

1

Relational Databases

Syllabus

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

Contents

1.1 Introduction to Database Management	
1.2 Purpose of Database System	May-07, 12, Dec.-04,..... Marks 8
1.3 Views of Data	May-16..... Marks 16
1.4 Data Models	Dec.-14,..... Marks 8
Database System Architecture	May-12, 13, 14, 16, 17,
.....	Dec.-08, 15, 17,..... Marks 16
Data Independence	
Introduction to Relational Databases	
Relational Model	
1.9 Keys	May-06, 07, 12, Dec.-06, Marks 4
1.10 Integrity Constraints	Dec.-05,..... Marks 10
1.11 Database integrity	
1.12 Relational Algebra.....	May-03,04,05,14,15,16,17,18,
.....	Dec.-02,07,08,11,15,16,17 . Marks 16
1.13 SQL Fundamentals	Dec.-15,..... Marks 16
1.14 Advanced SQL Features	
1.15 Dynamic SQL.....	May-17, Dec.-17,..... Marks 11
1.16 Two Marks Questions with Answers	

Part I : Introduction of DBMS**1.1 Introduction to Database Management**

- **Definition** : A Database Management System (DBMS) is a collection of **interrelated data** and various **programs** that are used to handle that data.
- The **primary goal** of DBMS is to provide a way to **store and retrieve** the required information from the database in convenient and efficient manner.
- For managing the data in the database two important **tasks** are conducted -
 - (i) **Define the structure** for storage of information.
 - (ii) Provide mechanism for **manipulation of information**.
- In addition, the database systems must ensure the **safety of information** stored.

Database System Applications

There are wide range of applications that make use of database systems. Some of the applications are -

- 1) **Accounting** : Database systems are used in maintaining information employees, salaries, and payroll taxes.
- 2) **Manufacturing** : For management of supply chain and tracking production of items in factories database systems are maintained.
- 3) For maintaining customer, product and purchase information the databases are used.
- 4) **Banking** : In banking sector, for customer information, accounts and loan and for performing banking applications the DBMS is used.
- 5) For purchase on credit cards and generation of monthly statements database systems are useful.
- 6) **Universities** : The database systems are used in universities for maintaining student information, course registration, and accounting.
- 7) **Reservation systems** : In airline/railway reservation systems, the database is used to maintain the reservation and schedule information.
- 8) **Telecommunication** : In telecommunications for keeping records of the calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about communication networks the database systems are used.

1.2 Purpose of Database System**AU : May-07, 12, Dec.-04**

- Earlier database systems are created in response to manage the commercial data. These data is typically stored in files. To allow users to manipulate these files various programs are written for

- 1) Addition of new data
 - 2) Updating the data
 - 3) Deleting the data.
- As per the addition of new need, separate application programs were required to write. Thus as the time goes by, the system acquires more files and more application programs.
 - This typical **file processing system** is supported by conventional operating system. Thus the file processing system can be described as –
 - The system that stores the permanent records in files and it needs different application programs to extract or add the records.
 - Before introducing database management system, this file processing system was in use. However, such a system has many drawbacks. Let us discuss them –

Disadvantages of Traditional File Processing System

The **traditional file system** has following **disadvantages** :

- 1) **Data redundancy** : Data redundancy means duplication of data at several places. Since different programmers create different files and these files might have different structures, there are chances that some information may appear repeatedly in some or more format at several places.
- 2) **Data inconsistency** : Data inconsistency occurs when various copies of same data may no longer get matched. For example changed address of an employee may be reflected in one department and may not be available (or old address present) for other department.
- 3) **Difficulty in accessing data** : The conventional file system does not allow to retrieve the desired data in efficient and convenient manner.
- 4) **Data isolation** : As the data is scattered over several files and files may be in different formats, it becomes to retrieve the desired data from the file for writing the new application.
- 5) **Integrity problems** : Data integrity means data values entered in the database fall within a specified range and are of correct format. With the use of several files enforcing such constraint on the data becomes difficult.
- 6) **Atomicity problems** : An atomicity means particular operation must be carried out entirely or not at all with the database. It is difficult to ensure atomicity in conventional file processing system.
- 7) **Concurrent access anomalies** : For efficient execution, multiple users update data simultaneously, in such a case data need to be synchronized. As in traditional file systems, data is distributed over multiple files, one cannot access these files concurrently.

- 8) **Security problems** : Every user is not allowed to access all the data of database system. Since application program in file system are added in an ad hoc manner, enforcing such security constraints become difficult.

Database systems offer solutions to all the above mentioned problems.

Difference between Database System and Conventional File System

Sr. No.	Database systems	Conventional file systems
1.	Data redundancy is less .	Data redundancy is more .
2.	Security is high .	Security is very low .
3.	Database systems are used when security constraints are high .	Conventional file systems are used where there is less demand for security constraints .
4.	Database systems define the data in a structured manner. Also there is well defined co-relation among the data.	File systems define the data in un-structured manner. Data is usually in isolated form.
5.	Data inconsistency is less in database systems.	Data inconsistency is more in file systems.
6.	User is unknown to the physical address of the data used in database systems.	User locates the physical address of file to access the data in conventional file systems.
7.	We can retrieve the data in any desired format using database systems.	We cannot retrieve the data in any desired format using file systems.
8.	There is ability to access the data concurrently using database systems.	There is no ability to concurrently access the data using conventional file system.

Characteristics of Database Systems

Following are the characteristics of database system -

- 1) Representation of some aspects of real world applications.
- 2) Systematic management of information.
- 3) Representing the data by multiple views.
- 4) Efficient and easy implementation of various operations such as insertion, deletion and updation.
- 5) It maintains data for some specific purpose.
- 6) It represents logical relationship between records and data.

Advantages of Database Systems

Following are the advantages of DBMS -

- 1) DBMS **removes the data redundancy** that means there is no duplication of data in database.
- 2) DBMS allows to **retrieve the desired data** in required format.

- 3) **Data** can be **isolated** in separate tables for convenient and efficient use.
- 4) Data can be **accessed efficiently** using a simple query language.
- 5) The **data integrity** can be maintained. That means – the constraints can be applied on data and it should be in some specific range.
- 6) The **atomicity** of data can be maintained. That means, if some operation is performed on one particular table of the database, then the change must be reflected for the entire database.
- 7) The DBMS allows **concurrent access** to multiple users by using the synchronization technique.
- 8) The **security policies** can be applied to DBMS to allow the user to access only desired part of the database system.

Disadvantages of Database Systems

- 1) **Complex design** : Database design is complex, **difficult and time consuming**.
- 2) **Hardware and software cost** : **Large amount of investment** is needed to setup the required hardware or to repair software failure.
- 3) **Damaged part** : If one part of database is corrupted or damaged, then **entire database** may get affected.
- 4) **Conversion cost** : If the current system is in conventional file system and if we need to convert it to database systems then large amount of cost is incurred in purchasing different tools, and adopting different techniques as per the requirement.
- 5) **Training** : For designing and maintaining the database systems, the people need to be trained

University Questions

1. Compare file system with database system

AU : May-07, Marks 8, May-12, Marks 2

2. What are the advantages and disadvantages of DBMS ?

AU : Dec.-04, Marks 4

1.3 Views of Data

AU : May-16

- Database is a collection of interrelated data and set of programs that allow users to access or modify the data.
- **Abstract view** of the system is a view in which the system hides certain details of how the data are stored and maintained.
- The **main purpose** of database systems is to provide users with abstract view of the data.
- The view of the system **helps the user to retrieve data efficiently**.
- For simplifying the user interaction with the system there are several levels of abstraction - these levels are - Physical level, logical level and view level.

1.3.1 Data Abstraction

Data abstraction : Data abstraction means retrieving only required amount of information of the system and hiding background details.

There are several levels of abstraction that simplify the user interactions with the system. These are

i) Physical level :

- This is the lowest level.
- This level describes how actually the data are stored.
- This level describes complex low level data structures.

2) Logical level :

- This is the next higher level, which describes the what data are stored in database.
- This level also describes the relationship among the data.
- The logical level thus describes then entire database in terms of small number of relatively simple structures.

The database administrators use logical level of abstraction for deciding what information to keep in database.

3) View level :

- This is highest level of abstraction that describes only part of the entire database.
- The view level can provide the access to only part of the database.
- This level helps in simplifying the interaction with the system.
- The system can provide **multiple views** of the same system.
- Clerk at the reservation system, can see only part of the database and can access the required information of the passenger.

Fig. 1.3.1 shows the relationship among the three levels of abstraction.

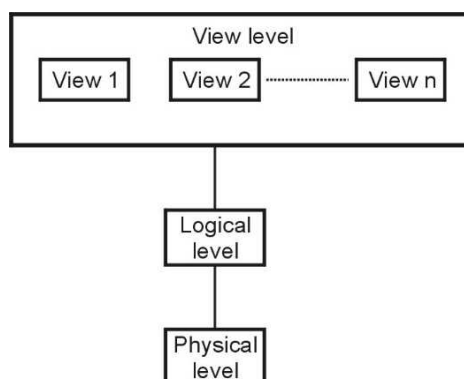


Fig. 1.3.1 : Levels of data abstraction

For example : Consider following record

```

type employee = record
    empID:numeric(10)
    empname:char(20)
    dept_no:numeric(10)
    salary:numeric(8,2)
end
  
```

This code defines a new record **employee** with four fields. Each field is associated with field name and its **type**. There are several other records such as

department with fields dept_no, dept_name, building

customer with fields cust_id, cust_name

- **At the physical level**, the record - **customer, employee, department** can be described as block of **consecutive storage locations**. Many database systems hide lowest level storage details from database programmer.
- The type definition of the records is decided **at the logical level**. The **programmer** work of the record at this level, similarly **database administrators** also work at this level of abstraction.
- There is specific view of the record is allowed **at the view level**. For instance - -customer can view the name of the employee, or id of the employee but cannot access employee's salary.

1.3.2 Instances and Schemas

Schema : The overall design of the database is called schema

For example - In a program we do variable declaration and assignment of values to the variable. The variable declaration is called schema and the value assigned to the variable is called instance. The schema for the student record can be

RollNo	Name	Marks
--------	------	-------

Instances : When information is inserted or deleted from the database then the database gets changed. The collection of information at particular moment is called **instances**. For example - following is an instance of **student** database

RollNo	Name	Marks
10	AAA	43
20	BBB	67

Types of Schema : The database has several schema based on the levels of abstraction.

- (1) **Physical Schema :** The physical schema is a database design described at the physical level of abstraction.
- (2) **Logical Schema :** The logical schema is a database design at the logical level of abstraction.
- (3) **Subschema :** A database may have several views at the view level which are called subschemas.

1.3.3 Database Languages

There are two types of languages supported by database systems. These are -

(1) DDL -

- Data Definition Language (DDL) is a specialized language used to specify a database schema by a set of definitions.
- It is a language which is used for **creating** and modifying the structures of tables, views, indexes and so on.
- DDL is also used to specify additional properties of data.
- Some of the common commands used in DDL are **-CREATE, ALTER, DROP**.
- The main use of CREATE command is to build a new table. Using ALTER command, the users can add up some additional column and drop existing columns. Using DROP command, the user can delete table or view.

(2) DML

- DML stands for Data Manipulation Language.
- This language enables users to access or manipulate data as organized by appropriate data model.
- The types of access are -
 - **Retrieval** of information stored in the database
 - **Insertion** of new information into the database.
 - **Deletion** of information from the database.
 - **Modification** of information stored in database.
- There are two types of DML -
 - **Procedural DML** - Require a user to specify what data are needed and how to get those data.
 - **Declarative DML** - Require a user to specify what data are needed without specifying how to get those data.

- **Query** is a statement used for requesting the retrieval of information. This retrieval of information using some specific language is called **query language**.

University Question

1. Briefly explain about views of data.

AU : May-16, Marks 16

1.4 Data Models

AU : Dec.-14

- **Definition :** It is a collection of conceptual tools for describing data, relationships among data, semantics (meaning) of data and constraints.
- Data model is a **structure below the database**.
- Data model provides a way to describe the design of database at physical, logical and view level.
- There are various data models used in database systems and these are as follows -

(1) Relational model :

- Relation model consists of collection of tables which stores data and also represents the relationship among the data.
- Table is also known as **relation**.
- The table contains one or more **columns** and each column has unique name.
- Each table contains record of particular type, and each record type defines a fixed number of fields or attributes.
- **For example** – Following figure shows the relational model by showing the relationship between Student and Result database. For example – Student Ram lives in city Chennai and his marks are 78. Thus the relationship between these two databases is maintained by the **SeatNo**. Column

SeatNo	Name	City
101	Ram	Chennai
102	Shyam	Pune

SeatNo	Marks
101	78
102	95

Advantages :

- (i) **Structural Independence** : Structural independence is an ability that allows us to make changes in one database structure without affecting other. The relational model have structural independence. Hence making required changes in the database is convenient in relational database model.
- (ii) **Conceptual Simplicity** : The relational model allows the designer to simply focus on logical design and not on physical design. Hence relational models are conceptually simple to understand.

(iii) **Query Capability** : Using simple query language (such as SQL) user can get information from the database or designer can manipulate the database structure.

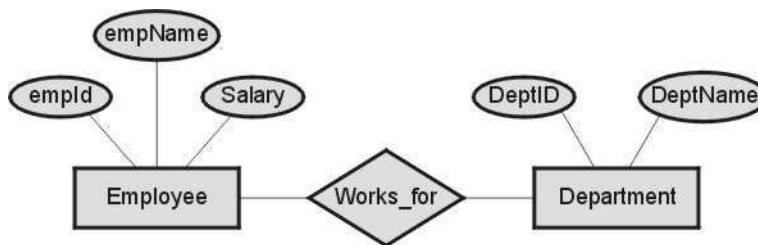
(iv) **Easy design,maintenance and usage** : The relational models can be designed logically hence they are easy to maintain and use.

Disadvantages :

- (i) Relational model requires powerful hardware and large data storage devices.
- (ii) May lead to slower processing time.
- (iii) Poorly designed systems lead to poor implementation of database systems.

(2) Entity relationship model :

- As the name suggests the entity relationship model uses collection of basic objects called **entities** and **relationships**.
- The entity is a thing or object in the real world.
- The entity relationship model is widely used in database design.
- For example - Following is a representation of Entity Relationship model in which the relationship **works_for** is between entities **Employee** and **Department**.



Advantages :

- i) **Simple** : It is simple to draw ER diagram when we know entities and relationships.
- ii) **Easy to understand** : The design of ER diagram is very logical and hence they are easy to design and understand.
- iii) **Effective**: It is effective communication tool.
- iv) **Integrated** : The ER model can be easily integrated with Relational model.
- v) **Easy conversion**: ER model can be converted easily into other type of models.

Disadvantages :

- i) **Loss of information** : While drawing ER model some information can be hidden or lost.
- ii) **Limited relationships** : The ER model can represent limited relationships as compared to other models.

iii) No Representation for data manipulation : It is not possible to represent data manipulation in ER model.

iv) No industry standard : There is no industry standard for notations of ER diagram.

(3) Object Based Data Model :

- The object oriented languages like C++, Java, C# are becoming the dominant in software development.
- This led to object based data model.
- The object based data model combines object oriented features with relational data model.

Advantages :

- i) Enriched modelling :** The object based data model has capability of modelling the real world objects.
- ii) Reusability :** There are certain features of object oriented design such as inheritance, polymorphism which help in reusability.
- iii) Support for schema evolution :** There is a tight coupling between data and applications, hence there is strong support for schema evolution.
- iv) Improved performance :** Using object based data model there can be significant improvement in performance using object based data model.

Disadvantages :

- i) Lack of universal data model :** There is no universally agreed data model for an object based data model, and most models lack a theoretical foundation.
- ii) Lack of experience :** In comparison with relational database management the use of object based data model is limited. This model is more dependent on the skilled programmer.
- iii) Complex :** More functionalities present in object based data model make the design complex.

(4) Semi-structured data model :

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

Advantages

- i) Data is not constrained by fixed schema.
- ii) It is flexible.
- iii) It is portable.

Disadvantage

- i) Queries are less efficient than other types of data model.

University Question

1. Write short note on : Data model and its types.

AU : Dec.-14, Marks 8

THREE SCHEMA ARCHITECTURE

The three schema architecture is used to visualize the schema levels in a database. The three schemas are only descriptions of data, the data only actually exists is at the physical level.

Three Schema Architecture

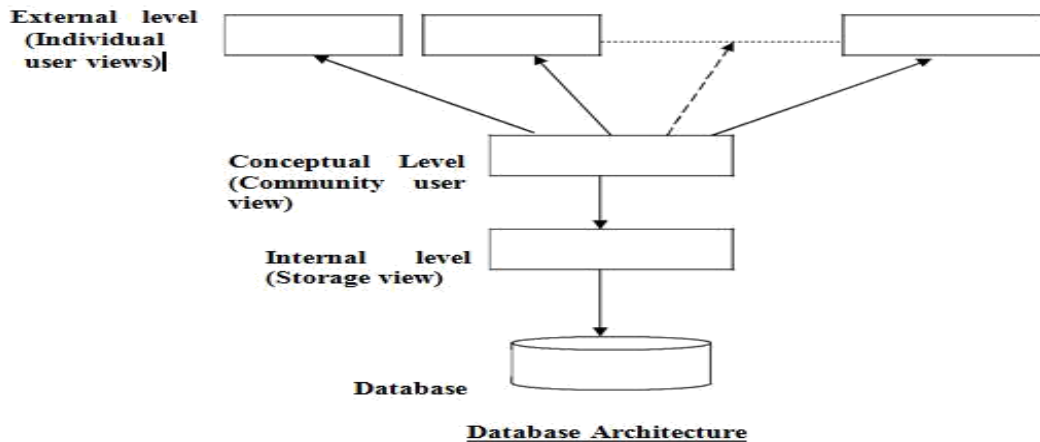
The goal of the three schema architecture is to separate the user applications and the physical database. The schemas can be defined at the following levels:

1.The internal level –has an internal schema which describes the physical storage structure of the database. Uses a physical data model and describes the complete details of data storage and access paths for the database.

2.The conceptual level –has a conceptual schema which describes the structure of the database for users. It hides the details of the physical storage structures, and concentrates on describing entities, data types, relationships, user operations and constraints. Usually a representational data model is used to describe the conceptual schema.

3.The External or View level –includes external schemas or user views. 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. Represented using the representational data model.

The three schema architecture is used to visualize the schema levels in a database. The three schemas are only descriptions of data, the data only actually exists is at the physical level.



1.5 Database System Architecture

AU : May-12, 13, 14, 16, 17, Dec.-08, 15, 17

- The typical structure of typical DBMS is based on relational data model as shown in Fig. 1.5.1. (Refer page 1-14).
- Consider the top part of Fig. 1.5.1. It shows **application interfaces** used by naïve users, application programs created by application programmers, query tools used by sophisticated users and administration tools used by database administrator
- The lowest part of the architecture is for **disk storage**.
- The two important components of database architecture are - **Query processor and storage manager**.

Query processor :

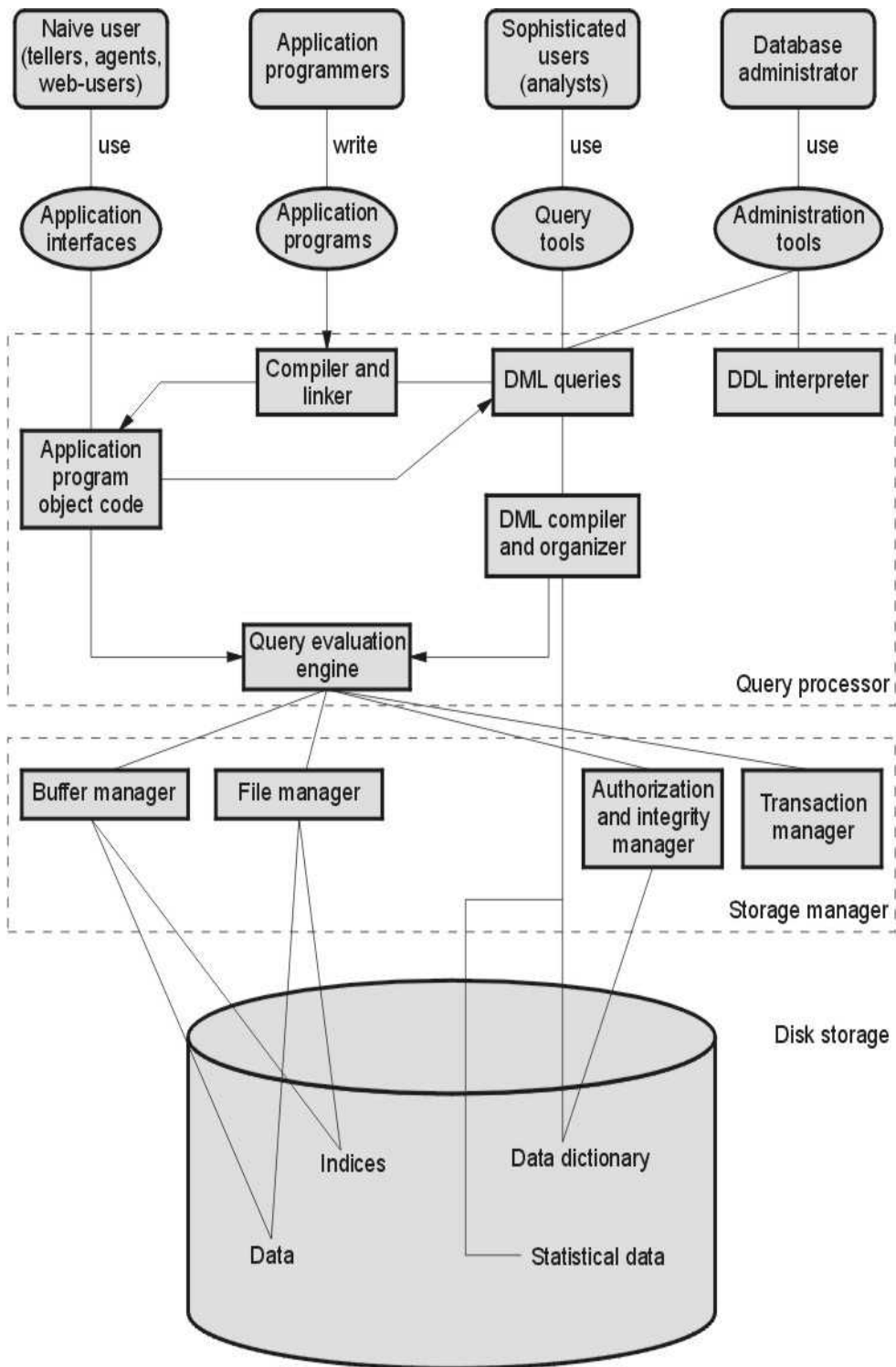
- The interactive query processor helps the database system to simplify and facilitate access to data. It consists of DDL interpreter, DML compiler and query evaluation engine.
- With the following components of query processor, various functionalities are performed -
 - DDL interpreter :** This is basically a translator which interprets the DDL statements in data dictionaries.
 - DML compiler :** It translates DML statements query language into an evaluation plan. This plan consists of the instructions which query evaluation engine understands.

iii) Query evaluation engine : It executes the low-level instructions generated by the DML compiler.

- When a user issues a query, the parsed query is presented to a query optimizer, which uses information about how the data is stored to produce an efficient execution plan for evaluating the query. An execution plan is a blueprint for evaluating a query. It is evaluated by **query evaluation engine**.

Storage manager :

- Storage manager is the component of database system that provides 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 storing, retrieving, and updating data in the database. The storage manager components include -
 - i) Authorization and integrity manager :** Validates the users who want to access the data and tests for integrity constraints.
 - ii) Transaction manager :** Ensures that the database remains in consistent despite of system failures and concurrent transaction execution proceeds without conflicting.
 - iii) File manager :** Manages allocation of space on disk storage and representation of the information on disk.
 - iv) Buffer manager :** Manages the fetching of data from disk storage into main memory. The buffer manager also decides what data to cache in main memory. Buffer manager is a crucial part of database system.
- Storage manager implements several data structures such as -
 - i) Data files :** Used for storing database itself.
 - ii) Data dictionary :** Used for storing metadata, particularly schema of database.
 - iii) Indices :** Indices are used to provide fast access to data items present in the database

**Fig. 1.5.1 Architecture of database**

University Questions

1. Explain the overall architecture of database system in detail.

AU : May-14,17, Dec.-17, Marks 8, May-16, Marks 16

2. With the help of a neat block diagram explain basic architecture of a database management system.

AU : May-12, May 13, Marks 16, Dec.-15, Marks 8

Q. Explain component modules of a DBMS and their interactions with the architecture

AU : Dec 08, Marks 10

1.6 Data Independence

Definition : Data independence is an ability by which one can change the data at one level without affecting the data at another level. Here level can be physical, conceptual or external.

Data independence is one of the important characteristics of database management system.

By this property, the structure of the database or the values stored in the database can be easily modified by without changing the application programs.

There are two types of data independence

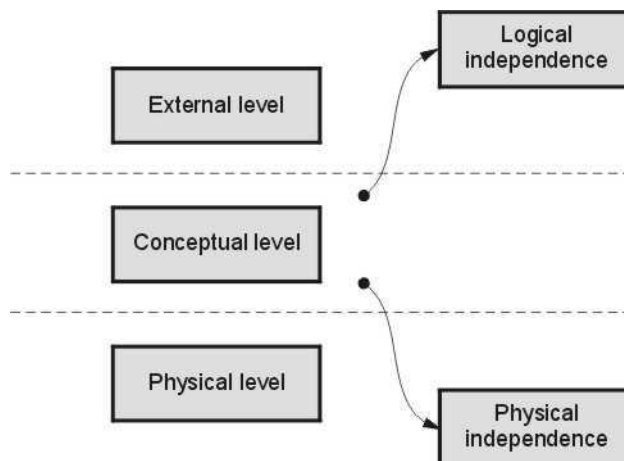


Fig. 1.6.1 Data independence

- 1. Physical Independence :** This is a kind of data independence which allows the modification of physical schema without requiring any change to the conceptual schema. For example - if there is any change in memory size of database server then it will not affect the logical structure of any data object.
- 2. Logical Independence :** This is a kind of data independence which allows the modification of conceptual schema without requiring any change to the external schema. For example - Any change in the table structure such as addition or deletion of some column does not affect user views.

By these data independence the time and cost acquired by changes in any one level can be reduced and abstract view of data can be provided to the user.

Part II Relational Databases

1.7 Introduction to Relational Databases

- Relation database is a **collection of tables** having unique names.
- For example – Consider the example of Student table in which the information about the student is stored.

RollNo	Name	Phone
001	AAA	1111111111
002	BBB	2222222222
003	CCC	3333333333

Fig. 1.7.1 Student table

The above table consists of three column headers RollNo, Name and Phone. Each row of the table indicates the information of each student by means of his Roll Number, Name and Phone number.

Similarly consider another table named Course as follows –

CourseID	CourseName	Credits
101	Mechanical	4
102	Computer Science	6
103	Electrical	5
104	Civil	3

Fig. 1.7.2 Course table

Clearly, in above table the columns are **CourseID**, **CourseName** and **Credits**. The CourseID **101** is associated with the course named **Mechanical** and associated with the course of mechanical there are **4 credit points**. Thus the relation is represented by the table in the relation model. Similarly we can establish the relationship among the two tables by defining the third table. For example – Consider the table Admission as

RollNo	CourseID
001	102
002	104
003	101

Fig. 1.7.3 Admission

From this third table we can easily find out that the course to which the RollNo 001 is admitted is computer Science.

1.8 Relational Model

There are some commonly used terms in Relational Model and those are -

Table or relation : In relational model, table is a collection of data items arranged in rows and columns. The table cannot have duplicate data or rows. Below is an example of student table

Roll No	Name	Marks	Phone
001	AAA	88	1111111111
002	BBB	83	2222222222
003	CCC	98	3333333333
004	DDD	67	4444444444

Tuple or record or row : The single entry in the table is called tuple. The tuple represents a set of related data. In above **Student** table there are four tuples. One of the tuple can be represented as

001	AAA	88	1111111111
-----	-----	----	------------

Attribute or columns : It is a part of table that contains several records. Each record can be broken down into several small parts of data known as attributes. For example the above table consists of four attributes such as **RollNo,Name,Marks** and **Phone**.

Relation schema : A relation schema describes the structure of the relation, with the name of the relation (i.e. name of table), its attributes and their names and type.

Relation Instance : It refers to specific instance of relation i.e. containing a specific set of rows. For example – the following is a relation instance – which contains the records with marks above 80.

RollNo	Name	Marks	Phone
001	AAA	88	1111111111
002	BBB	83	2222222222
003	CCC	98	3333333333

Domain : For each attribute of relation, there is a set of permitted values called domain. For example – in above table, the domain of attribute **Marks** is set of all possible permitted marks of the students. Similarly the domain of **Name** attribute is all possible names of students.

That means Domain of Marks attribute is (88,83,98)

Atomic : The domain is atomic if elements of the domain are considered to be indivisible units. For example in above **Student** table, the attribute **Phone** is non-atomic.

NULL attribute : A null is a special symbol, independent of data type, which means either unknown or inapplicable. It does not mean zero or blank. For example - Consider a salary table that contains NULL

Emp#	Job Name	Salary	Commission
E10	Sales	12500	32090
E11	Null	25000	8000
E12	Sales	44000	0
E13	Sales	44000	Null

Degree : It is nothing but total number of columns present in the relational database. In given Student table –

Roll No	Name	Marks	Phone
001	AAA	88	1111111111
002	BBB	83	2222222222
003	CCC	98	3333333333

The degree is 4.

Cardinality : It is total number of tuples present in the relational database. In above given table the cardinality is 3

Example 1.8.1 Find out following for given Staff table

i) No of Columns

ii) No of tuples

iii) Different attributes

iv) Degree

v) Cardinality

StaffID	Name	Sex	Designation	Salary	DOJ
S001	John	M	Manager	50000	1 Oct. 2012
S002	Ram	M	Executive	20000	20 Jan. 2015
S003	Meena	F	Supervisor	40000	12 Aug. 2011

Solution :

- i) No of Columns = 6
- ii) No of Tuples = 3
- iii) Different attributes are StaffID, Name, Sex, Designation, Salary, DOJ
- iv) Degree = Total number of columns = 6
- v) Cardinality = Total number of rows = 3

1.9 Keys

AU : May-06, 07, 12, Dec.-06

Keys are used to specify the tuples distinctly in the given relation.

Various types of keys used in relational model are – Superkey, Candidate Keys, primary keys, foreign keys. Let us discuss them with suitable example

- 1) **Super Key(SK):** It is a set of one or more attributes within a table that can uniquely identify each record within a table. For example – Consider the Student table as follows –

Reg No.	Roll No	Phone	Name	Marks
R101	001	1111111111	AAA	88
R102	002	2222222222	BBB	83
R103	003	3333333333	CCC	98
R104	004	4444444444	DDD	67

Fig. 1.9.1 Student

The superkey can be represented as follows

Superkey		(RollNo, Phone, Name) Superkey		(Name, Marks) Not a Superkey	
RegNo.		RollNo	Phone	Name	Marks
R101		001	1111111111	AAA	88
R102		002	2222222222	BBB	83
R103		003	3333333333	CCC	98
R104		004	4444444444	DDD	67

Clearly using the (RegNo) and (RollNo,Phone,Name) we can identify the records uniquely but (Name, Marks) of two students can be same, hence this combination not necessarily help in identifying the record uniquely.

- 2) **Candidate Key(CK)** : The candidate key is a subset of superset. In other words candidate key is a single attribute or least or minimal combination of attributes that uniquely identify each record in the table. For example - in above given **Student** table, the candidate key is RegNo, (RollNo,Phone). The candidate key can be

Candidate key		Candidate key			
RegNo.		RollNo	Phone	Name	Marks
R101		001	1111111111	AAA	88
R102		002	2222222222	BBB	83
R103		003	3333333333	CCC	98
R104		004	4444444444	DDD	67

Thus every candidate key is a superkey but every superkey is not a candidate key.

- 3) **Primary Key(PK)**: The primary key is a candidate key chosen by the database designer to identify the tuple in the relation uniquely. For example – Consider the following representation of primary key in the student table

Primary key

RegNo.	RollNo	Phone	Name	Marks
R101	001	1111111111	AAA	88
R102	002	2222222222	BBB	83
R103	003	3333333333	CCC	98
R104	004	4444444444	DDD	67

Other than the above mentioned primary key, various possible primary keys can be (RollNo), (RollNo,Name), (RollNo, Phone)

The relation among super key, candidate key and primary can be denoted by

Candidate Key=Super Key – Primary Key

Rules for Primary Key

- (i) The primary key may have one or more attributes.
- (ii) There is **only one primary key** in the relation.
- (iii) The value of primary key attribute can not be NULL.
- (iv) The value of primary key attribute does not get changed.

4) **Alternate key** : The alternate key is a candidate key which is not chosen by the database designer to uniquely identify the tuples. For example –

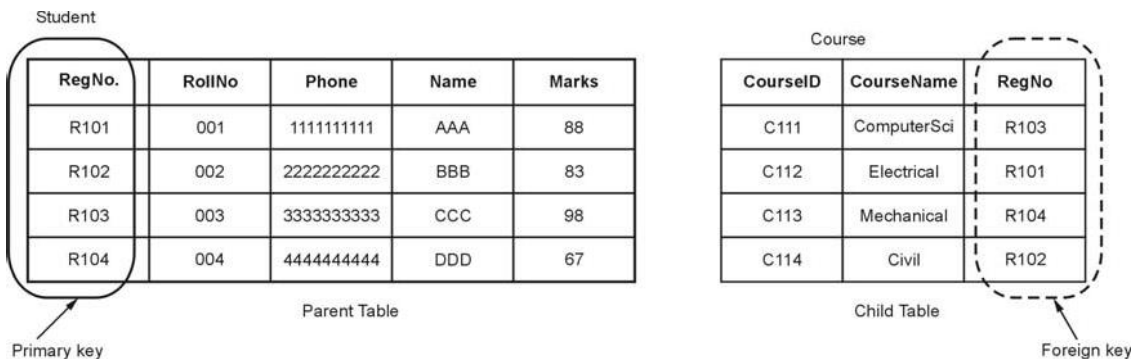
Primary key

Alternate key

RegNo.	RollNo	Phone	Name	Marks
R101	001	1111111111	AAA	88
R102	002	2222222222	BBB	83
R103	003	3333333333	CCC	98
R104	004	4444444444	DDD	67

5) Foreign key : Foreign key is a single attribute or collection of attributes in one table that refers to the primary key of other table.

- Thus foreign keys refer to primary key.
- The table containing the primary key is called **parent table** and the table containing foreign key is called **child table**.
- Example -



From above example, we can see that two tables are linked. For instance we could easily find out that the 'Student CCC has opted for ComputerSci course'

University Question

1. Explain distinction among the terms primary key, candidate key, foreign key and super key with suitable example

AU : May-06, 07, 12, Dec.-06, Marks 4

1.10 Integrity Constraints

AU : Dec.-05

Database integrity means correctness or accuracy of data in the database. A database may have number of integrity constraints. For example –

- The Employee ID and Department ID must consists of two digits.
- Every Employee ID must start with letter.

The integrity constraints are classified based on the concept of primary key and foreign key. Let us discuss the classification of constraints based on primary key and foreign key as follows –

1.10.1 Entity Integrity Rule

This rule states that “ In the relations , the value of attribute of primary key can not be null”.

The NULL represents a value for an attribute that is currently unknown or is not applicable for this tuple. The Nulls are always to deal with incomplete or exceptional data.

The primary key value helps in uniquely identifying every row in the table. Thus if the users of the database want to retrieve any row from the table or perform any action on that table, they must know the value of the key for that row. Hence it is necessary that the primary key should not have the NULL value.

1.10.2 Referential Integrity Rule

- Referential integrity refers to the **accuracy and consistency** of data within a relationship.
- In relationships, data is linked between two or more tables. This is achieved by having the foreign key (in the associated table) reference a primary key value (in the primary - or parent - table). Because of this, we need to ensure that data on both sides of the relationship remain intact.
- **The referential integrity rule states that** “whenever a **foreign key** value is used it must **reference a valid, existing primary key** in the parent table”.
- **Example :** Consider the situation where you have two tables : **Employees** and **Managers**. The **Employees** table has a foreign key attribute entitled **ManagedBy**, which points to the record for each employee’s manager in the **Managers** table.

Referential integrity enforces the **following three rules :**

- i) You cannot add a record to the **Employees** table unless the **ManagedBy** attribute points to a valid record in the **Managers** table. Referential integrity prevents the insertion of incorrect details into a table. Any operation that doesn't satisfy referential integrity rule fails.
- ii) If the primary key for a record in the **Managers** table changes, all corresponding records in the **Employees** table are modified.
- iii) If a record in the **Managers** table is deleted, all corresponding records in the **Employees** table are deleted.

Advantages of Referential Integrity

Referential integrity offers following advantages :

- i) Prevents the entry of duplicate data.
- ii) Prevents one table from pointing to a nonexistent field in another table.
- iii) Guaranteed consistency between "partnered" tables.
- iv) Prevents the deletion of a record that contains a value referred to by a foreign key in another table.
- v) Prevents the addition of a record to a table that contains a foreign key unless there is a primary key in the linked table.

University Question

1. Discuss the entity Integrity and referential integrity constraints. Why are they important ? Explain them with suitable examples.

AU : Dec.-05, Marks 10

1.11 Database integrity

- The foreign key is a key in one table that refers to the primary key of another table.
- The foreign key is basically used to link two tables. For example –

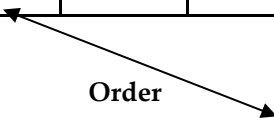
Consider Customer table as follows –

Customer

CustID	Name	City
C101	AAA	Chennai
C102	BBB	Mumbai
C103	CCC	Pune

Order

OrderID	Description	CustID
111	Bolts	C103
222	Nuts	C103
333	Beams	C101
444	Screws	C102
555	Disks	C101

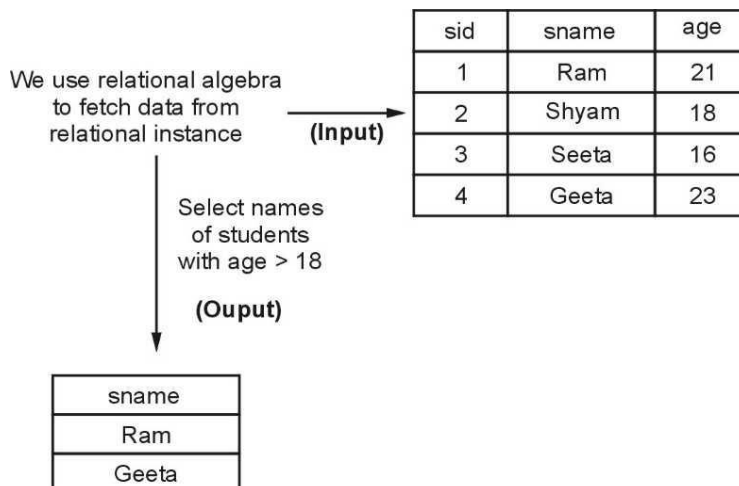


- Note that the "CustID" column in the "Order" table points to the "CustID" column in the "Customer" table.
- The "CustID" column in the "Customer" table is the PRIMARY KEY in the "Customer" table.
- The "CustID" column in the "Order" table is a FOREIGN KEY in the "Order" table.
- The table containing the foreign key is called the **child table**, and the table containing the primary key is called the **referenced** or **parent table**.
- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

- The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

1.12 Relational Algebra AU : May-03,04,05,14,15,16,17,18, Dec.-02,07,08,11,15,16,17

- There are two formal query languages associated with relational model and those are relational algebra and relational calculus.
- **Definition :** Relational algebra is a procedural query language which is used to access database tables to read data in different ways.
- The queries present in the relational algebra are denoted **using operators**.
- Every operator in relational algebra accepts relational instances (tables) as input and returns relational instance as output. For example :



- Each **relational algebra is procedural**. That means Each relational query describes a step-by-step procedure for computing the desired answer, based on the order in which operators are applied in the query.
- 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. The By composing the operators in relational expressions the complex relation can be defined.

1.12.1 Relational Operations

Types of operations in relational algebra

We have divided these operations in two categories:

1. Basic Operations
2. Derived Operations

Basic/Fundamental Operations:

1. Select (σ)
2. Project (Π)
3. Union (\cup)
4. Set Difference ($-$)
5. Cartesian product (\times)
6. Rename (ρ)

Derived Operations:

1. Natural Join (\bowtie)
2. Left, Right, Full outer join (\ltimes , \rtimes , $\ltimes\rtimes$)
3. Intersection (\cap)
4. Division (\div)

Lets discuss these operations one by one with the help of examples.

Select Operator (σ)

Select Operator is denoted by sigma (σ) and it is used to find the tuples (or rows) in a relation (or table) which satisfy the given condition.

If you understand little bit of SQL then you can think of it as a where clause in SQL, which is used for the same purpose.

Syntax of Select Operator (σ)

```
 $\sigma$  Condition/Predicate (Relation/Table name)
```

Select Operator (σ) Example

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Delhi

Query:

σ Customer_City="Agra" (CUSTOMER)

Output:

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra

Project Operator (π)

Project operator is denoted by π symbol and it is used to select desired columns (or attributes) from a table (or relation).

Project operator in relational algebra is similar to the Select statement in SQL.

Syntax of Project Operator (π)

π column_name1, column_name2, ..., column_nameN(table_name)

Project Operator (π) Example

In this example, we have a table CUSTOMER with three columns, we want to fetch only two columns of the table, which we can do with the help of Project Operator π .

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
-----	-----	-----
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Delhi

Query:

```
Π Customer_Name, Customer_City (CUSTOMER)
```

Output:

Customer_Name	Customer_City
-----	-----
Steve	Agra
Raghu	Agra
Chaitanya	Noida
Ajeet	Delhi
Carl	Delhi

Union Operator (U)

Union operator is denoted by U symbol and it is used to select all the rows (tuples) from two tables (relations).

Lets discuss union operator a bit more. Lets say we have two relations R1 and R2 both have same columns and we want to select all the tuples(rows) from these relations then we can apply the union operator on these relations.

Note: The rows (tuples) that are present in both the tables will only appear once in the union set. In short you can say that there are no duplicates present after the union operation.

Syntax of Union Operator (U)

```
table_name1 U table_name2
```

Union Operator (U) Example

Table 1: COURSE

Course_Id	Student_Name	Student_Id
C101	Aditya	S901
C104	Aditya	S901
C106	Steve	S911
C109	Paul	S921
C115	Lucy	S931

Table 2: STUDENT

Student_Id	Student_Name	Student_Age
S901	Aditya	19
S911	Steve	18
S921	Paul	19
S931	Lucy	17
S941	Carl	16
S951	Rick	18

Query:

```

[] Student_Name (COURSE) U [] Student_Name (STUDENT)

```

Output:

```

Student_Name
-----
Aditya
Carl
Paul
Lucy
Rick
Steve

```

Note: As you can see there are no duplicate names present in the output even though we had few common names in both the tables, also in the COURSE table we had the duplicate name itself.

Intersection Operator (\cap)

Intersection operator is denoted by \cap symbol and it is used to select common rows (tuples) from two tables (relations).

Lets say we have two relations R1 and R2 both have same columns and we want to select all those tuples(rows) that are present in both the relations, then in that case we can apply intersection operation on these two relations $R1 \cap R2$.

Note: Only those rows that are present in both the tables will appear in the result set.

Syntax of Intersection Operator (\cap)

```
table_name1  $\cap$  table_name2
```

Intersection Operator (\cap) Example

Lets take the same example that we have taken above.

Table 1: COURSE

Course_Id	Student_Name	Student_Id
-----	-----	-----
C101	Aditya	S901
C104	Aditya	S901
C106	Steve	S911
C109	Paul	S921
C115	Lucy	S931

Table 2: STUDENT

Student_Id	Student_Name	Student_Age
-----	-----	-----
S901	Aditya	19
S911	Steve	18
S921	Paul	19
S931	Lucy	17
S941	Carl	16
S951	Rick	18

Query:

```
 $\Pi$  Student_Name (COURSE)  $\cap$   $\Pi$  Student_Name (STUDENT)
```

Output:

```
Student_Name
-----
Aditya
Steve
Paul
Lucy
```

Set Difference (-)

Set Difference is denoted by $-$ symbol. Lets say we have two relations R1 and R2 and we want to select all those tuples(rows) that are present in Relation R1 but **not** present in Relation R2, this can be done using Set difference $R1 - R2$.

Syntax of Set Difference (-)

```
table_name1 - table_name2
```

Set Difference (-) Example

Lets take the same tables COURSE and STUDENT that we have seen above.

Query:

Lets write a query to select those student names that are present in STUDENT table but not present in COURSE table.

```
Π Student_Name (STUDENT) - Π Student_Name (COURSE)
```

Output:

```
Student_Name
-----
Carl
Rick
```

Cartesian product (X)

Cartesian Product is denoted by X symbol. Lets say we have two relations R1 and R2 then the cartesian product of these two relations ($R1 \times R2$) would combine each tuple of first relation R1 with the each tuple of second relation R2. I know it sounds confusing but once we take an example of this, you will be able to understand this.

Syntax of Cartesian product (X)

```
R1 X R2
```


Cartesian product (X) Example

Table 1: R

Col_A	Col_B
-----	-----
AA	100
BB	200
CC	300

Table 2: S

Col_X	Col_Y
-----	-----
XX	99
YY	11
ZZ	101

Query:

Lets find the cartesian product of table R and S.

R X S

Output:

Col_A	Col_B	Col_X	Col_Y
-----	-----	-----	-----
AA	100	XX	99
AA	100	YY	11
AA	100	ZZ	101
BB	200	XX	99
BB	200	YY	11
BB	200	ZZ	101
CC	300	XX	99
CC	300	YY	11
CC	300	ZZ	101

Note: The number of rows in the output will always be the cross product of number of rows in each table. In our example table 1 has 3 rows and table 2 has 3 rows so the output has $3 \times 3 = 9$ rows.

Rename (ρ)

Rename (ρ) operation can be used to rename a relation or an attribute of a relation.

Rename (ρ) Syntax:

$\rho(\text{new_relation_name}, \text{old_relation_name})$

Rename (ρ) Example

Lets say we have a table customer, we are fetching customer names and we are renaming the resulted relation to CUST_NAMES.

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
-----	-----	-----
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Delhi

Query:

```
 $\rho$  (CUST_NAMES,  $\Pi$  (Customer_Name) (CUSTOMER))
```

Output:

```
CUST_NAMES
-----
Steve
Raghu
Chaitanya
Ajeet
Carl
```

JOIN OPERATIONS:

A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied. It is denoted by \bowtie .

Example: **EMPLOYEE**

EMP_CODE	EMP_NAME
101	Stephan
102	Jack
103	Harry

SALARY

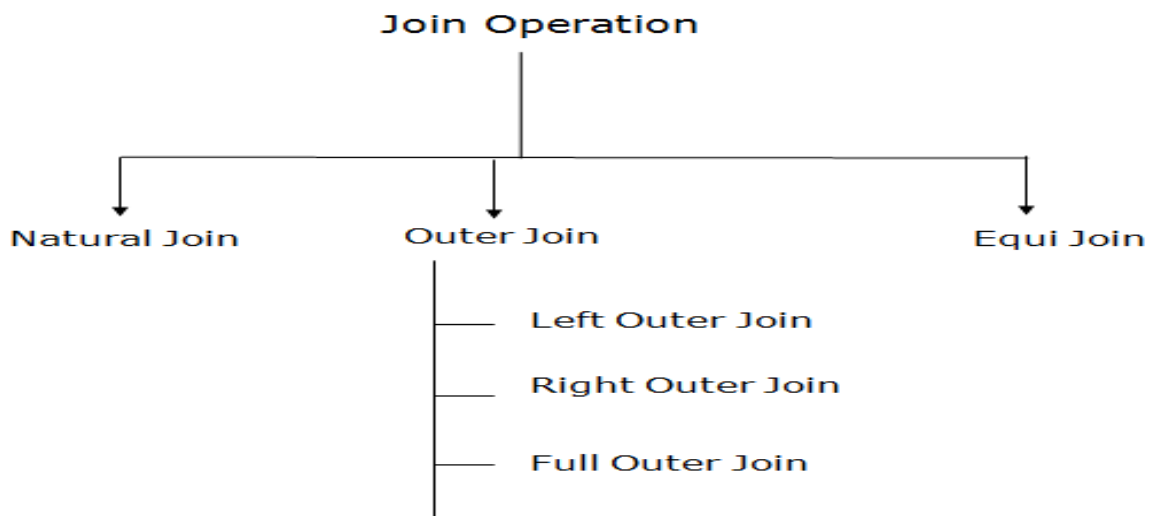
EMP_CODE	SALARY
101	50000
102	30000
103	25000

1. Operation: (EMPLOYEE \bowtie SALARY)

Result:

EMP_CODE	EMP_NAME	SALARY
101	Stephan	50000
102	Jack	30000
103	Harry	25000

TYPES OF JOIN OPERATIONS:



1. Natural Join:

- A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names.
- It is denoted by \bowtie .

Example: Let's use the above EMPLOYEE table and SALARY table:

Input:

1. \bowtie EMP_NAME, SALARY (EMPLOYEE \bowtie SALARY)

Output:

EMP_NAME	SALARY
Stephan	50000
Jack	30000
Harry	25000

2. Outer Join:

The outer join operation is an extension of the join operation. It is used to deal with missing information.

Example:

EMPLOYEE

EMP_NAME	STREET	CITY
Ram	Civil line	Mumbai
Shyam	Park street	Kolkata
Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

FACT_WORKERS

EMP_NAME	BRANCH	SALARY
Ram	Infosys	10000
Shyam	Wipro	20000
Kuber	HCL	30000
Hari	TCS	50000

Input:

1. (EMPLOYEE \bowtie FACT_WORKERS)

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru nagar	Hyderabad	TCS	50000

An outer join is basically of three types:

- a. Left outer join
- b. Right outer join
- c. Full outer join
 - a. Left outer join:
 - Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
 - In the left outer join, tuples in R have no matching tuples in S.
 - It is denoted by $\bowtie\leftarrow$.

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

1. EMPLOYEE \bowtie FACT_WORKERS

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL

b. Right outer join:

- Right outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In right outer join, tuples in S have no matching tuples in R.
- It is denoted by $\bowtie\leftarrow$.

Example: Using the above EMPLOYEE table and FACT_WORKERS Relation

Input: EMPLOYEE $\bowtie\leftarrow$ FACT_WORKERS

Output:

CITY	EMP_NAME	BRANCH	SALARY	STREET
Mumbai	Ram	Infosys	10000	Civil line
Kolkata	Shyam	Wipro	20000	Park street
Hyderabad	Hari	TCS	50000	Nehru street
NULL	Kuber	HCL	30000	NULL

c. Full outer join:

- Full outer join is like a left or right join except that it contains all rows from both tables.
- In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name.
- It is denoted by \bowtie .

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

1. EMPLOYEE \bowtie FACT_WORKERS

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL
Kuber	NULL	NULL	HCL	30000

3. Equi join:

It is also known as an inner join. It is the most common join. It is based on matched data as per the equality condition. The equi join uses the comparison operator(=).

Example:**CUSTOMER RELATION****PRODUCT**

CLASS_ID	NAME		PRODUCT_ID	CITY
1	John		1	Delhi
2	Harry		2	Mumbai
3	Jackson		3	Noida

Input: CUSTOMER \bowtie PRODUCT

Output:

CLASS_ID	NAME	PRODUCT_ID	CITY
1	John	1	Delhi
2	Harry	2	Mumbai
3	Harry	3	Noida

DIVIDE OPERATION

- The division operator is used when we have to evaluate queries which contain
- the keyword **ALL**.
- It is denoted by A/B where A and B are instances of relation.

For example - Find all the customers having accounts in all the branches. For that consider two tables - Customer and Account as

Customer	
Name	Branch
A	Pune
B	Mumbai
A	Mumbai
C	Pune

Account	
Branch	
Pune	
Mumbai	

Now A/B will give us

Name
A

Here We check all the branches from Account table against all the names from Customer table. We can then find that only customer A has all the accounts in all the branches.

Formal Definition of Division Operation : The operation A/B is define as the set of all xvalues (in the form of unary tuples) such that for every y value in (a tuple of) B, there is atuple $\langle x,y \rangle$ in A.

University Questions

1. Explain select, project, cartesian product and join operations in relational algebra with an example

AU : May-18, Marks 13, Dec.-16, Marks 6

2. List operations of relational algebra and purpose of each with example

AU : May-17, Marks 5

3. Differentiate between foreign key constraints and referential Integrity constraints with suitable example.

AU : Dec.-17, Marks 6

4. Explain various operations in relational algebra with examples

AU : May 03, Marks 10, Dec-07, Marks 8, Dec.- 08, Marks 10, May-14, Marks 16

5. Explain all join operations in relational algebra

AU : May 05, Marks 8

6. Briefly explain relational algebra

AU : May 04, Marks 8

7. What is rename operation in relational algebra ? Illustrate your answer with example

AU : Dec 02, Marks 2

Part III Structured Query Language(SQL)**1.13 SQL Fundamentals****AU : Dec.-15**

- Structure Query Language(SQL) is a database query language used for storing and managing data in Relational DBMS.
- Various parts of SQL are –
 - **Data Definition Language(DDL)** : It consists of a set of commands for defining relation schema, deleting relations, and modifying relation schemas.
 - **Data Manipulation Language(DML)** : It consists of set of SQL commands for inserting tuples into relational schema, deleting tuples from or modifying tuples in databases.
 - **Integrity** : The SQL DDL includes commands for specifying integrity constraints. These constraints must be satisfied by the databases.
 - **View definition** : The SQL DDL contains the commands for defining views for database.
 - **Transaction control** : The SQL also includes the set of commands that indicate beginning and ending of the transactions.
 - **Embedded SQL and Dynamic SQL** : There is a facility of including SQL commands in the programming languages like C,C++, COBOL or Java.
 - **Authorization** : The SQL DDL includes the commands for specifying access rights to relations and views.

1.13.1 Data Abstraction

The Basic data types used in SQL are –

- (1) **char(n)**: For representing the fixed length character string this data type is used. For instance – to represent name,designation, coursename, we use this data type. Instead of char we can also use **character**. The n is specified by the user.
- (2) **varchar(n)** : The varchar means character varying. That means – for denoting the variable length character strings this data type is used. The n is user specified maximum character length.
- (3) **int** : For representing the numeric values without precision, the int data type is used.
- (4) **numeric** : For representing, a fixed point number with user-specified precision this data type is used. The number consists of m digits plus sign k digits are to the right of precision. For instance the numeric(3,2) allows 333.11 but it does not allow 3333.11

- (5) **smallint** : It is used to store small integer value. It allows machine dependent subset of integer type.
- (6) **real** : It allows the floating point, double precision numbers.
- (7) **float(n)** : For representing the floating point number with precision of at least n digits this data type is used.

1.13.2 Basic Schema Definition

In this section, we will discuss various SQL commands for creating the schema definition.

There are three types of SQL Languages -

1. **DDL commands** : 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 are :

- **CREATE** – is used to create the database or its objects such as table, function, views and so on.
 - **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.
2. **DML commands** : DML stands for Data Manipulation Language. These commands deal with manipulation of data present in the database.

Examples of DML commands are :

- **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.
3. **DCL commands** : It stands for Data Control Language. It 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.

Let us discuss various commonly used SQL commands that help in building the basic schema.

(1) Create Table : The database relation can be created by using the **create table** command

Syntax

```
create table table_name;
```

Example

```
create table Student  
(RollNo int,  
Name varchar(10),  
Marks numeric(3,2),  
Primary key(RollNo));
```

The primary key attribute must be non null and unique.

(2) Insert : The insert command is used to insert data into the table. There are two syntaxes of inserting data into SQL

Syntax

- i) **Insert into** table_name (column1, column2, column3, ...) **values** (value1, value2, value3, ...);
- ii) **insert into** table_name **values** (value1, value2, value3, ...);

Example

- (i) **insert into** Student(RollNo,Name,Makrs) **values**(101,'AAA',56.45)
- (ii) **insert into** Student **values**(101,'AAA',56.45)

(3) Delete : This command is used to delete the existing record.

Syntax

```
delete from table_name  
where condition;
```

Example

```
Delete from student  
where RollNo=10
```

(4) Alter: 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.

Syntax for adding a columns

```
alter table table_name  
add column_name datatype;
```

Example

```
Alter table student  
Add address varchar(20)  
Syntax for dropping column  
Alter table table_name  
drop column column_name;
```

Example

```
Alter table student  
drop column address;
```

Basic Structure of SQL Queries

The **basic form** of SQL queries is

SELECT-FROM-WHERE. The syntax is as follows :

```
SELECT          [DISTINCT] target-list  
FROM            relation-list  
WHERE           qualification
```

- **SELECT** : This is one of the fundamental query command of SQL. It is similar to the projection operation of relational algebra. It selects the attributes based on the condition described by WHERE clause.
- **FROM** : This clause takes a relation name as an argument from which attributes are to be selected/projected. In case more than one relation names are given, this clause corresponds to Cartesian product.
- **WHERE** : This clause defines predicate or conditions, which must match in order to qualify the attributes to be projected.
- **Relation-list** : A list of relation names (tables)
- **target-list** : A list of attributes of relations from relation list (tables)
- **qualification** : Comparisons of attributes with values or with other attributes combined using AND, OR and NOT.
- **DISTINCT** is an optional keyword indicating that the answer should not contain duplicates. Normally if we write the SQL without DISTINCT operator then it does not eliminate the duplicates.

Example

```
SELECT sname  
FROM Student  
WHERE age > 18
```

- The above query will return names of all the students from student table where age of each student is greater than 18

Queries on Multiple Relations

Many times it is required to access multiple relations (tables) to operate on some information. For example consider two tables as **Student** and **Reserve**.

sid	sname	age
1	Ram	21
2	Shyam	18
3	Seeta	16
4	Geeta	23

sid	isbn	day
1	005	07-07-18
2	007	03-03-18
3	009	

Query : Find the names of students who have reserved the books with book isbn

```
Select Student.sname, Reserve.isbn
From Student, Reserve
Where Student.sid=Reserve.sid
```

Use of SQL Join

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.

Example : Consider two tables for using the joins in SQL. Note that **cid** is common column in following tables.

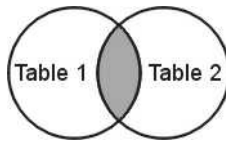
Student		
sid	cid	sname
1	101	Ram
2	101	Shyam
3	102	Seeta
4	NULL	Geeta

Reserve	
cid	cname
101	Pune
102	Mumbai
103	Chennai

1) Inner Join :

- The most important and frequently used of the joins is the INNER JOIN. They are also known as an EQUIJOIN.
- The INNER JOIN creates a new result table by combining column values of two tables (Table1 and Table2) based upon the **join-predicate**.
- The query compares each row of table1 with each row of Table2 to find all pairs of rows which satisfy the join-predicate.

- When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row. It can be represented as :



- Syntax :** The basic syntax of the INNER JOIN is as follows.

```
SELECT Table1.column1, Table2.column2...
FROM Table1
INNER JOIN Table2
ON Table1.common_field = Table2.common_field;
```

- Example :** For above given two tables namely Student and City, we can apply inner join. It will return the record that are matching in both tables using the common column **cid**. The query will be

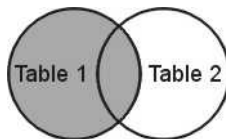
```
SELECT *
FROM Student Inner Join City on Student.cid=City.cid
```

The result will be

sid	cid	sname	cid	cname
1	101	Ram	101	Pune
2	101	Shyam	101	Pune
3	102	Seeta	102	Mumbai

2) Left Join :

- The SQL LEFT JOIN returns all rows from the left table, even if there are no matches in the right table. This means that if the ON clause matches 0 (zero) records in the right table; the join will still return a row in the result, but with NULL in each column from the right table.
- This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.
- It can be represented as -



- Syntax :** The basic syntax of a LEFT JOIN is as follows.

```
SELECT
SELECT Table1.column1, Table2.column2...
```

```
FROM Table1
LEFT JOIN Table2
ON Table1.common_field = Table2.common_field;
```

- **Example :** For above given two tables namely Student and City, we can apply Left join. It will Return all records from the left table, and the matched records from the right table using the common column **cid**. The query will be

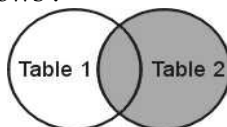
```
SELECT *
FROM Student Left Join City on Student.cid=City.cid
```

The result will be

sid	cid	sname	cid	cname
1	101	Ram	101	Pune
2	101	Shyam	101	Pune
3	102	Seeta	102	Mumbai
4	NULL	Geeta	NULL	NULL

3) Right Join :

- The SQL RIGHT JOIN returns all rows from the right table, even if there are no matches in the left table.
- This means that if the ON clause matches 0 (zero) records in the left table; the join will still return a row in the result, but with NULL in each column from the left table.
- This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.
- It can be represented as follows :



- **Syntax :** The basic syntax of a RIGHT JOIN is as follow -

```
SELECT Table1.column1, Table2.column2...
FROM Table1
RIGHT JOIN Table2
ON Table1.common_field = Table2.common_field;
```

- **Example :** For above given two tables namely Student and City, we can apply Right join. It will return all records from the right table, and the matched records from the left table using the common column **cid**. The query will be

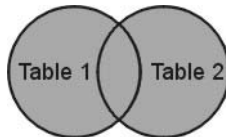
```
SELECT *
FROM Student Right Join City on Student.cid=City.cid
```


The result will be –

sid	cid	sname	cid	cname
1	101	Ram	101	Pune
2	101	Shyam	101	Pune
3	102	Seeta	102	Mumbai
NULL	NULL	NULL	103	Chennai

4) Full Join :

- The SQL FULL JOIN combines the results of both left and right outer joins.
- The joined table will contain all records from both the tables and fill in NULLs for missing matches on either side.
- It can be represented as



- **Syntax :** The basic syntax of a FULL JOIN is as follows :

```
SELECT Table1.column1, Table2.column2...
FROM Table1 FULL JOIN Table2 ON Table1.common_field = Table2.common_field;
```

The result will be -

- **Example :** For above given two tables namely **Student** and **City**, we can apply Full join. It will return returns rows when there is a match in one of the tables using the common column **cid**. The query will be -

```
SELECT *
FROM Student Full Join City on Student.cid=City.cid
```

The result will be -

sid	cid	sname	cid	cname
1	101	Ram	101	Pune
2	101	Shyam	101	Pune
3	102	Seeta	102	Mumbai
4	NULL	Geeta	NULL	NULL
NULL	NULL	NULL	103	Chennai

1.13.4 Additional Basic Operations

- 1) **The Rename Operation** : The SQL **AS** is used to assign temporarily a new name to a table column or table(relation) itself. One reason to rename a relation is to replace a long relation name with a shortened version that is more convenient to use elsewhere in the query. For example – “Find the names of students and isbn of book who reserve the books”.

Student

sid	sname	age
1	Ram	21
2	Shyam	18
3	Seeta	16
4	Geeta	23

Reserve

sid	isbn	day
1	005	07-07-18
2	007	03-03-18
3	009	05-05-18

```
Select S.sname,R.isbn
From Student as S, Reserve as R
Where S.sid=R.sid
```

In above case we could shorten the names of tables Student and Reserve as S and R respectively.

Another reason to rename a relation is a case where we wish to compare tuples in the same relation. We then need to take the Cartesian product of a relation with itself. For example –

If the query is – Find the names of students who reserve the book of isbn 005. Then the SQL statement will be –

```
Select S.sname,R.isbn
From Student as S, Reserve as R
Where S.sid=R.sid and S.isbn=005
```

- 2) **Attribute Specification in Select clause** : The symbol * is used in select clause to denote **all attributes**. For example – To select all the records from Student table we can write

```
Select* from Student
```

- 3) **Ordering the display of tuples** : For displaying the records in particular order we use **order by** clause.

The general syntax with ORDER BY is

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

- **Example** : Consider the Student table as follows –

sid	sname	marks	city
1	AAA	60	Pune
2	BBB	70	Mumbai
3	CCC	90	Pune
4	DDD	55	Mumbai

Query : Find the names of students from highest marks to lowest

Select sname
From Student
Order By marks

We can also use the **desc** for descending order and **asc** for ascending order. For example : .

In order to display names of the students in descending order of city – we can specify

Select sname
From Student
Order by city **desc**;

(4) Where clause Predicate :

- (i) The **between** operator can be used to simplify the where clause which is used to denote the value be less than or equal to some value and greater than or equal to some other value. For example – if we want the names of the students whose marks are between 80 and 90 then SQL statement will be

Select name
From Students
Where marks between 80 and 90;

Similarly we can make use of the comparison operators for various attributes. For example - If the query is – Find the names of students who reserve the book of isbn 005. Then the SQL statement will be –

Select sname
From Student , Reserve
Where (Student.sid,Reserve.isbn)=(Reserve.sid,005);

- (ii) We can use AND, OR and NOT operators in the Where clause. For filtering the records based on more than one condition, the AND and OR operators can be used. The NOT operator is used to demonstrate when the condition is not TRUE.

Consider following sample database – **Students** database, for applying AND, OR and NOT operators

sid	sname	marks	city
1	AAA	60	Pune
2	BBB	70	Mumbai
3	CCC	90	Pune
4	DDD	55	Mumbai

Syntax of AND

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

Example : Find the student having name "AAA" and lives in city "Pune"

```
SELECT *  
FROM Students  
Where sname='AAA' AND city='Pune'
```

Output

sid	sname	marks	city
1	AAA	60	Pune

Syntax OR

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

Example : Find the student having name "AAA" OR lives in city "Pune"

```
SELECT *  
FROM Students  
Where sname='AAA' OR city='Pune'
```

Output

sid	sname	marks	city
1	AAA	60	Pune
3	CCC	90	Pune

Syntax NOT

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

Example : Find the student who do not have city "Pune"

```
SELECT *  
FROM Students  
Where NOT city='Pune'
```

Output

sid	sname	marks	city
2	BBB	70	Mumbai
4	DDD	55	Mumbai

1.13.5 Domain and Key Constraint

Domain Constraint

- A domain is defined as the set of all unique values permitted for an attribute. For example, a domain of date is the set of all possible valid dates, a domain of Integer is all possible whole numbers, and a domain of day-of-week is Monday, Tuesday ... Sunday.
- This in effect is defining rules for a particular attribute. If it is determined that an attribute is a date then it should be implemented in the database to prevent invalid dates being entered.
- Domain constraints are user defined data type and we can define them like this :
- Domain constraint = Data type + Constraints
- The constraints can be specified using NOT NULL / UNIQUE / PRIMARY KEY / FOREIGN KEY / CHECK / DEFAULT.
- For example –

```
Create domain id_value integer
constraint id_test
check(value > 100); ← cheking if stud_id value is greater than 100

create table student (
stu_id id_value PRIMARY KEY,
stu_name CHAR(30),
stu_age integer
);
```

Key Constraint

- A key constraint is a statement that a certain minimal subset of the fields of a relation is a unique identifier for a tuple.
- For example - Consider the students relation and the constraint that no two students have the same student id. This IC is an example of a key constraint.
- The **definition** of key constraints contain **two parts** -
 - Two distinct tuples in a legal instance (an instance that satisfies all Integrity Constraints including the key constraint) **cannot have identical values** in all the fields of a key.
 - No subset of the set of fields in a key is a **unique identifier** for a tuple.
- The **first part** of the definition means that, in any legal instance, the values in the key fields uniquely identify a tuple in the instance. When specifying a key constraint, the DBA or user must be sure that this constraint will not prevent them from storing a 'correct' set of tuples. For example, several students may have the same name, although each student has a unique student id. If the name field is declared to be a key, the DBMS will not allow the Students relation to contain two tuples describing different students with the same name.

- The **second part** of the definition means, for example, that the set of fields {RollNo, Name} is not a key for Students, because this set properly contains the key {RollNo}. The set {RollNo, Name} is an example of a superkey, which is a set of fields that contains a key.
- The key constraint can be specified using SQL as follows -
 - In SQL, we can declare that a subset of the columns of a table constitute a key by using the UNIQUE constraint.
 - At most one of these candidate keys can be declared to be a primary key, using the PRIMARY KEY constraint. For example -

```
CREATE TABLE Student(RollNo integer,  
                      Name CHAR(20),  
                      age integer,  
                      UNIQUE(Name,age),  
                      CONSTRAINT StudentKey PRIMARY KEY(RollNo))
```

This definition says that **RollNo** is a **Primary key** and Combination of **Name and age** is also a key.

1.13.6 String Operations

- For string comparisons, we can use the comparison operators =, <, >, <=, >=, <> with the ordering of strings determined alphabetically as usual.
- SQL also permits a variety of functions on character strings such as concatenation using operator ||, extracting substrings, finding length of string, converting strings to upper case(using function **upper(s)**) and lowercase(using function **lower(s)**), removing spaces at the end of string(using function(trim(s)) and so on.
- Pattern matching can also be performed on strings using two types of special characters –
 - Percent(%): It matches zero, one or multiple characters
 - Underscore(_): The _ character matches any single character.
- The percentage and underscore can be used in combinations.
- Patterns are case sensitive. That means upper case characters do not match lowercase characters or vice versa.
- **For instance :**
 - 'Data%' matches any string beginning with "Data", For instance it could be with "Database", "DataMining", "DataStructure"
 - '___' matches any string of exactly three characters.
 - '___%' matches any string of at least length 3 characters.

- The **LIKE** clause can be used in **WHERE** clause to search for specific patterns.
- **For example** – Consider following **Employee Database**

EmpID	EmpName	Department	Date_of_Join
1	Sunil	Marketing	1-Jan
2	Mohsin	Manager	2-Jan
3	Supriya	Manager	3-Jan
4	Sonia	Accounts	4-Jan
5	Suraj	Sales	5-Jan
6.	Archana	Purchase	6-Jan

(1) Find all the employee with EmpName starting with “s”

SQL Statement:

```
SELECT * FROM Employee
WHERE EmpName LIKE 's%'
```

Output

EmpID	EmpName	Department	Date_of_Join
1	Sunil	Marketing	1-Jan
3	Supriya	Manager	3-Jan
4	Sonia	Accounts	4-Jan
5	Suraj	Sales	5-Jan

(2) Find the names of employee whose name begin with S and end with a

SQL Statement :

```
SELECT EmpName FROM Employee
WHERE EmpName LIKE 'S%a'
```

Output

EmpName
Supriya
Sonia

(3) Find the names of employee whose name begin with S and followed by exactly four characters

```
SELECT EmpName FROM Employee
WHERE EmpName LIKE 'S_____'
```

Output

EmpName
Sunil
Sonia
Suraj

1.13.7 Set Operations

1) **UNION** : To use this UNION clause, each SELECT statement must have

- i) The same number of columns selected
- ii) The same number of column expressions
- iii) The same data type and
- iv) Have them in the same order

This clause is used to combine two tables using UNION operator. It replaces the OR operator in the query. The union operator eliminates duplicate while the union all query will retain the duplicates.

Syntax

The basic syntax of a UNION clause is as follows –

SELECT column1 [, column2]

FROM table1 [, table2]

[WHERE condition]

UNION

SELECT column1 [, column2]

FROM table1 [, table2]

[WHERE condition]

Here, the given condition could be any given expression based on your requirement.

Consider Following relations –

Student		
sid	sname	age
1	Ram	21
2	Shyam	18
3	Seeta	16
4	Geeta	23

Reserve		
sid	isbn	day
1	005	07-07-18
2	005	03-03-17
3	007	08-11-16

Book		
isbn	bname	Author
005	DBMS	XYZ
006	OS	PQR
007	DAA	ABC

Example : Find the names of the students who have reserved the 'DBMS' book or 'OS' Book

The query can then be written by considering the Student, Reserve and Book table as

```
SELECT S.sname
FROM Student S, Reserve R, Book B
WHERE S.sid=R.sid AND R.isbn=B.isbn AND B.bname='DBMS'
UNION
SELECT S.sname
FROM Student S, Reserve R, Book B
WHERE S.sid=R.sid AND R.isbn=B.isbn AND B.bname='OS'
```

2) Intersect : The common entries between the two tables can be represented with the help of Intersect operator. It replaces the **AND** operator in the query.

Syntax

The basic syntax of a INTERSECT clause is as follows –

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
INTERSECT
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Example : Find the students who have reserved both the 'DBMS' book and 'OS' Book

The query can then be written by considering the Student, Reserve and Book table as

```
SELECT S.sid, S.sname
FROM Student S, Reserve R, Book B
WHERE S.sid=R.sid AND R.isbn=B.isbn AND B.bname='DBMS'
INTERSECT
SELECT S.sname
FROM Student S, Reserve R, Book B
WHERE S.sid=R.sid AND R.isbn=B.isbn AND B.bname='OS'
```

3) Except : The EXCEPT clause is used to represent the set-difference in the query. This query is used to represent the entries that are present in one table and not in other.

Syntax :

The basic syntax of a EXCEPT clause is as follows –

```
SELECT column1 [, column2 ]
```

```
FROM table1 [, table2 ]
```

```
[WHERE condition]
```

EXCEPT

```
SELECT column1 [, column2 ]
```

```
FROM table1 [, table2 ]
```

```
[WHERE condition]
```

Example : Find the students who have reserved both the 'DBMS' book but not reserved 'OS' Book

The query can then be written by considering the Student, Reserve and Book table as

```
SELECT S.sid, S.sname
```

```
FROM Student S, Reserve R, Book B
```

```
WHERE S.sid=R.sid AND R.isbn=B.isbn AND B.bname='DBMS'
```

```
EXCEPT
```

```
SELECT S.sname
```

```
FROM Student S, Reserve R, Book B
```

```
WHERE S.sid=R.sid AND R.isbn=B.isbn AND
```

```
B.bname='OS'
```

1.13.8 Aggregate Functions

- An aggregate function allows you to perform a calculation on a set of values to return a single scalar value.
- SQL offers five built-in aggregate functions :
 1. Average : avg
 2. Minimum : min
 3. Maximum : max
 4. Total: sum
 5. Count :

Basic Aggregation

- The aggregate functions that accept an expression parameter can be modified by the keywords DISTINCT or ALL. If neither is specified, the result is the same as if ALL were specified.

DISTINCT	Modifies the expression to include only distinct values that are not NULL
ALL	Includes all rows where expression is not NULL

- **Syntax of all the Aggregate Functions**

```
AVG( [ DISTINCT | ALL ] expression)
COUNT(*)
COUNT( [ DISTINCT | ALL ] expression )
MAX( [ DISTINCT | ALL ] expression)
MIN( [ DISTINCT | ALL ] expression)
SUM( [ DISTINCT | ALL ] expression)
```

- The **avg** function is used to compute average value. For example – To compute average marks of the students we can use

SQL Statement

```
SELECT AVG(marks)
FROM Students
```

- The **Count** function is used to count the total number of values in the specified field. It works on both numeric and non-numeric data type. COUNT (*) is a special implementation of the COUNT function that returns the count of all the rows in a specified table. COUNT (*) also considers Nulls and duplicates. For example Consider following table

Test	
id	value
11	100
22	200
33	300
NULL	400

SQL Statement

```
SELECT COUNT(*)
FROM Test
```

Output

4

```
SELECT COUNT(ALL id)
FROM Test
```

Output

3

- The **min** function is used to get the minimum value from the specified column. For example – Consider the above created **Test** table

SQL Statement

```
SELECT Min(value)
FROM Test
```

Output

100

- The **max** function is used to get the maximum value from the specified column.
For example – Consider the above created **Test** table

SQL Statement

```
SELECT Max(value)
FROM Test
```

Output

400

- The **sum** function is used to get total sum value from the specified column. For example – Consider the above created **Test** table

SQL Statement

```
SELECT sum(value)
FROM Test
```

Output

1000

Use of Group By and Having Clause

(i) Group By :

- The GROUP BY clause is a SQL command that is used to group rows that have the same values.
- The GROUP BY clause is used in the SELECT statement.
- Optionally it is used in conjunction with aggregate functions.
- The queries that contain the GROUP BY clause are called grouped queries
- This query returns a single row for every grouped item.
- Syntax :**

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

The general syntax with ORDER BY is

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

- Example :** Consider the Student table as follows -

sid	sname	marks	city
1	AAA	60	Pune
2	BBB	70	Mumbai
3	CCC	90	Pune
4	DDD	55	Mumbai

Query : Find the total marks of each student in each city

```
SELECT SUM(marks), city
FROM Student
GROUP BY city
```

Output

SUM(marks)	city
150	Pune
125	Mumbai

(ii) Having :

- HAVING filters records that work on summarized GROUP BY results.
- HAVING applies to summarized group records, whereas WHERE applies to individual records.
- Only the groups that meet the HAVING criteria will be returned.
- HAVING requires that a GROUP BY clause is present.
- WHERE and HAVING can be in the same query.
- Syntax :

```
SELECT column-names
FROM table-name
WHERE condition
GROUP BY column-names
HAVING condition
```

- **Example :** Consider the Student table as follows -

sid	sname	marks	city
1	AAA	60	Pune
2	BBB	70	Mumbai
3	CCC	90	Pune
4	DDD	55	Mumbai
5	EEE	84	Chennai

Query : Find the total marks of each student in the city named 'Pune' and 'Mumbai' only

```
SELECT SUM(marks), city
FROM Student
GROUP BY city
HAVING city IN('Pune','Mumbai')
```

Output

- The result will be as follows –

SUM(marks)	city
150	Pune
125	Mumbai

1.13.9 Nested Queries

In nested queries, a **query is written inside a query**. The result of inner query is used in execution of outer query.

There are two types of nested queries :

i) Independent Query :

- In independent nested queries, query execution starts from innermost query to outermost queries.
- The execution of inner query is independent of outer query, but the result of inner query is used in execution of outer query.
- Various operators like IN, NOT IN, ANY, ALL etc are used in writing independent nested queries.
- For example - Consider three tables namely **Student**, **City** and **Student_City** as follows -

Student		
sid	sname	phone
1	Ram	1111
2	Shyam	2222
3	Seeta	3333
4	Geeta	4444

City	
cid	cname
101	Pune
102	Mumbai
103	Chennai

Student_City	
sid	cid
1	101
1	103
2	101
3	102
4	102
4	103

- **Example 1** - If we want to find out **sid** who live in city 'Pune' or 'Chennai'. We can then write independent nested query using **IN** operator. Here we can use the IN operator allows you to specify multiple values in a WHERE clause. The IN operator is a shorthand for multiple OR conditions.

Step 1 : Find cid for cname='Pune' or 'Chennai'. The query will be

```
SELECT cid
FROM City
WHERE cname='Pune' or 'Chennai'
```

Step 2 : Using cid obtained in step 1 we can find the sid. The query will be

```
SELECT sid
FROM Student_City
WHERE cid IN
      (SELECT cid FROM City WHERE cname='Pune' or cname='Chennai')
```

The inner query will return a set with members 101 and 103 and outer query will return those **sid** for which **cid** is equal to any member of set (101 and 103 in this case). So, it will return 1, 2 and 4.

- **Example 2** : If we want to find out **sname** who live in city 'Pune' or 'Chennai'.

```
SELECT sname FROM Student WHERE sid IN
(SELECT sid FROM Student_City WHERE cid IN
(SELECT cid FROM City WHERE cname='Pune' or cname='Chennai'))
```

ii) Co-related Query :

In co-related nested queries, the output of inner query depends on the row which is being currently executed in outer query. For example

If we want to find out sname of Student who live in city with cid as 101, it can be done with the help of co-related nested query as :

```
SELECT sname FROM Student S WHERE EXISTS
(SELECT * FROM Student_City SC WHERE S.sid=SC.sid and SC.cid=101)
```

Here For each row of **Student S**, it will find the rows from **Student_City** where S.sid = SC.sid and SC.cid=101.

If for a **sid** from **Student S**, atleast a row exists in **Student_City** SC with **cid**=101, then inner query will return true and corresponding **sid** will be returned as output.

1.13.10 Modification of Databases

The modification of database is an operation for making changes in the existing databases. Various operations of modification of database are – insertion, deletion and updation of databases.

1. **Deletion** : The **delete** command is used to delete the existing record.

Syntax

```
delete from table_name  
where condition;
```

Example

```
delete from student  
where RollNo=10
```

- 2. Insertion :** The insert command is used to insert data into the table. There are two syntaxes of inserting data into SQL

Syntax

```
(i) Insert into table_name (column1, column2, column3, ...)  
values (value1, value2, value3, ...);  
(ii) insert into table_name  
values (value1, value2, value3, ...);
```

Example

```
(i) insert into Student(RollNo,Name,Makrs) values(101,'AAA',56.45)  
(ii) insert into Student values(101,'AAA',56.45)
```

- 3. Update :** The update statement is used to modify the existing records in the table.

```
update table_name  
set column1=value1, column2=value2,...  
where condition;
```

Example:

```
Delete student  
Set Name='WWW'  
where RollNo=101
```

Example 1.13.1 Write the DDL, DML, DCL for the students database. Which contains student details: name, id, DOB, branch, DOJ.

Course details : Course name, Course id, Stud.id, Faculty name, id, marks

AU : Dec.-17, Marks 15

Solution :

DDL Commands

```
CREATE TABLE Student  
(  
stud_name varchar(20),  
stud_id int(3),  
DOB varchar(15),  
branch varchar(10),  
DOJ varchar(15),  
);  
CREATE TABLE Course  
(
```



```
course_name varchar(20),  
course_id int(5),  
stud_id int(3),  
facult_name varchar(20),  
faculty_id varchar(5),  
marks real  
);
```

DML Commands

The commands which we will use here are insert and select. The insert command is used to insert the values into database tables. Using the select command, the database values can be displayed.

(1) Inserting values into Student table

```
insert into Student(stud_name,stud_id,DOB,branch,DOJ)  
values('AAA',11,'01-10-1999' , 'computers','5-3-2018')
```

```
insert into Student(stud_name,stud_id,DOB,branch,DOJ)  
values('BBB',12,'24-5-1988' , 'Mechanical','17-2-2016')
```

```
insert into Student(stud_name,stud_id,DOB,branch,DOJ)  
values('CCC',13,'8-1-1990' , 'Electrical','22-9-2017')
```

(2) Inserting values into Course table

```
insert into Course(course_name,course_id,stud_id,faculty_name,faculty_id,marks)  
values('Basic',101,11,'Archana' , 'F001','50')
```

```
insert into Course(course_name,course_id,stud_id,faculty_name,faculty_id,marks)  
values('Intermediate',102,12,'Rupali' , 'F002','70')
```

```
insert into Course(course_name,course_id,stud_id,faculty_name,faculty_id,marks)  
values('Advanced',103,13,'Sunil' , 'F003','100')
```

(3) Displaying records of Student table

```
Select * from Student;
```

(4) Displaying records of Course table

```
Select * from Course;
```

University Questions

- | | |
|---|------------------------------|
| 1. Explain aggregate functions in SQL with example. | AU : May 18, Marks 13 |
| 2. Write DDL, DML, DCL commands for the students database. | AU : Dec 17, Marks 7 |
| 3. Explain about SQL fundamentals. | AU : May 16, Marks 8 |
| 4. Explain about Data Definition Language. | AU : May 16, Marks 8 |
| 5. Explain the six clauses in the syntax of SQL query and show what type of constructs can be specified in each of the six clauses. Which of the six clauses are required and which are optional. | AU : Dec 15, Marks 16 |
| 6. Explain- DDL and DML | AU : Dec 14, Marks 8 |

1.14 Advanced SQL Features

1.14.1 Embedded SQL

- The programming module in which the SQL Statements are embedded is called Embedded SQL module.
- It is possible to embed SQL statements inside the programming language such as C, C++, PASCAL, Java and so on.
- It allows the application languages to communicate with DB and get requested result.
- The high level languages which supports embedding SQLs within it are also known as **host language**.
- An embedded SQL program must be processed by a special preprocessor prior to compilation. The preprocessor replaces embedded SQL requests with host-language declarations and procedure calls that allow runtime execution of the database accesses. Then, the resulting program is compiled by the host-language compiler. This is the main distinction between embedded SQL and JDBC or ODBC.

Example of Embedded SQL

To connect java application with the oracle database, we need to follow 5 following steps.

In this example, we are using Oracle 10g as the database. So we need to know following information for the oracle database:

Driver class: The driver class for the oracle database is oracle.jdbc.driver.OracleDriver.

Connection URL: The connection URL for the oracle10G database is jdbc:oracle:thin:@localhost:1521:xe where jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and XE is the Oracle service name. You may get all these information from the tnsnames.ora file.

Username: The default username for the oracle database is system.

Password: It is the password given by the user at the time of installing the oracle

database.

Create a Table

Before establishing connection, let's first create a table in oracle database. Following is the SQL query to create a table.

```
create table emp(id number(10),name varchar2(40),age number(3));
```

Example to Connect Java Application with Oracle database

In this example, we are connecting to an Oracle database and getting data from **emp** table. Here, **system** and **oracle** are the username and password of the Oracle database.

```
import java.sql.*;
class OracleCon{
public static void main(String args[]){
try{
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");

//step2 create the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

//step3 create the statement object
Statement stmt=con.createStatement();

//step4 execute query
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getInt(3));

//step5 close the connection object
con.close();

catch(Exception e){ System.out.println(e);}

}

}
```

Features of Embedded SQL

- (1) It is easy to use.
- (2) It is ANSI/ISO standard programming language.
- (3) It requires less coding
- (4) The precompiler can optimize execution time by generating stored procedures for the Embedded SQL statements.
- (5) It is identical over different host languages, hence writing applications using different programming languages is quite easy.

University Questions

1. What is the need of embedded SQL.

AU : May 17, Dec 17, Marks 2

2. What is embedded SQL ? Give an example

AU : Dec 16, Marks 5, May-14, Dec 14, Marks 8

1.15 Dynamic SQL

AU : May-17, Dec.-17

- Dynamic SQL is a programming technique which allows to build the SQL statements dynamically at runtime.
- Dynamic SQL statements are not embedded in the source program but stored as strings of characters that are manipulated during a program's runtime.
- These SQL statements are either entered by a programmer or automatically generated by the program.
- Dynamic SQL statements also may change from one execution to the next without manual intervention.
- Dynamic SQL facilitates automatic generation and manipulation of program modules for efficient automated repeating task preparation and performance.
- Dynamic SQL facilitates the development of powerful applications with the ability to create database objects for manipulation according to user input.

PreparedStatement interface

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

Let's see the example of parameterized query:

```
String sql="insert into emp values(?,?,?)";
```

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

Why use PreparedStatement?

Improves performance: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

Example of PreparedStatement to insert records until user press n

```
import java.sql.*;
import java.io.*;
class RS{
    public static void main(String args[])throws Exception{
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
            "system","oracle");
        PreparedStatement ps=con.prepareStatement("insert into emp130 values(?,?,?)");
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        do{
            System.out.println("enter id:");
            int id=Integer.parseInt(br.readLine());
            System.out.println("enter name:");
            String name=br.readLine();
            System.out.println("enter salary:");
            float salary=Float.parseFloat(br.readLine());
            ps.setInt(1,id);
            ps.setString(2,name);
            ps.setFloat(3,salary);
            int i=ps.executeUpdate();
            System.out.println(i+" records affected");
            System.out.println("Do you want to continue: y/n");
            String s=br.readLine();
            if(s.startsWith("n")){
                break;
            }
        }while(true);

        con.close();
    }
}
```
