
- ✓ **Double Rectangles**: Weak Entity Sets
- ✓ **Double Lines**: Total participation of an entity in a relationship set

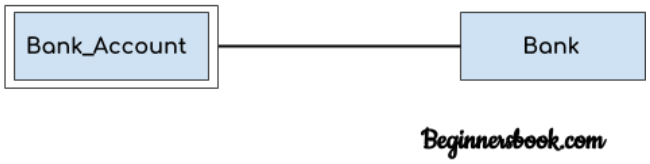
1. Entity

- ✓ An entity is an object or component of data. An entity is represented as rectangle in an ER diagram.
- ✓ For example: In the following ER diagram we have two entities Student and College and these two entities have many to one relationship as many student's study in a single college. We will read more about relationships later, for now focus on entities.



Weak Entity:

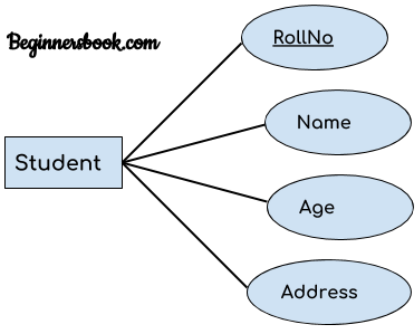
- ✓ An entity that cannot be uniquely identified by its own attributes and relies on the relationship with other entity is called weak entity. The weak entity is represented by a double rectangle. For example – a bank account cannot be uniquely identified without knowing the bank to which the account belongs, so bank account is a weak entity.



2. Attribute

An attribute describes the property of an entity. An attribute is represented as Oval in an ER diagram. There are four types of attributes:

a) Key attribute
b) Composite attribute
c) Multivalued attribute
d) Derived attribute

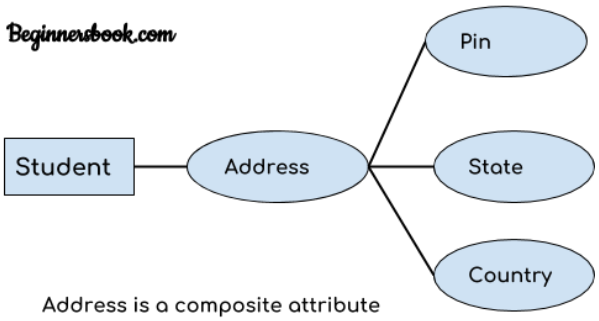


a) Key attribute

- ✓ A key attribute can uniquely identify an entity from an entity set. For example, student roll number can uniquely identify a student from a set of students.
- ✓ Key attribute is represented by oval same as other attributes however the **text of key attribute is underlined**.

b) Composite attribute

- ✓ An attribute that is a combination of other attributes is known as composite attribute. For example,
- ✓ In student entity, the student address is a composite attribute as an address is composed of other attributes such as pin code, state, country.

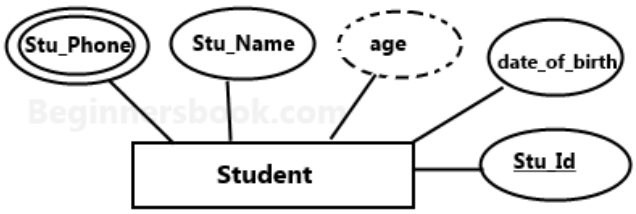


c) Multivalued attribute

- ✓ An attribute that can hold multiple values is known as multivalued attribute. It is represented with double ovals in an ER Diagram.
- ✓ For example – A person can have more than one phone numbers so the phone number attribute is multivalued.

d) Derived attribute

- ✓ A derived attribute is one whose value is dynamic and derived from another attribute. It is represented by dashed oval in an ER Diagram. For example – Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).



3. Relationship

A relationship is represented by diamond shape in ER diagram, it shows the relationship among entities. There are four types of relationships:

a) One to One
b) One to Many
c) Many to One
d) Many to Many

a) One to One

- ✓ When a single instance of an entity is associated with a single instance of another entity then it is called one to one relationship. For example, a person has only one passport and a passport is given to one person.

Beginnersbook.com

b) One to Many

- ✓ When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationships. For example – a customer can place many orders but an order cannot be placed by many customers.

Beginnersbook.com

c) Many to One

- ✓ When more than one instances of an entity is associated with a single instance of another entity then it is called many to one relationship. For example – many students can study in a single college but a student cannot study in many colleges at the same time.

Beginnersbook.com

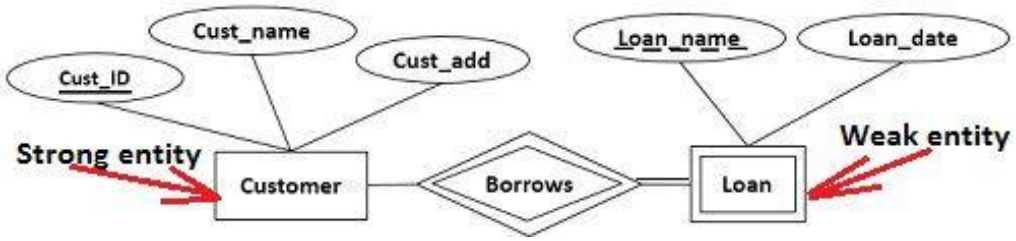
d) Many to Many

- ✓ When more than one instances of an entity are associated with more than one instances of another entity then it is called many to many relationships.
- ✓ For example, a can be assigned to many projects and a project can be assigned to many students.

Beginnersbook.com

Strong Entity Set Vs Weak Entity Set

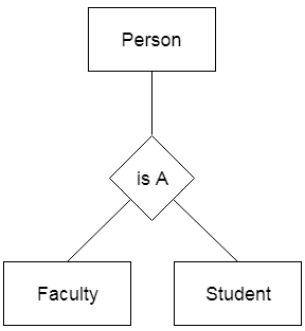
1. The basic difference between strong entity and a weak entity is that the strong entity has a **primary key** whereas, a weak entity has the **partial key** which acts as a discriminator between the entities of a weak entity set.
2. A weak entity always **depends** on the strong entity for its existence whereas, a strong entity is **independent** of any other entity's existence.
3. A strong entity is denoted with a **single rectangle** and a weak entity is denoted with a **double rectangle**.
4. The relationship between two strong entities is denoted with **single diamond** whereas, a relationship between a weak and a strong entity is denoted with double diamond called **Identifying Relationship**.

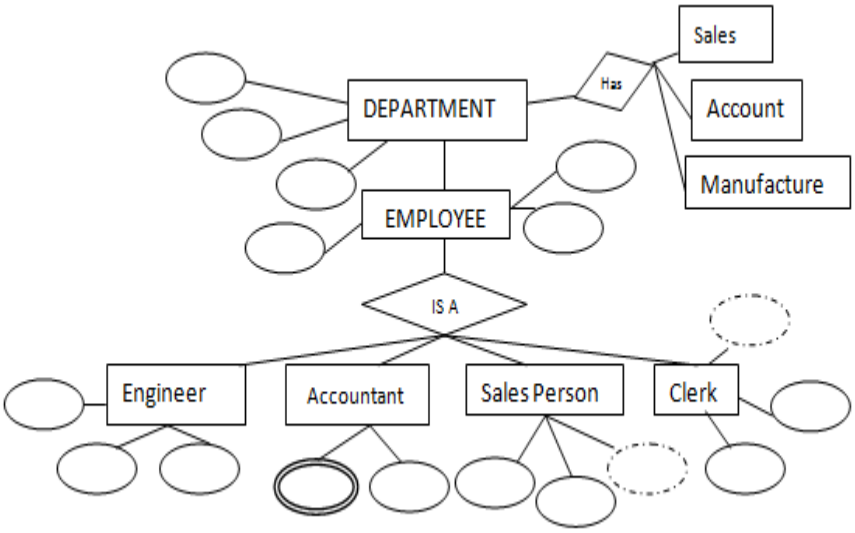


Generalization

Generalization is like a bottom-up approach in which **two or more entities of lower level** combine to form **a higher level entity** if they have some attributes in common. In generalization, an entity of a higher level can also combine with the entities of the lower level to form a further higher level entity.

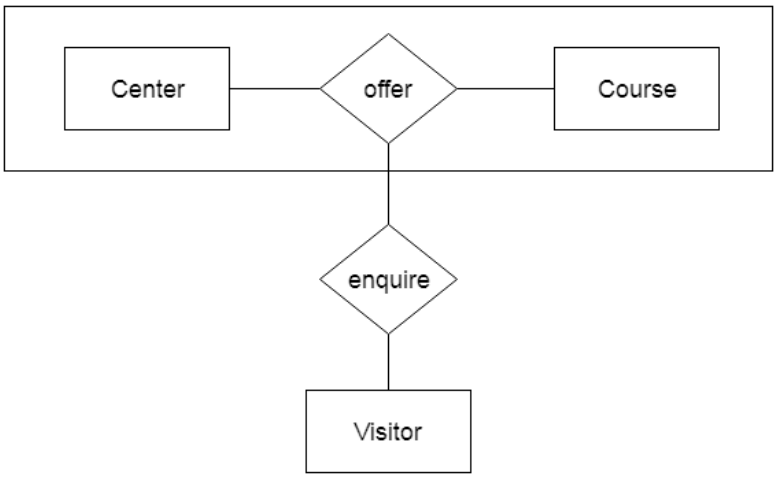
Generalization is more like subclass and superclass system, but the only difference is the approach. Generalization uses the bottom-up approach. In generalization, entities are combined to form a more generalized entity, i.e., subclasses are combined to make a superclass. **For example**, **Faculty and Student** entities can be generalized and create a higher level entity **Person**.

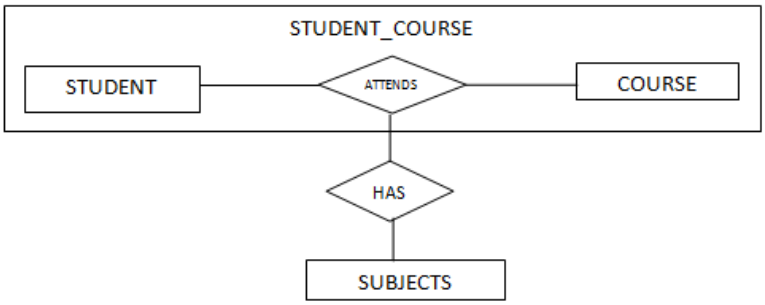




Aggregation

In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher level entity.





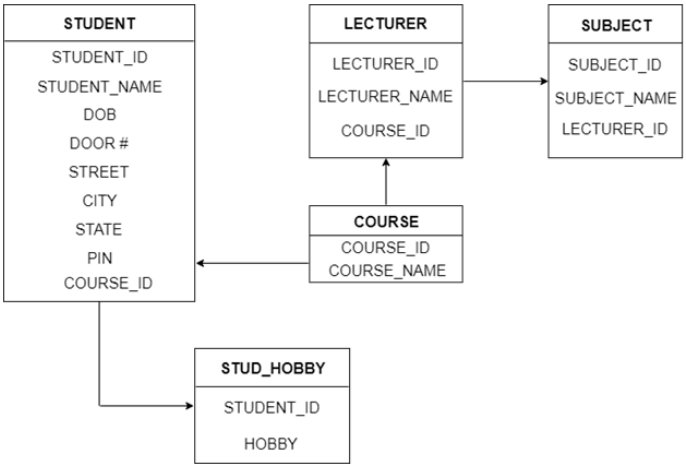
For example: Center entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity visitor.

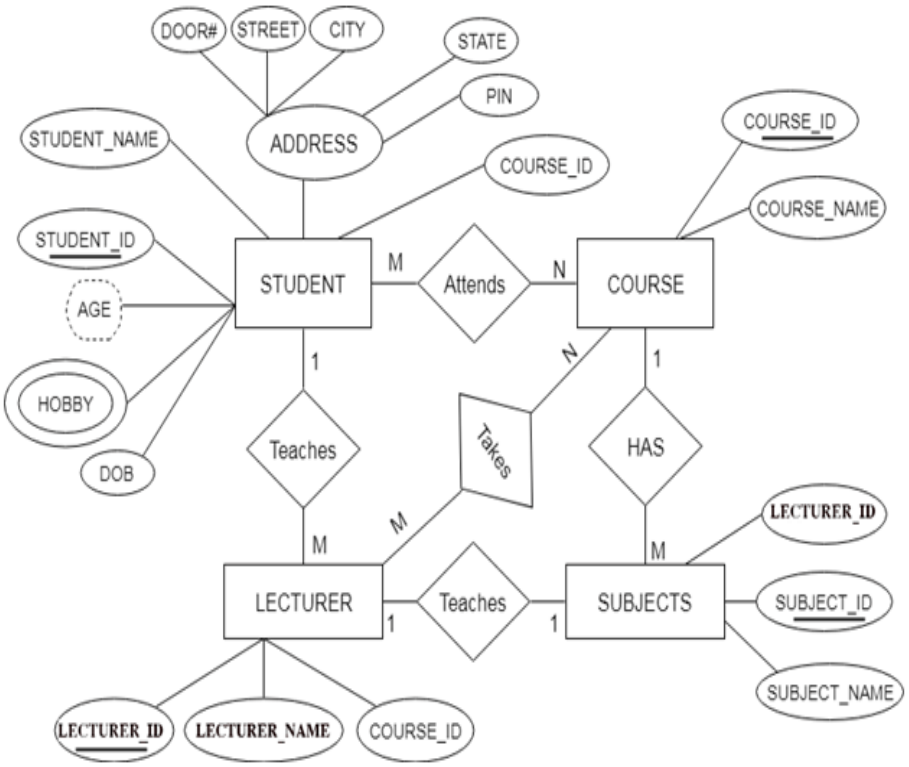
In the real world, if a visitor visits a coaching center then he will never enquiry about the Course only or just about the Center instead he will ask the enquiry about both.

Converting ER-Diagram to Table

Rules:

- ✓ Entity type becomes a table.
- ✓ All single-valued attribute becomes a column for the table.
- ✓ A key attribute of the entity type represented by the primary key.
- ✓ The multivalued attribute is represented by a separate table.
- ✓ Composite attribute represented by components.
- ✓ Derived attributes are not considered in the table.





UNIT III – RELATIONAL DATABASE MODEL - 4 HRS

Structure of RDBMS

A **relational database** is a type of database. It uses a structure that allows us to identify and access data in relation to another piece of data in the database. Often, data in a relational database is organized into tables.

A **relational database management system (RDBMS)** is a program that allows you to create, update, and administer a relational database. Most relational database management systems use the SQL language to access the database. MySQL, SQL-Server, MS-Access, Oracle are some of popular RDBMS.

Examples of RDBMS structure:

CUSTOMERS	
customer_id	customer_name
101	John Doe
102	Bruce Wayne

ORDERS			
order_id	customer_id	order_date	amount
555	101	12/24/09	$156.78
556	102	12/25/09	$99.99
557	101	12/26/09	$75.00

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

Table 1

STUDENT_COURSE

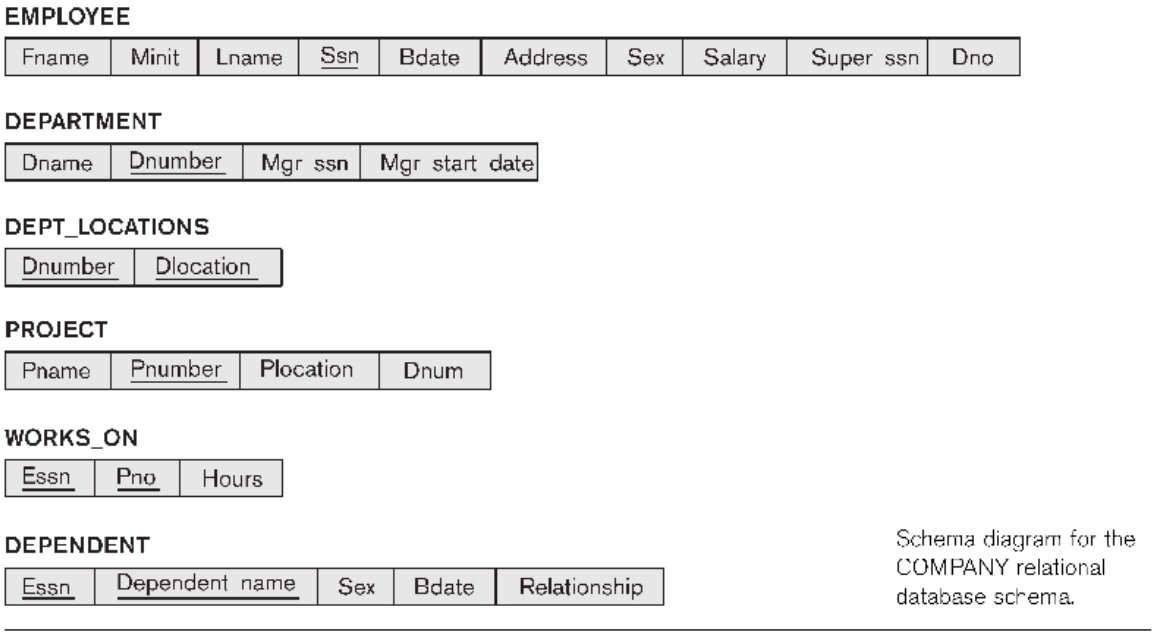
STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computer Networks
1	C2	Computer Networks

Table 2

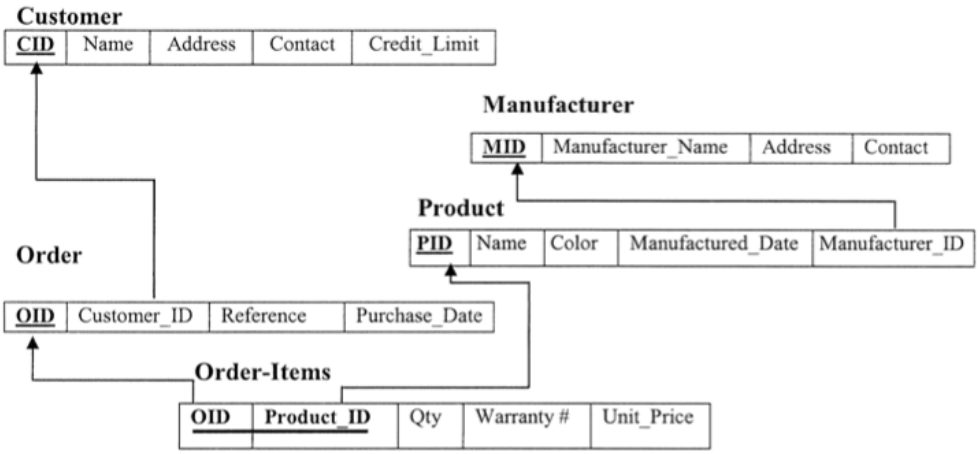
Database Schema

- ✓ 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.

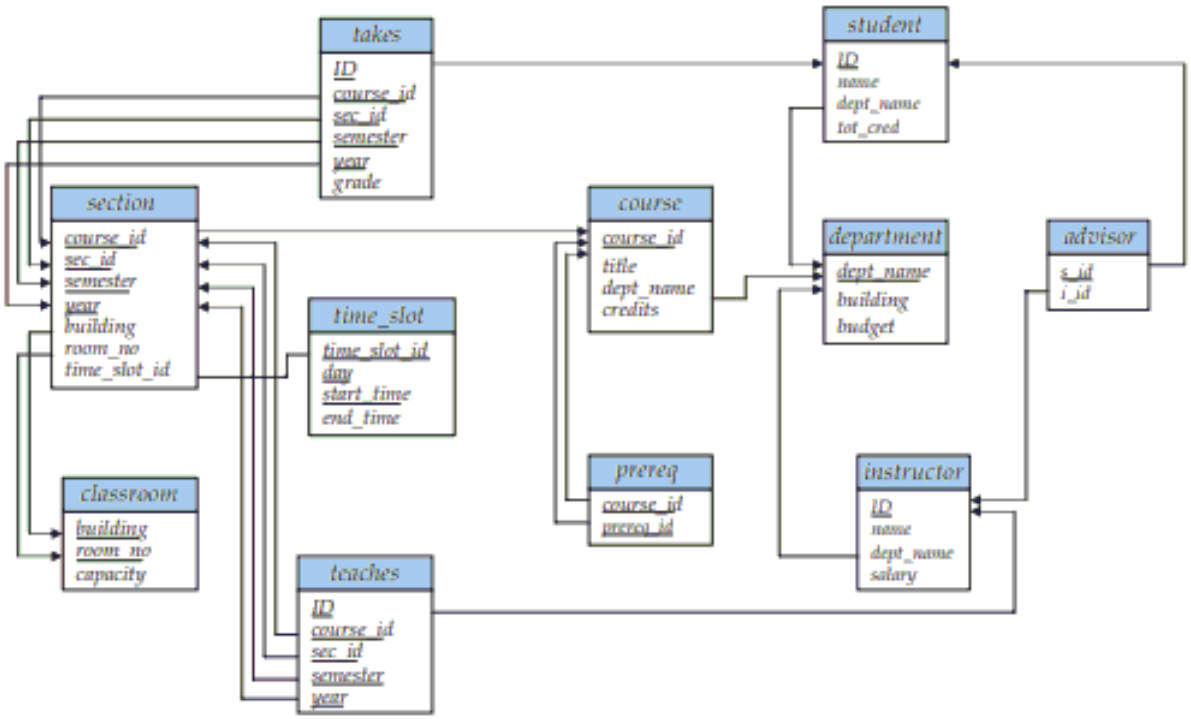
Schema Diagram for Employee Database



Schema diagram for the COMPANY relational database schema.



Schema Diagram for University Database



classroom(<u>*building*</u>, <u>*room_number*</u>, *capacity*)
department(<u>*dept_name*</u>, *building*, *budget*)
course(<u>*course_id*</u>, *title*, *dept_name*, *credits*)
instructor(<u>*ID*</u>, *name*, *dept_name*, *salary*)
section(<u>*course_id*</u>, <u>*sec_id*</u>, <u>*semester*</u>, <u>*year*</u>, *building*, *room_number*, *time_slot_id*)
teaches(<u>*ID*</u>, <u>*course_id*</u>, <u>*sec_id*</u>, <u>*semester*</u>, <u>*year*</u>)
student(<u>*ID*</u>, *name*, *dept_name*, *tot_cred*)
takes(<u>*ID*</u>, <u>*course_id*</u>, <u>*sec_id*</u>, <u>*semester*</u>, <u>*year*</u>, *grade*)
advisor(<u>*s_ID*</u>, *i_ID*)
time_slot(<u>*time_slot_id*</u>, <u>*day*</u>, <u>*start_time*</u>, *end_time*)
prereq(<u>*course_id*</u>, <u>*prereq_id*</u>)

Schema of the university database.

Keys

A **DBMS key** is an attribute or set of an attribute which helps you to identify a row(tuple) in a relation(table). They allow you to find the relation between two tables. Keys help you uniquely identify a row in a table by a combination of one or more columns in that table.

1. Super key

A **superkey** is a group of single or multiple keys which identifies rows in a table. A Super key may have additional attributes that are not needed for unique identification.

Example:

EmpSSN	EmpNum	Empname
9812345098	AB05	Shown
9876512345	AB06	Roslyn
199937890	AB07	James

In the above-given example, **EmpSSN** and **EmpNum** name are **superkeys**.

2. Primary Key

- ✓ A column or group of columns in a table which helps us to uniquely identifies every row in that table is called a primary key.
- ✓ This DBMS can't be a duplicate. The same value can't appear more than once in the table.

Rules for defining Primary key:

- Two rows can't have the same primary key value
- It must for every row to have a primary key value.
- The primary key field cannot be null.
- The value in a primary key column can never be modified or updated if any foreign key refers to that primary key.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com

- ✓ Here, **StudId** is a Primary Key.

3. Candidate Key

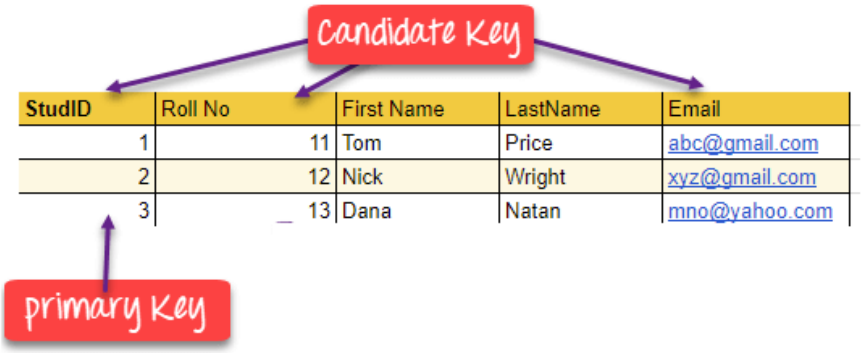
- ✓ A super key with no repeated attribute is called **candidate key**.
- ✓ The Primary key should be selected from the candidate keys. Every table must have at least a single candidate key.

Properties of Candidate key:

- ✓ It must contain unique values
- ✓ Candidate key may have multiple attributes
- ✓ Must not contain null values
- ✓ It should contain minimum fields to ensure uniqueness
- ✓ Uniquely identify each record in a table

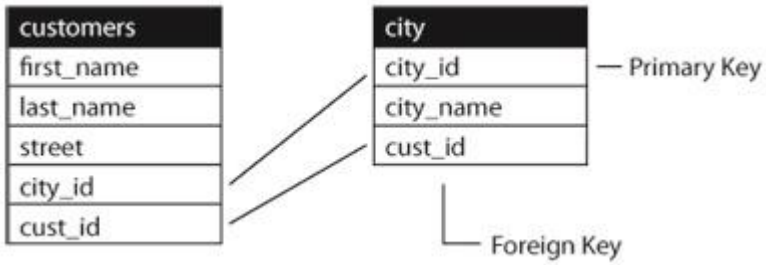
Example: In the given table Stud ID, Roll No, and email are candidate keys which help us to uniquely identify the student record in the table.

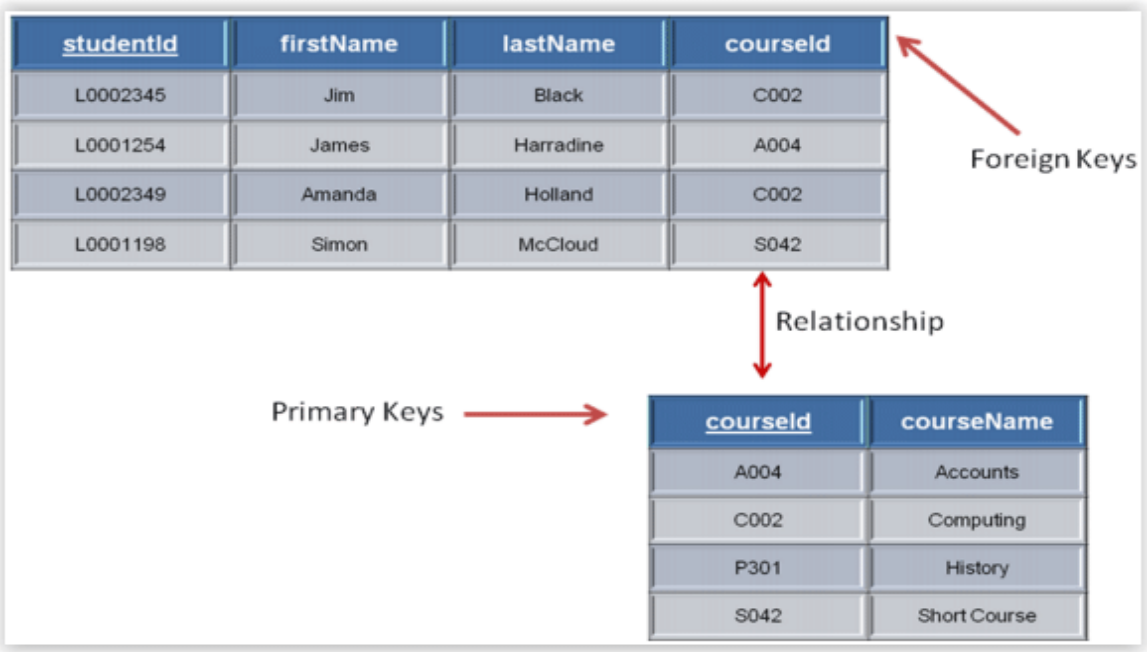
StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com



4. Foreign key

- ✓ A foreign key is a column which is added to create a relationship with another table. Foreign keys help us to maintain data integrity and also allows navigation between two different instances of an entity.
- ✓ Every relationship in the model needs to be supported by a foreign key.





5. Composite key

- ✓ A key which has multiple attributes to uniquely identify rows in a table is called a composite key.

Composite Key

ITEM

Supplier_ID	Item_ID	Item_Name	Quantity
S_1	I_1	AC	5
S_1	I_2	Inverter	8
S_2	I_2	Inverter	4
S_2	I_3	UPS	15
S_2	I_4	Generator	5
S_3	I_3	UPS	10

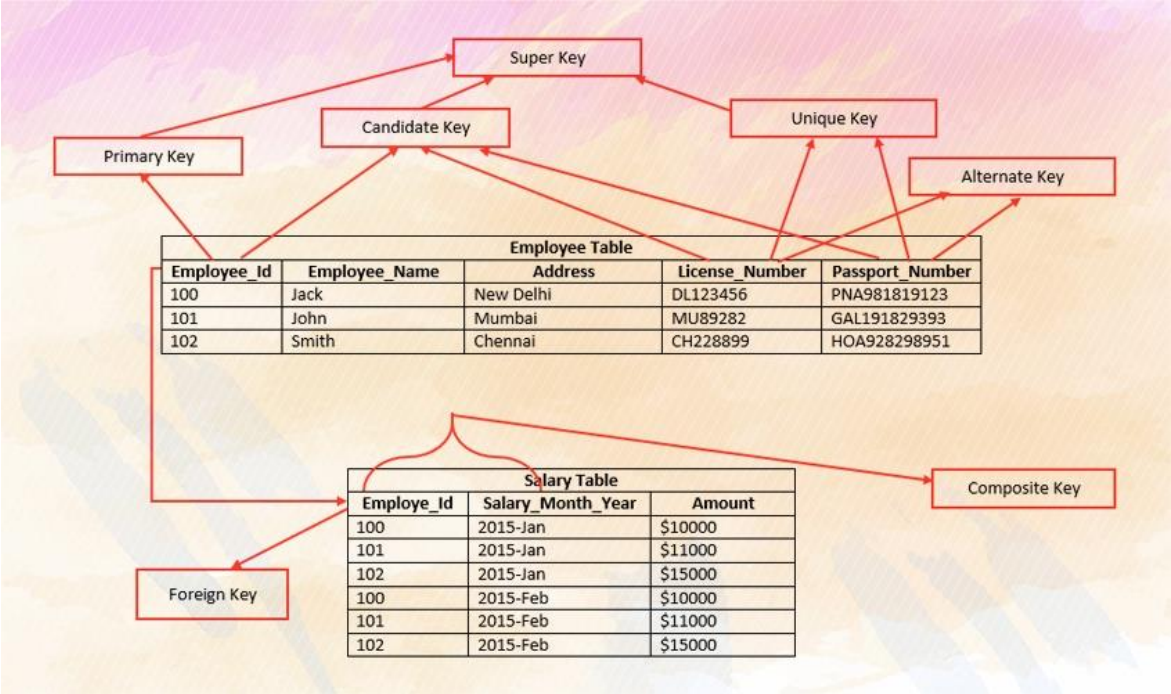
6. Alternate key

- ✓ All the keys which are not primary key are called an alternate key. It is a key which is currently not the primary key.
- ✓ However, a table may have single or multiple choices for the primary key.

StudID, Roll No, Email are qualified to become a primary key. But since StudID is the primary key, Roll No, Email becomes the alternative key.

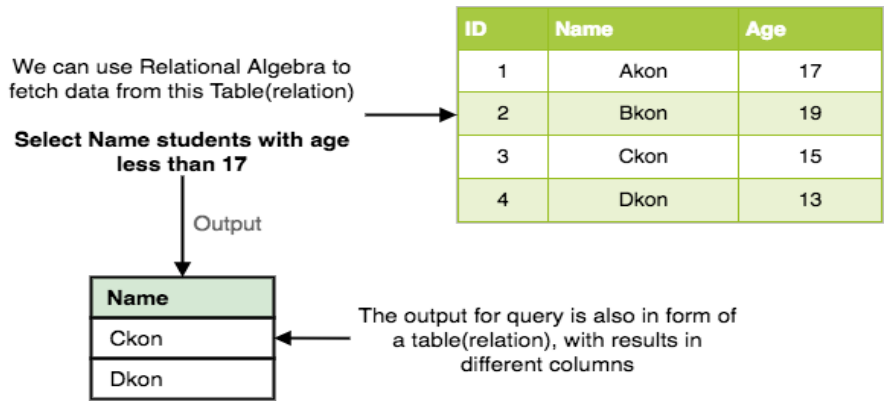
StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com

Summary



Relational Algebra

- ✓ Every database management system must define a query language to allow users to access the data stored in the database.
- ✓ Relational Algebra is a procedural query language used to query the database tables to access data in different ways.
- ✓ In relational algebra, input is a relation (table from which data has to be accessed) and output is also a relation (a temporary table holding the data asked for by the user).



- ✓ The primary operations that we can perform using relational algebra are: **Select, Project, Union, Set Difference, Cartesian Product and Join**.

a) Select Operation (σ)

- ✓ This is used to fetch rows(tuples) from table(relation) which satisfies a given condition.
- ✓ Let's take an example of the Student table we specified above in the Introduction of relational algebra, and fetch data for students with age more than 17.

  $\sigma_{age > 17}$ **(Student)**
- ✓ This will fetch the tuples(rows) from table Student, for which age will be greater than 17.
- ✓ You can also use, and, or operators, to specify two conditions, for example,

  $\sigma_{age > 17 \text{ and gender = 'Male'}}$ **(Student)**
- ✓ This will return tuples(rows) from table Student with information of male students, of age more than 17.

b) Project Operation (∏)

- ✓ Project operation is used to project only a certain set of attributes of a relation. In simple words, If you want to see only the names all of the students in the Student table, then you can use Project Operation.
- ✓ It will only project or show the columns or attributes asked for, and will also remove duplicate data from the columns.
- ✓ For example,

  $\prod_{Name, Age}$**(Student)**
- ✓ Above statement will show us only the Name and Age columns for all the rows of data in Student table.

- **Relational algebra expressions**

SQL	Result	Relational algebra
`select name, salary from E where salary < 200`	name | salary John | 100 Tom | 100	$\mathbf{PROJECT}_{name,\ salary}\ (\mathbf{SELECT}_{salary < 200}(E))$ *or, step by step, using an intermediate result* Temp <- $\mathbf{SELECT}_{salary < 200}(E)$ Result <- $\mathbf{PROJECT}_{name,\ salary}(Temp)$

c) Union Operation (∪)

- ✓ This operation is used to fetch data from two relations(tables).
- ✓ For this operation to work, the relations(tables) specified should have same number of attributes(columns) and same attribute domain. Also the duplicate rows are automatically eliminated from the result.
- ✓ Syntax: **A ∪ B** where A and B are relations.
- ✓ For example, if we have two tables **RegularClass** and **ExtraClass**, both have a column student to save name of student, then,

  $\prod_{Student}$**(RegularClass)** ∪ $\prod_{Student}$**(ExtraClass)**
- ✓ Above operation will give us name of Students who are attending both regular classes and extra classes, eliminating repetition.

d) Set Difference (-)

- ✓ This operation is used to find data present in one relation and not present in the second relation. This operation is also applicable on two relations, just like Union operation.
- ✓ **Syntax: A – B** where A and B are relations.
- ✓ For example, if we want to find name of students who attend the regular class but not the extra class, then, we can use the below operation:
  $\prod_{Student}$**(RegularClass) -** $\prod_{Student}$**(ExtraClass)**

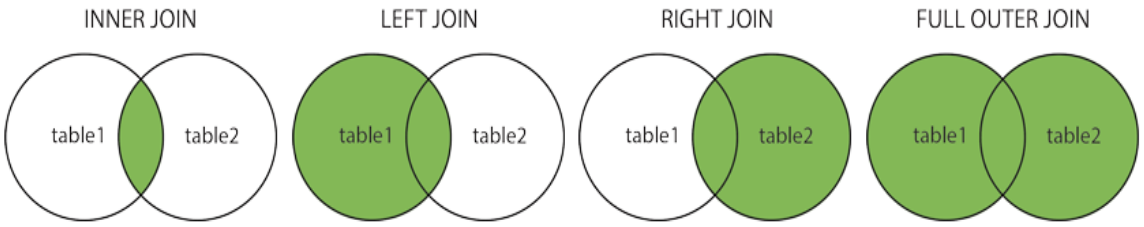
e) Cartesian Product (X)

- ✓ This is used to combine data from two different relations(tables) into one and fetch data from the combined relation. **Syntax: A X B**
- ✓ For example, if we want to find the information for Regular Class and Extra Class which are conducted during morning, then, we can use the following operation:
  $\sigma_{time = 'morning'}$ **(RegularClass X ExtraClass)**
- ✓ For the above query to work, both RegularClass and ExtraClass should have the attribute time.

f) Join

Here are the different types of the JOINs in SQL:

- **(INNER) JOIN (⋈)** : Returns records that have matching values in both tables.
- **LEFT (OUTER) JOIN (⟕):** Return all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN (⟖):** Return all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN (⟗):** Return all records when there is a match in either left or right table



Relational Algebra Practical Examples

Employee

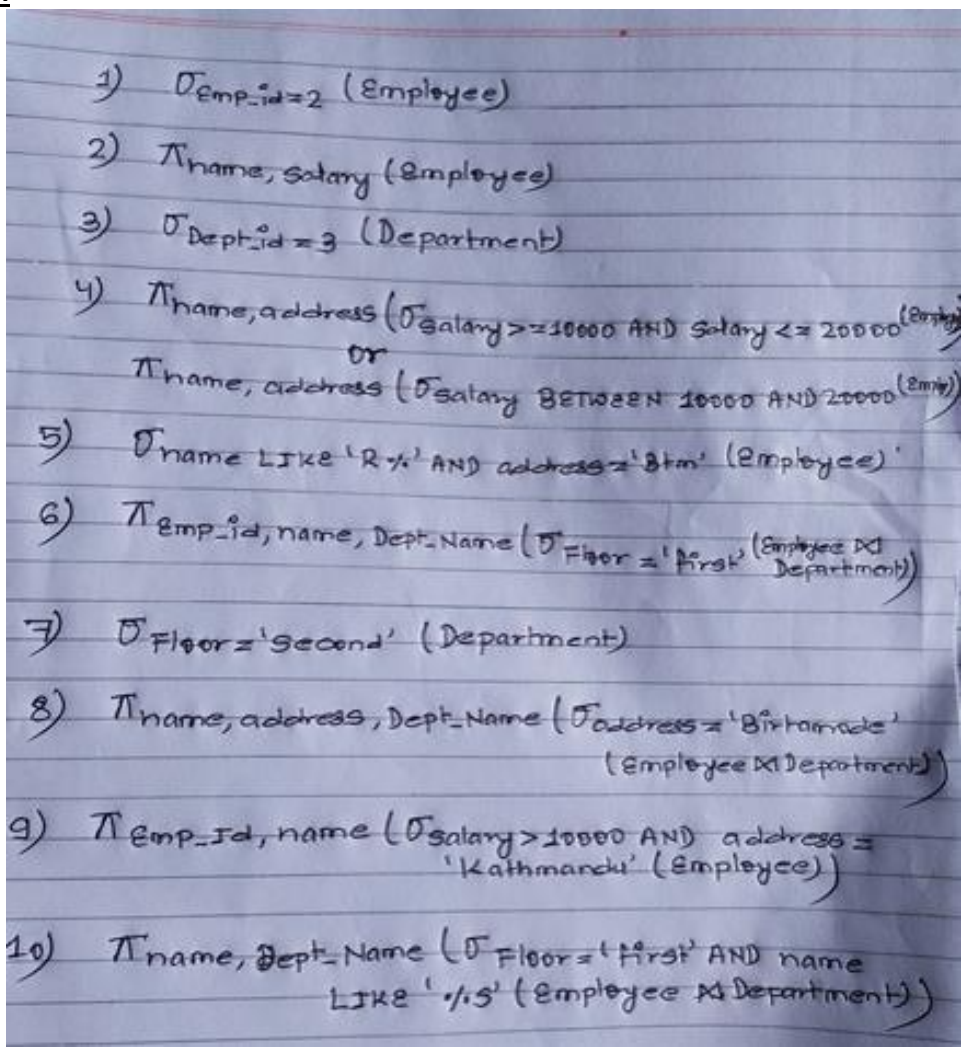
Emp_Id	Name	Address	Salary	Dept_Id

Department

Dept_Id	Dept_Name	Floor

1) Select all records of employee whose emp id is 2.
2) Select name and salary of all employees.
3) Select all records of department where department id is 3.
4) Select name and address of employees whose salary is between 10000 and 20000.
5) Select all records of employee whose name starts with letter 'R' and address is 'Btm'.
6) Select employee id, employee name and department name of employees working in first floor.
7) Select all records of department which are in second floor.
8) Select name, address and department name of employees which are from Birtamode.
9) Select employee id and name of employees having salary more than 10000 and from Kathmandu.
10) Select name and department name of employees having first floor and name ending with letter 's'.

Solution



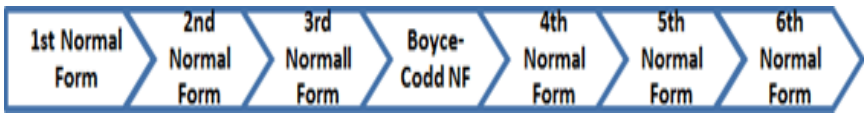
UNIT IV – DATABASE NORMALIZATION - 4 HRS

Introduction

Database normalization, or simply normalization, is the process of restructuring a relational database in accordance with a series of so-called normal forms in order to reduce data redundancy and improve data integrity.

- ✓ **Data redundancy** is the existence of data that is additional to the actual data.
- ✓ **Data integrity** is the maintenance of, and the assurance of the accuracy and consistency of, data over its entire life-cycle.

It divides larger tables to smaller tables and links them using relationships.



Advantages of Normalization

- A smaller database can be maintained as normalization eliminates the duplicate data. Overall size of the database is reduced as a result.
- As databases become lesser in size, the passes through the data becomes faster and shorter thereby improving response time and speed.
- Avoid redundant fields or columns.
- More flexible data structure i.e. we should be able to ad new rows and data values easily
- Better understanding of data.
- Easier to maintain data structure i.e. it is easy to perform operations and complex queries can be easily handled.

Disadvantages of Normalization

- Database systems are complex, difficult, and time-consuming to design.
- Substantial hardware and software start-up costs.
- Initial training required for all programmers and users.
- On Normalizing the relations to higher normal forms i.e. 4NF, 5NF the performance degrades.
- It is very time consuming and difficult process in normalizing relations of higher degree.
- Careless decomposition may lead to bad design of database which may leads to serious problems.

Functional Dependencies

- Functional dependency in DBMS, as the name suggests is a **relationship between attributes** of a table dependent on each other.
- Introduced by E. F. Codd, it helps in preventing data redundancy and gets to know about bad designs.
- For Example, consider the following table,

Employee number	Employee Name	Salary	City
1	Dana	50000	San Francisco
2	Francis	38000	London
3	Andrew	25000	Tokyo

- In this example, if we know the value of Employee number, we can obtain Employee Name, city, salary, etc.
- By this, we can say that the **city, Employee Name, and salary** are functionally depended on **Employee number**.
- A functional dependency is denoted by an arrow →
- The functional dependency of X on Y is represented by **X →Y**
- If column A of a table uniquely identifies the column B of same table, then it can have represented as **A->B** (Attribute B is functionally dependent on attribute A)

Types of Functional Dependencies

- Multivalued dependency
- Trivial functional dependency
- Non-trivial functional dependency
- Transitive dependency

Multivalued Dependency

- Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.
- A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.
- For example,

BIKE_MODEL	MANUF_YEAR	COLOR
M2011	2008	White
M2001	2008	Black
M3001	2013	White
M3001	2013	Black
M4006	2017	White
M4006	2017	Black

BIKE_MODEL → → MANUF_YEAR
BIKE_MODEL → → COLOR

- Here columns **COLOR** and **MANUF_YEAR** are dependent on **BIKE_MODEL** and independent of each other.
- In this case, these two columns can be called as **multivalued dependent** on **BIKE_MODEL.**

Trivial Functional Dependency

- The Trivial dependency is a set of attributes which are called a trivial if the set of attributes are included in that attribute.
- So, X -> Y is a trivial functional dependency if Y is a subset of X.
- A trivial functional dependency is a database dependency that occurs when describing a functional dependency of an attribute or of a collection of attributes that includes the original attribute.
- Consider this table with two columns emp_id and emp_name.

emp_id	emp_name
AS555	Harry
AS222	George
AS999	Kevin

- {emp_id, emp_name} -> emp_name [emp_name is a subset of {emp_id, emp_name}]

Non-Trivial Functional Dependency

- If a functional dependency X->Y holds true where Y is not a subset of X then this dependency is called non trivial Functional dependency.
- For example: An employee table with three attributes: emp_id, emp_name, emp_address.
- The following functional dependencies are non-trivial:
  emp_id -> emp_name (emp_name is not a subset of emp_id)
  emp_id -> emp_address (emp_address is not a subset of emp_id)

Transitive Dependency

- If non-primary key attributes depend upon other non-primary key attributes than there occurs transitive dependency.
- A transitive is a type of functional dependency which happens when t is indirectly formed by two functional dependencies.
- A transitive functional dependency is when changing a non-key column, might cause any of the other non-key columns to change.
- Consider the table, Changing the non-key column **Full Name** may change **Salutation**.

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3rd Street 34	Mr.
3	Robert Phil	5th Avenue	Mr.

- For example,

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Alibaba	Jack Ma	54

- {Company} -> {CEO} (if we know the company, we know its CEO's name)
- {CEO } -> {Age} If we know the CEO, we know the Age
- Therefore according to the rule of rule of transitive dependency:
- { Company} -> {Age} should hold, that makes sense because if we know the company name, we can know his age.
- **Note:** You need to remember that transitive dependency can only occur in a relation of three or more attributes.

Database Anomalies

- Database anomalies are the problems in relations that occur due to redundancy in the relations.
- They can occur in poorly planned, un-normalised databases where all the data is stored in one table.
- These anomalies affect the process of inserting, deleting and modifying data in the relations.
- Some important data may be lost if a relation is updated that contains database anomalies.
- It is important to remove these anomalies in order to perform different processing on the relations without any problem.

Types of Anomalies

- Insertion Anomalies
- Deletion Anomalies
- Modification Anomalies

Insertion Anomaly

- An Insert Anomaly occurs when certain attributes cannot be inserted into the database without the presence of other attributes.

sid	sname	cid	cname
S10	Ram	C03	Java
S11	Shyam	C03	Java
S12	Hari	C04	C
S13	Sita	C05	C++

- For example, we can't add a new course unless we have at least one student enrolled on the course.
- If we want to add a new course, then student details will become null. So, course can’t be inserted without having student details. This scenario forms insertion anomaly.

Deletion Anomaly

- A Delete Anomaly exists when certain attributes are lost because of the deletion of other attributes.

sid	sname	cid	cname
S10	Ram	C03	Java
S11	Shyam	C03	Java
S12	Hari	C04	C
S13	Sita	C05	C++

- For example, consider what happens if Student S13 is the last student to leave the course - All information about the course is lost.

Modification Anomaly

- The modification anomaly occurs when the record is updated in the relation. In this anomaly, the modification in the value of specific attribute requires modification in all records in which that value occurs.

sid	sname	cid	cname
S10	Ram	C03	Java
S11	Shyam	C03	Java
S12	Hari	C04	C
S13	Sita	C05	C++

- For example, if we update cid of student then we need to update cname of student too.
- So, normalization process is required to eliminate anomalies from database.

First Normal Form (1NF)

For a table to be in the First Normal Form, it should follow the following rules:

a) It should only have single(atomic) valued attributes/columns.
b) All the columns in a table should have unique names.

Consider the following table Student,

sid	sname	salutation	address	phone	sub_id	sub_name
1	Ram	Mr	Btm	9864912123, 9854964126	1	Database
2	Shyam	Mr	Ktm	9824912345	1	Database
3	Gita	Mrs	Btm	9824923456	2	Java
4	Sita	Ms	Ktm	9825612723	3	C
2	Shyam	Mr	Ktm	9824912345	2	Java

Above table does not satisfy 1NF because column phone contains multiple values. Hence, we need to create new table contact to store phone numbers.

Following are the normalized tables that satisfy 1NF,

Student

sid	sname	salutation	address	sub_id	sub_name
1	Ram	Mr	Btm	1	Database
2	Shyam	Mr	Ktm	1	Database
3	Gita	Mrs	Btm	2	Java
4	Sita	Ms	Ktm	3	C
2	Shyam	Mr	Ktm	2	Java

Contact

contact_id	phone	sid
1	9864912123	1
2	9854964126	1
3	9824912345	2
4	9824923456	3
5	9825612723	4

Second Normal Form (2NF)

For a table to be in the Second Normal Form, it must satisfy two conditions:

a) The table should be in the First Normal Form.

b) There should be no Partial Dependency

Partial Functional Dependency occurs only in relation with composite keys.

Partial functional dependency occurs when one or more non key attribute are depending on a part of the primary key.

Example:

Table: Stud_id, Course_id, Stud_name, Course_Name

Where: Primary Key = Stud_id + Course_id

Then: To determine name of student we use only Stud_id, which is part of primary key.

{Stud_id} -> {Stud_Name}

Hence, Stud_name is partially dependent on Stud_id. This is called **partial dependency.**

- Our **Student** table does not satisfy 2NF, because we have,
  Primary Key: sid + sub_id
- Here, to determine name of subject, we use sub_id which is a part of primary key. Hence, sub_name is partially dependent on sub_id.
- So, we need to remove partial dependency from Student table and create new table subject to store subject details.

Student

sid	sname	salutation	address	sub_id
1	Ram	Mr	Btm	1
2	Shyam	Mr	Ktm	1
3	Gita	Mrs	Btm	2
4	Sita	Ms	Ktm	3
2	Shyam	Mr	Ktm	2

Subject

sub_id	sub_name
1	Database
2	Java
3	C

Third Normal Form (3NF)

The official qualifications for 3NF are:

a) A table is already in 2NF.
b) Non primary key attributes do not depend on other non primary key attributes (i.e. no transitive dependencies)

All transitive dependencies are removed to place in another table.

Consider the following student table,

sid	sname	salutation	address	sub_id
1	Ram	Mr	Btm	1
2	Shyam	Mr	Ktm	1
3	Gita	Mrs	Btm	2
4	Sita	Ms	Ktm	3
2	Shyam	Mr	Ktm	2

In above table, there is transitive dependency on sname and salutation. Both are non-primary key attributes. Change in sname might cause change in salutiation. For eg, if we change name **Ram to Maya** then we need to change salutiation too.

Original table

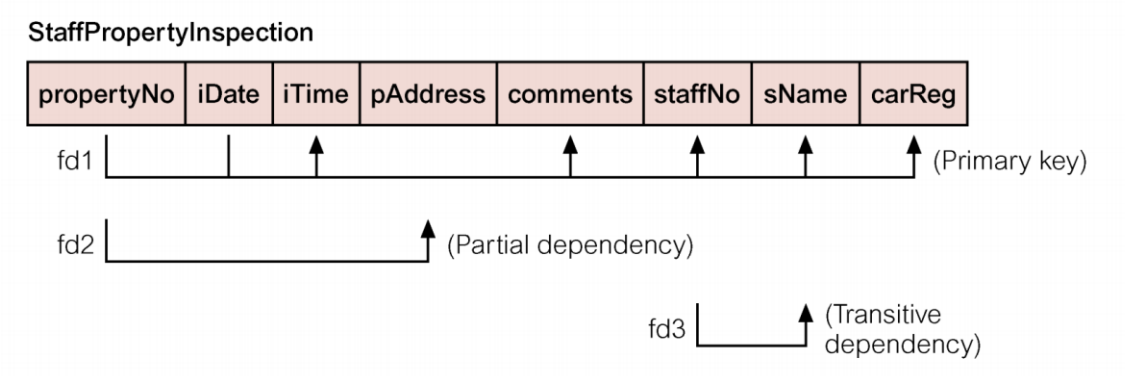
sid	sname	salutation	address	sub_id
1	Ram	Mr	Btm	1
2	Shyam	Mr	Ktm	1
3	Gita	Mrs	Btm	2
4	Sita	Ms	Ktm	3
2	Shyam	Mr	Ktm	2

Normalized tables.

sid	sname	salutation_id	address	sub_id
1	Ram	1	Btm	1
2	Shyam	1	Ktm	1
3	Gita	3	Btm	2
4	Sita	2	Ktm	3
2	Shyam	1	Ktm	2

salutation_id	salutation
1	Mr
2	Ms
3	Mrs

Functional Dependencies



Fourth Normal Form (4NF)

- A table is in the 4NF if it is in 3NF and has no **multivalued dependencies.**
- A **multivalued dependency** exists when there are at least 3 attributes (like X, Y and Z) in a relation and for value of X there is a well defined set of values of Y and a well defined set of values of Z. However, the set of values of Y is independent of set Z and vice versa.
- Suppose a student can have more than one subject and more than one activity.

Student_Info		
Student_Id	**Subject**	**Activity**
100	Music	Swimming
100	Accounting	Swimming
100	Music	Tennis
100	Accounting	Tennis
150	Math	Jogging

- Note that all three attributes make up the Primary Key.
- Note that Student_Id can be associated with many subject as well as many activities. This scenario is **multi valued dependency**.
- Databases with multivalued dependencies thus exhibit **redundancy**.

Original Table:

Student_Info		
Student_Id	**Subject**	**Activity**
100	Music	Swimming
100	Accounting	Swimming
100	Music	Tennis
100	Accounting	Tennis
150	Math	Jogging

Here are the tables Normalized-

Student_Id	Subject
100	Music
100	Accounting
150	Math

StudentId	Activity
100	Swimming
100	Tennis
150	Jogging

Original Table:

University courses

Course	Book	Lecturer
AHA	Silberschatz	John D
AHA	Nederpelt	John D
AHA	Silberschatz	William M
AHA	Nederpelt	William M
AHA	Silberschatz	Christian G
AHA	Nederpelt	Christian G
OSO	Silberschatz	John D
OSO	Silberschatz	William M

Here are the tables Normalized-

Course	Book
AHA	Silberschatz
AHA	Nederpelt

Course	Lecturer
AHA	John D
AHA	William M
AHA	Christian G
OSO	John D
OSO	William M

Boyce-Codd Normal Form (BCNF)

- **Boyce-Codd Normal Form (BCNF)** is one of the forms of database normalization. A database table is in BCNF if and only if there are **no non-trivial functional dependencies of attributes** on anything other than a superset of a candidate key.
- **BCNF** is also sometimes referred to as 3.5NF, or 3.5 Normal Form.
- For a table to satisfy the Boyce-Codd Normal Form, it should satisfy the following two conditions:
  a) It should be in the Third Normal Form.
  b) And, for any dependency A → B, A should be a super key.
- The second point sounds a bit tricky, right? In simple words, it means, that for a **dependency A → B, A cannot be a non-prime attribute, if B is a prime attribute**.
- Below we have a college enrolment table with columns **student_id, subject and professor**.

student_id	subject	professor
101	Java	P.Java
101	C++	P.Cpp
102	Java	P.Java2
103	C#	P.Chash
104	Java	P.Java

- In the above table student_id, subject together form the primary key, because using student_id and subject, we can find all the columns of the table.
- Also, there is a dependency between subject and professor, where subject depends on the professor name.
- This table satisfies the **1st Normal form** because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain.
- This table also satisfies the **2nd Normal Form** as their is no **Partial Dependency**.
- And, there is no **Transitive Dependency**, hence the table also satisfies the **3rd Normal Form**.
- But this table is not in **Boyce-Codd Normal Form**.

Why this table doesn't satisfy BCNF?

- In the table above, student_id, subject form primary key, which means subject column is a prime attribute.
- But, there is one more dependency, professor → subject.
- And while subject is a prime attribute, professor is a non-prime attribute, which is not allowed by BCNF.

To make this relation(table) satisfy BCNF, we will decompose this table into two tables, **student table** and **professor table**.
Below we have the structure for both the tables.

Student Table

student_id	p_id
101	1
101	2
and so on...	

Professor Table

p_id	professor	subject
1	P.Java	Java
2	P.Cpp	C++
and so on...		

Original Table

student_id	subject	professor
101	Java	P.Java
101	C++	P.Cpp
102	Java	P.Java2
103	C#	P.Chash
104	Java	P.Java

- And now, this relation satisfy **Boyce-Codd Normal Form**.

Unit – V Introduction to SQL Queries [6 Hrs]

Introduction to SQL

- SQL is a standard language for accessing and manipulating databases.
- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

SQL DML Statements

SQL CREATE DATABASE Statement

The CREATE DATABASE statement is used to create a new SQL database.
Syntax,

```
CREATE DATABASE databasename;
```

SQL DROP DATABASE Statement

The DROP DATABASE statement is used to drop an existing SQL database.
Syntax,

```
DROP DATABASE databasename;
```

SQL CREATE TABLE Statement

The CREATE TABLE statement is used to create a new table in a database.

Syntax,

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

The column parameters specify the names of the columns of the table. The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

Example,

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

SQL DROP TABLE Statement

The DROP TABLE statement is used to drop an existing table in a database.

Syntax,

```
DROP TABLE table_name;
```

SQL ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table. The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD Column

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
ADD column_name datatype;
```

The following SQL adds an "Email" column to the "Customers" table:

```
ALTER TABLE Customers
ADD Email varchar(255);
```

ALTER TABLE - DROP COLUMN

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

The following SQL deletes the "Email" column from the "Customers" table:

```
ALTER TABLE Customers
DROP COLUMN Email;
```

ALTER TABLE - ALTER/MODIFY COLUMN

To change the data type of a column in a table, use the following syntax:

SQL Server / MS Access:

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype;
```

My SQL / Oracle (prior version 10G):

```
ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```

Oracle 10G and later:

```
ALTER TABLE table_name
MODIFY column_name datatype;
```

SQL Constraints

SQL constraints are used to specify rules for data in a table. Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

Syntax

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```

The following constraints are commonly used in SQL:

a) **NOT NULL** - Ensures that a column cannot have a NULL value.
The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values:

```
CREATE TABLE Persons (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255) NOT NULL,
   Age int
);
```

b) **UNIQUE** - Ensures that all values in a column are different.
The following SQL creates a UNIQUE constraint on the "ID" column when the "Persons" table is created:

```
CREATE TABLE Persons (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   UNIQUE (ID)
);
```

c) **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table.

The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:

```
CREATE TABLE Persons (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   PRIMARY KEY (ID)
);
```

d) **FOREIGN KEY** - Uniquely identifies a row/record in another table.

```
CREATE TABLE Orders (
   OrderID int NOT NULL,
   OrderNumber int NOT NULL,
   PersonID int,
   PRIMARY KEY (OrderID),
   FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

e) **CHECK** - Ensures that all values in a column satisfies a specific condition.

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that you can not have any person below 18 years:

```
CREATE TABLE Persons (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   CHECK (Age>=18)
);
```

f) **DEFAULT** - Sets a default value for a column when no value is specified.

The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Birtamode'
);
```

Altering Constraints

1) Primary Key

To create a PRIMARY KEY constraint on the "ID" column when the table is already created, use the following SQL:

ALTER TABLE Persons
ADD PRIMARY KEY (ID);

To drop a PRIMARY KEY constraint, use the following SQL:

ALTER TABLE Persons
DROP PRIMARY KEY;

2) Unique Constraint

To create a UNIQUE constraint on the "ID" column when the table is already created, use the following SQL:

ALTER TABLE Persons
ADD UNIQUE (ID);

To drop a UNIQUE constraint, use the following SQL:

```
ALTER TABLE Persons
DROP CONSTRAINT constraint_name;
```

3) Foreign Key

To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

ALTER TABLE Orders
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

To drop a FOREIGN KEY constraint, use the following SQL:

ALTER TABLE Orders
DROP FOREIGN KEY foreign_key_name;

```
ALTER TABLE Orders
DROP CONSTRAINT constraint_name;
```

4) Check Constraint

To create a CHECK constraint on the "Age" column when the table is already created, use the following SQL:

ALTER TABLE Persons
ADD CHECK (Age>=18);

To drop a CHECK constraint, use the following SQL:
ALTER TABLE Persons
DROP CHECK Age;

5) Default Constraint
To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons
MODIFY City DEFAULT 'Sandnes';
```

To drop a DEFAULT constraint, use the following SQL:

```
ALTER TABLE Persons
ALTER COLUMN City DROP DEFAULT;
```

Unit – VI Manipulating and Querying Data [8 Hrs]

The SQL SELECT Statement

The SELECT statement is used to select data from a database. The data returned is stored in a result table, called the result-set.

```
SELECT column1, column2,                                    ...
FROM table_name;
```

Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

```
SELECT * FROM table_name;
```

SQL SELECT DISTINCT Statement

The SELECT DISTINCT statement is used to return only distinct (different) values. Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

Syntax,

```
SELECT DISTINCT column1, column2,                           ...
FROM table_name;
```

SQL WHERE Clause

The WHERE clause is used to filter records. The WHERE clause is used to extract only those records that fulfill a specified condition.

```
SELECT column1, column2,                                    ...
FROM table_name
WHERE condition;
```

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

```
SELECT * FROM Customers
WHERE CustomerID=1;
```

SQL AND, OR and NOT Operators

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

OR Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

NOT Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

Examples,

```
SELECT * FROM Customers
WHERE Country='Germany' AND City='Berlin';

SELECT * FROM Customers
WHERE Country='Germany' OR Country='Spain';

SELECT * FROM Customers
WHERE NOT Country='Germany';
```

Combining AND, OR and NOT

The following SQL statement selects all fields from "Customers" where country is "Germany" AND city must be "Berlin" OR "München".

```
SELECT * FROM Customers
WHERE Country='Germany' AND (City='Berlin' OR City='München');
```

The following SQL statement selects all fields from "Customers" where country is NOT "Germany" and NOT "USA":

```
SELECT * FROM Customers
WHERE NOT Country='Germany' AND NOT Country='USA';
```

Aggregate Functions

SQL MIN() and MAX() Functions

- The MIN() function returns the smallest value of the selected column.
- The MAX() function returns the largest value of the selected column.

Syntax,

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;

SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

The following SQL statement finds the price of the cheapest product:

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

The following SQL statement finds the price of the most expensive product:

```
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

SQL COUNT(), AVG() and SUM() Functions

- The COUNT() function returns the number of rows that matches a specified criteria.
- The AVG() function returns the average value of a numeric column.
- The SUM() function returns the total sum of a numeric column.

Syntax,

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;

SELECT AVG(column_name)
FROM table_name
WHERE condition;

SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

The following SQL statement finds the number of products:

```
SELECT COUNT(ProductID)
FROM Products;
```

The following SQL statement finds the average price of all products:

```
SELECT AVG(Price)
FROM Products;
```

The following SQL statement finds the sum of the "Quantity" fields in the "OrderDetails" table:

```
SELECT SUM(Quantity)
FROM OrderDetails;
```

SQL LIKE Operator

he LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
There are two wildcards often used in conjunction with the LIKE operator:

- % - The percent sign represents zero, one, or multiple characters
- _ - The underscore represents a single character

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

Here are some examples showing different LIKE operators with '%' and '_' wildcards:

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

The following SQL statement selects all customers with a CustomerName starting with "a":

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

The following SQL statement selects all customers with a CustomerName ending with "a":

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%a';
```

The following SQL statement selects all customers with a CustomerName that have "or" in any position:

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%or%';
```

The following SQL statement selects all customers with a CustomerName that have "r" in the second position:

```
SELECT * FROM Customers
WHERE CustomerName LIKE '_r%';
```

The following SQL statement selects all customers with a CustomerName that starts with "a" and are at least 3 characters in length:

```sql
SELECT * FROM Customers
WHERE CustomerName LIKE 'a__%';
```

The following SQL statement selects all customers with a ContactName that starts with "a" and ends with "o":

```sql
SELECT * FROM Customers
WHERE ContactName LIKE 'a%o';
```

The following SQL statement selects all customers with a CustomerName that does NOT start with "a":

```sql
SELECT * FROM Customers
WHERE CustomerName NOT LIKE 'a%';
```

The SQL IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.
- The IN operator is a shorthand for multiple OR conditions.

```sql
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

```sql
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

The following SQL statement selects all customers that are located in "Germany", "France" and "UK":

```sql
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');
```

The following SQL statement selects all customers that are NOT located in "Germany", "France" or "UK":

```sql
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
```

The following SQL statement selects all customers that are from the same countries as the suppliers:

```sql
SELECT * FROM Customers
WHERE Country IN (SELECT Country FROM Suppliers);
```

SQL BETWEEN Operator

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates. The BETWEEN operator is inclusive: begin and end values are included.

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

The following SQL statement selects all products with a price BETWEEN 10 and 20:

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

To display the products outside the range of the previous example, use NOT BETWEEN:

```
SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
```

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';
```

SQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set in ascending or descending order.
The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

```
SELECT column1, column2,                                    ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" column:

```
SELECT * FROM Customers
ORDER BY Country;
```

The following SQL statement selects all customers from the "Customers" table, sorted DESCENDING by the "Country" column:

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName:

```
SELECT * FROM Customers
ORDER BY Country, CustomerName;
```

The SQL GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".
The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

The following SQL statement lists the number of customers in each country:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

The following SQL statement lists the number of customers in each country, sorted high to low:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

SQL INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

The following SQL statement inserts a new record in the "Customers" table:

```
INSERT INTO Customers (CustomerName, ContactName, Address, City,
PostalCode, Country) VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen
21', 'Stavanger', '4006', 'Norway');
```

SQL UPDATE Statement

The UPDATE statement is used to modify the existing records in a table.

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

The following SQL statement updates the first customer (CustomerID = 1) with a new contact person and a new city.

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

The following SQL statement will update the contactname to "Juan" for all records where country is "Mexico":

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```

SQL DELETE Statement

The DELETE statement is used to delete existing records in a table.

```
DELETE FROM table_name WHERE condition;
```

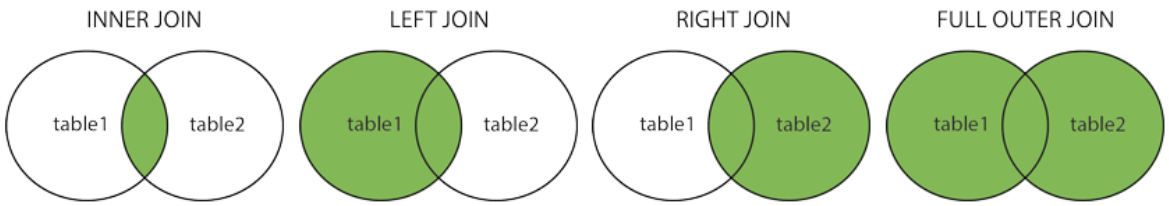
```
DELETE FROM table_name;
```

```
DELETE FROM Customers WHERE CustomerName='Ram';
```

SQL Joins

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.
Here are the different types of the JOINs in SQL:

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table



SQL INNER JOIN Keyword

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

Joining three tables,

```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

SQL LEFT JOIN, RIGHT JOIN and FULL JOIN Keyword

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;

SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;

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

SQL Views

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.

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

The following SQL creates a view that shows all customers from Brazil:

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = "Brazil";
```

We can query the view above as follows:

```
SELECT * FROM [Brazil Customers];
```

The following SQL creates a view that selects every product in the "Products" table with a price higher than the average price:

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AVG(Price) FROM Products);
```

We can query the view above as follows:

```
SELECT * FROM [Products Above Average Price];
```

SQL Updating a View

A view can be updated with the CREATE OR REPLACE VIEW command.

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

The following SQL adds the "City" column to the "Brazil Customers" view:

```
CREATE OR REPLACE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName, City
FROM Customers
WHERE Country = "Brazil";
```

SQL Dropping a View

A view is deleted with the DROP VIEW command.

```
DROP VIEW view_name;
```

```
DROP VIEW [Brazil Customers];
```

Unit VII – Developing Stored Procedures, DML Triggers and Indexing [5 Hrs]

What is a Stored Procedure?

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

Stored Procedure Syntax - MySQL

```
DELIMITER //
CREATE PROCEDURE procedure_name(param1,param2,…)
BEGIN
   Write Query
END //
DELIMITER ;
```

Execute a Stored Procedure

```
CALL procedure_name();
```

The following SQL statement creates a stored procedure named "SelectAllCustomers" that selects all records from the "Customers" table:

```
DELIMITER //
CREATE PROCEDURE SelectAllCustomers()
BEGIN
   SELECT * FROM customer;
END //
DELIMITER ;
```

The following SQL statement creates a stored procedure named "SelectAllCustomers" that selects name and age of customers whose age is greater than 20 from the "Customers" table:

```
DELIMITER //
CREATE PROCEDURE SelectAllCustomers()
BEGIN
   SELECT name,age FROM customer WHERE age>20;
END //
DELIMITER ;
```

(Only change query for insert, update and delete operations, other contents remain same)

DROP Stored Procedure

```
DROP PROCEDURE procedure_name;
```

Working with Parameters

In MySQL, a parameter has one of three modes: IN, OUT, or INOUT.

IN parameters

IN is the default mode. When you define an IN parameter in a stored procedure, the calling program has to pass an argument to the stored procedure. In addition, the value of an IN parameter is protected. It means that even the value of the IN parameter is changed inside the stored procedure, its original value is retained after the stored procedure ends. In other words, the stored procedure only works on the copy of the IN parameter.

OUT parameters

The value of an OUT parameter can be changed inside the stored procedure and its new value is passed back to the calling program. Notice that the stored procedure cannot access the initial value of the OUT parameter when it starts.

INOUT parameters

An INOUT parameter is a combination of IN and OUT parameters. It means that the calling program may pass the argument, and the stored procedure can modify the INOUT parameter, and pass the new value back to the calling program.

The IN parameter example

The following example creates a stored procedure that finds all offices that locate in a country specified by the input parameter `countryName`:

```
DELIMITER //

CREATE PROCEDURE GetOfficeByCountry(
    IN countryName VARCHAR(255)
)
BEGIN
    SELECT *
     FROM offices
    WHERE country = countryName;
END //

DELIMITER ;
```

Suppose that you want to find offices locating in the USA, you need to pass an argument (`USA`) to the stored procedure as shown in the following query:

```
CALL GetOfficeByCountry('USA');
```

The OUT parameter example

The following stored procedure returns the number of orders by order status.

```
DELIMITER $$

CREATE PROCEDURE GetOrderCountByStatus (
    IN  orderStatus VARCHAR(25),
    OUT total INT
)
BEGIN
    SELECT COUNT(orderNumber)
    INTO total
    FROM orders
    WHERE status = orderStatus;
END$$

DELIMITER ;
```

To get the number of orders that are in-process, you call the stored procedure `GetOrderCountByStatus` as follows:

```
CALL GetOrderCountByStatus('in process',@total);
SELECT @total AS  total_in_process;
```

	total_in_process
▶	6

Inserting Records Using Parameters

Inserting sid, name and address in student table.

```
DELIMITER //
CREATE PROCEDURE insertRecords(
   IN id INT,
   IN nm VARCHAR(30),
   IN ad VARCHAR(30)
)
BEGIN
  INSERT INTO student VALUES(id,nm,ad);
END //
```

Executing,

```
CALL insertRecords(5,'Hari','Ktm');
```

Updating Records Using Parameters

Update name and address on the basis of id in student table.

```
DELIMITER //
CREATE PROCEDURE updateRecords(
   IN id INT,
   IN nm VARCHAR(30),
   IN ad VARCHAR(30)
)
BEGIN
  UPDATE student set name=nm, address=ad WHERE sid=id;
END //
```

Executing,
CALL updateRecords(5,'Hari','Ktm');

Deleting Records Using Parameters

Delete records on the basis of id in student table.

```
DELIMITER //
CREATE PROCEDURE deleteRecords(
   IN id INT
)
BEGIN
  UPDATE FROM student WHERE sid=id;
END //
```

Executing,
CALL deleteRecords(3);

SQL Index

The CREATE INDEX statement is used to create indexes in tables. Indexes are used to retrieve data from the database very fast. The users cannot see the indexes; they are just used to speed up searches/queries.

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

The SQL statement below creates an index named "idx_lastname" on the "LastName" column in the "Persons" table:

```
CREATE INDEX idx_lastname
ON Persons (LastName);
```

If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

```
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
```

DROP INDEX Statement

The DROP INDEX statement is used to delete an index in a table.

```
DROP INDEX index_name;
```

SQL Triggers

A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

Syntax:

```
create trigger [trigger_name]
[before | after]
{insert | update | delete}
on [table_name]
[for each row]
[trigger_body]
```

Explanation of syntax:

1. create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.
2. [before | after]: This specifies when the trigger will be executed.
3. {insert | update | delete}: This specifies the DML operation.
4. on [table_name]: This specifies the name of the table associated with the trigger.
5. [for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.
6. [trigger_body]: This provides the operation to be performed as trigger is fired

BEFORE and AFTER of Trigger:

BEFORE triggers run the trigger action before the triggering statement is run.
AFTER triggers run the trigger action after the triggering statement is run.

Example:

Given Student Report Database, in which student marks assessment is recorded. In such schema, create a trigger so that the total and average of specified marks is automatically inserted whenever a record is insert.
Here, as trigger will invoke before record is inserted so, BEFORE Tag can be used.

Suppose the database Schema –

```
mysql> desc Student;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| tid   | int(4)      | NO   | PRI | NULL    | auto_increment |
| name  | varchar(30) | YES  |     | NULL    |                |
| subj1 | int(2)      | YES  |     | NULL    |                |
| subj2 | int(2)      | YES  |     | NULL    |                |
| subj3 | int(2)      | YES  |     | NULL    |                |
| total | int(3)      | YES  |     | NULL    |                |
| per   | int(3)      | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

SQL Trigger to problem statement.

```
create trigger stud_marks
before INSERT
on
Student
for each row
set new.total = new.subj1 + new.subj2 + new.subj3, new.per =
new.total * 60 / 100;
```

Above SQL statement will create a trigger in the student database in which whenever subject's marks are entered, before inserting this data into the database, trigger will compute those two values and insert with the entered values. i.e.,

```
mysql> insert into Student values(0, "ABCDE", 20, 20, 20, 0, 0);
Query OK, 1 row affected (0.09 sec)
mysql> select * from Student;
+-----+-------+-------+-------+-------+-------+------+
| tid | name  | subj1 | subj2 | subj3 | total | per  |
+-----+-------+-------+-------+-------+-------+------+
| 100 | ABCDE |    20 |    20 |    20 |    60 |   36 |
+-----+-------+-------+-------+-------+-------+------+
1 row in set (0.00 sec)
```

In this way trigger can be created and executed in the databases.

Drop Trigger

```
DROP TRIGGER trigger_name;
```

Trigger Limitations

- Only one INSTEAD OF trigger is allowed for each event type (INSERT, UPDATE, or DELETE) per table.
- If a table has a BEFORE trigger, it cannot have an INSTEAD OF trigger. The reverse is also true.
- If a table has an INSTEAD OF X trigger, AFTER X triggers defined for that table will not fire (where X is an event type).
- The __error table can only have one row. Delete operations are not currently allowed on the __error table.
- VarChar fields are not supported in the __old and __new tables.
- When using the Advantage Local Server, if a trigger fails for any reason, the database is left as is. This means any operations the trigger may have already performed will be persistent.
- Nested and recursive triggers are limited to 64 levels of server re-entrance before an error is returned.

UNIT VI – QUERY PROCESSING & SECURITY [5 Hrs]

Overview of Query Processing

Query Processing is a translation of high-level queries into low-level expression. It is a step wise process that can be used at the physical level of the file system, query optimization and actual execution of the query to get the result. It requires the basic concepts of relational algebra and file structure.

It refers to the range of activities that are involved in extracting data from the database. It includes translation of queries in high-level database languages into expressions that can be implemented at the physical level of the file system. In query processing, we will actually understand how these queries are processed and how they are optimized.

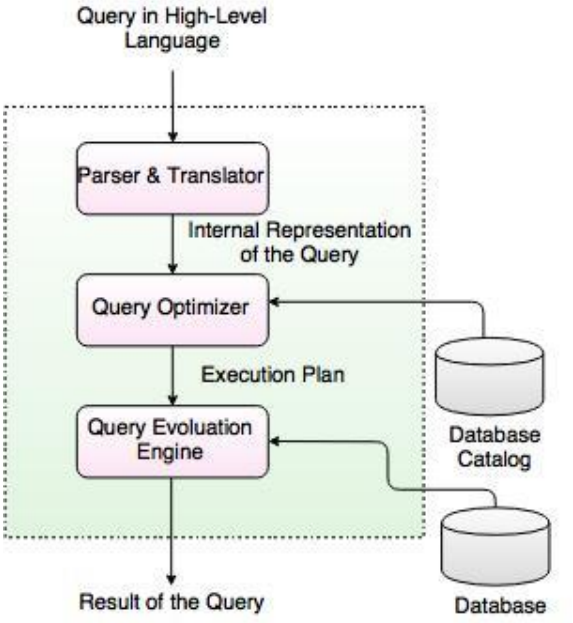


Fig. Query Processing

In the above diagram,

- The first step is to transform the query into a standard form.
- A query is translated into SQL and into a relational algebraic expression. During this process, Parser checks the syntax and verifies the relations and the attributes which are used in the query.
- The second step is Query Optimizer. In this, it transforms the query into equivalent expressions that are more efficient to execute.
- The third step is Query evaluation. It executes the above query execution plan and returns the result.

Measuring of Query Cost

- ✓ Cost is generally measured as total elapsed time for answering query.
- ✓ Many factors contribute to time cost - disk accesses, CPU, or even network communication.
- ✓ Typically disk access is the predominant cost, and is also relatively easy to estimate.
- ✓ Measured by taking into account –
  - Number of seeks
  - Number of blocks read
  - Number of blocks written
- ✓ Cost to write a block is greater than cost to read a block – data is read back after being written to ensure that the write was successful.
- ✓ For simplicity we just use number of block transfers from disk as the cost measure.
- ✓ We ignore the difference in cost between sequential and random I/O for simplicity.
- ✓ We also ignore CPU costs for simplicity. Costs depends on the size of the buffer in main memory. Having more memory reduces need for disk access.
- ✓ Amount of real memory available to buffer depends on other concurrent OS processes, and hard to determine ahead of actual execution.
- ✓ We often use worst case estimates, assuming only the minimum amount of memory needed for the operation is available.
- ✓ Real systems take CPU cost into account, differentiate between sequential and random I/O, and take buffer size into account. We do not include cost to writing output to disk in our cost formulae.

Selection Operation

File scan – search algorithms that locate and retrieve records that fulfill a selection condition.

- ✓ **Algorithm A1 (linear search).** Scan each file block and test all records to see whether they satisfy the selection condition.
  – Cost estimate (number of disk blocks scanned) = br
  - br denotes number of blocks containing records from relation r
  – If selection is on a key attribute, cost = (br/ 2) (stop on finding record)

  Linear search can be applied regardless of
  ∗ selection condition, or
  ∗ ordering of records in the file, or
  ∗ availability of indices

- **A2** (*binary search*). Applicable if selection is an equality comparison on the attribute on which file is ordered.
  - Assume that the blocks of a relation are stored contiguously
  - Cost estimate (number of disk blocks to be scanned):

$$= \lceil \log_2(b_r) \rceil + \left\lceil \frac{SC(A, r)}{f_r} \right\rceil - 1$$

    - $\lceil \log_2(b_r) \rceil$ — cost of locating the first tuple by a binary search on the blocks
    - $SC(A, r)$ — number of records that will satisfy the selection
    - $\lceil SC(A, r)/f_r \rceil$ — number of blocks that these records will occupy
  - Equality condition on a key attribute: $SC(A, r) = 1$; estimate reduces to $\lceil \log_2(b_r) \rceil$

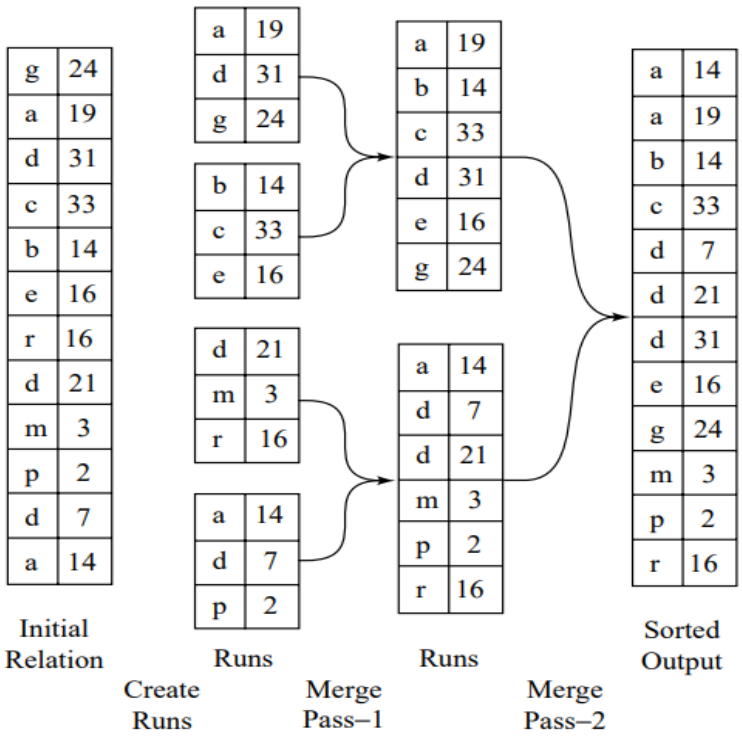
Sorting Operation

- ✓ We may build an index on the relation, and then use the index to read the relation in sorted order. May lead to one disk block access for each tuple.
- ✓ For relations that **fit in memory**, techniques like **quicksort** can be used. For relations that **don't fit in memory**, **external sort-merge** is a good choice.

External Sort–Merge

Let M denote memory size (in pages).

1. Create sorted **runs** as follows. Let i be 0 initially. Repeatedly do the following till the end of the relation:
   (a) Read M blocks of relation into memory
   (b) Sort the in-memory blocks
   (c) Write sorted data to run R_i; increment i.
2. Merge the runs; suppose for now that $i < M$. In a single merge step, use i blocks of memory to buffer input runs, and 1 block to buffer output. Repeatedly do the following until all input buffer pages are empty:
   (a) Select the first record in sort order from each of the buffers
   (b) Write the record to the output
   (c) Delete the record from the buffer page; if the buffer page is empty, read the next block (if any) of the run into the buffer.



Join Operation

- ✓ Several different algorithms to implement joins
  - – Nested-loop join
  - – Block nested-loop join
  - – Indexed nested-loop join
  - – Merge-join
  - – Hash-join

- ✓ Choice based on cost estimate
- ✓ Join size estimates required, particularly for cost estimates for outer-level operations in a relational-algebra expression.

Query Optimization

- ✓ A single query can be executed through different algorithms or re-written in different forms and structures.
- ✓ Hence, the question of **query optimization** comes into the picture – Which of these forms or pathways is the most optimal? The query optimizer attempts to determine the most efficient way to execute a given query by considering the possible query plans.
- ✓ A **query optimizer** is a critical database management system (DBMS) component that analyses Structured Query Language (SQL) queries and determines efficient execution mechanisms.
- ✓ A query optimizer generates one or more query plans for each query, each of which may be a mechanism used to run a query. The most efficient query plan is selected and used to run the query.
- ✓ Database users do not typically interact with a query optimizer, which works in the background.

Importance of Query Optimization

The goal of query optimization is to reduce the system resources required to fulfill a query, and ultimately provide the user with the correct result set faster.

- First, it provides the user with faster results, which makes the application seem faster to the user.
- Secondly, it allows the system to service more queries in the same amount of time, because each request takes less time than unoptimized queries.
- Thirdly, query optimization ultimately reduces the amount of wear on the hardware (e.g. disk drives), and allows the server to run more efficiently (e.g. lower power consumption, less memory usage).

Query Optimization Process

There are broadly two ways a query can be optimized:

- ✓ **Analyze and transform equivalent relational expressions**: Try to minimize the tuple and column counts of the intermediate and final query processes.
- ✓ **Using different algorithms for each operation**: These underlying algorithms determine how tuples are accessed from the data structures they are stored in, indexing, hashing, data retrieval and hence influence the number of disk and block accesses.

Database Administrator (DBA)

A **database administrator (DBA)** is a specialized computer systems administrator who maintains a successful database environment by directing or performing all related activities to keep the data secure.

The top **responsibility** of a **DBA professional** is to maintain **data integrity**. This means the DBA will ensure that data is secure from unauthorized access but is available to users.

A database administrator will often have a working knowledge and experience with a wide range of database management products such as Oracle-based software, and SQL, in addition to having obtained a degree in Computer Science and practical field experience and additional, related IT certifications.

Types of DBA

There are different types of DBAs depending on an organization's requirements:

1. **Administrative DBA –** maintains the servers and databases and keeps them running. Concerned with backups, security, patches, replication. These are activities mostly geared towards maintaining the database and software platform, but not really involved in enhancing or developing it.
2. **Development DBA -** works on building SQL queries, stored procedures, and so on, that meet business needs. This is the equivalent of a programmer, but specializing in database development. Commonly combined the role of Administrative DBA.
3. **Data Architect –** designs schemas, builds tables indexes, data structures and relationships. This role works to build a structure that meets a general business needs in a particular area.

4. **Data Warehouse DBA -** this is a relatively newer role, responsible for merging data from multiple sources into a data warehouse. May have to design the data warehouse as well as cleaning up and standardizing the data before loading using specialist data loading and transformation tools.

Roles & Responsibilities of DBA

1. Database installation, upgrade and patching
2. Install and configure relevant network components
3. Ensure database access, consistency and integrity
4. Resolving issues related to performance bottlenecks
5. Provide reporting on various metrics including availability, usage and performance
6. Performance testing and benchmark activities
7. Work with development staff on architectures, coding standards, and quality assurance policies
8. Create models for new database development or changes to existing ones
9. Respond to and resolve database access and performance issues
10. Monitor database system details
11. Design and implement redundant systems, policies, and procedures for disaster recovery
12. Monitor, optimize and allocate physical data storage for database systems
13. Plan and coordinate data migrations
14. Develop, implement, and maintain change control and testing processes
15. Perform database transaction and security audits
16. Establish end-user database access control levels
17. Implement database encryption and data encryption
18. Plan and ensure compliance with established best practices, related policies and legislation
19. Participate as a member of a team to move the team toward the completion of its goals
20. Capacity planning, installation, configuration, database design, migration, performance monitoring, security, troubleshooting, as well as backup and data recovery.

Database Security

- ✓ Database security refers to the collective measures used to protect and secure a database or database management software from illegitimate use and malicious threats and attacks.
- ✓ Database security covers and enforces security on all aspects and components of databases. This includes:
  - Data stored in database
  - Database server
  - Database management system (DBMS)
  - Other database workflow applications
- ✓ Database security is generally planned, implemented and maintained by a database administrator and or other information security professional.

Database Security Issues

1) No Security Testing Before Deployment
2) Poor Encryption and Data Breaches Come Together
3) Stolen Database Backups
4) Flaws in Features
5) Weak and Complex DB Infrastructure
6) Limitless Administration Access = Poor Data Protection
7) Inadequate Key Management
8) Irregularities in Databases

Types of Database Security

The database security can be broadly classified into physical and logic security.

- ✓ **Physical security -** security of hardware associated with the system and the protection of the site where the computer resides.
- ✓ **Logical security -** security measures residing in the operating system or the DBMS designed to handle threats to data.

Following measures can be used to secure database:

- Access authorization.
- Access controls.
- Backup and recovery of data.
- Data integrity.
- Encryption of data.

Access Protection/Control

Access control is a security technique that regulates who or what can view or use resources in a computing environment. It is a fundamental concept in security that minimizes risk to the business or organization.

There are two types of access control: physical and logical.

- ✓ **Physical access control** limits access to campuses, buildings, rooms and physical IT assets.
- ✓ **Logical access control** limits connections to computer networks, system files and data.

To secure a facility, organizations use electronic access control systems that rely on user credentials, access card readers, auditing and reports to track employee access to restricted business locations and proprietary areas, such as data centres.

Access control systems perform identification authentication and authorization of users and entities by evaluating required login credentials that can include passwords, personal identification numbers (PINs), biometric scans, security tokens or other authentication factors. Multifactor authentication, which requires two or more authentication factors, is often an important part of layered defence to protect access control systems.

User Account and Database Audits

- ✓ Database auditing involves observing a database so as to be aware of the actions of database users.
- ✓ Database administrators and consultants often set up auditing for security purposes, for example, to ensure that those without the permission to access information do not access it.
- ✓ Audit is an analysis of an organization's Computer and information systems in order to evaluate the efficiency, correctness & integrity of its database systems as well as to uncover potential Security Cracks.
- ✓ Auditing is done to verify that DBMS operations are properly implemented and executed.

Mandatory access control (MAC)

- ✓ Mandatory Access Control (MAC) is a set of security policies constrained according to system classification, configuration and authentication.
- ✓ MAC policy management and settings are established in one secure network and limited to system administrators.
- ✓ MAC defines and ensures a centralized enforcement of confidential security policy parameters.

Advantages

- MAC provides tighter security because only a system administrator may access or alter controls.
- MAC policies reduce security errors.

Discretionary access control (DAC)

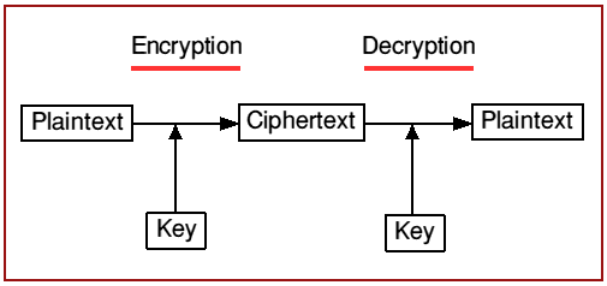
- ✓ Discretionary access control (DAC) is a type of security access control that grants or restricts object access via an access policy determined by an object's owner group and/or subjects.
- ✓ DAC mechanism controls are defined by user identification with supplied credentials during authentication, such as username and password.
- ✓ DACs are discretionary because the subject (owner) can transfer authenticated objects or information access to other users. In other words, the owner determines object access privileges.

Advantages

- ACL maintenance or capability
- Grant and revoke permissions maintenance

Data Encryption and Decryption

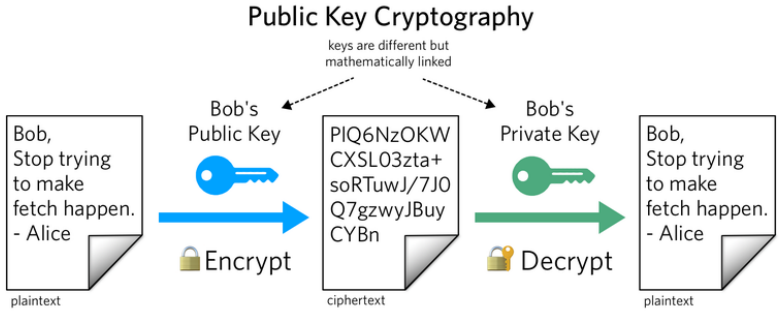
- ✓ **Database encryption** is the process of converting data, within a database, in plain text format into a meaningless cipher text by means of a suitable algorithm.
- ✓ **Database decryption** is converting the meaningless cipher text into the original information using keys generated by the encryption algorithms.



- ✓ Numerous algorithms are used for encryption. These algorithms generate keys related to the encrypted data.
- ✓ These keys set a link between the encryption and decryption procedures. The encrypted data can be decrypted only by using these keys.

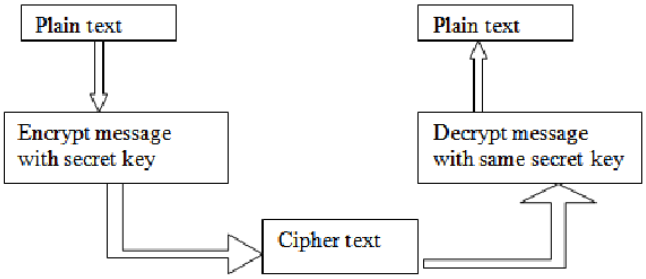
Public Key Cryptography

- ✓ Public key cryptography (PKC) is an encryption technique that uses a paired public and private key (or asymmetric key) algorithm for secure data communication.
- ✓ A message sender uses a recipient's public key to encrypt a message.
- ✓ To decrypt the sender's message, only the recipient's private key may be used.



Secret Key Cryptography

- ✓ Here only one key is used for both encryption and decryption. This type of encryption is also referred to as symmetric encryption.



UNIT IX – Transaction & Concurrency Control [4 Hrs]

Transaction Concept

The transaction is a set of logically related operation. It contains a group of tasks. A transaction is an action or series of actions. It is performed by a single user to perform operations for accessing the contents of the database.

Let's take an example of a simple transaction. Suppose a bank employee transfers Rs 500 from A's account to B's account. This very simple and small transaction involves several low-level tasks.

A's Account

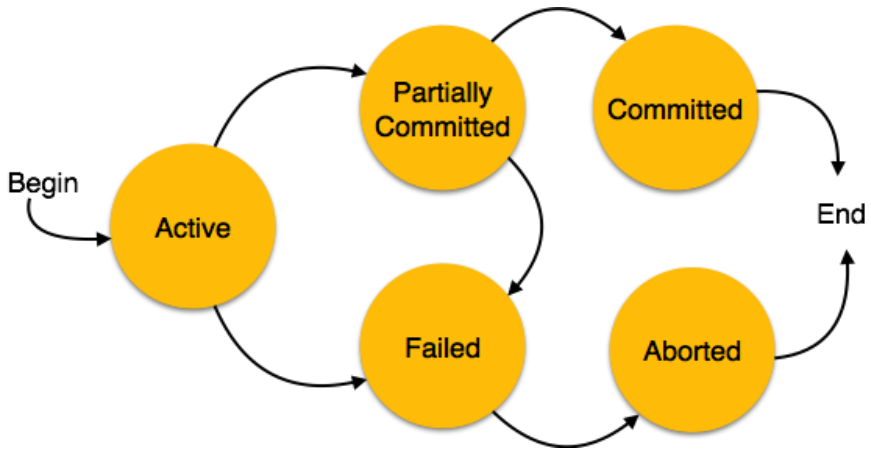
```
Open_Account(A)
Old_Balance = A.balance
New_Balance = Old_Balance - 500
A.balance = New_Balance
Close_Account(A)
```

B's Account

```
Open_Account(B)
Old_Balance = B.balance
New_Balance = Old_Balance + 500
B.balance = New_Balance
Close_Account(B)
```

States of Transactions / Transaction Model

A transaction in a database can be in one of the following states –



1) **Active –** In this state, the transaction is being executed. This is the initial state of every transaction.

2) **Partially Committed –** When a transaction executes its final operation, it is said to be in a partially committed state.

3) **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.

4) **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

5) **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.

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.

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.
- Consider the following transaction **T** consisting of **T1** and **T2**: Transfer of 100 from account **X** to account **Y**.

Before: X : 500	Y: 200
Transaction T	
T1	**T2**
Read (X) X: = X – 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

- If the transaction fails after completion of **T1** but before completion of **T2**. (say, after **write(X)** but before **write(Y)**), then amount has been deducted from **X** but not added to **Y**.
- This results in an inconsistent database state.
- Therefore, the transaction must be executed in entirety in order to ensure correctness of database state.

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.

- Referring to the example above, the total amount before and after the transaction must be maintained.

  Total **before T** occurs = **500 + 200 = 700**.

  Total **after T occurs** = **400 + 300 = 700**.
- Therefore, database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result, T is incomplete.

3) Isolation

- In a database system where more than one transaction is 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.

Let **X= 500, Y = 500.**
Consider two transactions **T** and **T".**

T	T''
Read (X) X: = X*100 Write (X) Read (Y) Y: = Y – 50 Write	Read (X) Read (Y) Z: = X + Y Write (Z)

Suppose **T** has been executed till **Read (Y)** and then **T''** starts. As a result, interleaving of operations takes place due to which **T''** reads correct value of **X** but incorrect value of **Y** and sum computed by
T'': (X+Y = 50, 000+500=50, 500)
is thus not consistent with the sum at end of transaction:
T: (X+Y = 50, 000 + 450 = 50, 450).
This results in database **inconsistency**, due to a **loss of 50 units**. Hence, transactions must take place in **isolation** and changes should be visible only after a they have been made to the main memory.

4) 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.

Serializability

When multiple transactions are running concurrently then there is a possibility that the database may be left in an inconsistent state. **Serializability** is a concept that helps us to check which schedules are serializable. A **serializable schedule** is the one that always leaves the database in consistent state.

- ✓ **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.

Types of Serializability

There are two types of Serializability –

1. Conflict Serializability
2. View Serializability

1) Conflict Serializability

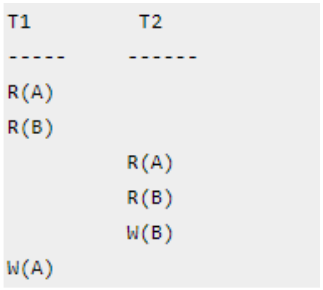
Conflict Serializability is one of the type of Serializability, which can be used to check whether a non-serial schedule is conflict serializable or not. A schedule is called conflict serializable if we can convert it into a serial schedule after swapping its non-conflicting operations.

Conflicting operations

Two operations are said to be in conflict, if they satisfy all the following three conditions:

1. Both the operations should belong to different transactions.
2. Both the operations are working on same data item.
3. At least one of the operation is a write operation.

Let's consider this schedule:



```
T1          T2
-----       ------
R(A)
R(B)
            R(A)
            R(B)
            W(B)
W(A)
```

To convert this schedule into a serial schedule we must have to swap the **R(A)** operation of transaction T2 with the **W(A)** operation of transaction T1.
However, we cannot swap these two operations because they are conflicting operations, thus we can say that this given schedule is **not Conflict Serializable**.

Example of Conflict Serializability

Lets take another example:

```
T1          T2
-----       ------
R(A)
            R(A)
            R(B)
            W(B)
R(B)
W(A)
```

Lets **swap non-conflicting operations**:

After swapping R(A) of T1 and R(A) of T2 we get:

```
T1          T2
-----       ------
            R(A)
R(A)
            R(B)
            W(B)
R(B)
W(A)
```

After swapping R(A) of T1 and R(B) of T2 we get:

```
T1          T2
-----       ------
            R(A)
            R(B)
R(A)
            W(B)
R(B)
W(A)
```

After swapping R(A) of T1 and W(B) of T2 we get:

```
T1          T2
-----       ------
            R(A)
            R(B)
            W(B)
R(A)
R(B)
W(A)
```

We finally got a serial schedule after swapping all the non-conflicting operations so we can say that the given schedule is **Conflict Serializable**.

2) View Serializability

View Serializability is a process to find out that a given schedule is view serializable or not. Two schedules S1 and S2 are said to be view equal if below conditions are satisfied:

a) **Initial Read**

If a transaction T1 reading data item A from initial database in S1 then in S2 also T1 should read A from initial database.

```
  T1      T2      T3
--------------------
        R(A)
 W(A)
                R(A)
        R(B)
```

Transaction T2 is reading A from initial database.

b) **Updated Read**

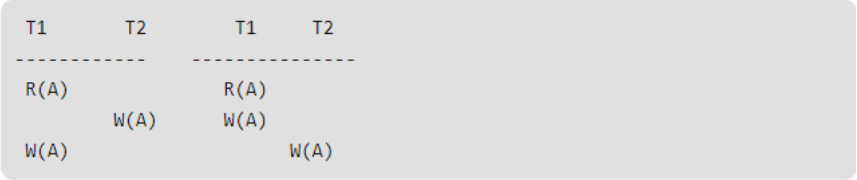
If Ti is reading A which is updated by Tj in S1 then in S2 also Ti should read A which is updated by Tj.

```
  T1      T2      T3          T1      T2      T3
--------------------        ----------------
  W(A)                        W(A)
        W(A)                                    R(A)
                R(A)                    W(A)
```

Above two schedule are not view equal as in S1 :T3 is reading A updated by T2, in S2 T3 is reading A updated by T1.

c) **Final Write operation**

If a transaction T1 updated A at last in S1, then in S2 also T1 should perform final write operations.



```
T1        T2          T1      T2
------------      ---------------
R(A)                R(A)
          W(A)      W(A)
W(A)                      W(A)
```

Above two schedule are not view as Final write operation in S1 is done by T1 while in S2 done by T2.

Concurrency Control

- ✓ Concurrency control is the procedure in DBMS **for managing simultaneous operations** without conflicting with each another.
- ✓ Concurrent access is quite **easy** if all users are **just reading data**. There is no way they can interfere with one another.
- ✓ Though for any practical database, would have a **mix of reading and WRITE operations** and hence the concurrency is a challenge.
- ✓ Concurrency control is used to address such conflicts which mostly occur with a **multi-user system**.
- ✓ It helps you to make sure that database transactions are performed concurrently without violating the data integrity of respective databases.
- ✓ Therefore, concurrency control is a most important element for the proper functioning of a system where two or multiple database transactions that require access to the same data, are executed simultaneously.

Why use Concurrency method?

Reasons for using Concurrency control method is DBMS:

- To apply Isolation through mutual exclusion between conflicting transactions
- To resolve read-write and write-write conflict issues
- To preserve database consistency through constantly preserving execution obstructions
- The system needs to control the interaction among the concurrent transactions. This control is achieved using concurrent-control schemes.
- Concurrency control helps to ensure serializability

Concurrency Control Protocols

Different concurrency control protocols offer different benefits between the amount of concurrency they allow and the amount of overhead that they impose.

- Lock-Based Protocols
- Two Phase
- Timestamp-Based Protocols
- Validation-Based Protocols

Lock-Based Protocols

A **lock** is a **data variable** which is **associated** with a **data item**. This lock signifies that operations that can be performed on the data item. Locks help synchronize access to the **database items by concurrent transactions.** All lock requests are made to the concurrency-control manager. Transactions proceed only once the lock request is granted.

- ✓ **Binary Locks:** A Binary lock on a data item can either locked or unlocked states.
- ✓ **Shared/exclusive:** This type of locking mechanism separates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is called an exclusive lock.

1. Shared Lock (S):

- ✓ A shared lock is also called a **Read-only lock**. With the shared lock, the data item can be shared between transactions.
- ✓ This is because you will never have permission to update data on the data item.
- ✓ For example, consider a case where two transactions are reading the account balance of a person.
- ✓ The database will let them read by placing a shared lock.
- ✓ However, if another transaction wants to update that account's balance, shared lock prevent it until the reading process is over.

2. Exclusive Lock (X):

- ✓ With the Exclusive Lock, a data item can be read as well as written.
- ✓ This is exclusive and can't be held concurrently on the same data item. X-lock is requested using lock-x instruction.
- ✓ Transactions may unlock the data item after finishing the 'write' operation.
- ✓ For example, when a transaction needs to update the account balance of a person. You can allow this transaction by placing X lock on it.
- ✓ Therefore, when the second transaction wants to read or write, exclusive lock prevents this operation.

Deadlock Handling

- ✓ Deadlock refers to a specific situation where two or more processes are waiting for each other to release a resource or more than two processes are waiting for the resource in a circular chain.
- ✓ A deadlock is a condition where two or more transactions are waiting indefinitely for one another to give up locks.
- ✓ Deadlock is said to be one of the most feared complications in DBMS as no task ever gets finished and is in waiting state forever.

Deadlock Example in DBMS

- ✓ For example: In the student table, transaction T1 holds a lock on some rows and needs to update some rows in the grade table.
- ✓ Simultaneously, transaction T2 holds locks on some rows in the grade table and needs to update the rows in the Student table held by Transaction T1.

- ✓ Now, the main problem arises. Now Transaction T1 is waiting for T2 to release its lock and similarly, transaction T2 is waiting for T1 to release its lock.
- ✓ All activities come to a halt state and remain at a standstill. It will remain in a standstill until the DBMS detects the deadlock and aborts one of the transactions.

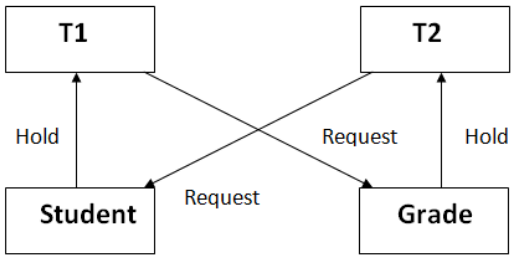


Figure: Deadlock in DBMS

Deadlock Avoidance

- ✓ When a database is stuck in a deadlock state, then it is better to avoid the database rather than aborting or restating the database. This is a waste of time and resource.
- ✓ Deadlock avoidance mechanism is used to detect any deadlock situation in advance. A method like "**wait for graph**" is used for **detecting** the deadlock situation but this method is suitable only for the smaller database. For the larger database**, deadlock prevention** method can be used.

Deadlock Prevention

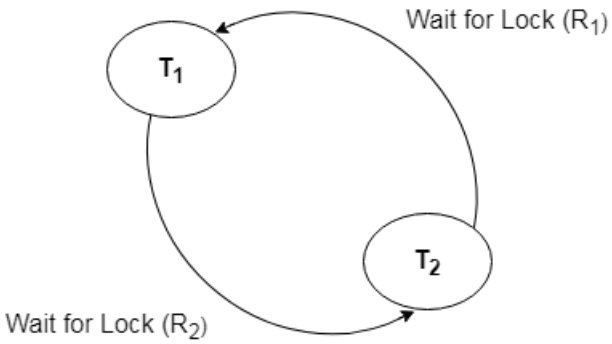
- Deadlock prevention method is suitable for a large database. If the resources are allocated in such a way that deadlock never occurs, then the deadlock can be prevented.
- The Database management system analyzes the operations of the transaction whether they can create a deadlock situation or not. If they do, then the DBMS never allowed that transaction to be executed.

Deadlock Detection

- ✓ In a database, when a transaction waits indefinitely to obtain a lock, then the DBMS should detect whether the transaction is involved in a deadlock or not.
- ✓ The lock manager maintains a Wait for the graph to detect the deadlock cycle in the database.

Wait for Graph

- This is the suitable method for deadlock detection. In this method, a graph is created based on the transaction and their lock. If the created graph has a cycle or closed loop, then there is a deadlock.
- The wait for the graph is maintained by the system for every transaction which is waiting for some data held by the others. The system keeps checking the graph if there is any cycle in the graph.



Timestamp-based Protocols

- ✓ The timestamp-based algorithm uses a timestamp to serialize the execution of concurrent transactions. This protocol ensures that every conflicting read and write operations are executed in timestamp order.
- ✓ The protocol uses the **System Time or Logical Count as** a Timestamp.
- ✓ **The older transaction is always given priority** in this method. It uses system time to determine the time stamp of the transaction. This is the most commonly used concurrency protocol.
- ✓ Lock-based protocols help you to manage the order between the conflicting transactions when they will execute. Timestamp-based protocols manage conflicts as soon as an operation is created. **Example**,

```
Suppose there are there transactions T1, T2, and T3.
T1 has entered the system at time 0010
T2 has entered the system at 0020
T3 has entered the system at 0030
Priority will be given to transaction T1, then transaction T2 and lastly Transaction T3.
```

Assignment to Students

Unit – I

1. What is DBMS? Explain features of DBMS.
2. Differentiate database and DBMS.
3. Explain advantages and disadvantages of DBMS.
4. Explain objective and importance of DBMS.
5. List and explain application of DBMS.
6. Differentiate traditional flat file system and database.

Unit –II

7. What do you mean by database design? Explain overall database designing process in detail.
8. What is abstraction? List and explain different levels of abstraction.
9. Explain structure of DBMS.
10. What do you mean by Data Independence? Explain physical and logical data independence in detail.
11. What is database architecture? Explain 1 tier, 2 tier and 3 tier database architecture.
12. Differentiate DDL and DML with examples.
13. What do you mean by data model? Explain different data models in detail.
14. What is QBE? Explain working and advantages of QBE.
15. Define entity and attributes. Differentiate strong and weak entity set.
16. Explain aggregation and generalization with examples.
17. Explain the process of converting ER Diagrams to Tables.
18. Construct an E-R diagram for a car insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents. Each insurance policy covers one or more cars, and has one or more premium payments associated with it. Each payment is for a particular period of time, and has an associated due date, and the date when the payment was received.
19. Design a database for an automobile company to provide to its dealers to assist them in maintaining customer records and dealer inventory and to assist sales staff in ordering cars. Each vehicle is identified by a vehicle identification number(VIN). Each individual vehicle is a particular model of a particular brand offered by the company (e.g., the XF is a model of the car brand Jaguar of Tata Motors). Each model can be offered with a variety of options, but an individual car may have only some (or none) of the available options. The database needs to store information about models, brands, and options, as well as information about individual dealers, customers, and cars. Your design should include an E-R diagram, a set of relational schemas, and a list of constraints, including primary-key and foreign-key constraints.
20. Design a database for an airline. The database must keep track of customers and their reservations, flights and their status, seat assignments on individual flights, and the schedule and routing of future flights. Your design should include an E-R

diagram, a set of relational schemas, and a list of constraints, including primary-key and foreign-key constraints.

21. Design an E-R diagram for keeping track of the exploits of your favourite sports team. You should store the matches played, the scores in each match, the players in each match, and individual player statistics for each match. Summary statistics should be modeled as derived attributes.

Unit – III

22. What is RDBMS? Explain the structure of RDBMS with example.
23. Define database schema. Develop database schema for the following:
    a) Hotel Management System
    b) Library Management System
    c) Bus Ticketing System
24. What do you mean by key? Explain different types of keys with example.
25. What is Relational Algebra? Explain different operations performed in relational algebra with examples.
26. Write relational algebra for the following relations:

Employee

Emp_Id	Name	Address	Salary	Dept_Id

Department

Dept_Id	Dept_Name	Floor

    i. Select name and address of employees whose salary is between 10000 and 20000.
    ii. Select employee id, employee name and department name of employees working in first floor.
    iii. Select all records of department which are in second floor.
    iv. Select name, address and department name of employees which are from Birtamode.
    v. Select employee id and name of employees having salary more than 10000 and from Kathmandu.

Unit – IV

27. What is Normalization? Explain advantages and disadvantages of Normalization.
28. What are differernt anomalies? Explain each of them with suitable examples.
29. What is functional dependency? Explain different types of functional dependencies with example.
30. Explain 1NF, 2NF, 3NF, BCNF and 4NF with suitable example.
31. Convert the following:

1. **Convert the following Student table to 2NF.**

sid	name	course_id	course	phone_no
1	Rohit	101	BCA	9833888831
2	Ravi	102	BBA	3555939392
3	Shyam	103	MCA	4994994993
4	Hari	104	MBA	4884484844
5	Ram	105	BBS	4908590495

3. **Normalize the following Employee table upto 3NF.**

emp_id	name	address	contact	branch_code	branch
1	Rohit	Btm	9866666,985555	101	Dhulabari
2	Ravi	Btm	983333333	102	Surunga
3	Shyam	Ktm	98777777	102	Surunga
4	Hari	Brt	98334444,9874774	101	Dhulabari
5	Ram	Ktm	98123444	103	Bhadrapur

2. **Convert the following Staff Table to 3NF.**

staff_id	name	address	department	department _contact
1	Rohit	Btm	Accounting	9866666666
2	Ravi	Btm	HR	9877777777
3	Shyam	Ktm	Accounting	9866666666
4	Hari	Brt	HR	9877777777
5	Ram	Ktm	Marketing	9833333333

Unit V, VI & VII

32. Define SQL. Explain importance of SQL with examples.
33. What are aggregate functions in SQL? Explain each of them with suitable examples.
34. What do you mean by constraint? Explain different types of constraints with examples.
35. Differentiate ORDER BY and GROUP BY clause with suitable example.
36. What do you mean by JOIN? Explain different types of join with suitable examples.
37. Explain importance of creating views in database. How views can be created in database?
38. Explain stored procedure with example.
39. How parameters can be created in stored procedure? Explain with example.
40. Explain trigger with example. What are the limitation of trigger?
41. Explain the importance of creating index in database. Explain the process of creating and dropping index.

Unit – VIII

42. Explain query processing with suitable diagram. How query cost is measured?
43. Explain different algorithms used for selection operation.
44. Explain sorting operation with the help of external merge sort.
45. What do you mean by query optimization? Explain query optimization process.
46. Define query optimizer. Explain importance of query optimization.
47. What do you mean by DBA? Explain different types of DBA.
48. What are the different roles and responsibilities of DBA?
49. Define database security. What are the different database security issues?
50. What are the different types of database security? Differentiate MAC and DAC with example.
51. What do you mean by encryption and decryption? Explain public key and secret key cryptography with example.

Unit – IX

52. Define transaction. Explain transaction model along with different states of transaction in detail.
53. Explain ACID properties of transaction with example.
54. What do you mean by serializability? Explain different types of serializability with example.
55. Describe concurrency control. What are the different types of concurrency control protocol? Explain lock based and time stamp based protocol in detail.
56. Explain deadlock in DBMS. How we can avoid deadlock in DBMS? Explain with example.

Lab Sheet - 1

1. Write SQL Query to create following table (Student).

Fileds	Datatype	Null	Key	Default	Check	Extra
student_id	int(11)	No	Primary			Auto_Increment
name	varchar(50)	No				Unique
address	varchar(100)	No		Birtamode		
class_id	int(11)	No	Foreign			
section	varchar(50)	Yes				
Age	Int(11)	No		16	Age>=15	

Note: Foreign key references to (Class) Table.

2. Write SQL query to drop primary key from above table.
3. Write SQL query to drop foreign key from above table.
4. Write SQL query to set student id as primary key.
5. Write SQL query to set class id as foreign key.
6. Write SQL query to remove unique constraint from name.
7. Write SQL query to remove default constraint from age.
8. Write SQL query to add unique constraint to section.
9. Write SQL query to add default value 18 to age.
10. Write SQL query to change column name address to location.
11. Write SQL query to add new column email and make it not null.
12. Write SQL query to remove column section from above table.
13. Write SQL query to add new column contact and make data type as integer.
14. Write SQL query to change data type of column contact to varchar and make it unique.
15. Write SQL query to change default value of address to Kathmandu.
16. Insert five set of records in above table.
17. Write SQL query to update name and address of student whose student id is 5.
18. Write SQL query to delete all the records of student having age greater than 20.
19. Write SQL query to update age of student having address btm.
20. Write SQL query to delete all records of student having student id 1.
21. Write SQL query to select all records of student.
22. Write SQL query to select all records of student having student id 3.
23. Write SQL query to select name and address of students whose age is greater than 21.
24. Write SQL query to select student id and name of students whose address in Birtamode.
25. Write SQL query to select records of students whose class id is 5 and address is Kathmandu.
26. Write SQL query to select maximum age from above table.
27. Write SQL query to select minimum age of students whose address is Birtamode.
28. Write SQL query to find total number of students having class id 5 and age greater than 19.

29. Write SQL query to find average age of students whose class id is 4 and section is B.
30. Write SQL query to select students whose address starts with letter 'B'.
31. Write SQL query to count those students whose name ends with letter 'R'.
32. Write SQL query to select name and age of students whose having address btm or ktm.
33. Write SQL query to select sum of age of students having id 1,2 and 3.
34. Write SQL query to select students whose age is between 18 and 22.
35. Write SQL query to select total students of each age group.
36. Write SQL query to select class id, name and maximum age of students studying in each class.
37. Write SQL query to select student's records by arranging in descending order on the basis of student id.
38. Write SQL query to select student id and name by of students whose age is greater than 20 after arranging records in alphabetical order on the basis of name.
39. Write SQL query to select records of student whose age is maximum among all the students.
40. Write SQL query to select student id and name of student whose student id is maximum among all the students.
41. Write SQL query to select name and age of student whose age is minimum than the average age of all students.
42. Write SQL query to list all the students except 'btm & 'ktm in asc order of age.
43. Write SQL query the students who does not belong to address 'btm'.
44. Write SQLL query to display the location of 'Ram''.
45. Write SQL query to display the total information of student table along with name and location of all the students having address 'Birtamode'.
46. Create table below with appropriate data type and constraints.

Employee

Emp_Id	Name	Address	Salary	Dept_Id

Department

Dept_Id	Dept_Name	Floor

47. Use all types of joins to select employee id, name and department name of employees.
48. Select name and address of employees whose salary is between 10000 and 20000.
49. Select employee id, employee name and department name of employees working in first floor.
50. Select all records of department which are in second floor.
51. Select name, address and department name of employees which are from Birtamode.
52. Select employee id and name of employees having salary more than 10000 and from Kathmandu.
53. Select name, department name and floor of employee whose name start with letter 'R' and age is greater than 30.
54. Select employee id and department name of employees whose floor is 'first' by arranging in ascending order on the basis of salary.
55. Select total number of employee working in each department.

56. Select maximum salary of employee working in each floor and whose department is ‘Finance’.
57. Select name and department name of employees whose salary is greater than average salary of all employees.
58. Select name and address of employee whore salary is between 20000 and 30000 and floor is ‘second’.
59. Select name and department name employee whose age is minimum.
60. Select sum of salary of all employees whose name ends with letter ‘s’ and department is ‘Account’.

Lab Sheet – 2

Consider the following table,

Employee

Emp_Id	Name	Address	Salary	Dept_Id

1. Create a view named Birtamode Employees that shows Employee id, name and address of employees having address Birtamode.
2. Create a view named Our Salary that shows employee name and salary whose salary is greater than 20,000.
3. Write SQL query to drop view Our Salary.
4. Write SQL query to add column salary on view Birtamode Employees.
5. Write SQL query to create stored procedure named SelectRecords that selects all records from Employee table.
6. Write SQL query to create stored procedure named MyEmployees that selects employees records of a particular address.
7. Write SQL query to create stored procedure named MyEmployees1 that selects employees records of a particular address and department id.
8. Write SQL query to drop above stored procedure SelectRecords.
9. Write SQL query to insert records in above table using stored procedure.
10. Write SQL query to delete record of employee whose employee id is given in parameter using stored procedure.
11. Write SQL query to update name and address of employee on the basis of salary. (Here, name, address and salary is given in parameter)
12. Write SQL query to update name and salary of employee on the basis of employee id and salary. (Here, name, employee id and salary is given in parameter)
13. Write SQL query to display maximum salary returned by stored procedure.
14. Write SQL query to display average salary of employees returned by stored procedure.
15. Write SQL query to create a trigger named MyTrigger.
16. Write SQL query to drop above trigger.
17. Write SQL query to create index named MyIndex on name of employee.
18. Write SQL query to create index named MyIndex1 on address and salary of employee.
19. Write SQL query to drop index MyIndex.
20. Write SQL query to drop index MyIndex1.