



# ABES Engineering College, Ghaziabad

## B.Tech Odd Semester (SEM V)

THEORY EXAMINATION 2021-22

### DATABASE MANAGEMENT SYSTEM

*Time: 3 Hours*

*Total Marks: 100*

**Note:** 1. Attempt all Sections. If require any missing data; then choose suitably.

#### SECTION A

1. Attempt all questions in brief.

2 x 10 = 20

a. What is the significance of Physical Data Independence?

[Marks-2]

**Data Independence:** It is defined as the capacity to make changes in three schema database architecture at one level of database system without having to change the schema at the next higher level.

There are two types of data independence: Physical Data Independence and Conceptual Data Independence

**Significance of Physical Data Independence:** The Physical Data Independence is ability to change in internal schema (physical level) without having to change in conceptual schema. This data independence is easy to achieve and implemented by most of the DBMS today.

Changes in the lowest or internal level (physical level) are: change in file and indexing location creating a new file structure, storing the new files in the system, creating a new index, etc.

b. List the four functions of DBA.

[Marks-2]

**Database Administrator (DBA):** A database administrator, or DBA, is responsible for maintaining, securing, and operating databases and also ensures that data is correctly stored and retrieved.

**The functions of a DBA include:**

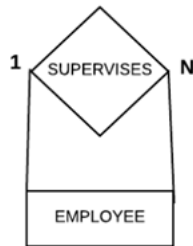
- I. **Schema definition:** The DBA creates the original database schema by executing a set of data definition statements in the DDL.
- II. **Schema and physical organization modification:** The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization or to alter the physical organization to improve performance.
- III. **Granting of authorization for data access:** By granting the different types of authorization, the database administrator can regulate which parts of the database various users can access.
- IV. **Routine maintenance:** Examples of database administrator's routine maintenance activities are:
  - Periodically backing up the database,
  - Physical data disk space management

c. When a relation set is called a recursive relationship set?

[Marks-2]

If there is a relationship between two entities and they both belong to the same entity type, then this is called a recursive relationship. We can say a relationship between the same entity set is called a recursive relationship.

**Example:** The some employees supervise other employee of the company as supervisor, hence employees and employees (supervisors) who supervise are same entity type.



**d. What do you mean by currency with respect to database?**

**[Marks-2]**

Data currency is monetary value assigned to data to identify its financial significance to an organization. Once the monetary value of data assets is identified, it may be used as the unit of exchange in a transaction, either as the sole payment or in combination with money. Another reason to assign a value to data is to quantify its importance to the business. That evaluation can help an organization in data management planning and related efforts like business continuance and disaster recovery planning.

**e. What is Relational Calculus?**

**[Marks-2]**

The relational calculus is non-procedural data base language. In this, users describe the desired information without giving the specific procedure (sequence of operations) for obtaining the information. It focuses on “What is to be done” not on “How is to be done”

It is divided into parts- **Tuple Relation Calculus** and **Domain Relation Calculus**

**Tuple Relation Calculus:** TRC is based on the concept of tuples, which are ordered sets of attribute values that represent a single row or record in a database table.

**Syntax:** it can be expressed as

$\{t \mid P(t)\}$

Where **t** is a tuple variable and **P (t)** is a logical formula that describes the conditions that the tuples in the result must satisfy. The curly braces {} are used to indicate that the expression is a set of tuples.

**Domain Relation Calculus:** It is similar to Tuple Relational Calculus, where it makes a list of the attributes that are to be chosen from the relations as per the conditions.

$\{ \langle a_1, a_2, a_3, \dots, a_n \rangle \mid P(a_1, a_2, a_3, \dots, a_n) \}$

where  $a_1, a_2, \dots, a_n$  are the attributes of the relation and P is the condition.

**f. What is Equi-Join in database?**

**[Marks-2]**

SQL's EQUI JOIN is an operation that allows us to combine data from two or more database tables based on a common column between them. This type of join uses the equals operator (=) in the WHERE clause to match the values in the columns. Alternatively, the EQUI JOIN can also be

performed using the JOIN keyword, followed by the ON keyword, and then specifying the column names and their associated tables that need to be checked for equality.

**Syntax:** EQUI JOIN:

```
SELECT table_a.column_a, table_b.column_b...  
FROM table_a  
INNER JOIN table_b  
ON table_a.common_field = table_b.common_field;
```

**Illustrative Example:**

**The EMPLOYEE Table –table-1**

ID	NAME	DEPARTMENT	SALARY
1	John	HR	5000
2	Jane	IT	6000
3	Bob	Finance	7000

**The DEPARTMENT Table – table-2**

DEPT_ID	DEPT_NAME
1	HR
2	IT
3	Finance

EQUI JOIN on these tables:

```
SQL> SELECT ID, NAME, SALARY, DEPT_NAME  
FROM EMPLOYEE  
INNER JOIN DEPARTMENT  
ON EMPLOYEE.DEPARTMENT = DEPARTMENT.DEPT_NAME;
```

**Output Table-table-3**

ID	NAME	SALARY	DEPT_NAME
1	John	5000	HR
2	Jane	6000	IT
3	Bob	7000	Finance

**g. What is a CLAUSE in terms of SQL?**

**[Marks-2]**

A clause in SQL is a built-in function that helps to fetch the required records from a database table.

There are various types of Clauses used in SQL like-

- **WHERE Clause:** It is used to filter the records with some conditions applied on the columns. The WHERE clauses are not mandatory clauses of SQL DML statements. But it

can be used to limit the number of rows affected by a SQL DML statement or returned by a query. It returns only those queries which fulfill the specific conditions.

- **GROUP BY Clause:** this statement is used to arrange identical data into groups in particular domain. The GROUP BY statement follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.
- **HAVING Clause:** The HAVING clause used only with SQL SELECT statement to specify that result of the query return rows or fetch the data/values from the groups according to the given condition. The HAVING clause is always executed with the GROUP BY clause.
- **ORDER BY:** If we need to arrange the query result set for any relation either in ascending (ASC) or descending (DESC) order then we use ORDER BY clause. Its main purpose is to sort records of the relation.

#### **h. Define the closure of an attribute set.**

**[Marks-2]**

Attribute closure of an attribute set can be defined as a set of attributes that can be functionally determined from it.

The set of attributes that are functionally dependent on attribute A is called the attribute closure of A and it can be represented as A<sup>+</sup>.

Steps to find the attribute closure of A (represented as A<sup>+</sup>)

Add A to A<sup>+</sup>

Recursively add attributes that can be functionally determined from attributes of the Relation R.

**Example:** Consider a relation R ( A , B , C , D , E , F , G ) with the functional dependencies --  
{ A → BC, BC → DE, D → F, CF → G }

**Closure of A,**

$$\begin{aligned}A^+ &= \{ A \} \\&= \{ A, B, C \} \text{ ( Using } A \rightarrow BC \text{ )} \\&= \{ A, B, C, D, E \} \text{ ( Using } BC \rightarrow DE \text{ )} \\&= \{ A, B, C, D, E, F \} \text{ ( Using } D \rightarrow F \text{ )} \\&= \{ A, B, C, D, E, F, G \} \text{ ( Using } CF \rightarrow G \text{ )}\end{aligned}$$

**Thus A<sup>+</sup> = { A , B , C , D , E , F , G }**

**Also BC<sup>+</sup> = { B , C , D , E , F , G }**

**And D<sup>+</sup> = { D , F }**

#### **i. When is a transaction rollback?**

**[Marks-2]**

ROLLBACK command is used in a transactional control language in SQL. It allows user to undo those transactions that aren't saved yet in the database. One can make use of this command if they wish to undo any changes or alterations since the execution of the last COMMIT.

In case an operation fails while completing a transaction, then it shows that the execution of all the changes is not successful. In this case, we can easily undo these changes with the ROLLBACK command.

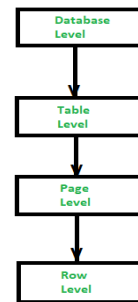
The syntax of ROLLBACK is: **Rollback;**

#### **j. List the various levels of locking?**

**[Marks-2]**

The locking in a database can be done at 4 levels, which start with the database at the highest level and down via table and page to the row at the lowest level.

1. **Database Level:** In this level, Lock is given to the complete database. Now, If let's say there are two relations in a database. If say, R1 and R2, where R1 uses tables, then R2 cannot use them.
2. **Table Level:** In this level, whole relation or table has been assigned a Lock for execution of process. This type of locking level is not suitable for multi-user database management systems.
3. **Page-Level:** The Page-level always consists of fixed size i.e, power of 2 or  $2^i$  type. A table can span several pages and a page can contain several tuples of one or more relations. At the page level, an intent shared lock (IS) will be imposed. This lock is capable of locking a table, shared, or exclusive page. An intent exclusive lock (IX) or intent update lock (IU) will be imposed if there's a case of DML statements (i.e. insert, update, delete).
4. **Row Level:** This level of locking is less restrictive as compared to other levels of locking. At the row level, if a concurrent transaction is accessing different rows in the same relationship, even if the rows are located on the same page, then this can be accepted by database systems.



*Hierarchy of levels of Locking*

## SECTION B

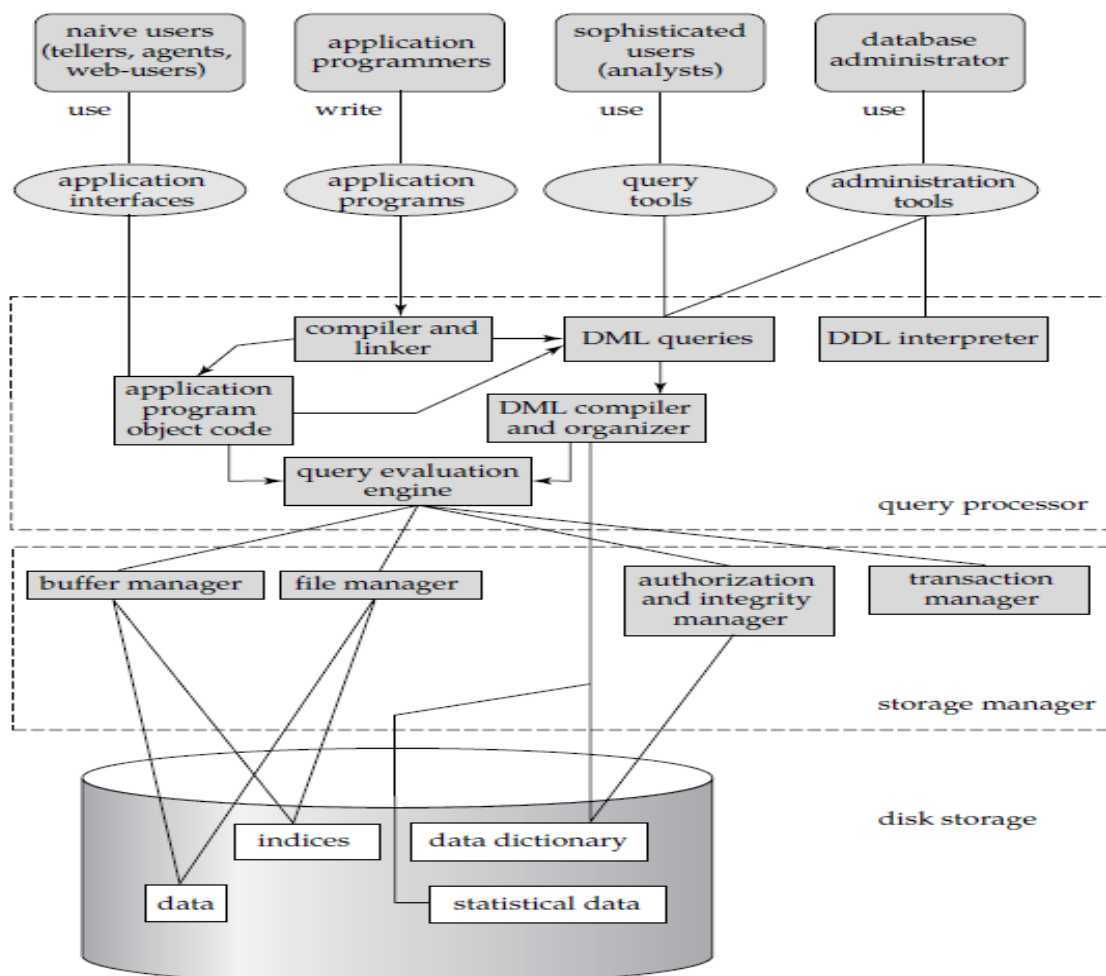
2. Attempt any three of the following:

10 x 3 = 30

a. Draw the overall structure of DBMS and explain its various components.

[Marks-10]

### Overall System architecture of DBMS



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. Corporate databases range in size from hundreds of gigabytes to, for the largest databases, terabytes of data. A gigabyte is 1000 megabytes (1 billion bytes), and a terabyte is 1 million megabytes (1 trillion bytes). Since the main memory of computers cannot store this much information, the information is stored on disks. Data are moved between disk storage and main memory as needed. Since the movement of data to and from disk is slow relative to the speed of the central processing unit, it is imperative that the database system structure the data so as to minimize the need to move data between disk and main memory.

The query processor is important because it helps the database system simplify and facilitate access to data. High-level views help to achieve this goal; with them, users of the system are not be burdened unnecessarily with the physical details of the implementation of the system. However, quick processing of updates and queries is important. 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.

### **Storage Manager**

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

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

The storage manager implements several data structures as part of the physical system implementation:

- **Data files**, which store the database itself.
- **Data dictionary**, which stores metadata about the structure of the database, in particular the schema of the database.
- **Indices**, which provide fast access to data items that hold particular values.

### **The Query Processor**

The query processor components include

- **DDL interpreter**, which interprets DDL statements and records the definitions in the data dictionary.
- **DML compiler**, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands. A query can usually be translated into any of a number of alternative evaluation plans that all give the same result. The DML compiler also performs
- **Query optimization**, that is, it picks the lowest cost evaluation plan from among the alternatives.
- **Query evaluation engine**, which executes low-level instructions generated by the DML compiler.

**b. Which relational algebra operations require the participating tables to be union-compatible? Give the Reason in detail. [Marks-10]**

Set operations in relational algebra require the participating tables to be union compatible.

Union compatibility of operands is required for the binary set operations.

If we take two relations R1 and R2 having attributes set as -

R1(A1, A2, ..., An) and R2(B1, B2, ..., Bn)

Thus R1 and R2 are union compatible if:

They have the same number of attributes, and The domains of corresponding attributes are type compatible

i.e.  $\text{dom}(A_i) = \text{dom}(B_i)$  for  $i=1, 2, \dots, n$

### **Reason for Union Compatibility**

Only similar types of tuples can be involved in set operations.

Suppose we have some tuples of Employees and some tuples of Department table.

Even if number of attributes is same in both the tables, both tuples are describing different meanings in the database.

UNION Operation (U)

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

Duplicate tuples are eliminated.

STUDENT	FN	LN
	Susan	Yao
	Ramesh	Shah
	Johnny	Kohler
	Barbara	Jones
	Amy	Ford
	Jimmy	Wang
	Ernest	Gilbert

INSTRUCTOR	FNAME	LNAME
	John	Smith
	Ricardo	Browne
	Susan	Yao
	Francis	Johnson
	Ramesh	Shah

(b)

FN	LN
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

STUDENT  $\cup$  INSTRUCTOR

**c. What do you understand by transitive dependencies? Explain with an example any two problems that can arise in the database if transitive dependencies are present in the database.** **[Marks-10]**

A transitive dependency in a database is an indirect relationship between values in the same table that causes a functional dependency. To achieve the normalization standard of Third Normal Form (3NF), you must eliminate any transitive dependency. By its nature, a transitive dependency requires three or more attributes (or database columns) that have a functional dependency between them, meaning that Column A in a table relies on Column B through an intermediate Column C. Let's see how this might work.

The given functional dependency can only be transitive when it is formed indirectly by two FDs. For example,

$P \rightarrow R$  happens to be a transitive dependency when the following functional dependencies hold true:

$P \rightarrow Q$ ,  $Q \text{ does not } \rightarrow P$  and  $Q \rightarrow R$

The transitive dependency can occur easily only in the case of some given relation of three or more attributes. Such a type of dependency helps us in normalizing the database in their 3rd Normal Form (3NF).

#### **Drawbacks of Transitive Dependency**

Some of the drawbacks of transitive dependency are as follows.

- Transitive dependencies can create problems in database design and management. For example, they can create redundancy issue.
- They can lead to inconsistency, and other issues. We can avoid these kinds of problems by normalizing a database by breaking down tables into smaller and more manageable units.
- They may reduce the performance and make database inefficient. Removing transitive dependencies by segregating the attributes into different tables can improve the database's performance and make it efficient.

#### **Transitive Dependencies bad for Database Design**

Transitive dependencies can create an issue in database design by violating the principle of database normalization. Database normalization is a process by which we can organize a database to minimize redundancy and avoid data anomalies, including update anomalies, insertion anomalies, and deletion anomalies, which can cause inconsistencies and errors in the data.

Moreover, transitive dependencies make it challenging to maintain the database by increasing the complexity and cost of database management.

All non-key attributes in the table are dependent only on the key attribute which helps to ensure data integrity and data consistency. It makes it easier to manage and modify the database over time.

#### **Example: Problem 1**

**Show\_Telecast**

Show_ID	Telecast_ID	Telecast_Type	CD_Cost (\$)
F08	S09	Thriller	50
F03	S05	Romantic	30
F05	S09	Comedy	20



The table above is not in its 3NF because it includes a transitive functional dependency.

Show\_ID → Telecast\_ID

Telecast\_ID → Telecast\_Type

Thus, the following has a transitive type of functional dependency.

Show\_ID → Telecast\_Type

The statement given above states the relation <Show\_Telecast> violates the 3NF (3rd Normal Form). If we want to remove this violation, then we have to split the tables for the removal of the transitive functional dependency.

**Show**

Show_ID	Telecast_ID	CD_Cost (\$)
F08	S09	50
F03	S05	30
F05	S09	20

**Telecast**

Telecast_ID	Telecast_Type
S09	Thriller
S05	Romantic
S09	Comedy

Now the above relation is in the Third Normal Form (3NF) of Normalization.

### Example: Problem 2

#### **STUDENT TABLE**

STUDENT ID	STUDENT	COURSE	CITY
101	NINJA 1	DSA	SHIMLA
102	NINJA 2	ML	DELHI
103	NINJA 3	JAVA	MUMBAI

As we see in the above table:

**Course→Student:** Here, the Course attribute determines the Student attribute. If we know the course, we can learn the student name.

**Student→City:** If we know the student name then we can determine City.

**Course→City:** If we know the course, we can determine the city via the Student column.

If we take a closer look into the functional dependencies discussed above, they are indeed forming a pattern

$A \rightarrow B$  and  $B \rightarrow C$ ; therefore,  $A \rightarrow C$ .

Similarly,

$A \rightarrow \text{Course}$ ,  $B \rightarrow \text{Student}$  and  $C \rightarrow \text{City}$

To ensure Third Normal Form(3NF) in the Student table, this transitive dependency must be removed and this process of removing transitive dependency is called Normalization.

The following Student table is not in 3NF. As it has the transitive dependency. Let's learn how?

Student ID → Student

Student → City

Here functional dependency also exist,

Student ID → City

Now let's eliminate the Transitive dependency. For removing transitive dependency we have to split the Student table in such a manner that Student ID will no longer functionally depend on Student city.

Now Let's split the table and create two new tables. One for {Student ID, Student} and another table containing {Student, Student City}.

#### STUDENT TABLE

STUDENT ID	STUDENT
101	NINJA 1
102	NINJA 2
103	NINJA 3

#### STUDENT CITY TABLE

STUDENT	CITY
NINJA 1	SHIMLA
NINJA 2	DELHI
NINJA 3	MUMBAI

Now this Student table and Author City table contains no Transitive dependency and the relation is now in 3NF.

**d. List ACID properties of transaction. Explain the usefulness of each. What is the importance of log?** **[Marks-10]**

ACID is a concept (and an acronym) that refers to the four properties of a transaction in a database system: Atomicity, Consistency, Isolation, and Durability.

These properties ensure the accuracy and integrity of the data in the database, ensuring that the data does not become corrupt due to some failure, guaranteeing the validity of the data even when errors or failures occur.

The ACID properties allow us to write applications without considering the complexity of the environment where the application is executed. These are essential for processing transactions in databases.

#### ACID Properties are

##### **1. Atomicity:**

A transaction must be an atomic unit of work, which means that all the modified data are performed, or none of them will be. The transaction should be executed completely or fail. If one part of the transaction fails, all the transactions will fail. Therefore, the transaction should not occur partially.

##### **2. Consistency:**

The consistency property ensures a completed execution of transaction from beginning to end without interference of other transactions.

- If the transaction completes successfully, then all changes to the system will have been properly made, and the system will be in a valid state.
- If any error occurs in a transaction, then any changes already made will be automatically rolled back. This will return the system to its state before the transaction was started.
- Since the system was in a consistent state when the transaction was started, it will once again be in a consistent state.
- The preservation of consistency is generally considered to be the responsibility of the programmers who write the database programs or of the DBMS module that enforces integrity constraints.

##### **3. Isolation:**

- Isolation property ensures that a transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executing concurrently.
- The isolation property is enforced by the concurrency control subsystem of the DBMS.
- This is important in that as the transaction is being executed, the state of the system may not be consistent. The transaction ensures that the system remains consistent after the transaction ends, but during an individual transaction, this may not be the case.
- If a transaction was not running in isolation, it could access data from the system that may not be consistent. By providing transaction isolation, this is prevented from happening.

### 3. **Durability:**

- Durability ensures that The changes applied to the database by a committed Transaction must persist in the database.
- The durability property is the responsibility of the recovery subsystem of the DBMS.
- The database system has checked the integrity constraints and won't need to abort the transaction.
- Many databases implement durability by writing all transactions into a transaction log that can be played back to recreate the system state right before a failure. A transaction can only be deemed committed after it is safely in the log.

Thus, features to consider for durability can be summarized as :

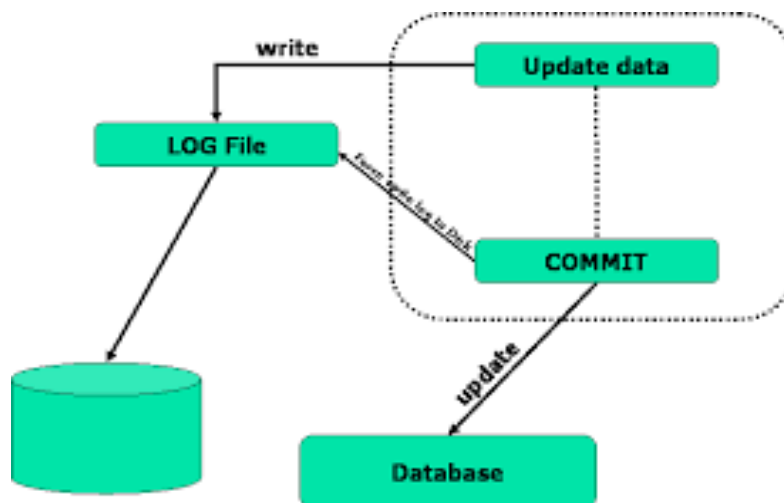
- recovery to the most recent successful commit after a database software failure
- recovery to the most recent successful commit after an application software failure
- recovery to the most recent successful commit after a CPU failure
- recovery to the most recent successful backup after a disk failure
- recovery to the most recent successful commit after a data disk failure

### **Log**

A log is basically a history of actions which have been executed by a Database Management System. Every SQL Server Database has a transaction log that records all the transactions and the Database modifications made on each transaction.

The log is an important and a critical component of the Database. If the system fails or something unexpected happens in the database, you can use the log to get the database back to consistent state. By looking at the log, it will give you an idea about what, and where things went wrong.

•



e. What do you mean by time stamping protocol for concurrency controlling? Discuss multi version scheme of concurrency control. [Marks-10]

**Timestamp** is a unique identifier created by the DBMS to identify a transaction. They are usually assigned in the order in which they are submitted to the system. Refer to the timestamp of a transaction  $T$  as  $TS(T)$ .

### **Timestamp Ordering Protocol –**

The main idea for this protocol is to order the transactions based on their Timestamps. A schedule in which the transactions participate is then serializable and the only equivalent serial schedule permitted has the transactions in the order of their Timestamp Values. Stating simply, the schedule is equivalent to the particular Serial Order corresponding to the order of the Transaction timestamps. An algorithm must ensure that, for each item accessed by Conflicting Operations in the schedule, the order in which the item is accessed does not violate the ordering. To ensure this, use two Timestamp Values relating to each database item  $X$ .

$W\_TS(X)$  is the largest timestamp of any transaction that executed  $write(X)$  successfully.

$R\_TS(X)$  is the largest timestamp of any transaction that executed  $read(X)$  successfully.

### **Basic Timestamp Ordering –**

Every transaction is issued a timestamp based on when it enters the system. Suppose, if an old transaction  $T_i$  has timestamp  $TS(T_i)$ , a new transaction  $T_j$  is assigned timestamp  $TS(T_j)$  such that  $TS(T_i) < TS(T_j)$ . The protocol manages concurrent execution such that the timestamps determine the serializability order. The timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. Whenever some Transaction  $T$  tries to issue a  $R\_item(X)$  or a  $W\_item(X)$ , the Basic TO algorithm compares the timestamp of  $T$  with  $R\_TS(X)$  &  $W\_TS(X)$  to ensure that the Timestamp order is not violated. This describes the Basic TO protocol in the following two cases.

1. Whenever a Transaction  $T$  issues a  **$W\_item(X)$**  operation, check the following conditions:
  - If  **$R\_TS(X) > TS(T)$**  and if  **$W\_TS(X) > TS(T)$** , then abort and rollback  $T$  and reject the operation. else,
  - Execute  $W\_item(X)$  operation of  $T$  and set  $W\_TS(X)$  to  $TS(T)$ .
2. Whenever a Transaction  $T$  issues a  **$R\_item(X)$**  operation, check the following conditions:
  - If  **$W\_TS(X) > TS(T)$** , then abort and reject  $T$  and reject the operation, else
  - If  **$W\_TS(X) \leq TS(T)$** , then execute the  $R\_item(X)$  operation of  $T$  and set  $R\_TS(X)$  to the larger of  $TS(T)$  and current  $R\_TS(X)$ .

### **Multiversion concurrency control (MVCC)**

Multi-version protocol aims to reduce the delay for read operations. It maintains multiple versions of data items. Whenever a write operation is performed, the protocol creates a new version of the transaction data to ensure conflict-free and successful read operations.

The newly created version contains the following information –

- **Content** – This field contains the data value of that version.
- **Write\_timestamp** – This field contains the timestamp of the transaction that created the new version.
- **Read\_timestamp** – This field contains the timestamp of the transaction that will read the newly created value.

By creating multiple versions of the data, the multi-version protocol ensures that read operations can access the appropriate version of the data without encountering conflicts. The protocol thus enables efficient concurrency control and reduces delays in read operations.

#### Various Types of MVCC

These are compared in the following table

MVCC Type	Description	Advantages	Disadvantages
Snapshot-based	Creates a snapshot of the database at the start of a transaction and uses it to provide necessary data for the transaction	Easy to implement	Significant overhead due to storing multiple versions of data
Timestamp-based	Assigns a unique timestamp to each transaction that creates a new version of a record; used to determine data visibility to transactions	More efficient than snapshot-based MVCC	Requires additional storage to store timestamps
History-based	Stores a complete history of all changes made to a record, allowing for easy rollback of transactions	Provides highest level of data consistency	Most complex of the MVCC techniques
Hybrid	Combines two or more MVCC techniques to balance performance and data consistency	Provides benefits of multiple MVCC techniques	More complex to implement than individual techniques

## SECTION C

#### 4. Attempt any one part of the following:

10 x 1 = 10

(a) What are the different types of Data Models in DBMS? Explain them.

[Marks-10]

A Data Model in Database Management System (DBMS) is the concept of tools that are developed to summarize the description of the database. Data Models provide a pictorial and transparent representation of data, which helps us in creating and manipulating data in database. It transforms database concepts from the design of the data to its proper implementation of data.

**Types of Database Model:** There are various types of Database Model like-

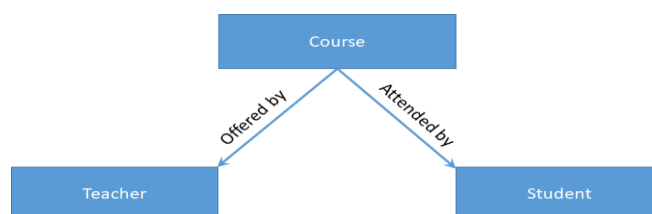
##### 1. Hierarchical Database Model:

The Hierarchical Model was the first database management system model. This concept uses a hierarchical tree structure to organise the data. The hierarchy begins at the root, which contains root data, and then grows into a tree as child nodes are added to the parent node. This model accurately depicts several real-world relationships such as food recipes, website sitemaps, and so on. A DDL for hierarchical data model must allow the definition of record types, fields types, pointers, and parent-child relationships. And the DML must support direct navigation using the

parent-child relationships and through pointers. Programs therefore navigate very close to the physical data structure level, implying that the hierarchical data model offers only very limited data independence.

Let us take an example of college students who take different courses. A course can be assigned to a student, but a student can take as many courses as they want.

Example:



Hierarchical structure

Records: Course, Teacher, Student

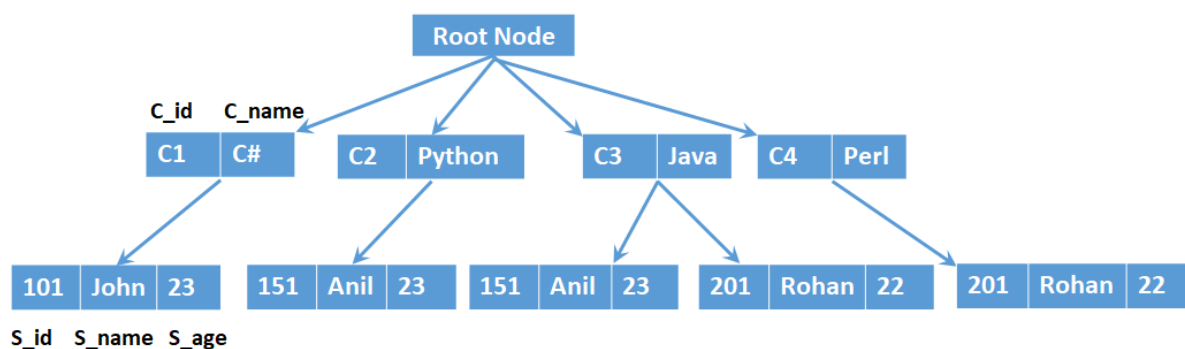
Relationship: Offered by, Attended by

Hierarchical model arranged as in Figure indicates the relationship and can answer the queries like:

- Who is teaching a particular course?
- Who has registered in a particular course?

It does not indicate the relationships and cannot answer the following queries:

- What are the courses being offered by a faculty?
- What are the courses being attended by a student?
- Whom are the students being taught by a faculty?
- Whom are the faculty teaching a student?



Example of Hierarchical structure (Course to Student)

### Advantages:

- **Simplicity:** It is conceptually simple due to the parent-child relationship. A clear chain of command or authority is there. Data can be retrieved easily due to the explicit links present between the table structures.
- **Data Integrity:** Referential integrity is always maintained, i.e., any changes made in the parent table are automatically updated in a child table.
- **Data Security:** It is provided and enforced by DBMS.
- **Efficiency:** It is efficient when a large number of one-to-many relationships exist in a database

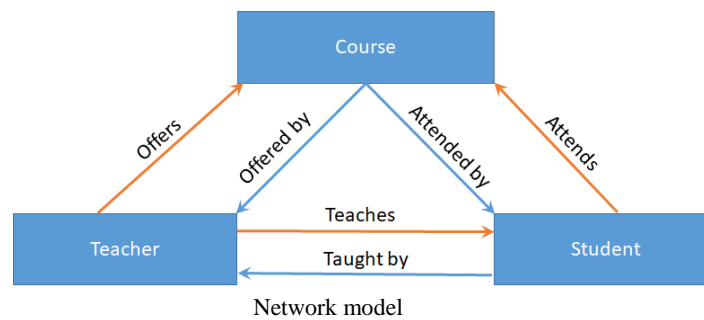
### Limitations:

- A child record cannot be created if it is not linked to any parent record. In our example above, we cannot create a course until it is opted by any student.
- M: N relationship (many-to-many) is not supported.
- Redundancy can result in data inconsistency.

### Network Mode

The network model replaces the hierarchical tree with a graph, thus allowing more general connections among the nodes. It allows a record to have more than one parent and multiple child records, thus permits many-to-many relationships. The network model was adopted by the Conference on Data System Language (CODASYL) Data Base Task Group in 1969 and underwent a major update in 1971. The operations on the network model are done with the help of the circular linked list. A child node in the network model can have more than one parent record, as shown in Figure.

### Example:

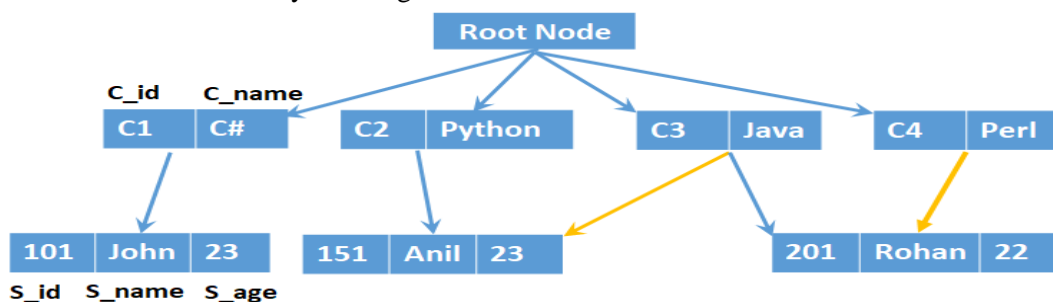


**Records:** Course, Teacher, Student

**Relationship:** Offered by, Attended by, Taught by, Offers, Attends, Teaches

Now the above network model can give the answer to all queries mentioned below, which was not possible in the case of Hierarchical model.

- What are the courses being offered by a faculty?
- What are the courses being attended by a student?
- Whom are the students being taught by a faculty?
- Whom is the faculty teaching a student?



**Some of the popular network databases are:**

- Integrated Data Store (IDS), IDMS (Integrated Database Management System), Raima Database Manager, TurboIMAGE, Univac DMS-1100

### Advantages:

- **Conceptual Simplicity:** It is conceptually simple and easy to design.

- **Capability to handle more relationship types:** It handles many to many relationships, which is a real help in modelling real-life situations.
- **Ease of data access:** The data access is easier and flexible as compared to the hierarchical model.
- **Data Integrity:** It ensures data integrity as a user must define the owner record before defining the member record.

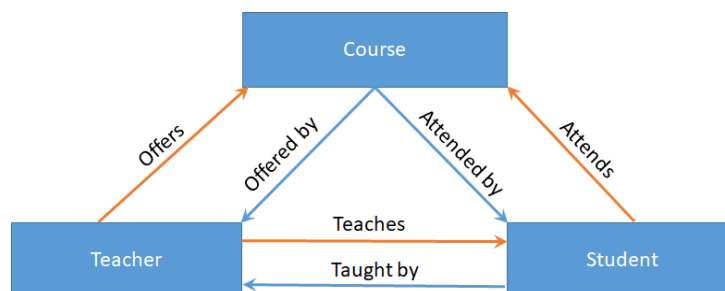
#### Limitations:

- **Implementation Complexity:** It requires the knowledge of actual physical data storage for navigational data access. The database administrators, designers, programmers, and even the end-users should be familiar with the internal data structure. It can't be used to create user-friendly DBMS.
- **Absence of structural Independence:** The data access method is navigational, which makes structural change very difficult and impossible in most cases. So if changes are made in the database structure, then all the application programs need to be modified.
- **Any change like updation, deletion, insertion is very complex.**

#### Relational Model

The relational model was introduced in 1970 by E. F. Codd (of IBM) in his landmark paper “A Relational Model of Data for Large Shared Databanks” (Communications of the ACM, June 1970, pp. 377–387). The relational model foundation is a mathematical concept known as a relation. A relation is also called a table, as a matrix composed of intersecting rows and columns. These tables represent both data and relationships amongst the data. Each table corresponds to an application entity. Each row in a relation is called a tuple and represents an instance of that entity. Each column represents an attribute. The relational model also describes a precise set of data manipulation constructs based on advanced mathematical concepts.

The relational data model is implemented through a very sophisticated relational database management system (RDBMS). The RDBMS performs the same basic functions provided by the hierarchical and network DBMS systems, in addition to other functions. Figure shows a relational model corresponding to the network model of records Course, Teacher and Student.



Relation	Attributes
Student :	(S_id, S_name, S_age)
Course :	(C_id, C_name)
Teacher :	(T_id, T_name)
Teacher Course :	(T_id, S_id)
Student Course:	(S_id, C_id)



Student Relation

S_id	S_name	S_age
101	John	23
151	Anil	23
201	Rohan	22

Student Course Relation

S_id	C_id
101	C1
151	C2
151	C3
201	C3
201	C4

Course Relation

C_id	C_name
C1	C#
C2	Python
C3	Java
C4	Pearl

Teacher Course Relation

T_id	C_id
T1	C1
T1	C2
T2	C3
T3	C3
T3	C4

Teacher Relation

T_id	T_name
T1	Anand
T2	Ritin
T3	Pratibha

Example of Relational Model

**Some popular Relational Database management systems are:** According to DB-Engines, in January 2021, the most widely used systems were Oracle, *MySQL (free software)*, *Microsoft SQL Server*, *PostgreSQL* (Open Source, a continuation development after INGRES), IBM DB2, SQLite (free software), *Microsoft Access*, *MariaDB* (free software), *Teradata*, *Microsoft Azure SQL Database*, *Apache Hive* (free software; specialized for data warehouses)

#### Advantages:

- Structural independence is promoted using independent tables. Database users need not be familiar with internal data storage.
- The tabular view improves conceptual simplicity. Database designers need to focus on logical design rather than knowing the physical data storage details.
- Ad hoc query capability is based on SQL. SQL is 4GL which permits the user to specify only what needs to be done without specifying how it will be done.
- Isolates the end-user from physical-level details
- Improves implementation and management simplicity
- Precision: The usage of relational algebra and relational calculus in the manipulation.

#### Limitations:

- Requires substantial hardware and system software overhead: It requires more powerful hardware and also physical storage devices to hide the complexity of implementation and data storage.
- Conceptual simplicity gives untrained people the tools to use a good system poorly. Poor database design degrades the performance when the database grows and it slows down a system.
- May promote information problems, i.e., information island phenomenon. As relational data model is easy to use, thus many departments may create their own database application, which creates the problem in integrating information and may create the issues of data inconsistency, redundancy, duplication, etc.
- Few relational databases have limits on the length of fields. It cannot be exceeded to accommodate more information.

There are some database model that extends the concept of object-oriented programming are discussed below in brief. The detailed discussion on these models is beyond the scope of this book.

### **Object-Oriented Data Model**

Relational database models may fail to handle the needs of complex information systems. The problem with RDBMS is that they force an information system to model into form of tables. Increasingly complex real-world problems demonstrated a need for a data model that more closely represented the real world. In the object-oriented data model (OODM), both data and their relationships are contained in a single structure known as an object. Object oriented models represent an entity as a class and its instance as an object. A class represents both attributes and the behavior of an entity. Object-oriented data models can typically be represented using Unified Modeling Language (UML) class diagrams. It is used to represent data and their relationships. Some object-oriented databases are designed to work well with object-oriented programming languages such as Delphi, Ruby, Python, JavaScript, Perl, Java, C#, Visual Basic .NET, C++, Objective-C and Smalltalk; others such as JADE have their own programming languages

#### **Advantages:**

- a. Capability to handle large number of different data types. It can store any type of data including text, numbers, pictures, voice and video.
- b. Object oriented programming with database technology
- c. Semantic content is added.
- d. Object oriented features improve productivity: Inheritance, Polymorphism and Dynamic binding

#### **Limitations:**

- a. Difficult to maintain
- b. Not suited for all applications.

### **Object-Relational Data Model**

Object relational systems combine the advantages of modern object oriented programming languages with relational database features such as multiple views of data and a high-level, non-procedural query language. Relational database systems are being enhanced by adding an object infrastructure to the database system itself, in the form of user defined data types, functions and rules and by building relational extenders that support specialized applications as image retrieval, advanced text searching and geographic applications. PostgreSQL is an Object-Relational Database Management System.

#### **Advantages:**

- a. Complex data sets can be saved and retrieved quickly and easily.
- b. Works well with object-oriented programming languages.

#### **Limitations:**

- a. Object databases are not widely adopted.
- b. In some situations, the high complexity can cause performance problems.
- c. High system overhead slows transactions

Summary Table

S.N.	Types of Database Model	Structure of Database
1	Hierarchical data Model	Tree
2	Network Database Model	Graph
3	ER Database model	Entities and relationships
4	Relation Model	Tables
5	Object Oriented Database Model	Object
6	Object Relational Database Model	Objects with Relation

**(b) State the procedural DML and nonprocedural DML with their differences. [Marks-10]**

Procedural Data Manipulation Language (DML) and non-procedural DML are two different types of DMLs that are used to manipulate data in a database.

**Procedural DMLs:**

Procedural DMLs (Data Manipulation Language) are a type of programming language that allows users to specify a series of actions to be taken on a database. These actions are often executed in a specific order, or “procedure,” hence the name. OR

In simple words, Procedural DML is a type of DML that requires the user to specify how to manipulate the data. It requires the user to specify the steps that the system should take to manipulate the data. Examples of procedural DMLs include languages such as COBOL, FORTRAN, and PL/SQL.

- It is a type of DML where we specify what data we want and how we get it.
- A procedural language is difficult to learn.
- Difficult to debug.
- Requires a large number of procedural instructions.
- Generally, it is used by professional programmers.
- File-oriented concept based.

**Non-Procedural DMLs:**

Non-procedural DMLs, on the other hand, do not require users to specify a specific series of actions to be taken on a database. Instead, they allow users to specify the desired result of a query, and the database system itself is responsible for determining the most efficient way to achieve that result. Non-procedural DMLs are often easier for users to learn and use, as they do not require a detailed understanding of how the database system works. OR,

Non-procedural DML is a type of DML that does not require the user to specify how to manipulate the data. It allows the user to specify what data they want, but not how to retrieve it. Non-procedural DMLs are often easier to use than procedural DMLs, because they do not require the user to have as much knowledge about the structure of the database. Examples of non-procedural DMLs include SQL and QBE (Query By Example).

- This type of DML specifies only what data we want, no need to mention how to get it.
- Easy to learn
- This language only requires a few non-procedural instructions.
- It can be used by both professional programmers and non-technical users.
- Database-oriented concept used in it.

In general, procedural DMLs offer more control and flexibility to the user, while non-procedural DMLs are simpler and easier to use.

**4. Attempt any one part of the following: 10 x 1 = 10**

**(a) Student (RollNo, Name, Father\_ Name, Branch)**

**Book (ISBN, Title, Author, Publisher)**

**Issue (RollNo, ISBN, Date-of –Issue)**

**Write the following queries in SQL and relational algebra:**

**[Marks-10]**

**I. List roll number and name of all students of the branch 'CSE'.**

**SQL:**

SELECT RollNo, Name FROM Student WHERE Branch = 'CSE';

**Relational Algebra:**

$\pi_{\text{RollNo, Name}} (\sigma_{\text{branch} = \text{'CSE'}} (\text{Student}))$

**II. Find the name of student who has issued a book published by 'ABC' publisher.**

**SQL**

SELECT S.Name FROM Student S, Issue I, Book B WHERE S.RollNo = I.RollNo  
AND I.ISBN = B.ISBN AND B.Publisher = 'ABC';

**Relational Algebra:**

$\pi_{\text{Name}} (\sigma_{\text{Publisher} = \text{'ABC'}} ((\text{Student} * \text{Issue}) * \text{Book}))$

**III. List title of all books and their authors issued to a student 'RAM'.**

**SQL**

SELECT B.Title, B.Author FROM Student S, Issue I, Book B WHERE S.RollNo =  
I.RollNo AND I.ISBN = B.ISBN AND S.Name = 'RAM';

**Relational Algebra:**

$\pi_{\text{Title, Author}} (\sigma_{\text{Name} = \text{'RAM'}} ((\text{Student} * \text{Issue}) * \text{Book}))$

**IV. List title of all books issued on or before December 1, 2020.**

**SQL**

SELECT B.Title FROM Issue I, Book B WHERE I.ISBN = B.ISBN AND I.Date-of-  
Issue <= '01-DEC-2020';

**Relational Algebra:**

$\pi_{\text{Title}} (\sigma_{\text{Date-of-Issue} \leq \text{'01-DEC-2020'}} (\text{Book} * \text{Issue}))$

**V. List all books published by publisher 'ABC'.**

**SQL**

SELECT \* FROM Book WHERE Publisher = 'ABC';

**Relational Algebra:**

$\sigma_{\text{Publisher} = \text{'ABC'}} (\text{Book})$

**(b) What do you mean by trigger? Explain it by a suitable example.**

**[Marks-10]**

A *trigger* defines a set of actions that are performed in response to an insert, update, or delete operation on a specified table. When such an SQL operation is executed, the trigger is said to have been *activated*. Triggers are optional and are defined using the CREATE TRIGGER statement.

Triggers can be used, along with referential constraints and check constraints, to enforce data integrity rules. Triggers can also be used to cause updates to other tables, automatically generate or transform values for inserted or updated rows, or invoke functions to perform tasks such as issuing alerts.

The following types of triggers are supported:

**BEFORE triggers**

Run before an update, or insert. Values that are being updated or inserted can be modified before the database is actually modified. You can use triggers that run before an update or insert in several ways:

- To check or modify values before they are actually updated or inserted in the database. This is useful if you must transform data from the way the user sees it to some internal database format.
- To run other non-database operations coded in user-defined functions.

**BEFORE DELETE triggers**

Run before a delete. Checks values (a raises an error, if necessary).

**AFTER triggers**

Run after an update, insert, or delete. You can use triggers that run after an update or insert in several ways:

- To update data in other tables. This capability is useful for maintaining relationships between data or in keeping audit trail information.
- To check against other data in the table or in other tables. This capability is useful to ensure data integrity when referential integrity constraints aren't appropriate, or when table check constraints limit checking to the current table only.
- To run non-database operations coded in user-defined functions. This capability is useful when issuing alerts or to update information outside the database.

**INSTEAD triggers**

Describe how to perform insert, update, and delete operations against views that are too complex to support these operations natively. They allow applications to use a view as the sole interface for all SQL operations (insert, delete, update and select).

**BEFORE triggers**

By using triggers that run before an update or insert, values that are being updated or inserted can be modified before the database is actually modified. These can be used to transform input from the application (user view of the data) to an internal database format where desired.

**AFTER triggers**

Triggers that run after an update, insert, or delete can be used in several ways.

### **INSTEAD OF triggers**

INSTEAD OF triggers describe how to perform insert, update, and delete operations against complex views. INSTEAD OF triggers allow applications to use a view as the sole interface for all SQL operations (insert, delete, update and select).

### **Examples of Trigger**

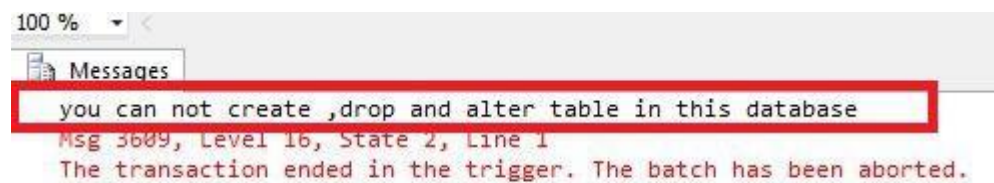
#### **DDL Triggers**

The Data Definition Language (DDL) command events such as Create\_table, Create\_view, drop\_table, Drop\_view, and Alter\_table cause the DDL triggers to be activated.

##### **SQL Server**

```
create trigger safety
on database
for
create_table,alter_table,drop_table
as
print 'you can not create,drop and alter tab
```

#### **Output:**



The screenshot shows a 'Messages' window in SQL Server. A red rectangle highlights the error message: 'you can not create ,drop and alter table in this database'. Below the highlighted message, the text 'Msg 3609, Level 16, State 2, Line 1' and 'The transaction ended in the trigger. The batch has been aborted.' is visible.

#### **DML Triggers**

The Data uses manipulation Language (DML) command events that begin with Insert, Update, and Delete set off the DML triggers. corresponding to insert\_table, update\_view, and delete\_table.

##### **SQL Server**

```
create trigger deep
on emp
for
insert,update ,delete
as
print 'you can not insert,update and delete this table i'
rollback;
```

#### **Output:**

100 % <

Messages

you can not insert,update and delete this table i

Msg 3609, Level 16, State 1, Line 1

The transaction ended in the trigger. The batch has been aborted.