Answer Key	

Internal Assessment Test 1 – JULY 2022									
Sub:	**DATABASE MANAGEMENT SYSTEMS**							**Sub Code:**	**20MCA21**
Date:	27-07-2022	Duration:	90 mins	Max Marks:	50	**Sem:**	II	**Branch:**	**MCA**

CMR
INSTITUTE OF
TECHNOLOGY

USN

Internal Assessment Test 1 – July 2022

Sub:	**Database Management System**							**Sub Code:**	**20MCA21**
Date:	**27-07-2022**	**Duration:**	**90 min's**	**Max Marks:**	**50**	**Sem:**	**II**	**Branch:**	**MCA**

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

	PART I	MARKS	OBE	
			CO	RBT
1	Define DBMS. Explain its characteristics. **OR**	[10]	CO1	L1
2	Explain with example advantages and disadvantages of DBMS.	[10]	CO1	L1
3	**PART II** Explain with necessary diagram the Database Component Modules. **OR**	[10]	CO1	L1
4	Write in detail the classification of Database Management Systems.	[10]	CO1	L1

5	**PART III** Explain about 2 tier and 3 tier client/server architecture with necessary diagram. **OR**	[10]	CO1	L1
6	Explain SQL data definition and data types in brief and explain different CREATE and ALTER command.	[10]	CO4	L2
7	**PART IV** What are key constraints? Explain the key constraints on relational database with suitable example. **OR**	[10]	CO4	L2

8	Define data independence and mapping. Explain the three-schema architecture with diagram.	[10]	CO1	L1
9	**PART V** Explain different insert, update and delete commands used in SQL with example. **OR**	[10]	CO4	L2
10	Explain different DDL and DML statements used in SQL.	[10]	CO4	L2

1. **Define DBMS. Explain its characteristics.**
   Definition:

A **database management system** (DBMS) is a collection of programs enabling users to create and maintain a database.

More specifically, a DBMS is a *general purpose* software system facilitating each of the following (with respect to a database):

- **definition:** specifying data types (and other constraints to which the data must conform) and data organization
- **construction:** the process of storing the data on some medium (e.g., magnetic disk) that is controlled by the DBMS
- **manipulation:** querying, updating, report generation
- **sharing:** allowing multiple users and programs to access the database "simultaneously"
- **system protection:** preventing database from becoming corrupted when hardware or software failures occur
- **security protection:** preventing unauthorized or malicious access to database.

Characteristics of DBMS:

Main Characteristics of database approach:

1. **Self-Description:** A database system includes —in addition to the data stored that is of relevance to the organization— a complete definition/description of the database's structure and constraints. This **meta-data** (i.e., data about data) is stored in the so-called **system catalog**, which contains a description of the structure of each file, the type and storage format of each field, and the various constraints on the data (i.e., conditions that the data must satisfy).

   See Figures 1.1 and 1.3.

   The system catalog is used not only by users (e.g., who need to know the names of tables and attributes, and sometimes data type information and other things), but also by the DBMS software, which certainly needs to "know" how the data is structured/organized in order to interpret it in a manner consistent with that structure. Recall that a DBMS is *general purpose*, as opposed to being a specific database application. Hence, the structure of the data cannot be "hard-coded" in its programs (such as is the case in typical *file processing* approaches), but rather must be treated as a "parameter" in some sense.

2. **Insulation between Programs and Data; Data Abstraction:**

   Program-Data Independence: In traditional file processing, the structure of the data files accessed by an application is "hard-coded" in its source code. (E.g., Consider a file descriptor in a COBOL program: it gives a detailed description of the layout of the records in a file by describing, for each field, how many bytes it occupies.)

   If, for some reason, we decide to change the structure of the data (e.g., by adding the first two digits to the YEAR field, in order to make the program Y2K compliant!), **every** application in which a description of that file's structure is hard-coded must be changed!

   In contrast, DBMS access programs, in most cases, do not require such changes, because the structure of the data is described (in the system catalog) separately from the programs that access it and those programs consult the catalog in order to ascertain the structure of the data (i.e., providing a means by which to determine boundaries between records and between fields within records) so that they interpret that data properly.

   In other words, the DBMS provides a conceptual or logical view of the data to application programs, so that the underlying implementation may be changed without the programs being modified. (This is referred to as *program-data independence*.)

   Also, which access paths (e.g., indexes) exist are listed in the catalog, helping the DBMS to determine the most efficient way to search for items in response to a query.

   Note: In fairness to COBOL, it should be pointed out that it has a COPY feature that allows different application programs to make use of the same file descriptor stored in a "library". This provides some degree of program-data independence, but not nearly as much as a good DBMS does. *End of note.*

3. **Multiple Views of Data:** Different users (e.g., in different departments of an organization) have different "views" or perspectives on the database. For example, from the point of view of a Bursar's Office employee, student data does not include anything about which courses were taken or which grades were earned. (This is an example of a **subset** view.)

As another example, a Registrar's Office employee might think that GPA is a field of data in each student's record. In reality, the underlying database might calculate that value each time it is needed. This is called **virtual** (or **derived**) data.

A view designed for an academic advisor might give the appearance that the data is structured to point out the prerequisites of each course.

4. **Data Sharing and Multi-user Transaction Processing:** As you learned about (or will) in the OS course, the simultaneous access of computer resources by multiple users/processes is a major source of complexity. The same is true for multi-user DBMS's.

Arising from this is the need for **concurrency control**, which is supposed to ensure that several users trying to update the same data do so in a "controlled" manner so that the results of the updates are as though they were done in some sequential order (rather than interleaved, which could result in data being incorrect).

This gives rise to the concept of a **transaction**, which is a process that makes one or more accesses to a database and which must have the appearance of executing in *isolation* from all other transactions (even ones that access the same data at the "same time") and of being *atomic* (in the sense that, if the system crashes in the middle of its execution, the database contents must be as though it did not execute at all).

Applications such as **airline reservation systems** are known as **online transaction processing** applications.

2. **Explain with example advantages and disadvantages of DBMS.**

i. Controlling Redundancy • redundancy in storing the same data multiple times leads to several problems o there is the need to perform a single logical update o storage space is wasted when the same data is stored repeatedly o Files that represent the same data may become inconsistent

ii. Restricting Unauthorized Access • A DBMS should provide a security and authorization subsystem • most users will not be authorized to access all information in the database

iii. Providing Persistent Storage for Program Objects • Databases can be used to provide persistent storage for program objects and data structures

iv. Providing Storage Structures and Search Techniques for Efficient Query Processing • the DBMS must provide specialized data structures and search techniques to speed up disk search for the desired records – index • The DBMS often has a buffering or caching module that maintains parts of the database in main memory buffers • The query processing and optimization module of the DBMS is responsible for choosing an efficient query execution plan for each query based on the existing storage structures.

v. Providing Backup and Recovery
responsible for recovery • the database is restored to the state it was in before the transaction started executing

vi. Providing Multiple User Interfaces • a DBMS should provide a variety of user interfaces o query languages for casual users, o programming language interfaces for application programmers, o forms and command codes for parametric users, and o menu-driven interfaces and natural language interfaces for standalone users

vii. Representing Complex Relationships among Data • A DBMS must have the capability to represent a variety of complex relationships among the data, to define new relationships as they arise, and to retrieve and update related data easily and efficientl

viii. Enforcing Integrity Constraints • The simplest type of integrity constraint involves specifying a data type for each data item • referential integrity constraint: specifying that a record in one file must be related to records in other files • Primary key constraint: uniqueness on data item values.

ix. Permitting inferencing and Actions Using Rules: A trigger is a form of a rule activated by updates to the table, which results in performing some additional operations to some other tables, sending messages, and so on.

x. Additional Implications of Using the Database Approach
Potential for Enforcing Standards: The DBA can enforce standards in a centralized database environment o Reduced Application Development Time: Development time using a DBMS is estimated to be one-

sixth to one-fourth of that for a traditional file system o Flexibility: Modern DBMSs allow certain types of evolutionary changes to the structure of the database without affecting the stored data and the existing application programs o Availability of Up-to-Date Information: As soon as one user's update is applied to the database, all other users can immediately see this update o Economies of Scale: reduce the wasteful overlap activities thereby reducing the overall costs of operation and management.

Disadvantages of DBMS:

In spite of the advantages of using a DBMS, there are a few situations in which a DBMS may involve unnecessary overhead costs that would not be incurred in traditional file processing. The overhead costs of using a DBMS are due to the following:

i. High initial investment in hardware, software, and training

ii. The generality that a DBMS provides for defining and processing data

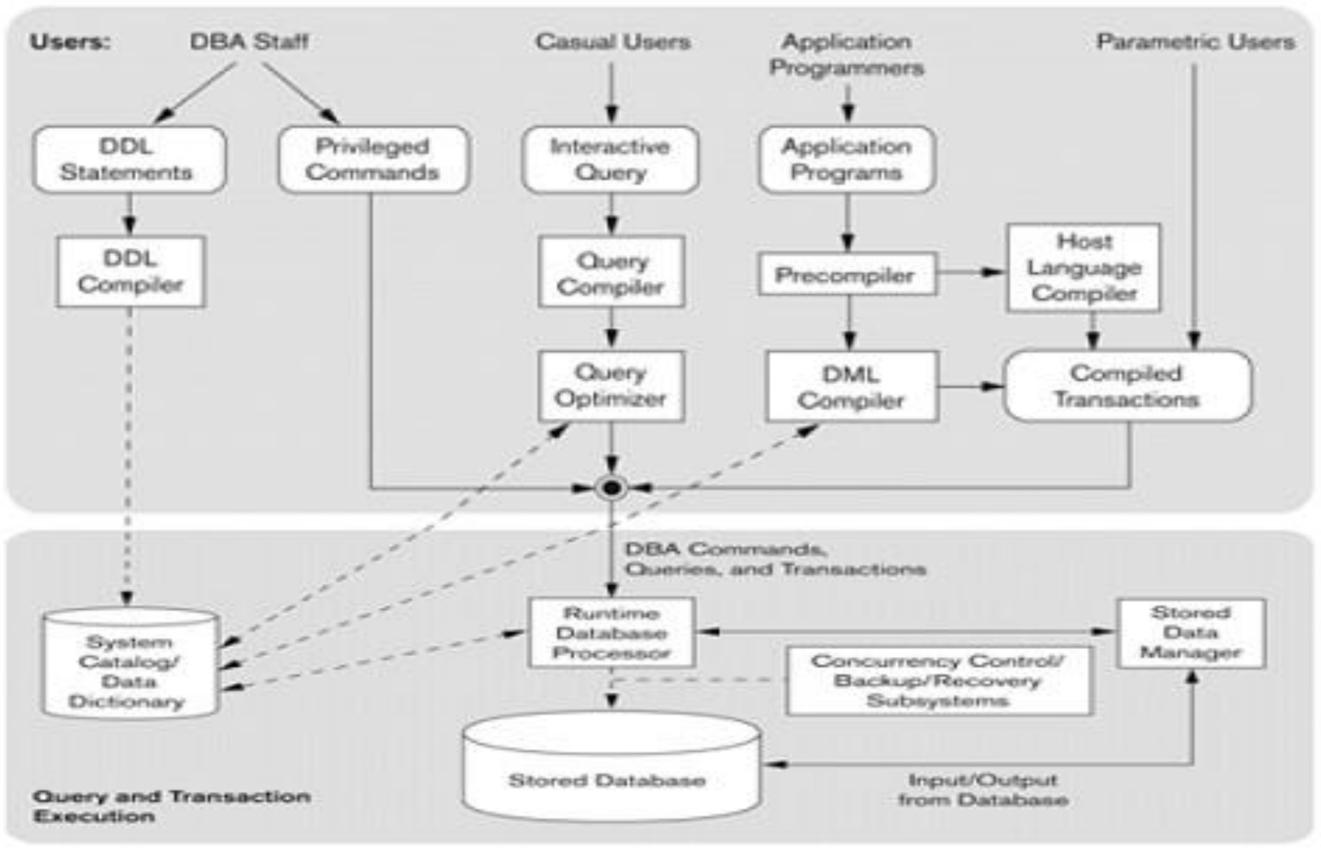
iii. Overhead for providing security, concurrency control, recovery, and integrity functions

iv. No multiple-user access to data

Certain industries and applications have elected not to use general-purpose DBMSs. For example, many computer-aided design (CAD) tools used by mechanical and civil engineers have proprietary file and data management software that is geared for the internal manipulations of drawings and 3D objects. Similarly, communication and switching systems designed by companies like AT&T were early manifestations of database software that was made to run very fast with hierarchically organized data for quick access and routing of calls. Similarly, GIS implementations often implement their own data organization schemes for efficiently implementing functions related to processing maps, physical contours, lines, polygons, and so on. General-purpose DBMSs are inadequate for their purpose.

3. **Explain with necessary diagram the Database Component Modules.**

A DBMS is a complex software system • discuss the types of software components that constitute a DBMS and the types of computer system software with which the DBMS interacts • two parts: o The top part: various users and their interfaces o The lower part: storage of data and processing of transactions • Access to the disk is controlled primarily by the operating system (OS) • buffer management module to schedule disk read/write, because this has a considerable effect on performance • A higher-level stored data manager module: controls access to DBMS information that is stored on disk, whether it is part of the database or the catalog • DDL Statement: used by the DBA for defining the database and tuning it • DDL Compiler: processes schema definitions and stores descriptions of the schemas in the DBMS catalog.
Privileged commands: The commands which are exclusively used by the DBA • Interactive query: The interface by which the casual users and persons with occasional need for information from the database interact • Query compiler: Queries are parsed and validated for correctness of the query syntax • Query optimizer: Responsible for optimising the query and its execution.
Pre-compiler: Extracts DML commands from an application program written in a host programming language • DML compiler: Compilation of pre-compiled DMLC commands into object code for database access
Host language compiler: The rest of the program is sent to the host language compiler. • Compiled transactions: Canned transactions are executed repeatedly by parametric users, who simply supply the parameters to the transactions.



4. Write in detail the classification of Database Management Systems.

Several criteria are normally used to classify DBMSs. The first is the **data model** on which the DBMS is based. The main data model used in many current commercial DBMSs is the **relational data model**. The **object data model** has been implemented in some commercial systems but has not had widespread use. Many legacy applications still run on database systems based on the **hierarchical** and **network data models**. Examples of hierarchical DBMSs include IMS (IBM) and some other sys-tems like System 2K (SAS Inc.) and TDMS. IMS is still used at governmental and industrial installations, including hospitals and banks, although many of its users have converted to relational systems. The network data model was used by many vendors and the resulting products like IDMS (Cullinet—now Computer Associates), DMS 1100 (Univac—now Unisys), IMAGE (Hewlett-Packard), VAX-DBMS (Digital—then Compaq and now HP), and SUPRA (Cincom) still have a fol-lowing and their user groups have their own active organizations. If we add IBM's popular VSAM file system to these, we can easily say that a reasonable percentage of worldwide-computerized data is still in these so-called **legacy database systems**.

The relational DBMSs are evolving continuously, and, in particular, have been incorporating many of the concepts that were developed in object databases. This has led to a new class of DBMSs called **object-relational DBMS**s. We can categorize DBMSs based on the data model: relational, object, object-relational, hierarchical, network, and other.

More recently, some experimental DBMSs are based on the XML (eXtended Markup Language) model, which is a tree-structured (hierarchical) data model. These have been called **native XML DBMSs.** Several commercial relational DBMSs have added XML interfaces and storage to their products.

The second criterion used to classify DBMSs is the **number of users** supported by the system. **Single-user systems** support only one user at a time and are mostly used with PCs. **Multiuser systems**, which include the majority of DBMSs, support con-current multiple users.

The third criterion is the **number of sites** over which the database is distributed. A DBMS is **centralized** if the data is stored at a single computer site. A centralized DBMS can support multiple users, but the DBMS and the database reside totally at a single computer site. A **distributed** DBMS (DDBMS) can have the actual database and DBMS software distributed over many sites, connected by a computer network. **Homogeneous** DDBMSs use the same DBMS software at all the sites, whereas **heterogeneous** DDBMSs can use different DBMS software at each site. It is also possible to develop **middleware software** to access several autonomous preexisting databases stored under heterogeneousDBMSs. This leads to a **federated** DBMS (or **multidatabase system**), in which the participating DBMSs are loosely coupled and have a degree of local autonomy. Many DDBMSs use client-server architecture, as we described in Section 2.5.

The fourth criterion is cost. It is difficult to propose a classification of DBMSs based on cost. Today we have open source (free) DBMS products like MySQL and PostgreSQL that are supported by third-party vendors with additional services. The main RDBMS products are available as free examination 30-day copy versions as well as personal versions, which may cost under $100 and allow a fair amount of functionality. The giant systems are being sold in modular form with components to handle distribution, replication, parallel processing, mobile capability, and so on, and with a large number of parameters that must be defined for the configuration. Furthermore, they are sold in the form of licenses—site licenses allow unlimited use of the database system with any number of copies running at the customer site. Another type of license limits the number of concurrent users or the number of user seats at a location. Standalone single user versions of some systems like Microsoft Access are sold per copy or included in the overall configuration of a desktop or laptop. In addition, data warehousing and mining features, as well as support for additional data types, are made available at extra cost. It is possible to pay millions of dollars for the installation and maintenance of large database sys-tems annually.

We can also classify a DBMS on the basis of the **types of access path** options for storing files. One well-known family of DBMSs is based on inverted file structures. Finally, a DBMS can be **general purpose** or **special purpose**. When performance is a primary consideration, a special-purpose DBMS can be designed and built for a specific application; such a system cannot be used for other applications without major changes. Many airline reservations and telephone directory systems devel-oped in the past are special-purpose DBMSs. These fall into the category of **online transaction processing** (**OLTP**) systems, which must support a large number of concurrent transactions without imposing excessive delays.

Let us briefly elaborate on the main criterion for classifying DBMSs: the data model. The basic **relational data model** represents a database as a collection of tables, where each table can be stored as a separate file. The database in Figure 1.2 resem-bles a relational representation. Most relational databases use the high-level query language called SQL and support a limited form of user views. We discuss the rela-tional model and its languages and operations in Chapters 3 through 6, and tech-niques for programming relational applications in Chapters 13 and 14.

The **object data model** defines a database in terms of objects, their properties, and their operations. Objects with the same structure and behavior belong to a **class**, and classes are organized into **hierarchies** (or **acyclic graphs**). The operations of each class are specified in terms of predefined procedures called **methods**. Relational DBMSs have been extending their models to incorporate object database concepts and other capabilities; these systems are referred to as **object-relational** or **extended relational systems**. We discuss object databases and object-relational systems in Chapter 11.

The **XML model** has emerged as a standard for exchanging data over the Web, and has been used as a basis for implementing several prototype native XML systems. XML uses hierarchical tree structures. It combines database concepts with concepts from document representation models. Data is represented as elements; with the use of tags, data can be nested to create complex hierarchical structures. This model conceptually resembles the object model but uses different terminology. XML capabilities have been added to many commercial DBMS products. We present an overview of XML in Chapter 12.

Two older, historically important data models, now known as **legacy data models**, are the network and hierarchical models. The **network model** represents data as record types and also represents a limited type of 1:N relationship, called a **set type**. A 1:N, or one-to-many, relationship relates one instance of a record to many record instances using some pointer linking mechanism in these models. Figure 2.8 shows a network schema diagram for the database of Figure 2.1, where record types are shown as rectangles and set types are shown as labeled directed arrows.

The network model, also known as the CODASYL DBTG model, has an associated record-at-a-time language that must be embedded in a host programming lan-guage. The network DML was proposed in the 1971 Database Task Group (DBTG) Report as an extension of the COBOL language. It provides commands for locating records directly (e.g., FIND ANY <record-type> USING <field-list>, or FIND DUPLICATE <record-type> USING <field-list>). It has commands to support traversals within set-types (e.g., GET OWNER, GET {FIRST, NEXT, LAST} MEMBER WITHIN <set-type> WHERE <condition>). It also has commands to store new data

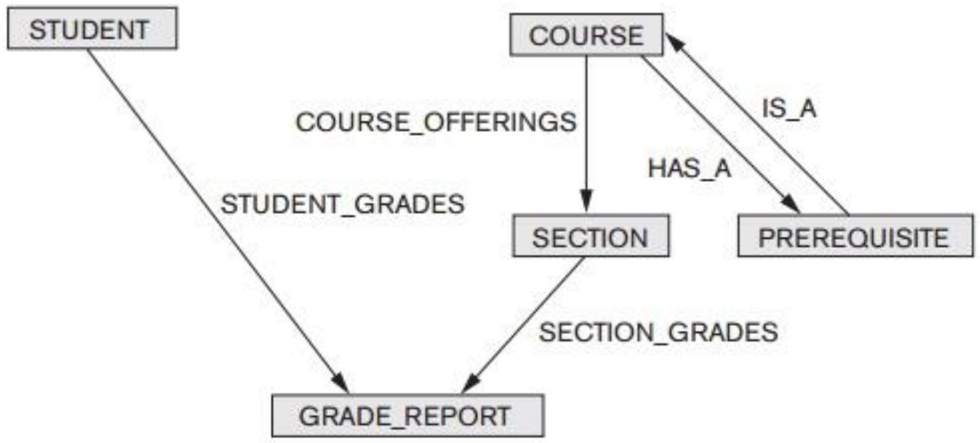
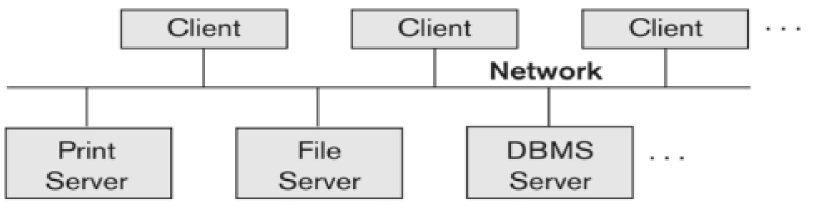


Figure 2.8
The schema of Figure 2.1 in network model notation.

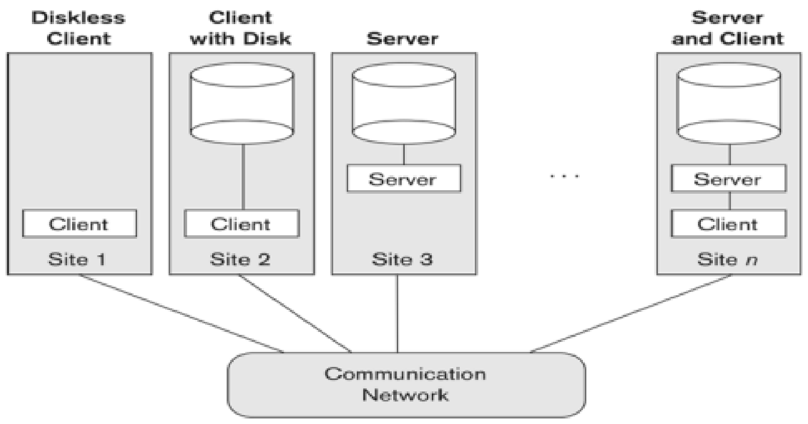
(e.g., STORE <record-type>) and to make it part of a set type (e.g., CONNECT <record-type> TO <set-type>). The language also handles many additional considerations, such as the currency of record types and set types, which are defined by the current position of the navigation process within the database. It is prominently used by IDMS, IMAGE, and SUPRA DBMSs today.

The **hierarchical model** represents data as hierarchical tree structures. Each hierar-chy represents a number of related records. There is no standard language for the hierarchical model. A popular hierarchical DML is DL/1 of the IMS system. It dominated the DBMS market for over 20 years between 1965 and 1985 and is still a widely used DBMS worldwide, holding a large percentage of data in governmental, health care, and banking and insurance databases. Its DML, called DL/1, was a de facto industry standard for a long time. DL/1 has commands to locate a record (e.g., GET { UNIQUE, NEXT} <record-type> WHERE <condition>). It has navigational facilities to navigate within hierarchies (e.g., GET NEXT WITHIN PARENT or GET {FIRST, NEXT} PATH <hierarchical-path-specification> WHERE <condition>). It has appropriate facilities to store and update records (e.g., INSERT <record-type>, REPLACE <record-type>). Currency issues during navigation are also handled with additional features in the language.

5. Explain about 2 tier and 3 tier client/server architecture with necessary diagram.



Logical two-tier client/server architecture



Physical two-tier client/server architecture

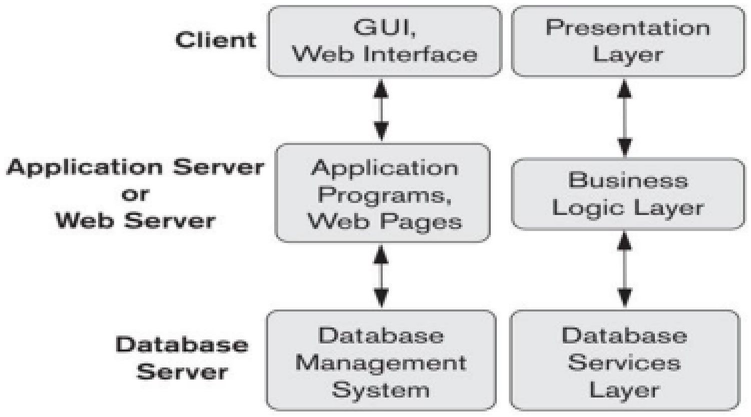
- Two main types of basic DBMS architectures were created on this underlying client/server framework: two-tier and three-tier
- Two-Tier Client/Server Architectures for DBMSs:

- the software components are distributed over two systems: client and server
  - client side: has the user interface and application programs
  - server side: the query and transaction functionality related to SQL processing
- The different approach to two-tier client/server architecture was taken by some object-oriented DBMSs
- the client level: handle the user interface; data dictionary functions; DBMS interactions with programming language compilers; global query optimization, concurrency control, and recovery across multiple servers; structuring of complex objects from the data in the buffers; and other such functions
- the server level: the DBMS software responsible for handling data storage on disk pages, local concurrency control and recovery, buffering and caching of disk pages, and other such functions
- Advantages: simplicity and seamless compatibility with existing systems

▪ Three-Tier and n-Tier Architectures for Web Applications

- The emergence of the Web changed the roles of clients and servers, leading to the three-tier architecture
- adds an intermediate layer (middle tier – application layer or web server) between the client and the database server
- runs the application programs and storing business rules (procedures or constraints) that are used to access data from the database server

- The presentation layer: displays information to the user and allows data entry.
- The business logic layer: handles intermediate rules and constraints before data is passed up to the user or down to the DBMS.
- The middle layer can also act as a Webserver, which retrieves query results from the database server and formats them into dynamic Web pages that are viewed by the Web browser at the client side
- The bottom layer includes all data management services.



6. **Explain SQL data definition and data types in brief and explain different CREATE and ALTER command.**

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 the database.DDL is a set of SQL commands used to create, modify, and delete database structures but not data. These commands are normally not used by a general user, who should be accessing the database via an application.

CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);
CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);
DROP TABLE ;
DROP TABLE EMPLOYEE;
ALTER TABLE table_name ADD column_name COLUMN-definition;
ALTER TABLE MODIFY(COLUMN DEFINITION....);

ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));

ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));

- **SQL Data Definition:**
- The set of relations in a database must be specified to the system by means of a data definition language (DDL).
- It allows specification of not only a set of relations, but also information about each relation, including:
- The schema for each relation.
- The types of values associated with each attribute.
- The integrity constraints.
- The set of indices to be maintained for each relation.
- The security and authorization information for each relation.
- The physical storage structure of each relation on disk
- Basic Types: The SQL standard supports a variety of built-in types, including:
  - Char(n): A fixed-length character string with user-specified length n
  - varchar(n): A variable-length character string with user-specified maximum length n
  - int: An integer (a finite subset of the integers that is machine dependent)
  - smallint: A small integer (a machine-dependent subset of the integer type)
  - numeric(p, d): A fixed-point number with user-specified precision.
  - 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

- The general form of the create table command is:
- create table r (A1 D1, A2 D2, . . . , An Dn,
  <integrity-constraint1>, . . . , <integrity-constraintk);
- **Data Types:**
- the SQL standard supports several data types relating to dates and times:
- date: A calendar date containing a (four-digit) year, month, and day of the month
- time: The time of day, in hours, minutes, and seconds
- timestamp: A combination of date and time
- Default Values
- SQL allows a default value to be specified for an attribute as illustrated by the following create table statement:
- Large-Object Types
- SQL provides large-object (LOB) data types for character data (clob) and binary data (blob).
  - book_review clob(10KB)
  - image blob(10MB)
  - movie blob(2GB)
- User-Defined Types
- SQL supports two forms of user-defined data types.

(i) Distinct types

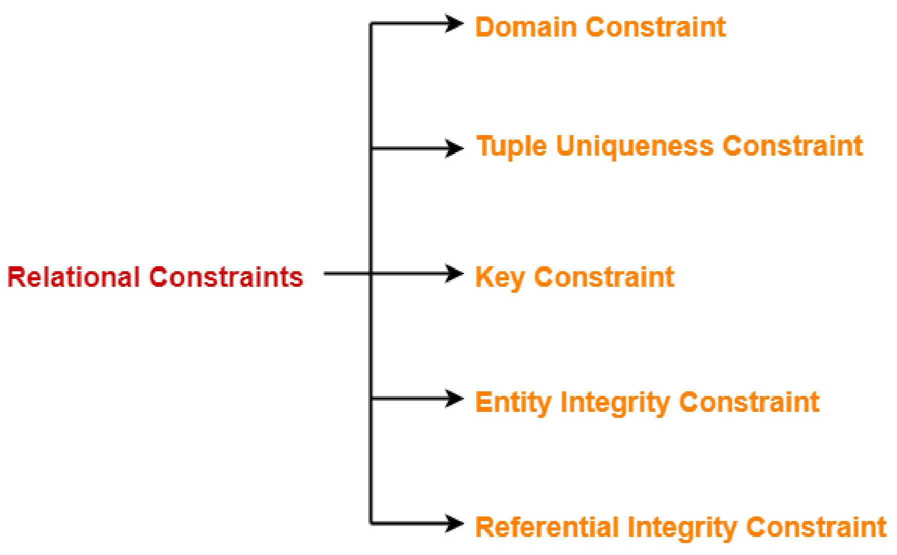
(ii) structured data types: allows the creation of complex data types with nested record structures, arrays, and multi-sets

(iii) Index Creation: An index on an attribute of a relation is a data structure that allows the database system to find those tuples in the relation that have a specified value for that attribute efficiently, without scanning through all the tuples of the relation

7. What are key constraints? Explain the key constraints on relational database with suitable example.

Types of Constraints in DBMS-

In DBMS, there are following 5 different types of relational constraints-


Relational Constraints
Domain Constraint
Tuple Uniqueness Constraint
Key Constraint
Entity Integrity Constraint
Referential Integrity Constraint

- Domain constraint
- Tuple Uniqueness constraint
- Key constraint
- Entity Integrity constraint
- Referential Integrity constraint

1. Domain Constraint-

Domain constraint defines the domain or set of values for an attribute.
It specifies that the value taken by the attribute must be the atomic value from its domain.

Example-

STU_ID	**Name**	**Age**
S001	Akshay	20
S002	Abhishek	21
S003	Shashank	20

S004	Rahul	**A**

Consider the following Student table-

Here, value **‘A’** is not allowed since only integer values can be taken by the age attribute.

2. Tuple Uniqueness Constraint-

Tuple Uniqueness constraint specifies that all the tuples must be necessarily unique in any relation.

Example-01:

Consider the following Student table-

STU_ID	**Name**	**Age**
S001	Akshay	20
S002	Abhishek	21
S003	Shashank	20
S004	Rahul	20

This relation satisfies the tuple uniqueness constraint since here all the tuples are unique.

Example-02:

Consider the following Student table-

STU_ID	**Name**	**Age**

S001	**Akshay**	**20**
S001	**Akshay**	**20**
S003	Shashank	20
S004	Rahul	20

This relation does not satisfy the tuple uniqueness constraint since here all the tuples are not unique.

3. Key Constraint-

Key constraint specifies that in any relation-

- All the values of primary key must be unique.
- The value of primary key must not be null.

Example-

Consider the following Student table-

STU_ID	**Name**	**Age**
S001	Akshay	20
S001	Abhishek	21
S003	Shashank	20
S004	Rahul	20

This relation does not satisfy the key constraint as here all the values of primary key are not unique.

4. Entity Integrity Constraint-

- Entity integrity constraint specifies that no attribute of primary key must contain a null value in any relation.
- This is because the presence of null value in the primary key violates the uniqueness property.

Example-

Consider the following Student table-

STU_ID	**Name**	**Age**
S001	Akshay	20
S002	Abhishek	21
S003	Shashank	20
	Rahul	20

This relation does not satisfy the entity integrity constraint as here the primary key contains a NULL value.

5. Referential Integrity Constraint-

- This constraint is enforced when a foreign key references the primary key of a relation.
- It specifies that all the values taken by the foreign key must either be available in the relation of the primary key or be null.

Important Results-

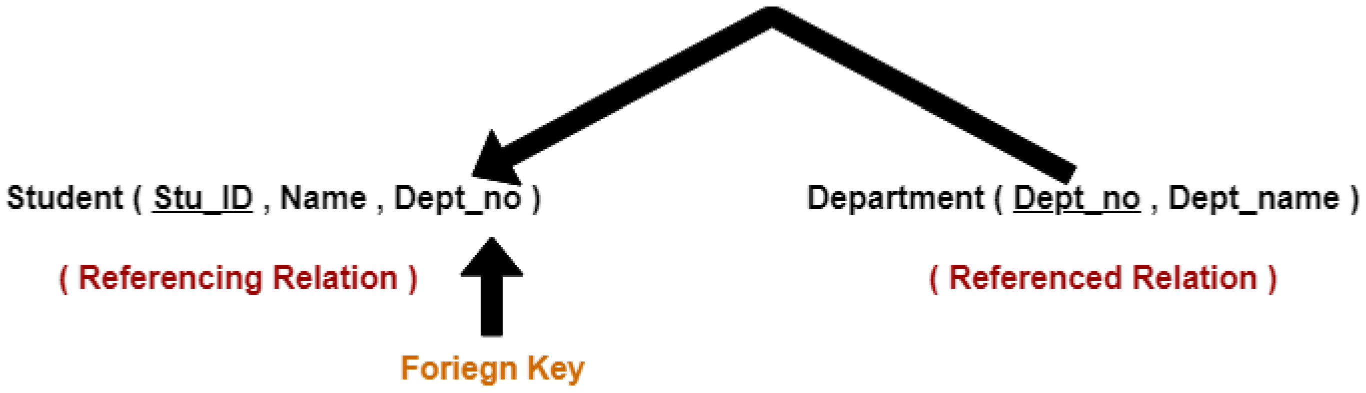
The following two important results emerges out due to referential integrity constraint-

- We can not insert a record into a referencing relation if the corresponding record does not exist in the referenced relation.
- We can not delete or update a record of the referenced relation if the corresponding record exists in the referencing relation.

Example-

Consider the following two relations- 'Student' and 'Department'.

Here, relation 'Student' references the relation 'Department'.


Student (Stu_ID , Name , Dept_no)
Department (Dept_no , Dept_name)
(Referencing Relation)
(Referenced Relation)
Foriegn Key

Student

STU_ID	Name	Dept_no
S001	Akshay	D10
S002	Abhishek	D10
S003	Shashank	D11
S004	Rahul	**D14**

Department

Dept_no	**Dept_name**
D10	ASET
D11	ALS
D12	ASFL
D13	ASHS

Here,

- The relation ‘Student’ does not satisfy the referential integrity constraint.
- This is because in relation ‘Department’, no value of primary key specifies department no. 14.
- Thus, referential integrity constraint is violated.

Handling Violation of Referential Integrity Constraint-

To ensure the correctness of the database, it is important to handle the violation of referential integrity constraint properly.

8. **Define data independence and mapping. Explain the three-schema architecture with diagram.**

The Three-Schema Architecture

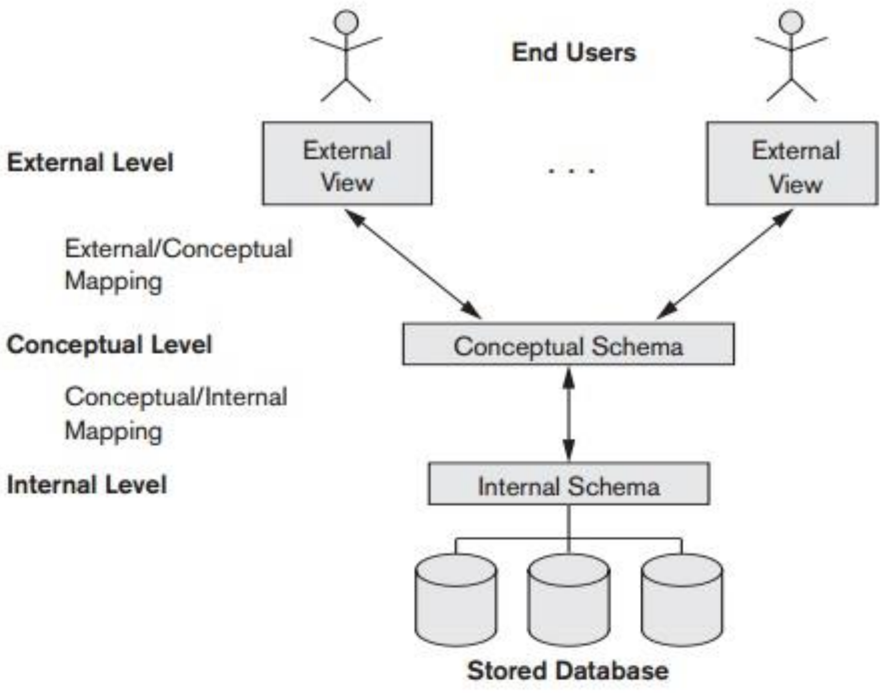
The goal of the three-schema architecture, illustrated in Figure 2.2, is to separate the user applications from the physical database. In this architecture, schemas can be defined at the following three levels:

The **internal level** has an **internal schema**, which describes the physical stor-age structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.

The **conceptual level** has a **conceptual schema**, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of 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 when a database system is implemented. This *implementation conceptual schema* is often based on a *conceptual schema design* in a high-level data model.

The **external** or **view level** includes a number of **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. As in the previous level, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level data model.

Figure 2.2
The three-schema architecture.



The three-schema architecture is a convenient tool with which the user can visualize the schema levels in a database system. Most DBMSs do not separate the three levels completely and explicitly, but support the three-schema architecture to some extent. Some older DBMSs may include physical-level details in the conceptual schema. The three-level ANSI architecture has an important place in database technology development because it clearly separates the users' external level, the database's con-ceptual level, and the internal storage level for designing a database. It is very much applicable in the design of DBMSs, even today. In most DBMSs that support user views, external schemas are specified in the same data model that describes the conceptual-level information (for example, a relational DBMS like Oracle uses SQL for this). Some DBMSs allow different data models to be used at the conceptual and external levels. An example is Universal Data Base (UDB), a DBMS from IBM, which uses the relational model to describe the conceptual schema, but may use an object-oriented model to describe an external schema.

Notice that the three schemas are only *descriptions* of data; the stored data that *actually* exists is at the physical level only. In a DBMS based on the three-schema architecture, each user group refers to its own external schema. Hence, the DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database. If the request is a database retrieval, the data extracted from the stored database must be reformatted to match the user's external view. The processes of transforming requests and results between levels are called **mappings**. These mappings may be time-consuming, so some DBMSs—especially those that are meant to support small databases—do not support external views. Even in such systems, however, a certain amount of mapping is necessary to transform requests between the conceptual and internal levels.

2. Data Independence

The three-schema architecture can be used to further explain the concept of **data independence**, which can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence:

Logical data independence is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand the database (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item). In the last case, external schemas that refer only to the remaining data should not be affected. For example, the external schema of Figure 1.5(a) should not be affected by changing the GRADE_REPORT file (or record type) shown in Figure 1.2 into the one shown in Figure 1.6(a). Only the view definition and the mappings need to be changed in a DBMS that supports logical data independence. After the conceptual schema undergoes a logical reorganization, application pro-grams that reference the external schema constructs must work as before.

Changes to constraints can be applied to the conceptual schema without affecting the external schemas or application programs.

Physical data independence is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well. Changes to the internal schema may be needed because some physical files were reorganized—for example, by creating additional access structures—to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema. For example, providing an access path to improve retrieval speed of section records (Figure 1.2) by semester and year should not require a query such as *list all sections offered in fall 2008* to be changed, although the query would be executed more efficiently by the DBMS by utilizing the new access path.

Generally, physical data independence exists in most databases and file environments where physical details such as the exact location of data on disk, and hard-ware details of storage encoding, placement, compression, splitting, merging of records, and so on are hidden from the user. Applications remain unaware of these details. On the other hand, logical data independence is harder to achieve because it allows structural and constraint changes without affecting application programs—a much stricter requirement.

Whenever we have a multiple-level DBMS, its catalog must be expanded to include information on how to map requests and data among the various levels. The DBMS uses additional software to accomplish these mappings by referring to the mapping information in the catalog. Data independence occurs because when the schema is changed at some level, the schema at the next higher level remains unchanged; only the *mapping* between the two levels is changed. Hence, application programs refer-ring to the higher-level schema need not be changed.

The three-schema architecture can make it easier to achieve true data independence, both physical and logical. However, the two levels of mappings create an overhead during compilation or execution of a query or program, leading to inefficiencies in the DBMS. Because of this, few DBMSs have implemented the full three-schema architecture.

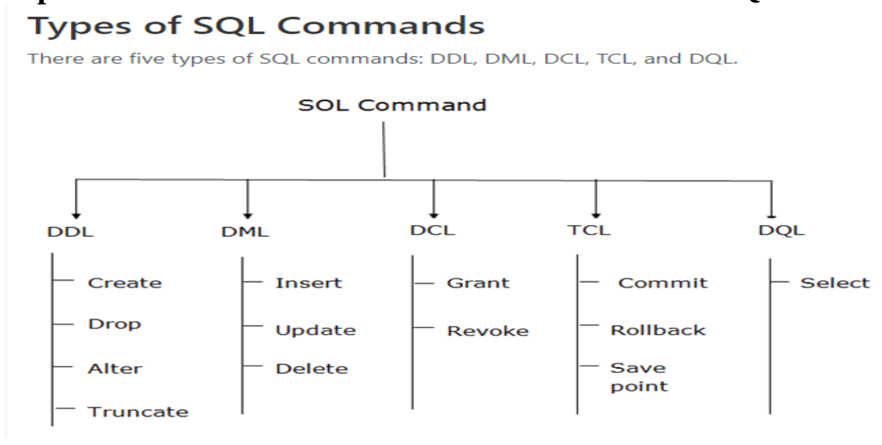
9. Explain different insert, update and delete commands used in SQL with example.

The SQL commands that deals with the manipulation of data present in the database belong to **DML** or Data Manipulation Language and this includes most of the SQL statements. It is the component of the SQL statement that controls access to data and to the database. Basically, DCL statements are grouped with DML statements.

```
INSERT INTO TABLE_NAME
(col1, col2, col3,.... col N)
VALUES (value1, value2, value3, .... valueN);
OR
INSERT INTO TABLE_NAME
VALUES (value1, value2, value3, .... valueN);
OR
INSERT INTO w3cschoool (Author, Subject) VALUES ("Sonoo", "DBMS");

UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CONDITION]

UPDATE students
SET User_Name = 'Sonoo'
WHERE Student_Id = '3'

UPDATE students
SET User_Name = 'Sonoo'
WHERE Student_Id = '3'

DELETE FROM table_name [WHERE condition];
DELETE FROM BOOK
```

WHERE Author="Sonoo";

10. Explain different DDL and DML statements used in SQL.



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 the database.DDL is a set of SQL commands used to create, modify, and delete database structures but not data. These commands are normally not used by a general user, who should be accessing the database via an application.

```
CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);
CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);
DROP TABLE ;
DROP TABLE EMPLOYEE;
ALTER TABLE table_name ADD column_name COLUMN-definition;
ALTER TABLE MODIFY(COLUMN DEFINITION....);

ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));

ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));
```

DML

The SQL commands that deals with the manipulation of data present in the database belong to **DML** or Data Manipulation Language and this includes most of the SQL statements. It is the component of the SQL statement that controls access to data and to the database. Basically, DCL statements are grouped with DML statements.

```
INSERT INTO TABLE_NAME

(col1, col2, col3,.... col N)

VALUES (value1, value2, value3, .... valueN);
```

OR

```
INSERT INTO TABLE_NAME
VALUES (value1, value2, value3, .... valueN);
OR
INSERT INTO w3cschoool (Author, Subject) VALUES ("Sonoo", "DBMS");

UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CONDITION]

UPDATE students
SET User_Name = 'Sonoo'
WHERE Student_Id = '3'

UPDATE students
SET User_Name = 'Sonoo'
WHERE Student_Id = '3'

DELETE FROM table_name [WHERE condition];
DELETE FROM BOOK
WHERE Author="Sonoo";
```