

UNIT I RELATIONAL DATABASES

Purpose of Database System – Views of data – Data Models – Database System Architecture – Introduction to relational databases – Relational Model – Keys – Relational Algebra – SQL fundamentals – Advanced SQL features – Embedded SQL– Dynamic SQL

1.1 PURPOSE OF DATABASE SYSTEM

File System

A file system is the methods and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organized on the disk. File systems can differ between operating systems

Types of File System:

There are number of file systems with different logical structures and properties(speed, size). Example:File Allocation Table (FAT),Global File System (GFS),Net Technology File System (NTFS), Hierarchical File system(HFS), Universal Disk Format(UDF)

Drawbacks of using file systems to store data:

- Data redundancy and inconsistency
 - Multiple file formats, duplication of information in different files
- Difficulty in accessing data
 - Need to write a new program to carry out each new task
- Data isolation — multiple files and formats
- Integrity problems
 - Integrity constraints (e.g. account balance > 0) become “buried” in program code rather than being stated explicitly
 - Hard to add new constraints or change existing one
- Atomicity of updates
 - Failures may leave database in an inconsistent state with partial updates carried out
 - Example: Transfer of funds from one account to another should either complete or not happen at all
- Concurrent access by multiple users
 - Concurrent accessed needed for performance
 - Uncontrolled concurrent accesses can lead to inconsistencies

- Example: Two people reading a balance and updating it at the same time
- Security problems
 - Hard to provide user access to some, but not all, data

These difficulties provoked the development of database management systems

Data:

Data could be any fact that can be recorded. It could be in any form including text, number, image, audio and video.

Example

Student_name - Text form

Student_Mobile_no - Number form

Student_photo - Images

Information:

A collection of related data which will provide a meaningful message.

Table 1: Comparison between data and information

Data	Information
Data are atomic level pieces of the information	Information is a collection of data
Data do not help in decision making	Information help in decision making
Data are generally in unorganized form	Information is in organized form
Data are collected from the source directly and hence is not dependent on information	Information is dependent on the data that is gathered.

Metadata:

A metadata or data dictionary is defined as the data about the data. It is also known as system catalog. It is used by developers to develop software, query, and procedure. Data dictionary is a repository of information about database.

Database:

Collection of logically interrelated data is called as database.

Example:

- Traditional database (Contains only text and numbers)
- Multimedia database (Contains video and audio)
- Geographic Information system (Contains only images)
- Real-time database (Example: database which is used in supermarket)

Database Management System:

The software or a set of program to define, construct, manipulate database is called database management system. One database system could not access all type of database. Based on the type of database, the database system will vary.

The collection of database and database management system is called as database system.

Database Applications:

- Banking: all transactions
- Airlines: reservations, schedules
- Universities: registration, grades
- Sales: customers, products, purchases
- Online retailers: order tracking, customized recommendations
- Manufacturing: production, inventory, orders, supply chain
- Human resources: employee records, salaries, tax deductions

Advantages of DBMS:**Data independence:**

Application programs should be as independent as possible from details of data representation and storage. The DBMS can provide an abstract view of the data to insulate application code from such details.

Efficient data access:

A DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently. This feature is especially important if the data is stored on external storage devices.

Data integrity and security:

If data is always accessed through the DBMS, the DBMS can enforce integrity constraints on the data. For example, before inserting salary information for an employee, the DBMS can check that the department budget is not exceeded. Also, the DBMS can enforce access controls that govern what data is visible to different classes of users.

Data administration:

When several users share the data, centralizing the administration of data can offer significant improvements. Experienced professionals, who understand the nature of the data being managed, and how different groups of users use it, can be responsible for organizing the data representation to minimize redundancy and for fine-tuning the storage of the data to make retrieval efficient.

Concurrent access and crash recovery:

A DBMS schedules concurrent accesses to the data in such a manner that users can think of the data as being accessed by only one user at a time. Further, the DBMS protects users from the effects of system failures.

Reduced application development time:

Clearly, the DBMS supports many important functions that are common to many applications accessing data stored in the DBMS. This, in conjunction with the high-level interface to the data, facilitates quick development of applications. Such applications are also likely to be more robust than applications developed from scratch because many important tasks are handled by the DBMS instead of being implemented by the application.

Disadvantages of DBMS

DBMS is a complex piece of software, optimized for certain kinds of workloads (e.g., answering complex queries or handling many concurrent requests), and its performance may not be adequate for certain specialized applications.

DBMS is that an application may need to manipulate the data in ways not supported by the query language.

1.2 VIEWS OF DATA (LEVELS OF ABSTRACTION)

The data in a DBMS is described at three levels of abstraction, as illustrated in Figure 1.1 . The objective of three-level architecture is to separate each user's view of the database from the way the database is physically represented.

The collection of information stored in the database at a particular time period is called an instance of the database. The overall design of the database is called the database schema.

DBMS architecture has three levels:

- External Schema(view level)
- Conceptual Schema (logical level)
- Internal Schema (Physical level)

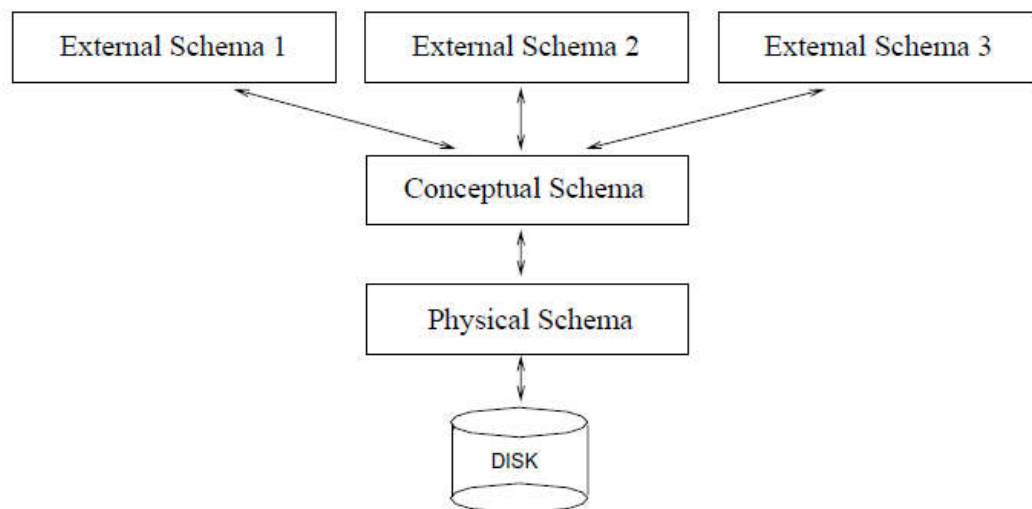


Figure 1.1: Levels of abstraction

External schema or view level:

This is the highest level of abstraction. The external schema deals with the way in which data is viewed by the individual user at the conceptual level. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group.

External view would depend on the user who is accessing the database. Any number of user views (even identical) may exist for a conceptual database.

Conceptual schema or logical level:

The logical level is the global view of the database. The Conceptual schema describes what data is stored in the database and relationships among the data. It also describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures.

It represents the following

- All entities, attributes and relationship
- The constraints of data
- Schematic information about the data
- Security and integrity of data

Conceptual view contains the logical view of the database. Any database has only one conceptual schema.

Internal schema (physical schema) or storage level:

The physical schema of the internal level describes how the data stored in database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database. This physical schema specifies additional storage details.

It concerns the following

- Storage space allocation
- Record description
- Record placement
- Data compression and data encryption

The internal view represents the physical location of each element on the disk of the server. Any database has only one internal schema.

Logical Data Independence: The ability to change the logical schema without changing the external schema or application programs. Changes in logical schema may include The addition or removal of new entities, attributes of relationship

Physical Data Independence: The ability to change the physical schema without changing the Logical schema. Changes in physical schema may include using new storage devices, different data structures and modifying indices

DATA MODEL:

A data model is 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.

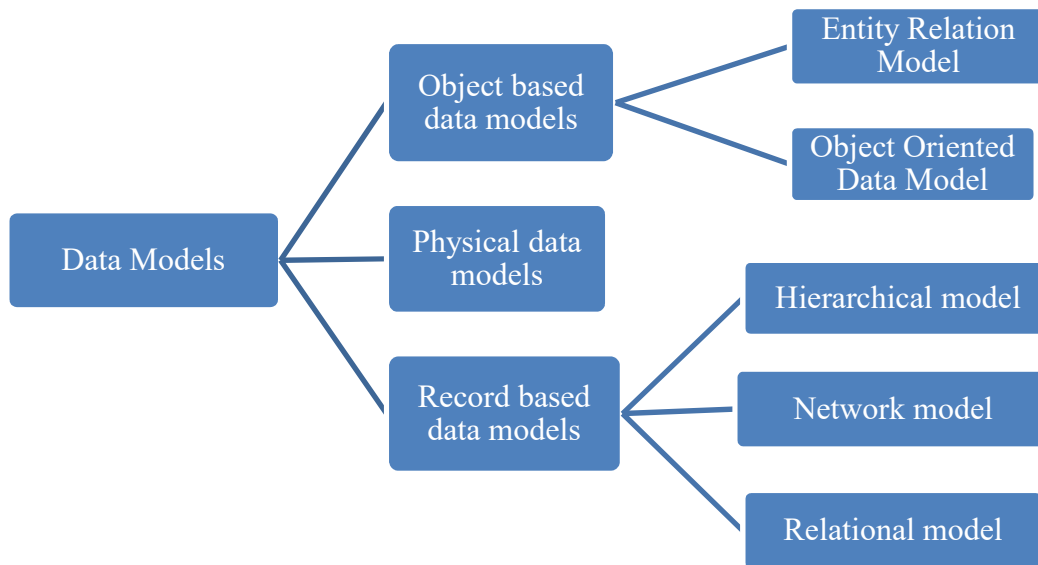


Figure 1.2: Types of Data models

1.3.1 Object Based Data Model

Object based data model is designed using the entities in the real world, attributes of each entity and their relationship

Entity Relationship Model

The entity relationship (ER) data model is well suited to data modelling for use with databases because it is fairly abstract and is easy to discuss and explain. ER models are readily translated to relations. ER models, also called an ER schema, are represented by ER diagrams.

ER modelling is based on two concepts:

- entities, defined as tables that hold specific information (data)
- relationships, defined as the associations or interactions between entities

An entity is an object in the real world with an independent existence that can be differentiated from other objects. An entity might be

- an object with physical existence (e.g., a lecturer, a student, a car)
- an object with conceptual existence (e.g., a course, a job, a position)

Advantages:

- It makes the requirement simple and easily understandable by representing simple diagrams.
- One can convert ER diagrams into record based data model easily.

Disadvantages:

- No standard notations are available for ER diagram. There is great flexibility in the notation. It's all depends upon the designer.
- It is meant for high level designs which cannot be simplified for low level design like coding.

Object Oriented Data Model

This data model is a method of representing real world objects. It considers each object in the world as objects and isolates it from each other. It groups its related functionalities together and allows inheriting its functionality to other related sub-groups. Object based data model is an extension of ER model with the notions of encapsulation, methods and object identity.

Advantages:

- New features can be added by easily added.
- It reduces the overhead and maintenance cost
- More flexible in the case of any changes.

Disadvantages:

- It is not widely developed and complete to use it in the database systems. Hence it is not accepted by the users.
- It is an approach for solving the requirement. It is not a technology. Hence it fails to put it in the database management systems.

1.3.2 Physical Data Model

Physical data model represents the actual design of a relational database which is dependent upon a specific version of a data persistent technology. A complete physical data model will include all the database artifacts required to create relationships between tables or to achieve performance goals, such as indexes, constraint definitions, linking tables, partitioned tables. Physical model is based on the table structure in the database.

Advantages:

It can have denormalized structure

It is independent on the RDBMS

Diasadvantage:

It depends on the user how he specifies the diagram and the RDBMS servers

1.3.3 Record Based Data Model

They describe data at the view and conceptual levels. These models specify logical structure of the database with records fields and attributes

Hierarchical Data Model

A hierarchical model represents the data in a tree-like structure in which there is a single parent for each record. This model structure allows the one-to-one and a one-to-many relationship between two/ various types of data. This structure is very helpful in describing many relationships in the real world, table of contents, any nested and sorted information. The hierarchical structure is used as the physical order of records in storage. One can access the records by navigating down through the data structure using pointers which are combined with sequential accessing. Therefore, the hierarchical structure is not suitable for certain database operations when a full path is not also included for each record.

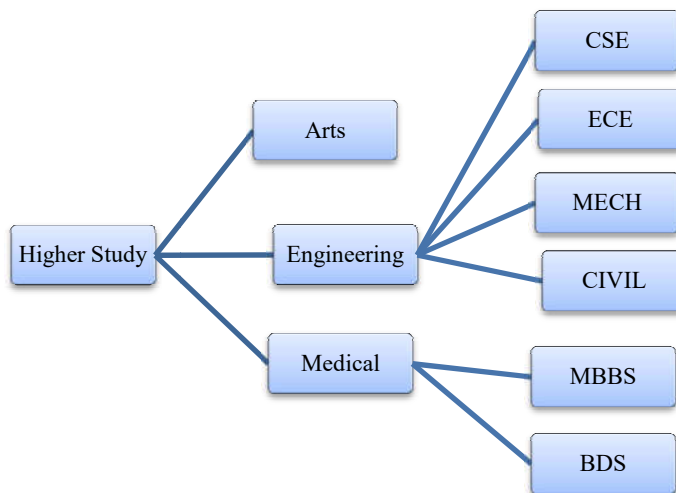


Figure 1.3: Hierarchical Data Model

Advantages

- Simplicity and Security
- Database Integrity and Efficiency

Disadvantages:

- Complexity of Implementation
- Lack of structural independence

- Operational Anomalies

Network Data Model

A network model is a database model that is designed as a flexible approach to representing objects and their relationships. A unique feature of the network model is its schema, which is viewed as a graph where relationship types are arcs and object types are nodes.



Figure 1.4: Network Data Model

Advantages:

- Simplicity and High speed retrieval
- Data integrity and data independence
- Ability to handle more relationship types

Disadvantages

- User friendly database management system cannot be created using the network model
- Lack of structural independence

Relational Model

A relational database represents all data in the database as simple two-dimensional tables called relations. Each row of a relational table, called tuple, represents a data entity with columns of the table representing attributes(fields). The allowable values for these attributes are called the domain. Each row in a relational table must have a unique primary key and also has some secondary keys which correspond with primary keys in other tables. Relational model is an example for record

based model. Relational model can have records of fixed structure. Most of the current databases are based on relational model.

Advantage:

- Structural Independence
- Conceptual simplicity
- Design, Implementation, maintenance and usage ease.

Disadvantage:

- Hardware overheads
- Ease of design can lead to bad design

1.3.5 Semi Structured Data Model

Semi structured data is used where individual data items of the same type may have different sets of attributes. The extensible markup language (xml) is widely used to represent semi structured data.

1.3 DATABASE ARCHITECTURE

Database architecture illustrates the components of the database systems and the connections among them. Database architecture can be centralized or client server architecture.

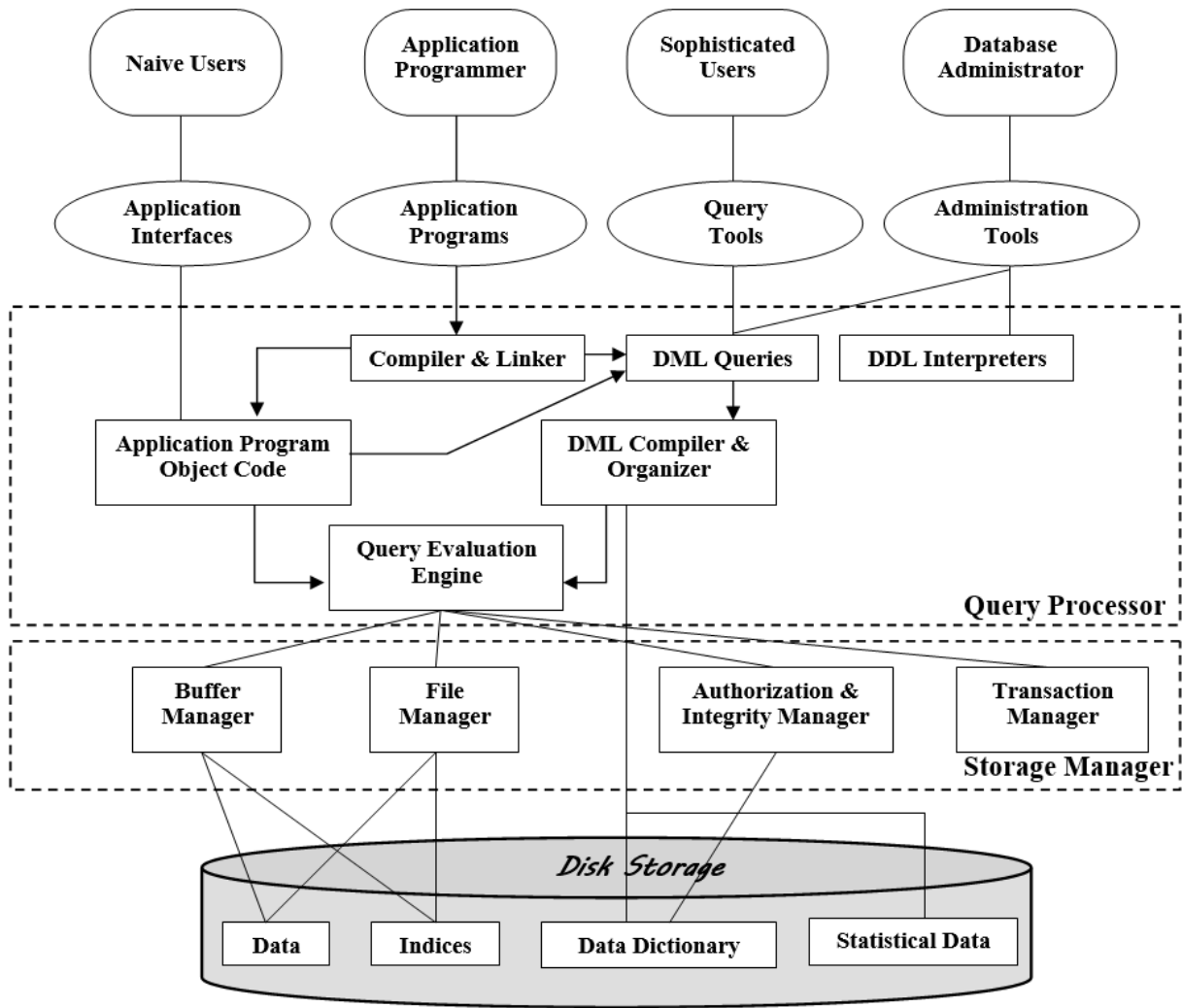


Figure 1.5: Database System Architecture

Users

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

- **Naive users** are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.
- **Application programmers** are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. Rapid application development (RAD) tools are tools that enable an application programmer to construct forms and reports with minimal programming effort.

- **Sophisticated users** interact with the system without writing programs. Instead, they form their requests either using a database query language or by using tools such as data analysis software. Analysts who submit queries to explore data in the database fall in this category. Specialized users are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework. Among these applications are computer-aided design systems, knowledgebase and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modelling systems.
- **Database Administrator** One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a database administrator (DBA). The functions of a DBA include:
 - **Schema definition.** The DBA creates the original database schema by executing a set of data definition statements in the DDL.
 - **Storage structure and access-method definition.**
 - **Schema and physical-organization modification.** The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.
 - **Granting of authorization for data access.** By granting different types of authorization, the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.
 - **Routine maintenance.** Examples of the database administrator's routine maintenance activities are:
 - Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding.
 - Ensuring that enough free disk space is available for normal operations and upgrading disk space as required.
 - Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

Query processor

- **DDL interpreter**, which interprets DDL statements and records the definitions in the data dictionary.
- **DML compiler**, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.

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

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

Storage Manager

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

- **Authorization and integrity manager**, which tests for the satisfaction of integrity constraints and checks the authority of users to access data.
- **Transaction manager**, which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.
- **File manager**, which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
- **Buffer manager**, which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory. The buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory. The storage manager implements several data structures as part of the physical system implementation:

- **Data files**, which store the database itself.
- **Data dictionary**, which stores metadata about the structure of the database, in particular the schema of the database.
- **Indices**, which can provide fast access to data items. Like the index in this textbook, a database index provides pointers to those data items that hold a particular value. For example, we could use an index to find the instructor record with a particular ID, or all instructor records with a particular name. Hashing is an alternative to indexing that is faster in some but not all cases.

Statistical data

A collection of data points or numerical values that can be categorized and subject to analysis; statistics are the raw material on which conclusions about cause-and-effect relationships are based.

SYSTEM ARCHITECTURE

The design of a DBMS depends on its architecture. It can be centralized or decentralized or hierarchical. The architecture of a DBMS can be seen as either single tier or multi-tier. N-tier architecture divides the whole system into related but independent **n** modules, which can be independently modified, altered, changed, or replaced

Two-tier Architecture

The application at the client end directly communicates with the database at the server side. The server side is responsible for providing query processing and transaction management functionalities. The user interfaces and application programs are run on the client side. The application on the client side establishes a connection with the server side in order to communicate with the DBMS.

An advantage of this type is that maintenance and understanding is easier, compatible with existing systems. However this model gives poor performance when there are a large number of users.

Three-tier architecture

The client does not directly communicate with the server. Instead, it interacts with an application server which further communicates with the database system and then the query processing and transaction management takes place. This intermediate layer acts as a medium for exchange of partially processed data between server and client. This type of architecture is used in case of large web applications.

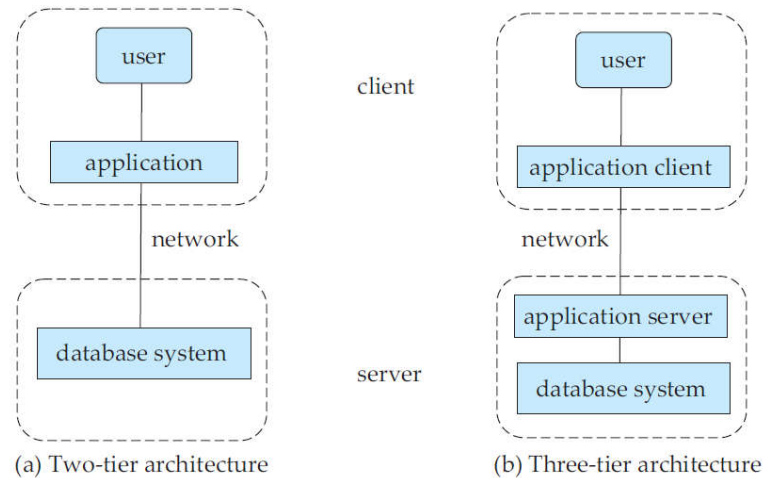


Figure 1.6: DBMS Architecture

Advantages:

- **Enhanced scalability** due to distributed deployment of application servers. Now, individual connections need not be made between client and server.
- **Data Integrity** is maintained. Since there is a middle layer between client and server, data corruption can be avoided/removed.
- **Security** is improved. This type of model prevents direct interaction of the client with the server thereby reducing access to unauthorized data.

Disadvantages:

Increased complexity of implementation and communication. It becomes difficult for this sort of interaction to take place due to presence of middle layers.

1.4 INTRODUCTION TO RELATIONAL DATABASE

A relational database is a set of formally described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables. Relational Database is derived from the mathematical function concept of mapping data sets and was developed by Edgar F. Codd. To access the database, you execute a structured query language (SQL) statement, which is the American National Standards Institute (ANSI) standard language for operating relational databases.

The language contains a large set of operators for partitioning and combining relations. The database can be modified by using the SQL statements.

A relational database uses relations or two-dimensional tables to store information. For example, you might want to store information about all the employees in your company. In a relational database, you create several tables to store different pieces of information about your employees, such as an employee table, a department table, and a salary table.

1.4.1 Relational Model

The principles of the relational model were first outlined by Dr. E. F. Codd in a June 1970 paper called “A Relational Model of Data for Large Shared Data Banks.” In this paper, Dr. Codd proposed the relational model for database systems.

Components of the Relational Model

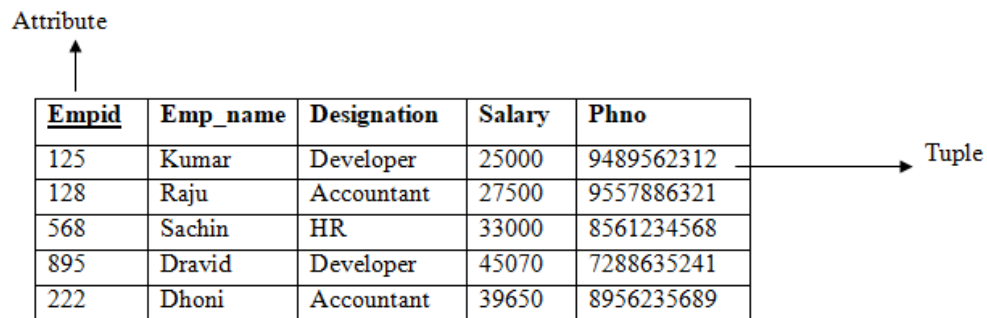
- Collections of objects or relations that store the data
- A set of operators that can act on the relations to produce other relations
- Data integrity for accuracy and consistency

Terminology Used in a Relational Database

A relational database can contain one or many tables. A table is the basic storage structure of an RDBMS. A table holds all the data necessary about something in the real world, such as employees, invoices, or customers.

- **Relation:** Relation is defined as a table comprising rows and columns. For example Employee is a relation as shown in figure
- **Tuple(Record):** A tuple is also called as row of a relation. Tuple is an each individual entry that exists in a table. Each row in a table should be identified by a primary key, which allows no duplicate rows. The order of rows is insignificant. For Example every row(tuple) or record represents all data required for a particular employee
- **Attribute(Field):** Named column of a relation. Empid, Empname, Designation, Salary, Phno are the attributes of the Employees relation. Column order is insignificant when storing data.
- **Domain:** It is defined as a set of allowed values(data types) for each attribute. For example the Empid attribute can have only integer values
- **Cardinality:** The number of tuples in a relation is called the cardinality of the relation. The cardinality of employees relation is 6

- **Degree:** The number of attributes in the relation. The degree of Employees relation is 5



<u>Empid</u>	Emp_name	Designation	Salary	Phno
125	Kumar	Developer	25000	9489562312
128	Raju	Accountant	27500	9557886321
568	Sachin	HR	33000	8561234568
895	Dravid	Developer	45070	7288635241
222	Dhoni	Accountant	39650	8956235689

Figure 1.7 Employees Relation

Properties of a Relational Database

- Values are atomic
- Each row is unique
- The sequence of columns is insignificant
- The sequence of rows is insignificant
- Each column has a unique name
- All values in a column should be in a same domain

Keys

An identifier is used to identify a row in a database uniquely. This identifier may be a attribute or a group attributes.

Super Key – A super key is a set of one or more columns (attributes) to uniquely identify rows in a table. A super set of the candidate key is the super key. Super key consists of primary key and candidate key

Few super keys of employees relation given in figure 1.7 are {Emp_id}, { Emp_id, Emp_name}, { Emp_id, Emp_name, Designation},{ Emp_id, Emp_name, Designation, Salary} and { Emp_id, Emp_name, Designation, Salary,Phno}

Candidate Key – A super key with no redundant attribute is known as candidate key. Candidate key is the minimal subset of superkey. All the candidate keys are super keys. There may be several candidate keys for a relation.

{ Emp_id, Emp_name} is one of candidate key of Employees relation in Figure 1.7

All the candidate keys are super keys

Primary Key – A primary is a column or set of columns in a table that uniquely identifies tuples (rows) in a table. Primary key cannot contain any NULL values. Primary key is a subset of candidate key

Properties of primary key

Stable- Value of primary key must not change

Minimal-Should be composed of minimum number of attributes

Definitive-Value must exist for every record during creation of a record

In the figure 1.7 Empid is the primary key

Alternate Key – Out of all candidate keys, only one gets selected as primary key, remaining keys are known as alternate or secondary keys.

Composite Key – A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called composite key.

Foreign Key – Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables. A foreign key must match an existing primary key value of a same or another relation

Projectid	Empid	Plocation
P123	125	Coimbatore
P225	568	Bangalore
P556	222	Delhi
P212	568	Pune

Figure 1.8 Emp_project Relation

In the Figure 1.8 Empid is a foreign key for the Emp_project relation which refers to the primary key(Emp_id) of the relation Employees given in the figure 1.7

1.7 RELATIONAL ALGEBRA

Relational algebra is procedural language with a set of operations performed on one or more relation to derive the result relation without changing the original relations . These operations enable a user to specify basic retrieval requests as relational algebra expressions. A sequence of

relational algebra operations forms a relational algebra expression, whose result will also be a relation that represents the result of a database query

Basic Operations:

- Select (σ)
- Project(π)
- Union(\cup)
- Set difference($-$)
- Cartesian Product(\times)
- Rename(ρ)

The SELECT Operation(σ)

The select operation return rows of the relation that satisfies the given condition. The syntax of selection operation is

$\sigma_{\langle \text{selection condition} \rangle}(\text{Relation R})$

Relation Name: Employee

Employee Relation

<u>Eid</u>	Name	Designation	Salary	Phno
125	Kumar	Developer	25000	9489562312
128	Raju	Accountant	27500	9557886321
568	Sachin	HR	33000	8561234568
895	Dravid	Developer	45070	7288635241
222	Dhoni	Accountant	39650	8956235689

Query:

$\sigma_{(\text{salary} \geq 30000)}(\text{Employee})$

Result

<u>Eid</u>	Name	Designation	Salary	Phno
568	Sachin	HR	33000	8561234568
895	Dravid	Developer	45070	7288635241
222	Dhoni	Accountant	39650	8956235689

Project Operation(π)

Project operation returns specified attributes from all rows of the input relation. The syntax of project operation is

$\pi_{\langle \text{Attribute List} \rangle}(\text{Relation R})$

Example

Relation Name: Employee

Query:

$\pi_{(\text{name, salary})}(\text{employee})$

Result:

Name	Salary
Kumar	25000
Raju	27500
Sachin	33000
Dravid	45070
Dhoni	39650

Union Operation(\cup)

The result of this operation, denoted by $R \cup S$, is a relation that includes all tuples that are either in R or in S or in both R and S

Example

Relation: R, S

R

Name
Sachin
Dhoni
Dravid
Kohli

S

Name
Dravid
Kohli
Raju
Kumar

Query: $R \cup S$

Result:

R ∪ S

Name
Sachin
Dhoni
Dravid
Kohli
Raju
Kumar

Set Difference(-)

The result of this operation, denoted by $R - S$, is a relation that includes all tuples that are in R but not in S

Example

Relation: R, S

Query: R-S

Output:

$R - S$

Name
Sachin
Dhoni

Cartesian Product(X)

This set operation produces a new element by combining every member from one relation with every member from the other relation

Example

Relation: Employee, Project;

Project Relation

Pid	Empid	Plocation
P123	125	Coimbatore
P225	568	Bangalore
P556	222	Delhi

Query:

Employee X project

Result:

<u>Eid</u>	Name	Designation	Salary	Phno	Empid	Pid	Plocation
125	Kumar	Developer	25000	9489562312	125	P123	Coimbatore
125	Kumar	Developer	25000	9489562312	568	P225	Bangalore
125	Kumar	Developer	25000	9489562312	222	P556	Delhi
128	Raju	Accountant	27500	9557886321	125	P123	Coimbatore
128	Raju	Accountant	27500	9557886321	568	P225	Bangalore
128	Raju	Accountant	27500	9557886321	222	P556	Delhi
568	Sachin	HR	33000	8561234568	125	P123	Coimbatore
568	Sachin	HR	33000	8561234568	568	P225	Bangalore
568	Sachin	HR	33000	8561234568	222	P556	Delhi
895	Dravid	Developer	45070	7288635241	125	P123	Coimbatore
895	Dravid	Developer	45070	7288635241	568	P225	Bangalore
895	Dravid	Developer	45070	7288635241	222	P556	Delhi
222	Dhoni	Accountant	39650	8956235689	125	P123	Coimbatore
222	Dhoni	Accountant	39650	8956235689	568	P225	Bangalore
222	Dhoni	Accountant	39650	8956235689	222	P556	Delhi

Rename operation(ρ)

It is used to rename a relation or to rename the output of any relational algebraic operation.

Example

$\rho_{(\text{worker})}(\text{employee})$

Other Operations**Intersection (\cap):**

The result of this operation, denoted by $R \cap S$, is a relation that includes all tuples that are in both R and in S

Example

Relation: R, S

Query:

$R \cap S$

Result

Name
Dravid
Kohli

Joins

A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied. Three types of joins available namely inner join, outer join, semi join.

Inner Join :

An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation. Theta Join, Equijoin, and Natural Join are called inner joins.

Theta Join (θ)

Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol θ . Theta join can use all kinds of comparison operators. The general form is

$R1 \bowtie_{\theta} R2$

Example

Relations: Employee, Project

Query:

Employee $\bowtie_{\text{Employee.Eid=Project.Empid}}$ Project

Result

<u>Eid</u>	Name	Designation	Salary	Phno	Empid	Pid	Plocation
125	Kumar	Developer	25000	9489562312	125	P123	Coimbatore
568	Sachin	HR	33000	8561234568	568	P225	Bangalore
222	Dhoni	Accountant	39650	8956235689	222	P556	Delhi

Equi Join (\bowtie_c)

Equijoin is a special case of join where only equality condition holds between a pair of attributes.
 As values of two attributes will be equal in result of equijoin, only one attribute will appear in result
 The general form is

$R1 \bowtie R2$

Example

Relations: Employee, Project

Query:

Employee $\bowtie_{\text{Employee.Eid=Project.Empid}}$ Project

Result

<u>Eid</u>	Name	Designation	Salary	Phno	Empid	Pid	Plocation
125	Kumar	Developer	25000	9489562312	125	P123	Coimbatore
568	Sachin	HR	33000	8561234568	568	P225	Bangalore
222	Dhoni	Accountant	39650	8956235689	222	P556	Delhi

Natural Join (\bowtie)

It is used to combine related tuples from two relations into single based on equality condition. There is no need to specify equality condition explicitly

Example

Relation: Employee, Project

Query:

employee \bowtie project

Result:

<u>Eid</u>	Name	Designation	Salary	Phno	Pid	Plocation
125	Kumar	Developer	25000	9489562312	P123	Coimbatore
568	Sachin	HR	33000	8561234568	P225	Bangalore
222	Dhoni	Accountant	39650	8956235689	P556	Delhi

Semi Join (\bowtie)(\ltimes)

A join where the result only contains the columns from one of the joined relations. A semi join between two tables returns rows from the first table where one or more matches are found in the second table.

Example

Relation: Employee, Project

Query:

employee ⋈ **project**

Result:

<u>Eid</u>	Name	Designation	Salary	Phno
125	Kumar	Developer	25000	9489562312
568	Sachin	HR	33000	8561234568
222	Dhoni	Accountant	39650	8956235689

Outer Join

Outer joins include all the tuples from the participating relations in the resulting relation. There are three kinds of outer joins – left outer join, right outer join, and full outer join.

Left Outer Join (⋈_L)

All the tuples from the Left relation, R, are included in the resulting relation. If there are tuples in R without any matching tuple in the Right relation S, then the S-attributes of the resulting relation are made NULL.

Example:

Customer Relation:

customer_id	first_name	last_name	email	address	city	state	zipcode
1	George	Washington	gswashington@usa.gov	3200 Mt Vernon Hwy	Mount Vernon	VA	22121

2	John	Adams	jadams@usa.gov	1250 Hancock St	Quincy	MA	02169
3	Thomas	Jefferson	tjefferson@usa.gov	931 Thomas Jefferson Pkwy	Charlottesville	VA	22902
4	James	Madison	jmadison@usa.gov	11350 Constitution Hwy	Orange	VA	22960
5	James	Monroe	jmonroe@usa.gov	2050 James Monroe Parkway	Charlottesville	VA	22902

Orders Relations

order_id	order_date	amount	customer_id
1	07/04/1776	\$234.56	1
2	03/14/1760	\$78.50	3
3	05/23/1784	\$124.00	2
4	09/03/1790	\$65.50	3
5	07/21/1795	\$25.50	10
6	11/27/1787	\$14.40	9

Query:

Customer ⋈ Orders

Result

first_name	last_name	order_date	order_amount
George	Washington	07/04/1776	\$234.56
John	Adams	05/23/1784	\$124.00
Thomas	Jefferson	03/14/1760	\$78.50
Thomas	Jefferson	09/03/1790	\$65.50
James	Madison	NULL	NULL
James	Monroe	NULL	NULL

Right Outer Join(⋈)

All the tuples from the Right relation, S, are included in the resulting relation. If there are tuples in S without any matching tuple in R, then the R-attributes of resulting relation are made NULL.

Example

Relation: Customer, Orders

Query:

Customer ⋈ Orders

Result:

first_name	last_name	order_date	order_amount
George	Washington	07/04/1776	\$234.56
Thomas	Jefferson	03/14/1760	\$78.50
John	Adams	05/23/1784	\$124.00
Thomas	Jefferson	09/03/1790	\$65.50
NULL	NULL	07/21/1795	\$25.50
NULL	NULL	11/27/1787	\$14.40

Full Outer Join (⋈)

All the tuples from both participating relations are included in the resulting relation. If there are no matching tuples for both relations, their respective unmatched attributes are made NULL.

Relation: Customer, Orders

Query:

Customer ⋈ Orders

Result

first_name	last_name	order_date	order_amount
George	Washington	07/04/1776	\$234.56
Thomas	Jefferson	03/14/1760	\$78.50
John	Adams	05/23/1784	\$124.00
Thomas	Jefferson	09/03/1790	\$65.50
NULL	NULL	07/21/1795	\$25.50
NULL	NULL	11/27/1787	\$14.40
James	Madison	NULL	NULL

James	Monroe	NULL	NULL
-------	--------	------	------

Division \div

The division operator returns those tuples from relation1 which are associated to every relation 2's tuple. Division operator can be applied only if Attributes of Relation2 is proper subset of attributes of Relation 1

Example

Relation: students, sports

students relation

roll	sports_name
1	Hockey
3	Tennis
3	Hockey
4	Hockey

sports relation

sports_name
Hockey
Tennis

Query: students \div sports

Result:

roll
3

1.8 SQL FUNDAMENTALS

IBM developed the original version of SQL, originally called Sequel, as part of the System R project in the early 1970s. The Sequel language has evolved since then, and its name has changed to SQL (Structured Query Language)

1.9.1 Data Definition Language (DDL)

The Data Definition Language is used to define the overall schema of the database. The DDL is also used to specify additional properties of the data. DDL deals with database schemas and descriptions, of how the data should reside in the database.

Example of DDL are:

- **CREATE** : to create database
- **ALTER** : alters the structure of the existing database
- **DROP** : delete objects from the database
- **TRUNCATE** : remove all records from a table, including all spaces allocated for the records are removed
- **RENAME**: Used to rename a relation

SQL Basic Data types

Basic Types

The SQL standard supports a variety of built-in types, including:

- **char(*n*)**: A fixed-length character string with user-specified length *n*. The full form, **character**, can be used instead.
- **varchar(*n*)**: A variable-length character string with user-specified maximum length *n*. The full form, **character varying**, is equivalent.
- **int**: An integer (a finite subset of the integers that is machine dependent). The full form, **integer**, is equivalent.
- **smallint**: A small integer (a machine-dependent subset of the integer type).
- **numeric(*p*, *d*)**: A fixed-point number with user-specified precision. The number consists of *p* digits (plus a sign), and *d* of the *p* digits are to the right of the decimal point. Thus, **numeric(3,1)** allows 44.5 to be stored exactly, but neither 444.5 or 0.32 can be stored exactly in a field of this type.
- **real, double precision**: Floating-point and double-precision floating-point numbers with machine-dependent precision.
- **float(*n*)**: A floating-point number, with precision of at least *n* digits.

CREATE:

It is used to create a table or relation.

Syntax :

CREATE TABLE *tablename* (*A1 D1, A2 D2, . . . ,An Dn, <integrity-constraint1>, . . . ,<integrity-constraintk>*);

where

tablename - the name of the table

Ai - the attribute name

Di - the domain(data type) of the attribute

integrity-constraint_i - data integrity constraint name

Integrity Constraints:

Integrity Constraints enforce rules on the data in a table whenever a row is inserted, updated, or deleted from that table. The constraint must be satisfied for the operation to succeed. It prevent the deletion of a table if there are dependencies from other tables

Primary Key Constraint: The Primary Key constraint uniquely identifies each record in a table. Primary keys must contain unique values, and cannot contain NULL values. A table can have only one primary key, which may consist of single or multiple fields.

Foreign Key Constraint: A foreign key is a key used to link two tables together. A foreign key is a field (or collection of fields) in one table that refers to the primary key in another table. The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

Unique Constraint: The unique constraint ensures that all values in a column are different. Both the unique and primary key constraints provide a guarantee for uniqueness for a column or set of columns. A primary key constraint automatically has a unique constraint.

NOT NULL Constraint: The NOT NULL constraint enforces a column to not accept NULL values. This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field

Check Constraint: The check constraint is used to limit the value range that can be placed in a column.

Default Constraint: The default constraint is used to provide a default value for a column.

Example

```
CREATE TABLE Employee (
    Eid int,
    Name varchar(255) NOT NULL,
    Designation varchar(255),
    Salary int , check(Salary>0),
    Phno varchar(255) , PRIMARY KEY(Eid)
);
```

Result:

Table created.

To view the structure of the table, **DESCRIBE table_name**

DESCRIBE Employee

Name	NULL	Type
Eid		int
Name	NOT NULL	varchar(255)
Designation		varchar(255)
Salary		Int
Phno		varchar(255)

ALTER :

The alter table statement is used to add, delete, or modify columns in an existing table.

Add Clause:

The new column becomes the last column. If a table already contains rows when a column is added, then the new column is initially null for all the rows.

Modify Clause:

Column modification can include changes to a column's data type, size, and default value. the data type is changed only if the column contains null values. A change to the default value of a column affects only subsequent insertions to the table.

Drop Clause:

only one column can be dropped at a time. The table must have at least one column remaining in it after it is altered. Once a column is dropped, it cannot be recovered.

Syntax:

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


ALTER TABLE table_name DROP column_name;

Where,

table_name - the name of the table

column_name – name of the attribute

datatype – domain of the attribute

Example 1:

ALTER TABLE employee ADD (Address varchar(255));

Result:

Table Altered.

DESCRIBE Employee

Name	NULL	Type
Eid		int
Name	NOT NULL	varchar(255)
Designation		varchar(255)
Salary		Int
Phno		varchar(255)
Address		varchar(255)

Example 2:

ALTER TABLE employee DROP Address;

Result:

Table Altered.

DESCRIBE Employee

Name	NULL	Type
Eid		int
Name	NOT NULL	varchar(255)
Designation		varchar(255)
Salary		Int
Phno		varchar(255)

DROP :

The DROP TABLE statement is used to drop an existing table in a database. The drop command deletes the entire table along with its content. The DROP TABLE statement cannot be roll backed.

Syntax:

```
DROP TABLE table_name;
```

Example:

```
DROP TABLE Employee;
```

Result:

Table Dropped.

```
DESC Employee;
```

No table found

RENAME :

It is used to rename a table.

Syntax

```
RENAME old_name TO new_name;
```

Where,

old_name - the old name of the table, view, sequence, or synonym.

new_name - the new name of the table, view, sequence, or synonym.

Example:

```
RENAME Employee TO emp;
```

Result:

Table Renamed.

TRUNCATE :

Truncate is used to remove all rows from a table and to release the storage space used by that table. It does not delete the table. The removed rows can not be roll backed.

Syntax:

```
TRUNCATE TABLE table_name;
```

Example:

```
TRUNCATE TABLE Employee.
```

1.9.2 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.

DML commands are not auto-committed. It means that the changes are not permanent to database, they can be rolled back. Data manipulation is retrieval of information, insertion of new information, deletion of information or modification of information stored in the database.

DML statements are :

- **SELECT** : retrieve data from the database
- **INSERT** : insert data into a table
- **UPDATE** : updates existing data in a table
- **DELETE** : delete all records from a database table

There are two types of DML :

1. **Procedural DML** : it requires a user to specify what data are needed and how to get those data.
2. **Non-Procedural DML** : it requires a user to specify what data are needed without specifying how to get those data.

The basic structure of an SQL query consists of three clauses: select, from, and where. The query takes as its input the relations listed in the from clause, operates on them as specified in the where and select clauses, and then produces a relation as the result. The select clause is used to list the attributes desired in the result of a query. The from clause is a list of the relations to be accessed in the evaluation of the query. The where clause is a predicate involving attributes of the relation in the from clause.

SELECT:

The select statement is used to select data from a database.

Syntax:

```
SELECT * | { [DISTINCT] column|expression [alias],...} FROM table_name [WHERE  
condition(s)] [GROUP BY group_by_expression] [HAVING group_condition] [ORDER BY  
column];
```

Where,

* - selects all columns

DISTINCT - suppresses duplicates

column|expression - selects the named column or the expression

alias - gives selected columns different headings

Where Clause : restricts the query to rows that meet a condition

Group by -

Example :

Consider the relation Employee

<u>Eid</u>	Name	Designation	Salary	Phno
125	Kumar	Developer	25000	9489562312
128	Raju	Accountant	27500	9557886321
568	Sachin	HR	33000	8561234568
895	Dravid	Developer	45070	7288635241
222	Dhoni	Accountant	39650	8956235689

Query: Select * from Employee;

Result:

<u>Eid</u>	Name	Designation	Salary	Phno
125	Kumar	Developer	25000	9489562312

128	Raju	Accountant	27500	9557886321
568	Sachin	HR	33000	8561234568
895	Dravid	Developer	45070	7288635241
222	Dhoni	Accountant	39650	8956235689

Query: Select Name, Salary from Employee;

Result:

Name	Salary
Kumar	25000
Raju	27500
Sachin	33000
Dravid	45070
Dhoni	39650

Where Clause

It Restrict the rows returned by select statement.

Comparison Conditions:

= - Equal to

> - Greater than

>= - Greater than or equal to

< - Less than

<= - Less than or equal to

<> - Not equal to

BETWEEN - Between two values (inclusive)

IN(set) - Match any of a list of values (%)

LIKE - Match a character pattern

IS NULL - Is a null value

Logical Conditions:

AND - if all the conditions separated by AND is TRUE

OR - if any of the conditions separated by OR is TRUE

NOT - if the condition(s) is NOT TRUE

Rules of Precedence:

Arithmetic operators

Concatenation operator

Comparison conditions

IS [NOT] NULL, LIKE, [NOT] IN

NOT] BETWEEN

NOT logical condition

AND logical condition

OR logical condition

Query: Select Name, Salary from employee where Salary between 27500 and 40000

Result:

Name	Salary
Raju	27500
Sachin	33000
Dhoni	39650

Query: Select Name, Designation, Salary from Employee where Salary>25000 and Designation='Developer';

Result:

Name	Designation	Salary
Dravid	Developer	45070

Query: Select Eid, Name, Designation from Employee where Name like 'D%'

<u>Eid</u>	Name	Designation
895	Dravid	Developer
222	Dhoni	Accountant

Aggregate Functions

Aggregate functions are functions that take a collection of values as input and return a single value. SQL offers five built-in aggregate functions:

- **avg**- returns the average value of a numeric column.
- **min**- returns the smallest value of the selected column.
- **max**- returns the largest value of the selected column.
- **sum**- returns the total sum of a numeric column.
- **count**- returns the number of rows that matches a specified criteria.

Example:

Query : Select min(Salary) from Employee;

Result:

Min(Salary)
25000

Query: Select avg(Salary) from Employee where designation='Accountant';

Result: shows the average salary of employees with the designation Accountant

Avg(Salary)
33575

Group By Clause:

the GROUP BY clause to divide the rows in a table into groups. The attribute or attributes given in the **group by** clause are used to form groups. By default, rows are sorted by ascending order of the columns included in the GROUP BY list.

Query: select Designation, avg(Salary) from Employee group by Designation;

Result:

Designation	Avg(Salary)
Accountant	33575.0
Developer	36250
HR	33000

Having Clause:

The HAVING clause is used to restrict **groups**. The groups that match the criteria in the HAVING clause are displayed.

Query:

Select Designation, avg(Salary) as avg from Employee group by Designation having avg(Salary)>34000;

Result:

Designation	Avg
Developer	36250

INSERT:

It is used to insert a new row that contains values for each column. **Only one row is inserted at a time.**

Syntax:

INSERT INTO *table* (column1, coulmn2, column3,) VALUES (*value1*, *value2*, *value3*, ...);.

If the order of the values is in the same order as the columns in the table

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

Example

Query:

Insert into Employee(Eid, Designation, Name, Salary, Phno)values(587,'Designer','Walter',32570, '8978567780')

Result:

1 row created.

<u>Eid</u>	Name	Designation	Salary	Phno
125	Kumar	Developer	25000	9489562312
128	Raju	Accountant	27500	9557886321

568	Sachin	HR	33000	8561234568
895	Dravid	Developer	45070	7288635241
222	Dhoni	Accountant	39650	8956235689
587	Walter	Designer	32570	8978567780

UPDATE :

The UPDATE command is used to modify attribute values of one or more selected tuples.

Syntax:

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

Example:

Query:

Update Employee set salary=43575 where Eid=587;

Result:

1 row updated.

<u>Eid</u>	Name	Designation	Salary	Phno
125	Kumar	Developer	25000	9489562312
128	Raju	Accountant	27500	9557886321
568	Sachin	HR	33000	8561234568
895	Dravid	Developer	45070	7288635241
222	Dhoni	Accountant	39650	8956235689
587	Walter	Designer	43575	8978567780

DELETE:

The DELETE statement is used to delete existing records in a table. The delete operation is not made permanent until the data transaction is committed. If the WHERE clause is omitted, all rows in the table are deleted.

Syntax:

DELETE FROM table_name WHERE condition;

Example:

Query:

Delete from Employee where EID=587

Result:

1 row deleted.

<u>Eid</u>	Name	Designation	Salary	Phno
125	Kumar	Developer	25000	9489562312
128	Raju	Accountant	27500	9557886321
568	Sachin	HR	33000	8561234568
895	Dravid	Developer	45070	7288635241
222	Dhoni	Accountant	39650	8956235689

Query:

Delete from Employee;

Result:

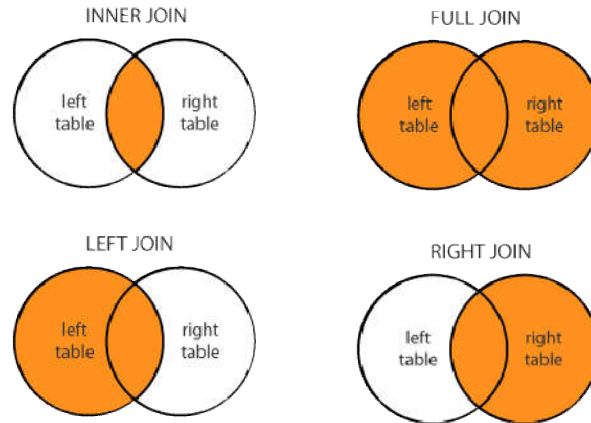
5 row deleted.

Joins in SQL

The SQL Joins clause is used to combine records from two or more tables in a database. A join is a means for combining fields from two tables by using values common to each.

There are different types of joins available in SQL –

- Inner Join – returns rows when there is a match in both tables.
- Left Join – returns all rows from the left table, even if there are no matches in the right table.
- Right Join – returns all rows from the right table, even if there are no matches in the left table.
- Full Join – returns rows when there is a match in one of the tables.
- Self Join – is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.



Example:

Relation: Customer

customer_id	first_name	last_name	Email	address	city	state	zipcode
1	George	Washington	gwashington@usa.gov	3200 Mt Vernon Hwy	Mount Vernon	VA	22121
2	John	Adams	jadams@usa.gov	1250 Hancock St	Quincy	MA	02169
3	Thomas	Jefferson	tjefferson@usa.gov	931 Thomas Jefferson Pkwy	Charlottesville	VA	22902
4	James	Madison	jmadison@usa.gov	11350 Constitution Hwy	Orange	VA	22960
5	James	Monroe	jmonroe@usa.gov	2050 James Monroe Parkway	Charlottesville	VA	22902

Relation: Orders

order_id	order_date	amount	customer_id
1	07/04/1776	\$234.56	1
2	03/14/1760	\$78.50	3
3	05/23/1784	\$124.00	2
4	09/03/1790	\$65.50	3
5	07/21/1795	\$25.50	10
6	11/27/1787	\$14.40	9

Query : (Inner Join)

select first_name, last_name, order_date, order_amount from customers c inner join orders o

on c.customer_id = o.customer_id;

Result:

first_name	last_name	order_date	order_amount
George	Washington	07/4/1776	\$234.56
John	Adams	05/23/1784	\$124.00
Thomas	Jefferson	03/14/1760	\$78.50
Thomas	Jefferson	09/03/1790	\$65.50

Query : (Left Join)

select first_name, last_name, order_date, order_amount from customers c left join orders o
on c.customer_id = o.customer_id;

Result:

first_name	last_name	order_date	order_amount
George	Washington	07/04/1776	\$234.56
John	Adams	05/23/1784	\$124.00
Thomas	Jefferson	03/14/1760	\$78.50
Thomas	Jefferson	09/03/1790	\$65.50
James	Madison	NULL	NULL
James	Monroe	NULL	NULL

Query: (Right Join)

select first_name, last_name, order_date, order_amount from customers c right join orders o
on c.customer_id = o.customer_id

Result:

first_name	last_name	order_date	order_amount
George	Washington	07/04/1776	\$234.56
Thomas	Jefferson	03/14/1760	\$78.50
John	Adams	05/23/1784	\$124.00
Thomas	Jefferson	09/03/1790	\$65.50

NULL	NULL	07/21/1795	\$25.50
NULL	NULL	11/27/1787	\$14.40

Query: (Full Join)

select first_name, last_name, order_date, order_amount from customers c full join orders o
on c.customer_id = o.customer_id

Result:

first_name	last_name	order_date	order_amount
George	Washington	07/04/1776	\$234.56
Thomas	Jefferson	03/14/1760	\$78.50
John	Adams	05/23/1784	\$124.00
Thomas	Jefferson	09/03/1790	\$65.50
NULL	NULL	07/21/1795	\$25.50
NULL	NULL	11/27/1787	\$14.40
James	Madison	NULL	NULL
James	Monroe	NULL	NULL

1.9.3 Transaction Control Language(TCL)

TCL commands are to keep a check on other commands and their affect on the database. These commands can annul changes made by other commands by rolling back to original state. It can also make changes permanent.

TCL Statements are:

- **COMMIT** : to permanently save
- **ROLLBACK** : to undo change
- **SAVEPOINT** : to save temporarily

COMMIT:

It makes the change permanent.

Example:

Query:

Delete from Employee where EID=587

Result: Remove Employee 587 in the Employee table.

1 row deleted.

Query:

Commit;

Result: makes the change permanent

Commit complete.

ROLLBACK:

Data changes are undone. The previous state of the data is restored.

Example:

Query:

DELETE FROM Employee;

Result :all rows in the employee table is deleted

5 rows deleted.

Query:

ROLLBACK;

Result: deleted rows in the employee table is Roll backed.

SAVEPOINT:

It is used to create a marker in the current transaction which divides the transaction into smaller sections. Roll back to that marker by using the ROLLBACK TO SAVEPOINT statement.

Query:

Delete from Employee where EID=587

Result: Remove Employee 587 in the Employee table.

1 row deleted.

Query:

savepoint A

Result:

Savepoint created.

Query:

Delete from Employee where EID=222

Result: Remove Employee 222 in the Employee table.

1 row deleted.

Query:

ROLLBACK TO A;

Result:

Rollback complete.

<u>Eid</u>	Name	Designation	Salary	Phno
125	Kumar	Developer	25000	9489562312
128	Raju	Accountant	27500	9557886321
568	Sachin	HR	33000	8561234568
895	Dravid	Developer	45070	7288635241
587	Walter	Designer	43575	8978567780

1.9.4 Data Control Language (DCL)

It is used to control database access. It gives access to specific objects in the database. Roles, permissions, and referential integrity are created using data control language. Users can be given the privileges to access database. Two types of privileges are there.

System Privilege :

More than 100 distinct system privileges are available for users and roles. System privileges typically are provided by the database administrator.

Example:

```
CREATE SESSION
CREATE TABLE
CREATE SEQUENCE
CREATE VIEW
CREATE PROCEDURE
```

Object Privilege:

An *object privilege* is a privilege or right to perform a particular action on a specific table, view, sequence, or procedure.

Example:

```
ALTER
```

DELETE
EXECUTE
INDEX
INSERT
REFERENCES
SELECT
UPDATE

DCL statements:

GRANT
REVOKE

Create user:

The DBA creates users by using the CREATE USER statement.

Syntax:

CREATE USER *user_name* IDENTIFIED BY *password*;

Example:

Query:

CREATE USER Student IDENTIFIED BY welcome;

Result:

User created.

Where, user name is Student and the password is welcome.

Role:

A role is a named group of related privileges that can be granted to the user. This method makes it easier to revoke and maintain privileges. The DBA must create the role. Then the DBA can assign privileges to the role and users to the role.

Create role:

Syntax:

CREATE ROLE *role*;

Example:

CREATE ROLE Manager;

Role created.

GRANT:

Database Administrator (DBA) can grant specific privileges to a user or role. DBA can create user and role. Then the privilege is granted.

Grant System Privilege to user:

Syntax:

GRANT *privilege* [, *privilege*...] TO *user* [, *user* | *role*, *PUBLIC*...];

Example:

Query:

GRANT create session, create table, create sequence, create view TO Student;

Result:

Grant succeeded.

For the student user, privileges (create session, create table, create sequence, create view) are granted.

Grant privileges to a role:

Syntax:

GRANT privilege(s) TO role_name;

Example:

GRANT create table, create view TO manager;

Grant succeeded.

Grant a role to users:

Syntax:

GRANT role_name TO user_name;

Example:

GRANT manager TO Student;

Result:

Grant succeeded.

create table, create view privileges are granted to the user student.

Grant Object privilege to user:

Syntax:

GRANT *object_priv* [(*columns*)] ON *object* TO {*user*|*role*|PUBLIC} [WITH GRANT OPTION];

Example:

GRANT select ON Employee TO Student;

Grant succeeded.

REVOKE:

The REVOKE statement is used to revoke privileges granted to other users.

Syntax:

REVOKE {*privilege* [, *privilege*...]|ALL} ON *object* FROM {*user*[, *user*...]|*role*|PUBLIC} [CASCADE CONSTRAINTS];

Example:

REVOKE select ON Employee FROM Student;

Revoke succeeded.

1.9 Advanced SQL features

1.9.1 Dynamic SQL:

A general-purpose program can connect to and communicate with a database server using a collection of functions (for procedural languages) or methods (for object-oriented languages). Dynamic SQL allows the program to construct an SQL query as a character string at runtime, submit the query, and then retrieve the result into program variables a tuple at a time. The dynamic SQL component of SQL allows programs to construct and submit SQL queries at runtime.

There are two standards for connecting to an SQL database and performing queries and updates. i)JDBC ii)ODBC

i) JDBC (Java Database Connectivity)

The JDBC standard defines an application program interface (API) that Java programs can use to connect to database servers.

Steps:

- i. Connecting to the Database
- ii. Shipping SQL Statements to the Database System
- iii. Retrieving the Result of a Query

i. Connecting to the Database

The first step in accessing a database from a Java program is to open a connection to the database. This step is required to select which database to use. After opening a connection, Java program execute SQL statements.

A connection is opened using the `getConnection` method of the `Driver-Manager` class (within `java.sql`). This method takes three parameters.

The first parameter to the `getConnection` call is a string that specifies the URL, or machine name, where the server runs (in our example, `db.yale.edu`), along with possibly some other information such as the protocol to be used to communicate with the database the port number the database system uses for communication and the specific database on the server to be used. JDBC specifies only the API, not the communication protocol. A JDBC driver may support multiple protocols, and we must specify one supported by both the database and the driver. The protocol details are vendor specific.

The second parameter to `getConnection` is a database user identifier, which is a string.

The third parameter is a password, which is also a string.

ii. Shipping SQL Statements to the Database System

Once a database connection is open, the program can use it to send SQL statements to the database system for execution. This is done via an instance of the class `Statement`. A `Statement` object is not the SQL statement itself, but rather an object that allows the Java program to invoke methods that ship an SQL statement given as an argument for execution by the database system.

To execute a statement, we invoke either the `executeQuery` method or the `executeUpdate` method, depending on whether the SQL statement is a query (and, thus, returns a result set) or nonquery statement such as update, insert, delete, create table, etc.

iii. Retrieving the Result of a Query

It retrieves the set of tuples in the result into a `ResultSet` object `rset` and fetches them one tuple at a time. The next method on the result set tests whether or not there remains at least one unfetched tuple in the result set and if so, fetches it. The method `getString` can retrieve any of the basic SQL data types. The statement and connection are both closed at the end of the Java program.

ii) ODBC (Open Database Connectivity)

The **Open Database Connectivity** (ODBC) standard defines an API that applications can use to open a connection with a database, send queries and updates, and get back results. Applications such as graphical user interfaces, statistics packages, and spreadsheets can make use of the same ODBC API to connect to any database server that supports ODBC.

Each database system supporting ODBC provides a library that must be linked with the client program. When the client program makes an ODBC API call, the code in the library communicates with the server to carry out the requested action and fetch results.

Steps:

The first step in using ODBC to communicate with a server is to set up a connection with the server. The program then opens the database connection by using `SQLConnect`. This call takes several parameters, including the connection handle, the server to which to connect, the user identifier, and the password for the database.

Once the connection is set up, the program can send SQL commands to the database by using `SQLExecDirect`.

A result tuple is fetched using `SQLFetch`.

At the end of the session, the program frees the statement handle, disconnects from the database, and frees up the connection and SQL environment handles.

1.10.2 Embedded SQL

Embedded SQL provides a means by which a program can interact with a database server. However, under embedded SQL, the SQL statements are identified at compile time using a preprocessor.

The preprocessor submits the SQL statements to the database system for precompilation and optimization; then it replaces the SQL statements in the application program with appropriate code and function calls before invoking the programming-language compiler.

The SQL standard defines embeddings of SQL in a variety of programming languages, such as C, C++, Cobol, Pascal, Java, PL/I, and Fortran. A language in which SQL queries are embedded is referred to as a host language, and the SQL structures permitted in the host language constitute embedded SQL.

Steps:

- To identify embedded SQL requests to the preprocessor, we use the EXEC SQL statement. it has the form: EXEC SQL <embedded SQL statement >;
- Before executing any SQL statements, the program must first connect to the database. EXEC SQL **connect to** server **user** user-name **using** password;
- To write a relational query, we use the **declare cursor** statement.

Example:

Consider the university schema. Assume that we have a host-language variable *credit amount* in our program, declared as we saw earlier, and that we wish to find the names of all students who have taken more than *credit amount* credit hours. We can write this query as follows:

```
EXEC SQL
declare c cursor for
select ID, name
from student
where tot cred > :credit amount;
```

- The program must use the **open** and **fetch** commands to obtain the result tuples.

EXEC SQL **open** *c*; - This statement causes the database system to execute the query and to save the results within a temporary relation.

- The **fetch** statement requires one host-language variable for each attribute of the result relation. A single **fetch** request returns only one tuple.
- The **close** statement to tell the database system to delete the temporary relation that held the result of the query. EXEC SQL **close** *c*;

EXERCISE

1. Consider the relation

Employee

<u>Eid</u>	Name	Designation	Salary	Phno
125	Kumar	Developer	25000	9489562312
128	Raju	Accountant	27500	9557886321
568	Sachin	HR	33000	8561234568
895	Dravid	Developer	45070	7288635241
222	Dhoni	Accountant	39650	8956235689
258	Priya	Developer	28500	8345696952
345	Rohit	HR	23575	9884564562

Project

Pid	Empid	Plocation
P123	125	Coimbatore
P225	568	Bangalore
P556	222	Delhi
P125	895	Coimbatore
P255	258	Chennai
P010	222	Ranchi
P585	258	Chennai

Give an expression in the relational algebra to express each of the following queries

- Find the names of all employees who work in the designation Developer
- Find the names of all employees whose salary is greater than 30000
- Find the names of all employees whose salary is greater than 30000 and works for Accounts department
- Find the names of the developer who works in Coimbatore
- Find the names of all employees who work in Chennai having salary greater than 30000

2. Consider the relations

Employee

<u>Eid</u>	Name	Designation	Salary	Phno
125	Kumar	Developer	25000	9489562312
128	Raju	Accountant	27500	9557886321
568	Sachin	HR	33000	8561234568
895	Dravid	Developer	45070	7288635241
222	Dhoni	Accountant	39650	8956235689
258	Priya	Developer	28500	8345696952

345	Rohit	HR	23575	9884564562
-----	-------	----	-------	------------

Works

Eid	Company Name
128	Infosys
895	Microsoft Corporation
345	Scava Systems
568	IBM Corporation
222	Microsoft Corporation
258	IBM Corporation

Company

Company Name	Location	Phone No
Infosys	Mysore	022 2563897
IBM Corporation	Bengaluru	04258978592
Microsoft Corporation	Delhi	012 2698745
Scava Systems	Coimbatore	0422 2536356

Give an expression in SQL for each of the following queries

- i. Find all employees in the database who do not work for “Wipro”
- ii. Find the highest salary of employee
- iii. Find the company that has the most employees
- iv. Find all employees in the database who earn more than each employee of ”Scava Systems”
- v. Give all employees of “IBM Corporation” a 10 percent raise in salary.
- vi. Find all employees who earn more than the average salary of all employees
- vii. Find all employees who earn more than the average salary of employees working for Infosys

3. Consider the following relational database

Employee(Employee-Name, Street, City)

Works(Employee-Name, Company-Name, Salary)

Company(Company-Name, City)

Manager(Employee-Name, Manager-Name)

Give an SQL DDL definitions of this database. Identify referential integrity constraints that should hold and include them in DDL Definitions

4. Consider the relation Employee(empno, name, age, salary). Write a embedded SQL statement in C language to retrieve all the employee record whose salary is between 10,000 and 20,000.

5. A relational schema for a train reservation database is given below

Passenger(pid, pname, age)

Reservation(pid,cass,tid)

Table : Passenger

pid	'pname	Age
0	'Sachin'	65
1	'Rahul'	66
2	'Sourav'	67
3	'Anil'	69

Table : Re s ervation

pid	class	tid
0	'AC'	8200
1	'AC'	8201
2	'SC'	8201
5	'AC'	8203
1	'SC'	8204
3	'AC'	8202

What pids are returned by the following SQL query for the above instance of the tables?

SELECT pid FROM Reservation WHERE class= 'AC' AND EXISTS (SELECT * FROM Passenger WHERE age > 65 AND Passenger.pid = Reservation.pid)

6. Consider the following relations:

Student

Roll_No	Student Name
1	Raj
2	Rohit
3	Raj

Performance

Roll_No	Course	Marks
1	Math	80
1	English	70
2	Math	75
3	English	80
2	Physics	65
3	Math	80

Consider the following SQL query.

SELECT S. Student_Name, sum (P.Marks) FROM Student S, Performance P
WHERE S. Roll_No =P.Roll_No GROUP BY S.Student_Name

The number of rows that will be returned by the SQL query is _____

7. Consider a database that has the relation schema CR (StudentName, CourseName). An instance of the schema CR is as given below.

Student Name	Course Name
SA	CA
SA	CB
SA	CC

SB	CB
SB	CC
SC	CA
SC	CB
SC	CC
SD	CA
SD	CB
SD	CC
SD	CD
SE	CD
SE	CA
SE	CB
SF	CA
SF	CB
SF	CC

What is the output of following query that is made on the database.

$$T1 \leftarrow \pi_{\text{CourseName}} (\sigma_{\text{StudentName}='SA'} (CR))$$

$$T2 \leftarrow CR \div T1$$

8. Consider tables students and sports below:

students	
roll	sports name
1	Hockey
3	Tennis
3	Hockey
4	Hockey

sports
sports name
Hockey
Tennis

What does (students \div sports) return?

9. Consider the relational database given below:

^ employee(person name, street, city)

^ works(person name, company name, salary)

^ company(company name, city)

Give the relational algebraic expression to find the names and cities of residence of all employees who work for 'UCO'.

POSSIBLE QUESTIONS

PART-A

1. Differentiate File processing system with Database Management system.
2. What is the purpose of database management system?
3. Define instance and schema?
4. Define physical data independence
5. Give the difference between physical and logical data independence.
6. State the levels of abstraction in a DBMS.
7. What is the role of database administrator?
8. What is data dictionary?
9. What is a data model? List the types of data model used
10. What are the problems caused by redundancy
11. What are primary key constraints?
12. What is referential integrity?
13. Define a super key
14. List the basic relational algebra operations
15. What is the use of truncate and drop command in SQL?
16. What are the categories of SQL command?
17. What is data definition language? Give example.
18. Differentiate between static and dynamic SQL.
19. Define DML, DDL, TCL and DCL
20. Distinguish between key and super key
21. Why does SQL allow duplicate tuples in a table or in a query result?
22. Differentiate between Cartesian product and natural join operations used in relational algebra.
23. List few applications of DBMS.

PART-B

1. Explain the overall architecture of the data base system in detail.
2. List the operations of relational algebra and the purpose of each with example
3. Justify the need of embedded SQL. Consider the relation student (Reg No, name, mark, and grade). Write embedded dynamic SQL Program in C language to retrieve all the student's records whose mark is more than 90

4. Explain the aggregate functions in SQL with an example
5. Consider a student registration database comprising of the given table schema

Student File (Student Number, Student Name, Address, Telephone)

Course File(Course Number, Description, Hours, Professor Number)

Professor File(Professor Number, Name, Office)

Registration File(Student Number, Course Number, Date)

Consider a suitable sample of records for the above tables and write SQL statements to answer the queries listed below:

- i. Which course does a specific professor teach?
 - ii. What courses are taught by two specific professor?
 - iii. Who teaches a specific course and where is his office?
 - iv. For a specific student number, in which course is the student registered and what is his/her name?
 - v. Who are the professors for a specific student?
 - vi. Who are students registered in a specific course?
6. Write notes on:
 - i) Embedded SQL
 - ii) Data Models
7.
 - i) Describe about DDL and DML commands in SQL.
 - ii) Consider the relation employee (empid, empname, desg, mgr, hiredate, salary, deptno, age).
 - a) Write SQL query to List all employee names and their and their manager whose manager is 7902 or 7566 or 7789.
 - b) Write SQL query to Display all employee names and salary whose salary is greater than average salary of the company and job title starts with 'M'.