UNIT: I

# UNIT I RELATIONAL DATABASES

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

## Introduction to Database Management System

As the name suggests, the database management system consists of two parts. They are:

1. Database and
2. Management System

**What is a Database**?

To find out what database is, we have to start from data, which is the basic building block of any DBMS.

**Data:** Facts, figures, statistics etc. having no particular meaning (e.g. 1, ABC, 19 etc).

**Record:** Collection of related data items, e.g. in the above example the three data items had no meaning. But if we organize them in the following way, then they collectively represent meaningful information

| Roll | Name | Age |
|---|---|---|
| 1 | ABC | 19 |

**Table or Relation**: Collection of related records.

| Roll | Name | Age |
|---|---|---|
| 1 | ABC | 19 |
| 2 | DEF | 22 |
| 3 | XYZ | 28 |

The columns of this relation are called **Fields, Attributes or Domains**. The rows are called **Tuples or Records**.

**Database:** Collection of related relations.

Consider the following collection of tables:

**T1**

| Roll | Name | Age |
|---|---|---|
| 1 | ABC | 19 |
| 2 | DEF | 22 |
| 3 | XYZ | 28 |

**T2**

| Roll | Address |
|---|---|
| 1 | KOL |
| 2 | DEL |
| 3 | MUM |

**T3**

| Roll | Year |
|---|---|
| 1 | I |
| 2 | II |
| 3 | I |

**T4**

| Year | Hostel |
|---|---|
| I | H1 |
| II | H2 |

We now have a collection of 4 tables. They can be called a “related collection” because we can clearly find out that there are some common attributes existing in a selected pair of tables. Because of these common attributes we may combine the data of two or more tables together to find out the complete details of a student. Questions like “Which hostel does the youngest student live in?” can be answered now, although Age and Hostel attributes are in different tables.

A database in a DBMS could be viewed by lots of different people with different responsibilities.

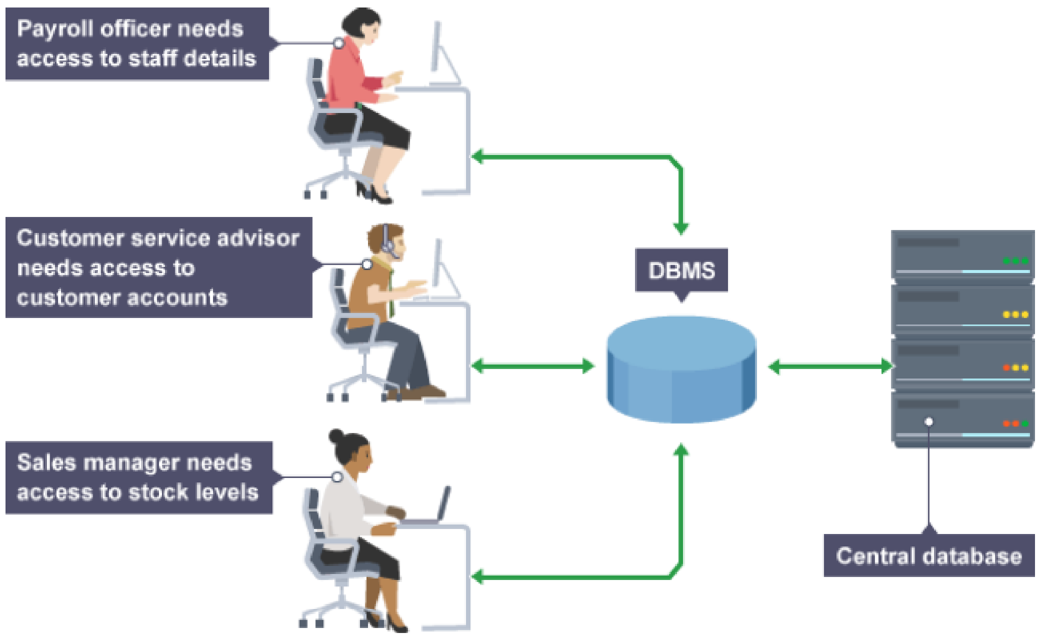

Payroll officer needs access to staff details
Customer service advisor needs access to customer accounts
Sales manager needs access to stock levels
DBMS
Central database

**Figure 1.1: Employees are accessing Data through DBMS**

For example, within a company there are different departments, as well as customers, who each need to see different kinds of data. Each employee in the company will have different levels of access to the database with their own customized front-end application.

**What is Management System?**

A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data. This is a collection of related data with an implicit meaning and hence is a database. The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient. By data, we mean known facts that can be recorded and that have implicit meaning.

The management system is important because without the existence of some kind of rules and regulations it is not possible to maintain the database. We have to select the particular attributes which should be included in a particular table; the common attributes to create relationship between two tables; if a new record has to be inserted or deleted then which tables should have to be handled etc. These issues must be resolved by having some kind of rules to follow in order to maintain the integrity of the database.

Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results.

Because information is so important in most organizations, computer scientists have developed a large body of concepts and techniques for managing data. These concepts and technique form the focus of this book.

**Database Management System (DBMS) and Its Applications:**

A Database management system is a computerized record-keeping system. It is a repository or a container for collection of computerized data files. The overall purpose of DBMS is to allow he users to define, store, retrieve and update the information contained in the database on demand. Information can be anything that is of significance to an individual or organization.

Databases touch all aspects of our lives. Some of the major areas of application are as follows:

1. Banking
2. Airlines
3. Universities
4. Manufacturing and selling
5. Human resources

**Enterprise Information**

- **Sales**: For customer, product, and purchase information.
- **Accounting**: For payments, receipts, account balances, assets and other accounting information.
- **Human resources**: For information about employees, salaries, payroll taxes, and benefits, and for generationof paychecks.
- **Manufacturing**: For management of the supply chain and for tracking production of items in factories, inventories of items inwarehouses and stores, and orders for items.
- **Online retailers**: For sales data noted above plus online order tracking,generation of recommendation lists,and maintenance of online product evaluations.

**Banking and Finance**

- **Banking**: For customer information, accounts, loans, and banking transactions.
- **Credit card transactions**: For purchases on credit cards and generation of monthly statements.
- **Finance:** For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also for storing real-time market data to enable online trading by customers and automated trading by the firm.
- **Universities:** For student information, course registrations, and grades (in addition to standard enterprise information such as human resources and accounting).
- **Airlines**: For reservations and schedule information. Airlines were among the first to use databases in ageographically distributed manner.
- **Telecommunication**: For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

**1. What is database? (2m)**

**2. What is DBMS? (2m)**

**3. List out some representative application of databases (2m)**

- **Explain the characteristics of database approach. (U)(APRIL 2008) (15m)**

## 1.0 Purpose of Database System

Database systems arose in response to early methods of computerized management of commercial data. As an example of such methods, typical of the 1960s, consider part of a university organization that, among other data, keeps information about all instructors, students, departments, and course offerings. One way to keep the information on a computer is to store it in operating system files. To allow users to manipulate the information, the system has a number of application programs that manipulate the files, including programs to:

✓ Add new students, instructors, and courses

✓ Register students for courses and generate class registers

✓ Assign grades to students, compute grade point averages (GPA), and generate transcripts

System programmers wrote these application programs to meet the needs of the university.New application programs are added to the system as the need arises. For example, suppose that a university decides to create a new major (say, computer science).As a result, the university creates a new department and creates new permanent files (or adds information to existing files) to record information about all the instructors in the department, students in that major, course offerings, degree requirements, etc. The university may have to write new application programs to deal with rules specific to the new major. New application programs may also have to be written to handle new rules in the university. Thus, as time goes by, the system acquires more files and more application programs.

This typical file-processing system is supported by a conventional operating system. The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files. Before database management systems (DBMSs) were introduced, organizations usually stored information in such systems. Keeping organizational information in a fileprocessing system has a number of major disadvantages:

**Data redundancy and inconsistency**. Since different programmers create the files and application programs over a long period, the various files are likely to have different structures and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places (files).

For example, if a student has a double major (say, music and mathematics) the address and telephone number of that student may appear in a file that consists of student records of students in the Music department and in a file that consists of student records of students in the Mathematics department. This redundancy leads to higher storage and access cost. In addition, it may lead to data inconsistency; that is, the various copies of the same data may no longer agree. For example, a changed student address may be reflected in the Music department records but not elsewhere in the system.

**Difficulty in accessing data**. Suppose that one of the university clerks needs to find out the names of all students who live within a particular postal-code area. The clerk asks the data-processing department to generate such a list. Because the designers of the original system did not anticipate this request, there is no application program on hand to meet it. There is, however, an application program to generate the list of all students.

The university clerk has now two choices: either obtain the list of all students and extract the needed information manually or ask a programmer to write the necessary application program. Both alternatives are obviously unsatisfactory. Suppose that such a program is written, and that, several days later, the same clerk needs to trim that list to include only those students who have taken at least 60 credit hours. As expected, a program to generate such a list does not exist. Again, the clerk has the preceding two options, neither of which is satisfactory. The point here is that conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. More responsive data-retrieval systems are required for general use.

**Data isolation**. Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

**Integrity problems**. The data values stored in the database must satisfy certain types of consistency

**constraints**. Suppose the university maintains an account for each department, and records the balance amount in each account. Suppose also that the university requires that the account balance of a departmentmay never fall below zero. Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.

**Atomicity problems.** A computer system, like any other device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure.

Consider a program to transfer $500 from the account balance of department A to the account balance of department B. If a system failure occurs during the execution of the program, it is possible that the $500 was removed from the balance of department A but was not credited to the balance of department B, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur.

**Concurrent-access anomalies**. For the sake of overall performance of the system and faster response, manysystems allow multiple users to update the data simultaneously. Indeed, today, the largest Internet retailers may have millions of accesses per day to their data by shoppers.

In such an environment, interaction of concurrent updates is possible and may result in inconsistent data. Consider department A, with an account balance of $10,000. If two department clerks debit the account balance (by say $500 and $100, respectively) of department A at almost exactly the same time, the result of the concurrent executions may leave the budget in an incorrect (or inconsistent) state. Suppose that the programs executing on behalf of each withdrawal read the old balance, reduce that value by the amount being withdrawn, and write the result back. If the two programs run concurrently, they may both read the value $10,000, and write back $9500 and $9900, respectively.

Depending on which one writes the value last, the account balance of department A may contain either $9500 or $9900, rather than the correct value of $9400. To guard against this possibility, the system must maintain some form of supervision.

But supervision is difficult to provide because data may be accessed by many different application programs that have not been coordinated previously.

As another example, suppose a registration program maintains a count of students registered for a course, in order to enforce limits on the number of students registered. When a student registers, the program reads the current count for the courses, verifies that the count is not already at the limit, adds one to the count, and stores the count back in the database. Suppose two students register concurrently, with the count at (say) 39. The two program executions may both read the value 39, and both would then write back 40, leading to an incorrect increase of only 1, even though two students successfully registered for the course and the count should be 41.

Furthermore, suppose the course registration limit was 40; in the above case both students would be able to register, leading to a violation of the limit of 40 students.

**Security problems**. Not every user of the database system should be able to access all the data. For example, in a university, payroll personnel need to see only that part of the database that has financial information. They do not need access to information about academic records. But, since application programs are added to the file-processing system in an ad hoc manner, enforcing such security constraints is difficult.

These difficulties, among others, prompted the development of database systems. In what follows, we shall see the concepts and algorithms that enable database systems to solve the problems with file-processing systems.

1. **What is the purpose of data base system? (2m)**
2. **What are problems caused by redundancy? (2m)**

## 1.1 VIEWS OF DATA

**What are the different views of data in DBMS?**

In a DBMS, the different views of data are the internal view, dealing with physical storage; the conceptual view, describing the logical structure and relationships; and the external view, providing customized data access for individual users or groups.

**Data Abstraction:**

Data abstractions in DBMS refer to the hiding of unnecessary data from the end-user. Database systems have complex Data Structures and relationships. These difficulties are masked so that users may readily access the data, and just the relevant section of the database is made accessible to them through data abstraction. Let's understand this more with an example.

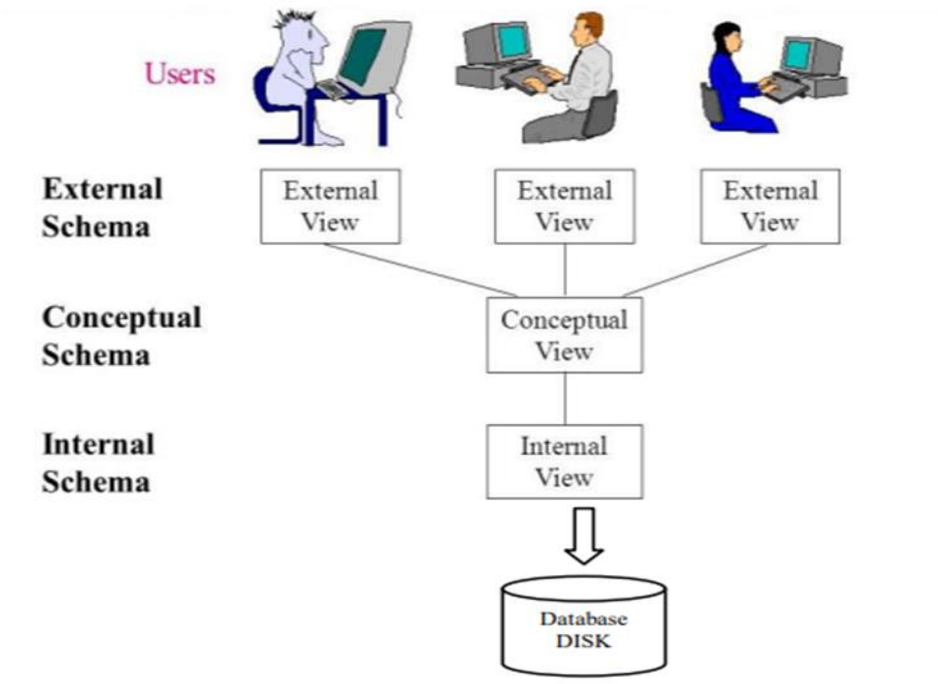


**Figure 1.2 : Levels of Abstraction in a DBMS**

**Example of Data Abstraction**

Our dependency on email communication in the modern day is apparent. However, data location and storage specifics are frequently hidden from us. Regardless of their physical storage or data type, reading and managing our emails is our top priority while using Gmail.

If we want to retrieve any email from Gmail, we don't know where that data is physically kept, such as in India or the United States, or what data model was utilized to store it. These things are not essential to us. Only our email is of interest to us. Levels of Data Abstraction in DBMS

Database Management Systems (DBMS) are essential to effectively managing and organizing data in data management. Three interrelated levels of abstraction, each performing a different function, make up the hierarchical framework through which these systems operate.

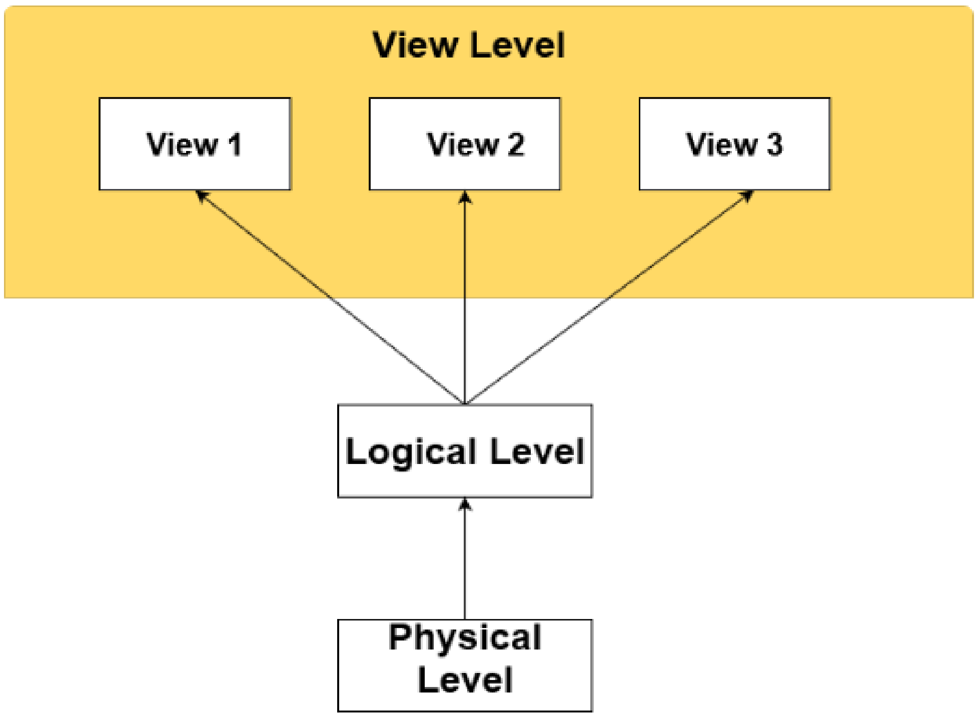
There are three levels of abstraction for DBMS are:

External Level / View Level

Conceptual Level/ Logical Level

View or External Level

Now let's look at the levels of data abstractions in DBMS and discuss them in detail.



**1. Physical or Internal Level**

It is the lowest level of abstraction for DBMSs, defining how data is stored, data structures for storing data, and database access mechanisms.

Developers or database application programmers decide how to store data in the database. It is complex to understand.

**Example**

The physical level, being the lowest level of abstraction, can be understood with an example, like how information about a customer is stored in tables while the data is stored in the form of blocks.

Another example of physical-level abstraction would be sequential file organization due to the continuous storage of records. While in indexed file organizations, we can access the records with the help of indexes.

**2. Logical or Conceptual Level**

The logical level is the next higher level or intermediate level. It explains what data is stored in the database and how those data are related. It seeks to explain the complete or entire data by describing what tables should be constructed and what the linkages between those tables should be. It is less complex than the physical level.

**Example**

The logical level in DBMS is used for representing entities and relationships among the data stored. For example, defining tables and their attributes and specifying relationships between them. A table named 'class' may have different attributes like student_name, Roll_no, student ID, and Marks.

A table named 'IDs' contains details about the address of the teacher's ID (foreign key), and student ID (foreign key).

**3. View or External Level**

This is the top level. There are various views at the view level, with each view defining only a portion of the total data. It also facilitates user engagement by providing a variety of views or numerous views of a single database. All users have access to the view level. This is the easiest and most simple level.

**Example**

The external level in DBMS defines a part of the entire data and simplifies interaction with the user by providing multiple views of a similar database. For example, interacting with a system using a graphical user interface (GUI) to access an application's features. Here GUI is the view level, and the user does not know how and what data is exactly stored, i.e hiding the details from the user.

**What is Data Independence?**

The primary goal of data abstractions in DBMS is to obtain data independence in order to save time and money when modifying or altering a database.

Data independence is known as the ability to change the scheme without impacting the programmes and applications to be rewritten. Data is isolated from programmes so that changes to the data do not influence the program's or application's execution.

**Types of Data Independence**

Data Independence is mainly of two types :

**1. Physical level independence**

It refers to the ability to change the physical schema without changing the conceptual or logical schema, which is done for optimization purposes.

**2. Logical level independence**

This feature is referred to as the ability to change the logical schema without changing the external schema or application program.

Any modifications to the conceptual representation of the data would not affect the user's perception of the data.

**Advantages of Data Abstraction in DBMS**

It reduces the complexity for the users.

While retrieval of data abstractions in DBMS makes the system efficient.

Increases the usability of the users.

Increases the security aspect of the application as implementation details are hidden from the users.

Increases the code duplicity and reusability.

**Disadvantages of Data Abstraction in DBMS**

Some of the disadvantages of data abstraction are mentioned below.

Data abstraction might be confusing for developers as there are complexities at multiple levels of the database.

Whenever the extra layer is added to the code, navigation becomes challenging.

At lower levels changing the behavior of DBMS might prove to be a challenging task or might even be impossible because of abstraction.

1. **Describe the three levels of data abstraction? (2m)**
2. **What is a physical level? (2m)**
3. **What is a logical level? (2m)**
4. **What is a view level? (2m)**
5. **What is a schema? (2m)**
6. **What are the different types of schemas? (2m)**
7. **What is an instance? (2m)**

- **Explain the system structure of a database system with neat block diagram. (R)(DEC 2007), (MAY 2010) (MAY 2012) (15m)**

- **Explain the different between physical and logical data independence with an example. (U)(APRIL 2008) (15m)**
- **Describe the three-schema architecture. Why do we need mapping between schema Levels? How do different schema definition languages support this architecture? (R) (DEC 2008) (15m)**
- **What is a view? How can it be created? Explain with an example. (R) (MAY 2010) (15m)**

## 1.2 DATA MODELS

Underlying the structure of a database is the data model: a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. A data model provides a way to describe the design of a database at the physical, logical, and view levels.

• **Relational Model**. The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Tables are also known as relations. The relational model is an example of a record-based model. Record-based models are so named because the database is structured in fixed-format records of several types. Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes. The columns of the table correspond to the attributes of the record type. The relational data model is the most widely used data model, and a vast major ity of current database systems are based on the relational model. Chapters 2 through 8 cover the relational model in detail.

• **Entity-Relationship Model**. The entity-relationship (E-R) data model uses a collection of basic objects, called entities, and relationships among these objects. An entity is a "thing" or "object" in the real world that is distinguishable from other objects. The entity-relationship model is widely used in database design, and Chapter 7 explores it in detail.

• **Object-Based Data Model**. Object-oriented programming (especially in Java, C++, or C#) has become the dominant software-development methodology. This led to the development of an object-oriented data model that can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity. The object-relational data model combines features of the object-oriented data model and relational data model. Chap ter 22 examines the object-relational data model.

• **Semi structured Data Model**. The semi structured data model permits the specification of data where individual data items of the same type may have different sets of attributes. This is in contrast to the data models mentioned earlier, where every data item of a particular type must have the same set of attributes. The Extensible Markup Language (XML) is widely used to represent semi structured data. Chapter 23 covers it.

Historically, the network data model and the hierarchical data model pre ceded the relational data model. These models were tied closely to the underlying implementation, and complicated the task of modeling data. As a result they are used little now, except in old database code that is still in service in some places. They are outlined online in Appendices D and E for interested readers.

1. **What do you mean by data model? (2m)**
2. **What are the different data models? (2m)**
3. **What is a relational model? (2m)**
4. **What is an entity-relationship model? (2m)**
5. **What is object based data model? (2m)**
6. **What is a semi structured data model? (2m)**

- **What is a data model? Explain various data model for describing the design of a database at the logical level. (U) (APRIL 2008), (MAY 2010) (15m)**

## 1.3 DATABASE SYSTEM ARCHITECTURE

The architecture of a database system is greatly influenced by the underlying computer system on which the database system runs. Database systems can be centralized, or client-server, where one server machine executes work on behalf of multiple client machines. Database systems can also be designed to exploit parallel computer architectures. Distributed databases span multiple geographically separated machines. Database applications are usually partitioned into two or three parts, as in

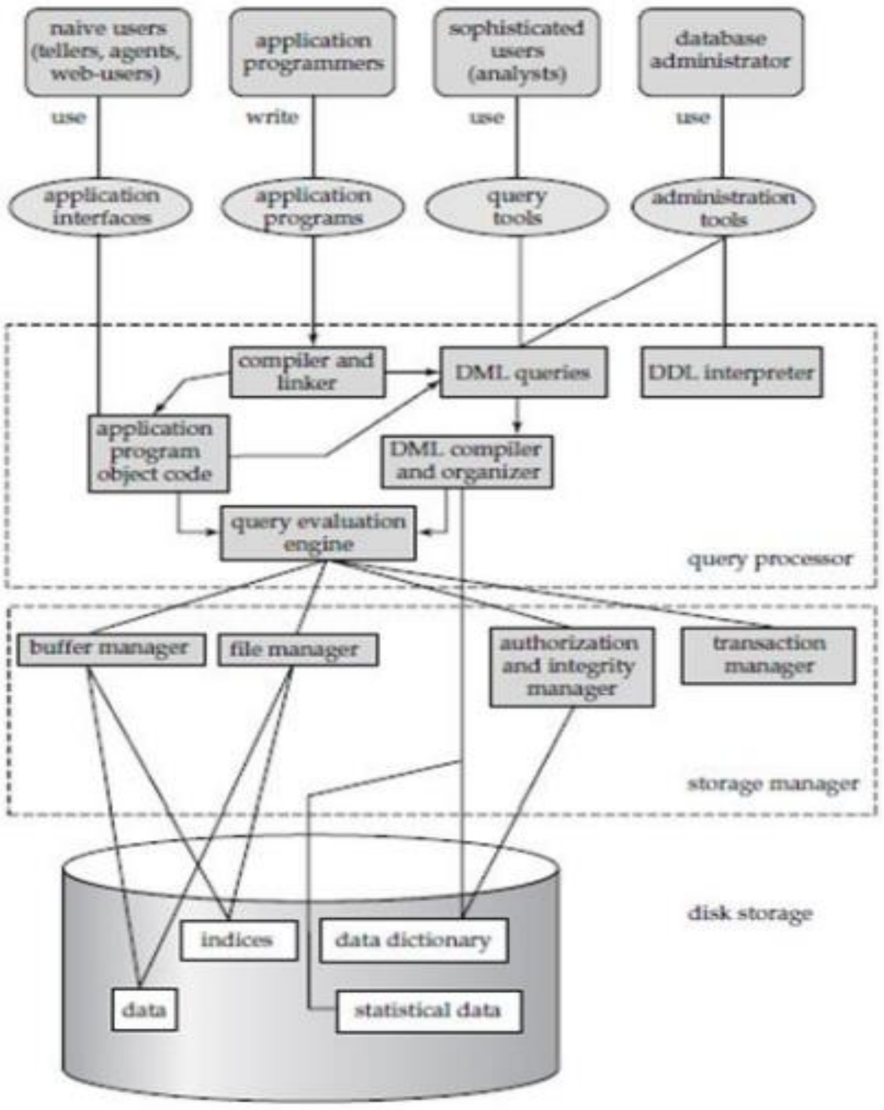


**Figure 1.3: Database System Architecture**

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the storage manager and the query processor components**. *The storage manager*** is important because databases typically require a large amount of storage space. The query processor is important because it helps the database system simplify and facilitate access to data.

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

Database applications are usually partitioned into two or three parts, as in Figure 1.4. In a two-tier architecture, the application resides at the client machine, where it invokes database system functionality at the server machine through query language statements.

Application program interface standards like ODBC and JDBC are used for interaction between the client and the server. In contrast, in a three-tier architecture, the client machine acts as merely a front end and does not contain any direct database calls. Instead, the client end communicates with an application server, usually through a forms interface.

The application server in turn communicates with a database system to access data. The business logic of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead of being distributed across multiple clients. Three-tier applications are more appropriate for large applications, and for applications that run on the Worldwide Web.

1. **What is storage manager? (2m)**
2. **What are the components of storage manager? (2m)**
3. **List out the data structure used to implement the storage manager. (2m)**

- **State and explain the Architecture of DBMS. (Nov/Dec-2017) (15m)**

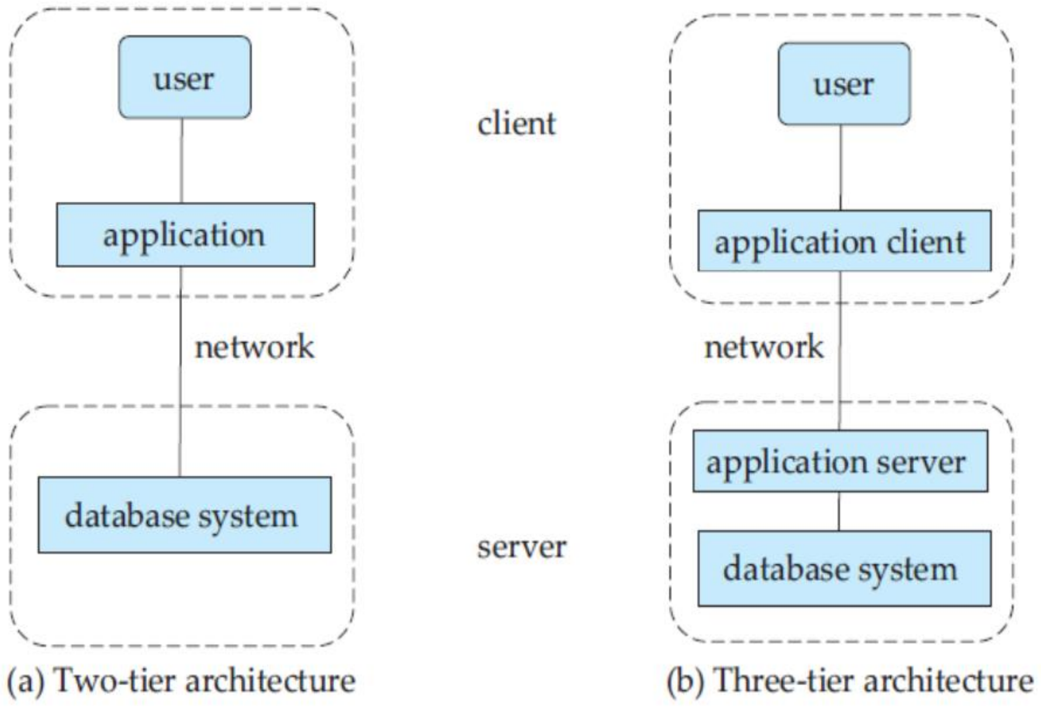


**Figure 1.4: Two-tier and three-tier architectures.**

## 1.4. INTRODUCTION TO RELATIONAL DATABASES:

### 1.4.1 RELATIONAL MODEL:

The relational model uses a collection of tables to represent both data and the relationships among those data. The relational model describes data at the logical and view levels, abstracting away low-level details of data storage.

### 1.4.2 KEYS:

**Super Key: A** super key is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation.

**Example:**

Consider the relation instructor with attributes ID, name, Dept name and salary.

The super keys are {ID},{ID,name},{name,dept name},{ID,name,dept name},{ID,name,dept name, salary}The name attribute of instructor, on the other hand, is not a superkey, because several instructors might have the same name.

**Candidate Keys:**

The minimal superkeys are called candidate keys.It is possible that several distinct sets of attributes could serve as a candidate key. Both {ID} and {name, dept name} are candidate keys. Although the attributes ID and name together can distinguish instructortuples, their combination, {ID, name}, does not form a candidate key, since the attribute ID alone is a candidate key.

**Primary Key:**

A candidate key that is chosen by the database designer as the principal means of identifying tuples within a relation. The primary key should be chosen such that its attribute values are never, or very rarely, changed. For instance, the address field of a person should not be part of the primary key, since it is likely to change. Social-security numbers, on the other hand, are guaranteed never to change. Primary key attributes are also underlined.

**Foreign Key:**

An attribute in a relation r1 is the primary key of another relation, say r2. This attribute is called a foreign key from r1, referencing r2. The relation r1 is also called the referencing relation of the foreign key dependency, and r2 is called the referenced relation of the foreign key.

**Example:**

Consider the relation instructor with attributes ID, name, Dept name and salary.

ID is the Primary Key

Consider the relation Department with attributes Dept name and instructor name.

Dept name is the Primary Key

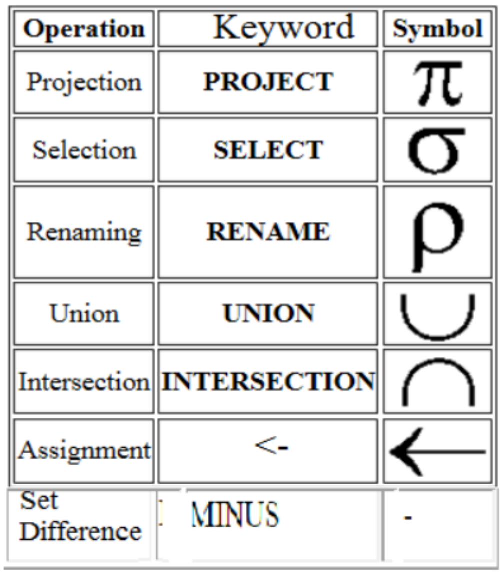
The attribute dept name in instructor is a foreign key from instructor, referencingdepartment, since dept name is the primary key of department.

1. **What is a candidate key? (2m)**
2. **What is a primary key? (2m)**
3. **What is a foreign key? (2m)**
4. **Distinguish between Key and Super Key? (2m)**

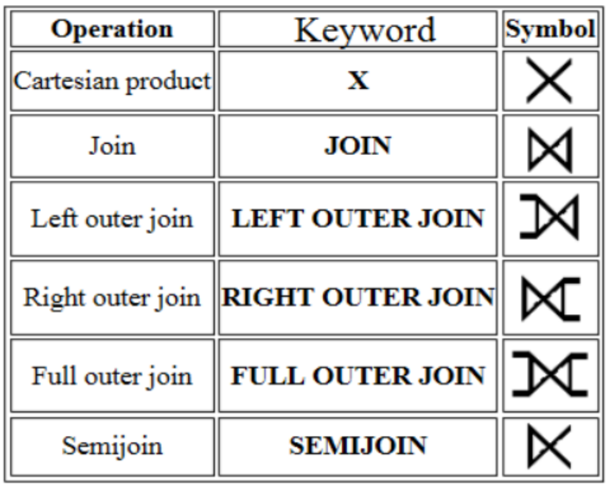
## 1.4.3RELATIONAL ALGEBRA

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output.

- ✓ It uses operators to perform queries.
- ✓ An operator can be either unary or binary.
- ✓ They accept relations as their input and yield relations as their output.
- ✓ Relational algebra is performed recursively on a relation and intermediateresults are also considered relations.



| Operation | Keyword | Symbol |
|---|---|---|
| Projection | PROJECT | $\pi$ |
| Selection | SELECT | $\sigma$ |
| Renaming | RENAME | $\rho$ |
| Union | UNION | $\cup$ |
| Intersection | INTERSECTION | $\cap$ |
| Assignment | <- | $\leftarrow$ |
| Set Difference | MINUS | - |



| Operation | Keyword | Symbol |
|---|---|---|
| Cartesian product | X | $\times$ |
| Join | JOIN | $\bowtie$ |
| Left outer join | LEFT OUTER JOIN | ⟕ |
| Right outer join | RIGHT OUTER JOIN | ⟖ |
| Full outer join | FULL OUTER JOIN | ⟗ |
| Semijoin | SEMIJOIN | ⋉ |

The fundamental operations of relational algebra are as follows −

- Select
- Project
- Union
- Set different
- Cartesian product
- Rename

**Select Operation (σ):**

The SELECT operation is used to choose a subset of the tuples from a relation that satisfies a selection condition.

**Syntax:**

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

**Example:**

$\sigma_{Dno=4}(EMPLOYEE)$

Output − Selects tuples from Employee where Dno is '4'.

$\sigma_{Salary>30000}(EMPLOYEE)$

Output − Selects tuples from Employee where salary is greater than '30000'.

$\sigma_{(Dno=4 \text{ AND } Salary>25000) \text{ OR } (Dno=5 \text{ AND } Salary>30000)}(EMPLOYEE)$

Output − Selects tuples from Employee where Dno is '4' and salary is greater than '25000' or Dno is '5' and salary is greater than '30000' .

**Project Operation (∏):**

The PROJECT operation is used to select certain columns from the table and discards the other columns.

**Syntax:**

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

**Example:**

$\pi_{Lname,\ Fname,\ Salary}(EMPLOYEE)$

Output - Selects and projects columns named as Lname, Fname and salary fromthe relation Employee.

**Union Operation (∪):**

The union operation used to create a realtion that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.

**Example:**

$\prod_{author}$ (Books) ∪ $\prod_{author}$ (Articles)

Output − Projects the names of the authors who have either written a book or an article or both.

**Set Difference (−):**

The set difference is used to find tuples that are in one relation but are not inanother.

**Example:**

$\prod_{author}$ (Books) − $\prod_{author}$ (Articles)

Output − Provides the name of authors who have written books but not
articles.

**Cartesian Product (X):**

The cartesian product allows us to combine information from any two different relations.

**Example:**

$\prod_{author}$ = 'tutorialspoint'$^{(Books\ X\ Articles)}$

Output − Yields a relation, which shows all the books and articles written by tutorialspoint.

**Rename Operation (ρ):**

The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter rho ρ.

Notation − $\rho_{x}(E)$

Where the result of expression E is saved with name of x.

**Intersection Operation (∩):**

The union operation used to create a realtion that includes all tuples that are in
both R and S.

**Example:**

$\prod_{author}$ (Books) ∩ $\prod_{author}$ (Articles)

Output − Projects the names of the authors who have written both book and
article.

**Assignment Operation (←):**

The Assignment Operation (←) works like assignment in a programming language.

The result of the expression to the right of the ← is assigned to the variable on the left of the ←.

**Example:**

temp←$\sigma_{Dno=4}$(EMPLOYEE)

**Natural join (⋈):**

Natural join (⋈) is a binary operator that is written as (R ⋈ S) where R and S are relations.

The result of the natural join is the set of all combinations of tuples in R and S that are equal on their common attribute names.

**Example**: Consider the tables Employee and Dept and their natural join:

*Employee*

| Name | EmpId | DeptName |
|---|---|---|
| Harry | 3415 | Finance |
| Sally | 2241 | Sales |
| George | 3401 | Finance |
| Harriet | 2202 | Sales |

*Dept*

| DeptName | Manager |
|---|---|
| Finance | George |
| Sales | Harriet |
| Production | Charles |

*Employee ⋈ Dept*

| Name | EmpId | DeptName | Manager |
|---|---|---|---|
| Harry | 3415 | Finance | George |
| Sally | 2241 | Sales | Harriet |
| George | 3401 | Finance | George |
| Harriet | 2202 | Sales | Harriet |

**Semi join ( ⋉ ):**

The left semijoin is joining similar to the natural join and written as R ⋉ S where R and S are relations.

The result of this semijoin is the set of all tuples in R for which there is a tuple in S that is equal on their common attribute names.

**Example:** Consider the tables Employee and Dept and their semi join:

*Employee*

| Name | EmpId | DeptName |
|---|---|---|
| Harry | 3415 | Finance |
| Sally | 2241 | Sales |
| George | 3401 | Finance |
| Harriet | 2202 | Production |

*Dept*

| DeptName | Manager |
|---|---|
| Sales | Bob |
| Sales | Thomas |
| Production | Katie |
| Production | Mark |

*Employee ⋉ Dept*

| Name | EmpId | DeptName |
|---|---|---|
| Sally | 2241 | Sales |
| Harriet | 2202 | Production |

**Division (÷)**

The division is a binary operation that is written as R ÷ S. The result consists of the restrictions of tuples in R to the attribute names unique to R.

**Example:**

Consider the tables Completed and DB Project

*Completed*

| Student | Task |
|---|---|
| Fred | Database1 |
| Fred | Database2 |
| Fred | Compiler1 |
| Eugene | Database1 |
| Eugene | Compiler1 |
| Sarah | Database1 |
| Sarah | Database2 |

*Completed ÷ DBProject*

| Student |
|---|
| Fred |
| Sarah |

*DBProject*

| Task |
|---|
| Database1 |
| Database2 |

**Left outer join ( )**

The left outer join is written as R S where R and S are relations.

The result of the left outer join is the set of all combinations of tuples in R and S that are equal on their common attribute names, in addition to tuples in R that have nomatching tuples in S.

**Example:**Consider the tables Employee and Dept and their left outer join:

*Employee*

| Name | EmpId | DeptName |
|---|---|---|
| Harry | 3415 | Finance |
| Sally | 2241 | Sales |
| George | 3401 | Finance |
| Harriet | 2202 | Sales |
| Tim | 1123 | Executive |

*Dept*

*Employee* ⟕ *Dept*

| Name | EmpId | DeptName | Manager |
|---|---|---|---|
| Harry | 3415 | Finance | ω |
| Sally | 2241 | Sales | Harriet |
| George | 3401 | Finance | ω |
| Harriet | 2202 | Sales | Harriet |
| Tim | 1123 | Executive | ω |

| DeptName | Manager |
|---|---|
| Sales | Harriet |
| Production | Charles |

In the resulting relation, tuples in S which have no common values in common attribute names with tuples in R take a null value, ω.

**Right outer join ( ):**

The right outer join of relations R and S is written as R S.

The result of the right outer join is the set of all combinations of tuples in R and Sthat are equal ontheir common attribute names, in addition to tuples in S that have no matching tuples in R.

**Example:** Consider the tables Employee and Dept and their right outer join: their common attribute names, in addition to tuples in S that have no matching tuples in R.

*Employee*

| Name | EmpId | DeptName |
|---|---|---|
| Harry | 3415 | Finance |
| Sally | 2241 | Sales |
| George | 3401 | Finance |
| Harriet | 2202 | Sales |
| Tim | 1123 | Executive |

*Dept*

| DeptName | Manager |
|---|---|
| Sales | Harriet |
| Production | Charles |

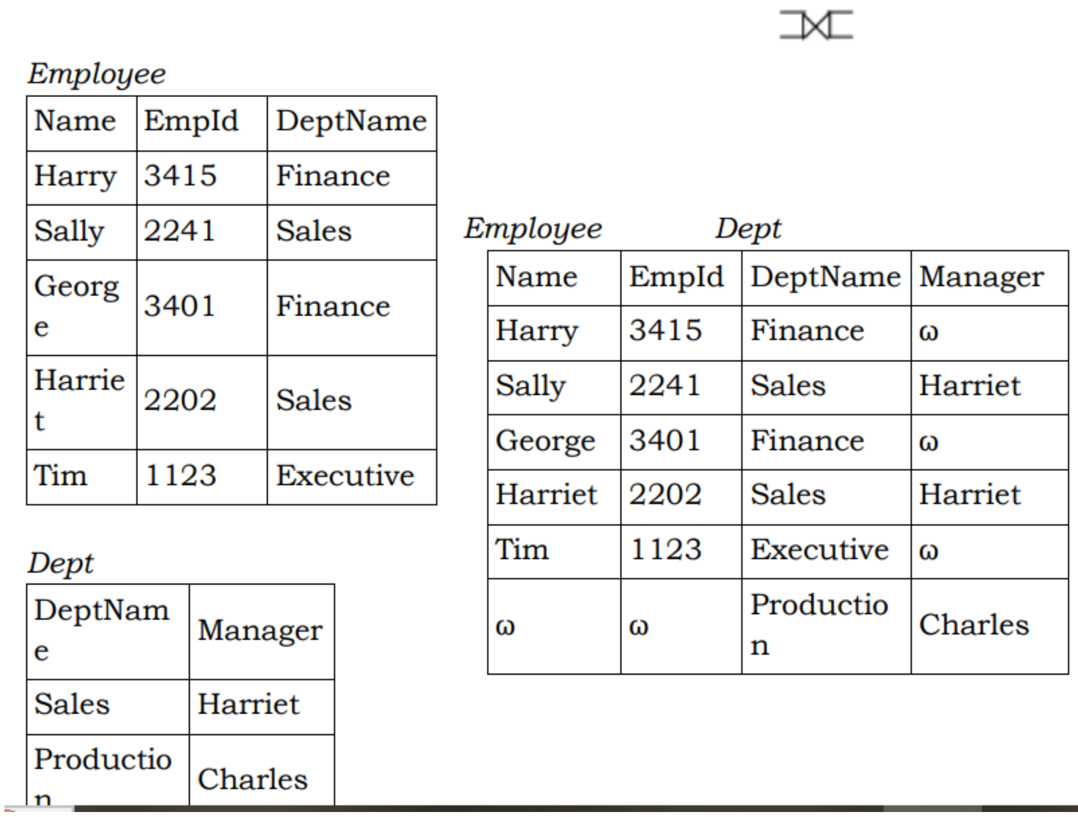
*Employee* *Dept*

| Name | EmpId | DeptName | Manager |
|---|---|---|---|
| Sally | 2241 | Sales | Harriet |
| Harriet | 2202 | Sales | Harriet |
| ω | Ω | Production | Charles |

**Full outer join ( ):**

The outer join or full outer join in effectcombines the results of the left and right outer joins.

The full outer join is written as ⟗ where R and S are relations.
The result of the full outer join is the set of all combinations of tuples in R and S that are equal on their common attribute names, in addition to tuples in S that have no matching tuples in R and tuples in R that have no matching tuples in S in their common attribute names.
**Example**: Consider the tables Employee and Dept and their full outer join:



*Employee*

| Name | EmpId | DeptName |
|---|---|---|
| Harry | 3415 | Finance |
| Sally | 2241 | Sales |
| Georg e | 3401 | Finance |
| Harrie t | 2202 | Sales |
| Tim | 1123 | Executive |

*Dept*

| DeptNam e | Manager |
|---|---|
| Sales | Harriet |
| Productio n | Charles |

*Employee* ⟗ *Dept*

| Name | EmpId | DeptName | Manager |
|---|---|---|---|
| Harry | 3415 | Finance | ω |
| Sally | 2241 | Sales | Harriet |
| George | 3401 | Finance | ω |
| Harriet | 2202 | Sales | Harriet |
| Tim | 1123 | Executive | ω |
| ω | ω | Productio n | Charles |

1. **Define Relational Database. (2m)**
2. **What are the relational operators? (2m)**
3. **Define Cartesian product. (2m)**
4. **Give the usage of the rename operation with an example. (2m)**

## 1.5 SQL FUNDAMENTALS:

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in relational database.

SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language.

Also, they are using different dialects, such as:

- MS SQL Server using T-SQL,
- Oracle using PL/SQL,
- MS Access version of SQL is called JET SQL (native format) etc.

**Data types**:

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

- **char(n)**: A fixed-length character string with user-specified length n. The full form,character, can be used instead.
- **varchar(n)**: A variable-length character string with user-specified maximumlength n. The full form, character varying, is equivalent.
- **int:** An integer represents a signed integer decimal or binary.
- **smallint:** A small integer is a machine-independent subset of the integer type.
- **numeric(p,d):** A fixed-point number with user-specified precision. The number consists of p digits (plus a sign), and d of the p digits are to the right of the decimal point. Thus, numeric(3,1) allows 44.5 to be stored exactly, but neither444.5 or 0.32 can be stored exactly in a field of this type.
- **real, double precision**: Floating-point and double-precision floating-point numbers with machine-dependent precision.
- **float(n)**: A floating-point number, with precision of at least n digits.
- **date**: A calendar date containing a year, month and day of the month.
- **time:** The time of day, in hours, minutes and seconds.

**DDL:**

**DDL** stands for **Data-Definition language**. The SQL DDL provides commands for defining relation schemas, deleting relations, and modifying relation schemas.

**1. COMMAND NAME**: create

**DESCRIPTION**: It is used to create objects in the database.

The types of objects aretables, indexes, users and databases.

**SYNTAX:** create table <tablename>(columnname1 datatype(size),columnname2 datatype(size),...);

**EXAMPLE:** SQL>create table emp (empno number(4), ename varchar(10), designation varchar(10),salary number(8,2));

**2. COMMAND NAME:** desc

**DESCRIPTION:** It is used to describe the table.

**SYNTAX:** desc <table name>;

**EXAMPLE:** SQL>desc emp;

**3. COMMAND NAME**: drop

**DESCRIPTION:** It is used to delete the object from the database.

**SYNTAX:** drop table <tablename>;

**EXAMPLE:** SQL>drop table emp;

**4. COMMAND NAME:** alter

**DESCRIPTION:** It is used to alter the structure of database objects.

**SYNTAX**: a. alter table <tablename> add(new columnname1 datatype(size), new columnname2 datatype(size),...);

b. alter table <tablename> modify(column definition);

**EXAMPLE:** a. SQL>alter table emp add (addrdess varchar(50));

b. SQL>alter table emp modify(empno number(7));

**5. COMMAND NAME:** truncate

**DESCRIPTION:** It is used to delete the records but retain the structure of database objects.

**SYNTAX:** truncate table <tablename>;

**EXAMPLE:** SQL>truncate table emp;

**DML:**

**DML** stands for **Data-Manipulation language**. The SQL DML provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.

**1. COMMAND NAME**: insert

**DESCRIPTION:** It is used to insert a new record in the database.

**SYNTAX:** insert into <tablename>values(a list of data values);

**EXAMPLE:** SQL>insert into emp values(50,'ABC','manager',10000);

**2. COMMAND NAME**: update

**DESCRIPTION:** This allows the user to update the particular column value.

**SYNTAX:** update <tablename> set <col1=value> where<condition>;

**EXAMPLE:** SQL>update emp set empno=100 where designation= 'manager';

**3.COMMAND NAME**: delete

**DESCRIPTION:** This allows you to delete the particular column values.

**SYNTAX:** delete from <tablename> where <condition>;

**EXAMPLE:** SQL>delete from emp where empno=100;

**4. COMMAND NAME**: select

**DESCRIPTION:** This command is used to select the record from the table or view.

**SYNTAX:** select<col1…….colN>/* from <tablename>;

**EXAMPLE:** SQL>select * from emp;

**DCL:**

**Data Control Language(DCL)** is used to control privilege in Database

DCL defines two commands,

- **Grant :** Gives user access privileges to database.
- **Revoke :** Take back permissions from user.

**1. COMMAND NAME**: grant

**DESCRIPTION:** It is used to provide access or privileges on the database objects to the users.

**SYNTAX:** GRANT <object privileges>
ON <object_name>
TO <User_Name>
[WITH GRANT OPTION];

object privileges -> is the access right or privilege granted to the user

object_name -> is the name of the database object like table, view, stored proc and sequence.

User_Name -> is the name of the user whom an
access right is being granted.
WITH GRANT OPTION -> allows a user to grant access
rights to other users.

**EXAMPLE:** SQL>grant create, update,drop,select on table to ABC;

**2. COMMAND NAME:** revoke

**DESCRIPTION:** It is used to remove access rights or privileges on the database objects to the users.

**SYNTAX:** REVOKE <object privileges> ON <object_name> FROM <User_Name>

**EXAMPLE:** SQL>revoke create, update,drop,select on table from ABC;

**TCL:**

**Transaction Control Language(TCL)** commands are used to manage transactions in database.These are used to manage the changes made by DML statements. It also allows statements to be grouped together into logical transactions.

**1. COMMAND NAME:** commit

**DESCRIPTION:** Commit command is used to permanently save any transaaction into database.

- Commit command is used to end a transaction.
- Commit command also erases all save points in the
transaction thus releasing the transaction locks.

**SYNTAX**: commit;

**EXAMPLE:** SQL>commit;

**2. COMMAND NAME:** savepoint

**DESCRIPTION**: The savepoint makes names and masks the current point in the processing of transaction.

- They are used to identify a point in a transaction to
which we can later rollback.
- Thus savepoint is used in conjuction with Rollback,
to rollback portions of the current transaction.
- Maximum number of transaction is 5.

**SYNTAX:** savepoint savepoint_name;

**EXAMPLE:** save point table1;

**3. COMMAND NAME:** rollback

**DESCRIPTION:** It is used to undo the work done in the current
transaction.

**SYNTAX:**rollback [work] [to savepoint];

**EXAMPLE:** rollback;

1. **What are different access types in DML? (2m)**

- **Explain the following:**

i.Data Definition Language. ii.Data Manipulation Language. iii.Data control Language. iv.Views **(R) (APRIL 2008) (NOV 2014) (15m)**

- State and explain the command DDL,DML,DCL with suitable example. **(Nov/Dec-2017) (15m)**
- Consider the following relations for a company database Application**.(AP)(MAY 2009) (15m)**

Employee (**Eno**, Name, Sex, DOB, Doj, Designation, Basic_Pay, Dept_No)
Department (**DeptNo**,Name)
Project (**ProjNo**, Name, Dept_No)
Works for (**Eno,ProjNo,Date,** Hours)

The attributes specified for each relation is self-explanatory. However the business rules are stated as follows. A department can control a project. An employee can work on any number of projects on a day. However an employee cannot work more than once on a project he/she worked on that day. The primary key are underlined.

(i). Identify the forging keys, Develop DDL to implement the above schema.

(ii)Develop an SQL query to list the department number and the number of Employee in each department.

(iii)Develop a View that will keep track of the department number, the number of employees in the department, and the total basic pay expenditure for each department.

(iv)Develop an SQL query to list the details of employees who have worked in more than three projects on a day.

- Create an EMPLOYEE table and write the steps for various functions in an SQL Like add, update, delete, save and join the various attributes of an EMPLOYEE Table. **(C)(MAY 2007) (15m)**

## 1.6 ADVANCED SQL FEATURES:

### Tuple Variables

Tuple variables are defined in the from clause via the use of the **as clause.**

For example, find the customer names and their loan numbers for all customers

having a loan at some branch in the banking database.

SQL>**select** B.customer_name, B.loan_no, L.amount

**from** borrower **as** B, loan as L **where**

L. loan_no = B. loan_no;

### String Operations

SQL includes a string matching operator for comparisons on character strings.

Patterns are described using 2 special characters.

**Percent (%):** The % character, matches any substring.

**Underscore (-):** The-character matches any character.

**Example: Find the names of customers where the 1st 2 characters are 'Ba'.**

SQL>**select** customer_name **from** customer **where**

customer_name **like** 'Ba%';

Find the names of customer where the 2nd character is 'n' or 'a'.

SQL>**select** customer_name **from** customer **where**

customer_name **like** '_n%' or '_a%';

### Order by Clause

The order by clause causes the tuples in the result of a query to appear in sorted

order.

SQL>select *from employee order by salary;

SQL>select * from employee order by salary desc, eid asc;

### Aggregate Functions

Aggregate functions are functions that take a collection of values as input and return a single value as output.

SQL offers 5 built in aggregate funtions:

avg - average value

min - minimum value

max - maximum value

sum - sum of values

count - number of values.

**Examples**

1. Find the average account balance at the perryridge branch:

SQL>select avg (balance) from account where
branch-name = “Perryridge”;

2. Find the number of tuples in the customer relation

SQL>select count (*) from customer;

**Aggregate functions with group by clause**

Group by clause is used to group the rows based on certain criteria. Group by is used in conjunction with aggregate functions like sum, avg, min, max, count, etc.

**Example:** Find the average account balance at each branch.

SQL>select branch_name, avg (balance) from account
group by branch_name;

**Aggregate functions with having clause**

Find the name of all branches where the average account balance is more than $2,000.

SQL>select branch_name, avg (balance) from account
group by branch_name having avg (balance) > 2000;

**Null Values**

SQL allows the use of null values to indicate absence of information about the value of an attribute.

Null signifies an unknown value or that a value does not exist.

The predicate is null can be used to check for null values.

**Example:**

Find all loan numbers which appear in the loan relation with null values for amount.

SQL>select loan_no from loan where amount is null;

**PROCEDURE:**

A procedure is a block that can take parameters (sometimes referred to as arguments) and be invoked.

Procedures promote reusability and maintainability. Once validated, they can be used in number of applications. If the definition changes, only the procedure are affected, this greatly simplifies maintenance.

**Modularized program development:**

- Group logically related statements within blocks.
- Nest sub-blocks inside larger blocks to build powerful programs.

- Break down a complex problem into a set of manageable well defined logical modules and implement the modules with blocks.

**KEYWORDS AND THEIR PURPOSES:**

**REPLACE:** It recreates the procedure if it already exists.

**PROCEDURE:** It is the name of the procedure to be created.

**ARGUMENT:** It is the name of the argument to the procedure. Parenthesis can be omitted if no arguments are present.

**IN:** Specifies that a value for the argument must be specified when calling the procedure ie., used to pass values to a sub-program. This is the default parameter.

**OUT:** Specifies that the procedure passes a value for this argument back to it"s calling environment after execution ie. used to return values to a caller of the sub-program.

**INOUT:** Specifies that a value for the argument must be specified when calling the procedure and that procedure passes a value for this argument back to it"s calling environment after execution.

**RETURN:** It is the data type of the function"s return value because every function must return a value, this clause is required.

**SYNTAX :**

```
create or replace procedure <procedure name> (argument {in,out,inout} datatype )
{is,as} variable declaration;
constant declaration;
begin
PL/SQL subprogram body;
exception
exception PL/SQL block;
end;
```

**EXECUTION:**

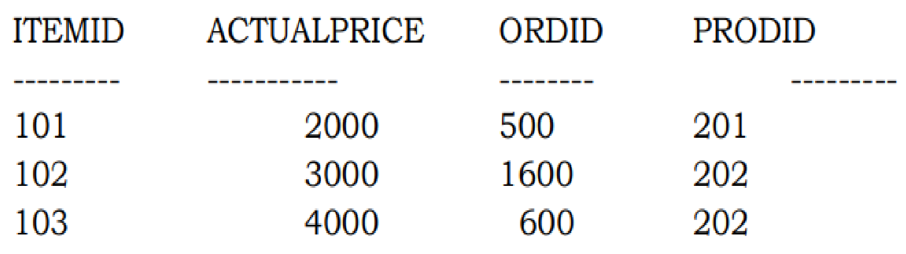
Setting serveroutput on

```
SQL> set serveroutput on
```

**PROGRAM:**

Tables used:

```
SQL> select * from ititems;
```



```
ITEMID     ACTUALPRICE    ORDID     PRODID
---------  -----------    --------       ---------
101               2000    500       201
102               3000    1600      202
103               4000      600     202
```

**PROCEDURE FOR _IN‘ PARAMETER – CREATION, EXECUTION**

```
SQL> set serveroutput on;
SQL> create procedure yyy (a IN number) is price number;
begin
select actualprice into price from ititems where itemid=a;
dbms_output.put_line('Actual price is ' || price);
if price is null then
dbms_output.put_line('price is null');
end if;
end;
/
Procedure created.
SQL> exec yyy(103);
Actual price is 4000
PL/SQL procedure successfully completed.
```

**PROCEDURE FOR _OUT‘ PARAMETER – CREATION, EXECUTION**

```
SQL> set serveroutput on;
SQL> create procedure zzz (a in number, b out number) is identity number;
begin
select ordid into identity from ititems where itemid=a;
if identity<1000 then
b:=100;
end if;
end;
/
Procedure created.
SQL> declare
a number;
```

```
b number;
begin
zzz(101,b);
dbms_output.put_line('The value of b is '|| b);
end;
/
The value of b is 100
PL/SQL procedure successfully completed.
SQL> create procedure itit ( a in out number) is
begin
a:=a+1;
end;
/
Procedure created.
SQL> declare
a number:=7;
begin
itit(a);
dbms_output.put_line('The updated value is '||a);
end;
/
The updated value is 8
PL/SQL procedure successfully completed.
```

**FUNCTIONS:**

A PL/SQL function is same as a procedure except that it returns a value.

**SYNTAX:**

```
create or replace function <function name> (argument in datatype,……) return datatype
{is,as}
variable declaration;
constant declaration;
begin
PL/SQL subprogram body;
exception
exception PL/SQL block; end;
```

**PROCEDURE:**

1. Start.
2. Create the table with essential attributes.
3. Initialize the functions to carry out the searching procedure.
4. Frame the searching procedure for both positive and negative searching.
5. Execute the function for both positive and negative result.
6. Stop.

```
set serveroutput on
SQL>set serveroutput on
```

**IMPLEMENTATION OF FACTORIAL USING FUNCTION:**

```
SQL>create function fnfact(n number) return number is b number;
begin
b:=1;
for i in 1..n
loop
b:=b*i;
end loop;
return b;
end;
/
SQL>declare
n number:=&n;
y number;
begin
y:=fnfact(n);
dbms_output.put_line(y);
end;
/
```

**PROGRAM TO FIND THE ADDRESS FROM THE PHONEBOOK:**

```
1)SQL>create table phonebook(phone_no number(6) primary key,username varchar
(20),doorno varchar2(20),street varchar2(20),place varchar2(30),pincode char(6));
Table created
```

```
SQL>insert into phonebook values(20132,'vijay','120/5D','Sweet Street','NGO
colony','650000');
1 row created
SQL>insert into phonebook values (29457,'vasanth','95/RS','Daniel Street','POG
colony','650002');
1 row created
SQL>select *from phonebook;
2)SQL>create or replace function findaddress(phone in number)return varchar2 as
address varchar2(100);
begin
select username||','||doorno||','||street||','||place||','||pincode into address from
phonebook where phone_no=phone;
return address;
exception
when no_data_found then return 'address not found';
end;
/
Function created
3) SQL>declare
address varchar2(100);
begin
address:=findaddress(20132);
dbms_output.put_line(address);
end;
/
SQL>declare
address varchar2(100);
begin
address:=findaddress(23556);
dbms_output.put_line(address);
end;
/
```

**TRIGGERS:**

A data base trigger is stored pl/sql. Program unit associated with a specific database table or view. The code in the trigger defines the action the database needs to perform whenever some database manipulation (Insert, Update, and Delete) takes place.

A database trigger has three parts,

1. Trigger Event.
2. A Trigger Constraint.
3. Trigger Action.

**SYNTAX:**

The syntax for creating a trigger is:

```
CREATE[or Replace] TRIGGER trigger_name
{Before|After|Instead of}
{Insert [or]|Update[or]|Delete}
[of col_name]
ON table_name
[Referencing old As o new As n]
[For each row]
DECLARE
declaration_statements
BEGIN
Executable_statements
EXCEPTION
Exception_handling_statements
END;
```

**PROCEDURE:**

1.Start.
2.Initialize the trigger with specific table id.
3.Specify the operations (Update, Delete, Insert) for which trigger has to be executed.
4.Execute the trigger procedure for both before and after sequences.
5.Carry out the operation on the table to check for trigger execution.
6.Stop.

**EXECUTION:**

**1) TRIGGER AFTER UPDATE**

```
Create or replace trigger t
After
Insert OR
Update of salary, dept_id OR
Delete
ON employees
for each row
Begin
if inserting then
dbms_output.put_line('Inserting');
elsif Updating('Salary') then
dbms_output.put_line('Updating Salary');
elsif Updating('Dept_id') then
dbms_output.put_line('Updating Department ID');
elsif deleting then
dbms_output.put_line('Deleting');
End if;
End;
/
1)SQL>update employees set salary=1700 where dept_id=1;
2)SQL>insert into employees values(15000,101);
3)SQL>insert into employees values(1000,2);
4)SQL>delete from employees where dept_id=2;
```

**2) SIMPLE TRIGGER THAT DOES NOT ALLOW INSERT UPDATE AND DELETE OPERATIONS ON THE TABLE**

Table used:

```
SQL> select * from itempls;
ENAME EID SALARY
---------- --------- ---------
xxx 11 10000
yyy 12 10500
zzz 13 15500
```

**Trigger**:

```
SQL> create trigger ittrigg before insert or update or delete on itempls for each
row
begin
raise_application_error(-20010,'You cannot do manipulation');
end;
/
Trigger created.
1)SQL> insert into itempls values('aaa',14,34000);
2)SQL>delete from itempls where ename='xxx';
```

1. **What is a trigger? (2m)**

- **What are aggregate functions? Explain five built-in aggregate functions. (U) (MAY 2008) (15m)**
- **Explain the use of trigger with your own example. (U) (MAY 2010) (15m)**

## 1.7 EMBEDDED SQL:

Embedded SQL is a method of combining the power of a programming language and the database.

Embedded SQL statements are processed by a special SQL precompiler. The embedded SQL statements are parsed by an embedded SQL.

The output from the preprocessor is then compiled by the host compiler. This allows programmers to embed SQL statements in programs written in any number of languages such as: C Programming language family, COBOL, FORTRAN and Java. The

SQL standard defines embeddings of SQL in a variety of programming languages such as C, Java, and COBOL.

A language to which SQL queries are embedded is referred to as a host language, and the SQL structures permitted in the host language comprise embedded SQL.

The basic form of these languages follows that of the System are embedding of SQL into PL/I. EXEC SQL statement is used to identify embedded SQL request to the preprocessor EXEC SQL <embedded SQL statement > END_EXEC

The statement causes the query to be evaluated EXEC SQL .. END_EXEC

The fetch statement causes the values of one tuple in the query result to be placed on host language variables. For example, we need one variable to hold the customer name value and another to hold the customer city value. Suppose that those variables are cn and cc the statement is

EXEC SQL fetch c into :cn, :cc END_EXEC

Repeated calls to fetch get successive tuples in the query result .

A variable called SQLSTATE in the SQL communication area (SQLCA) gets set to ‗02000‘ to indicate no more data is available.

The close statement causes the database system to delete the temporary relation that holds the result of the query. EXEC SQL close c END_EXEC

Example:

```
/*variable declaration in Language C*/
int loop;
EXEC SQL BEGIN DECLARE SECTION;
varchar dname[16],fname[16],...;
char ssn[10],bdate[11],...;
int dno,dnumber,SQLCODE,....;
EXEC SQL END DECLARE SECTION;
/*conditional and looping statements in Language C*/
loop=1;
while(loop){
prompt("Enter SSN:",ssn);
EXEC SQL
select FNAME,LNAME,ADDRESS,SALARY
into :fname,:lname,:address,:salary
from EMPLOYEE where SSN==:ssn;
if(SQLCODE==0)printf(fname,....);
else printf("SSN does not exist:",ssn);
prompt("More SSN?(1=yes,0=no):",loop);
END-EXEC
}
```

1. **What is the use of embedded SQL? (2m)**

- **What is Embedded SQL? Give example. (R)(NOV 2016) (15m)**

## 1.7 DYNAMIC SQL

Dynamic SQL allows the program to construct an SQL query as a character string at runtime, submit the query, and then retrieve the result into program variables a tuple at a time. The dynamic SQL component of SQL allows programs to construct and submit SQL queries at runtime.

1. **Discuss about dynamic SQL. (2m)**

### 1.7.1 DIFFERENCE BETWEEN EMBEDDED SQL AND DYNAMIC SQL:

| S.No. | Static (embedded) SQL | Dynamic (interactive) SQL |
|---|---|---|
| 1 | In static SQL how database will be accessed is predetermined in the embedded SQL statement. | In dynamic SQL, how database will be accessed is determined at run time. |
| 2 | It is more swift and efficient. | It is less swift and efficient. |
| 3 | SQL statements are compiled at compile time. | SQL statements are compiled at run time. |
| 4 | Parsing, validation, optimization, and generation of application plan are done at compile time | Parsing, validation, optimization, and generation of application plan are done at run time. |
| 5 | It is generally used for situations where data is distributed uniformly. | It is generally used for situations where data is distributed non-uniformly. |
| 6 | EXECUTE IMMEDIATE, EXECUTE and PREPARE statements are not used | EXECUTE IMMEDIATE, EXECUTE and PREPARE statements are used. |
| 7 | It is less flexible. | It is more flexible. |