**MODULE: 5 (Database)**

(Q.1) What do you understand By Database?

In computing, a database is an organized collection of data (also known as a data store) stored and accessed electronically through the use of a database management system. Small databases can be stored on a file system, while large databases are hosted on computer clusters or cloud storage. The design of databases spans formal techniques and practical considerations, including data modeling, efficient data representation and storage, query languages, security and privacy of sensitive data, and distributed computing issues, including supporting concurrent access and fault tolerance.

A database management system (DBMS) is the software that interacts with end users, applications, and the database itself to capture and analyze the data. The DBMS software additionally encompasses the core facilities provided to administer the database. The sum total of the database, the DBMS and the associated applications can be referred to as a database system. Often the term "database" is also used loosely to refer to any of the DBMS, the database system or an application associated with the database.

(Q.2) What is Normalization?

Normalization is the process to eliminate data redundancy and enhance data integrity in the table. Normalization also helps to organize the data in the database. It is a multi-step process that sets the data into tabular form and removes the duplicated data from the relational tables. Normalization organizes the columns and tables of a database to ensure that database integrity constraints properly execute their dependencies. It is a systematic technique of decomposing tables to eliminate data redundancy (repetition) and undesirable characteristics like Insertion, Update, and Deletion anomalies.

1. 1st Normal Form (1NF)

A table is referred to as being in its First Normal Form if atomicity of the table is 1. Here, atomicity states that a single cell cannot hold multiple values. It must hold only a single-valued attribute. The First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

1. Second Normal Form (2NF)

The first condition for the table to be in Second Normal Form is that the table has to be in First Normal Form. The table should not possess partial dependency. The partial dependency here means the proper subset of the candidate key should give a non-prime attribute.

1. Third Normal Form (3NF)

The first condition for the table to be in Third Normal Form is that the table should be in the Second Normal Form. The second condition is that there should be no transitive dependency for non-prime attributes, which indicates that non-prime attributes (which are not a part of the candidate key) should not depend on other non-prime attributes in a table. Therefore, a transitive dependency is a functional dependency in which A → C (A determines C) indirectly, because of A → B and B → C (where it is not the case that B → A). The third Normal Form ensures the reduction of data duplication. It is also used to achieve data integrity.

1. Boyce CoddNormal Form (BCNF)

Boyce Codd Normal Form is also known as 3.5 NF. It is the superior version of 3NF and was developed by Raymond F. Boyce and Edgar F. Codd to tackle certain types of anomalies which were not resolved with 3NF. The first condition for the table to be in Boyce Codd Normal Form is that the table should be in the third normal form. Secondly, every Right-Hand Side (RHS) attribute of the functional dependencies should depend on the super key of that particular table.

(Q.3) What is Difference between DBMS and RDBMS?

|  |  |  |
| --- | --- | --- |
| **Parameter** | **DBMS** | **RDBMS** |
| Function | DBMS is system software for creating, storing and managing, updating and retrieving data from databases. | RDBMS is software that allows the creation and management of databases in a tabular format for efficient retrieval, updation, and storage of data. |
| Storage | In DBMS, the storage of data is in the files. It is arranged either in a hierarchical form or navigational form. | In RDBMS, the data is stored in tables. There is no hierarchy and instead, follow a [relational model](https://scaler.com/topics/relational-model-in-dbms/). Columns are the headers and rows contain the corresponding values. |
| Number of Users | Database Management System can only support a single user. | Relational Database Management System allows access to multiple users to the databases. |
| Normalization (organizing data in the database) | DBMS does not support normalization. | Normalization is enabled in RDBMS. In fact, it was introduced by Edgar F. Codd for his relational database model. |
| Data Type | DBMS cannot store large quantities of data. | RBMS allows users to store a large set of data. |
| Data Relationships | In the database management system, there are no relationships amongst the data stored. | In RDBMS, there are relationships formed amongst the data stored in tables. |
| Data Fetching | The process of data fetching in DBMS is slow. | The process of Data Fetching is faster, and efficient in RDBMS because of its relational model. |
| Distributed Databases | DBMS does not support the distribution of databases.  A distributed database is a database that can be stored at different locations. | RDBMS supports distributed databases. |
| Data Redundancy | The version of DBMS increases data redundancy (repetition of data). | In RDBMS, data redundancy is eliminated that reduces wastage of time and resources. |
| Hardware and Software Requirement | DBMS needs minimum software and hardware requirements. | In RDBMS, hardware and software requirements are higher than the classic DBMS. |
| Data Integrity (the [integrity constraints](https://www.scaler.com/topics/integrity-constraints-in-dbms/) that allows to maintain the accuracy and consistency in the databases) | Database Management System does not support any data integration constraints or methods. | The Relational Database Management System supports data integrity constraints. |
| Data Access | In DBMS, you can access only a single file from a single database. | In RDBMS, you can access multiple data at a single time. |
| Data Security | DBMS is more prone to data theft, and access to unauthorized users because it does not allow any data security measures. | RDBMS supports security measures and is more secure than the traditional RDBMS. |
| ACID Properties **(**The ACID properties are necessary to ensure data consistency) | DBMS does not support any ACID properties. | RDBMS supports ACID properties and ensures no data inconsistencies. |
| Data Client-Server | DBMS does not support client-server architecture. | RDBMS supports client-server architecture |
| Examples | XML, File system, window registry, etc. are some of the examples of database management systems. | Oracle, MYSQL, SQL Server, etc. are some of the examples of Relational Database Management Systems. |

(Q.4) What is EF Cod Rule of RDBMS Systems?

The Codd's 12 rules are as follows.

Rule zero

This rule states that for a system to qualify as an RDBMS, it must be able to manage database entirely through the relational capabilities. Powered By VDO.AI

Rule 1:

Information rule all information (including metadata) is to be represented as stored data in cells of tables. The rows and columns have to be strictly unordered.

Rule 2:

Guaranteed Access Each unique piece of data (atomic value) should be accesible by: Table Name + Primary Key (Row) + Attribute (column). NOTE: Ability to directly access via POINTER is a violation of this rule.

Rule 3:

Systematic treatment of NULL Null has several meanings, it can mean missing data, not applicable or no value. It should be handled consistently. Also, Primary key must not be null, ever. Expression on NULL must give null.

Rule 4:

Active Online Catalog Database dictionary (catalog) is the structure description of the complete Database and it must be stored online. The Catalog must be governed by same rules as rest of the database. The same query language should be used on catalog as used to query database.

Rule 5:

Powerful and Well-Structured Language One well-structured language must be there to provide all manners of access to the data stored in the database. Example: SQL, etc. If the database allows access to the data without the use of this language, then that is a violation.

Rule 6:

View Updation Rule All the view that are theoretically updatable should be updatable by the system as well.

Rule 7:

Relational Level Operation There must be Insert, Delete and Update operations at each level of relations. Set operation like Union, Intersection and minus should also be supported.

Rule 8:

Physical Data Independence The physical storage of data should not matter to the system. If say, some file supporting table is renamed or moved from one disk to another, it should not effect the application.

Rule 9:

Logical Data Independence If there is change in the logical structure (table structures) of the database the user view of data should not change. Say, if a table is split into two tables, a new view should give result as the join of the two tables. This rule is most difficult to satisfy.

Rule 10:

Integrity Independence The database should be able to enforce its own integrity rather than using other programs. Key and Check constraints, trigger etc. should be stored in Data Dictionary. This also make RDBMS independent of front-end.

Rule 11:

Distribution Independence a database should work properly regardless of its distribution across a network. Even if a database is geographically distributed, with data stored in pieces, the end user should get an impression that it is stored at the same place. This lays the foundation of distributed database.

Rule 12:

Nonsubversion Rule If low level access is allowed to a system it should not be able to subvert or bypass integrity rules to change the data. This can be achieved by some sort of looking or encryption

(Q.5) What do you understand By Data Redundancy?

In DBMS, when the same data is stored in different tables, it causes data redundancy. Sometimes, it is done on purpose for recovery or backup of data, faster access of data, or updating data easily. Redundant data costs extra money, demands higher storage capacity, and requires extra effort to keep all the files up to date. Sometimes, unintentional duplicity of data causes a problem for the database to work properly, or it may become harder for the end user to access data. Redundant data unnecessarily occupy space in the database to save identical copies, which leads to space constraints, which is one of the major problems. Redundancy in DBMS gives rise to anomalies, and we will study it further. In a database management system, the problems that occur while working on data include inserting, deleting, and updating data in the database is done on purpose for recovery or backup of data, faster access of data, or updating data easily. Redundant data costs extra money, demands higher storage capacity, and requires extra effort to keep all the files up to date. Sometimes, unintentional duplicity of data causes a problem for the database to work properly, or it may become harder for the end user to access data. Redundant data unnecessarily occupy space in the database to save identical copies, which leads to space constraints, which is one of the major problems.

Insertion Anomaly:

Insertion anomaly arises when you are trying to insert some data into the database, but you are not able to insert it.

Deletion Anomaly:

Deletion anomaly arises when you delete some data from the database, but some unrelated data is also deleted; that is, there will be a loss of data due to deletion anomaly.

Updating Anomaly:

An update anomaly arises when you update some data in the database, but the data is partially updated, which causes data inconsistency.

(Q.6) What is DDL Interpreter?

A DDL (Data Definition Language) Interpreter is a language in a database that is used to create and modify the structure of objects in a database using predefined commands and a specific syntax. These objects include tables, sequences, locations, aliases, schemas, and indexes. DDL statements create, modify, and remove database objects. The interpreter interprets DDL statements and records them in tables containing metadata. DDL is a standardized language with commands to define the storage groups, different structures, and objects in a database. DDL statements are similar to a computer programming language for defining data structures, especially database schemas. Common examples of DDL statements include CREATE, ALTER, and DROP. In contrast, DML (Data Manipulation Language) is used to manipulate data itself, such as inserting, updating, and deleting data. The main difference between DDL and DML is that DDL is used to create and modify database objects, while DML is used to perform operations on the data.

(Q.7) What is DML Compiler in SQL?

A DML (data manipulation language) refers to a computer programming language that allows you to add (insert), delete (delete), and alter (update) data in a database. A DML is typically a sublanguage of a larger database language like SQL, with the DML containing some of the language’s operators. A DML (data manipulation language) is a group of computer languages that provide commands for manipulating data in databases. The majority of SQL statements are categorised as DML (Data Manipulation Language), which includes SQL commands that deal with modifying data in a database. It’s the section of the SQL statement that controls who has access to the database and data. DML statements and DCL statements are grouped together. Because the DML command isn’t auto-committed, it won’t be able to save all database changes permanently. There’s a chance they’ll be rolled back. Here are some different DML commands:

INSERT INTO Command

This command can be used to insert data into a row of a table.

UPDATE Command

This statement in SQL is used to update the data that is present in an existing table of a database. The UPDATE statement can be used to update single or multiple columns on the basis of our specific needs.

DELETE Command

The DELETE statement can be used in SQL to delete various records from a given table. On the basis of the condition that has been set in the WHERE clause, one can delete single or multiple records.

(Q.8) What is SQL Key Constraints writing an Example of SQL Key Constraints

SQL constraints are used to specify rules for the data in a table. Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted. Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The constraints used in SQL are:

|  |  |
| --- | --- |
| Constraint | Description |
| NOT NULL | values cannot be null |
| UNIQUE | values cannot match any older value |
| PRIMARY KEY | used to uniquely identify a row |
| FOREIGN KEY | references a row in another table |
| CHECK | validates condition for new value |
| DEFAULT | set default value if not passed |
| CREATE INDEX | used to speedup the read process |

NOT NULL Constraint

The NOT NULL constraint in a column means that the column cannot store NULL values.

For example,

CREATE TABLE Colleges

(college\_id INT NOT NULL, college\_code VARCHAR (20) NOT NULL,

college\_name VARCHAR(50) );

UNIQUE Constraint

The UNIQUE constraint in a column means that the column must have unique value.

For example,

CREATE TABLE Colleges

( college\_id INT NOT NULL UNIQUE, college\_code VARCHAR(20) UNIQUE,

college\_name VARCHAR(50));

Similarly, the value of college\_id must be unique as well as it cannot store NULL values.

PRIMARY KEY Constraint

The PRIMARY KEY constraint is simply a combination of NOT NULL and UNIQUE constraints. It means that the column value is used to uniquely identify the row.

For example,

CREATE TABLE Colleges

(college\_id INT PRIMARY KEY, college\_code VARCHAR (20) NOT NULL,

college\_name VARCHAR(50) );

Similarly, it cannot store NULL value and must be UNIQUE.

FOREIGN KEY Constraint

The FOREIGN KEY (REFERENCES in some databases) constraint in a column is used to reference a record that exists in another table.

For example,

CREATE TABLE Orders

( order\_id INT PRIMARY KEY,

customer\_id int REFERENCES Customers(id) );

CHECK Constraint

The CHECK constraint checks the condition before allowing values in a table.

For example,

CREATE TABLE Orders

( order\_id INT PRIMARY KEY,

amount int CHECK (amount >= 100) );

DEFAULT Constraint

The DEFAULT constraint is used to set the default value if we try to store NULL in a column.

For example,

CREATE TABLE College

( college\_id INT PRIMARY KEY,

college\_code VARCHAR(20),

college\_country VARCHAR(20) DEFAULT 'US' );

CREATE INDEX

Constraint If a column has CREATE INDEX constraint, it's faster to retrieve data if we use that column for data retrieval.

For example, --

CREATE TABLE Colleges

( college\_id INT PRIMARY KEY,

college\_code VARCHAR(20) NOT NULL,

college\_name VARCHAR(50) );

CREATE INDEX college\_index ON Colleges(college\_code);

(Q.9) What is save Point? How to create a save Point write a Query?

Savepoint is a command in SQL that is used with the rollback command. It is a command in Transaction Control Language that is used to mark the transaction in a table. Consider you are making a very long table, and you want to roll back only to a certain position in a table then; this can be achieved using the savepoint. If you made a transaction in a table, you could mark the transaction as a certain name, and later on, if you want to roll back to that point, you can do it easily by using the transaction's name. Savepoint is helpful when we want to roll back only a small part of a table and not the whole table. In simple words, we can say savepoint is a bookmark in SQL.

A SAVEPOINT is a point in a transaction in which you can roll the transaction back to a certain point without rolling back the entire transaction.

Syntax for Savepoint command:

**SAVEPOINT SAVEPOINT\_NAME;**

This command is used only in the creation of SAVEPOINT among all the transactions. In general ROLLBACK is used to undo a group of transactions.

Syntax for rolling back to Savepoint command:

**ROLLBACK TO SAVEPOINT\_NAME;**

you can ROLLBACK to any SAVEPOINT at any time to return the appropriate data to its original state.

Example: Delete those records from the table which have age = 20 and then ROLLBACK the changes in the database by keeping Savepoints.

Queries:

SAVEPOINT SP1; //Savepoint created.

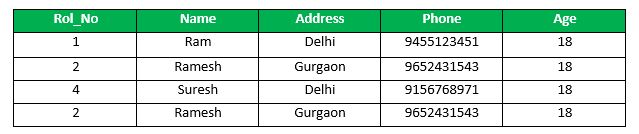
DELETE FROM Student WHERE AGE = 20; //deleted

SAVEPOINT SP2; //Savepoint created.

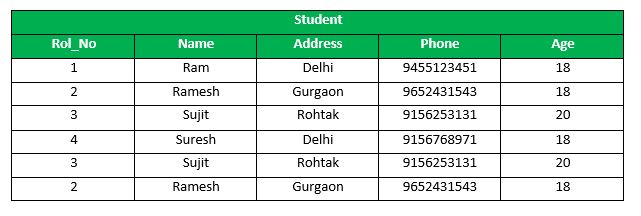
Here SP1 is first SAVEPOINT created before deletion

.In this example one deletion have taken place. After deletion again SAVEPOINT SP2 is created.

OUTPUT :



Deletion have been taken place, let us assume that you have changed your mind and decided to ROLLBACK to the SAVEPOINT that you identified as SP1 which is before deletion. deletion is undone by this statement ,

ROLLBACK TO SP1; //Rollback completed.   


(Q.10) What is trigger and how to create a Trigger in SQL?

A trigger in SQL is a procedural code that is automatically executed in response to certain events on a specified table. It is important to understand how these small codes make such a huge difference in database performance. In this article, you will learn how to implement triggers along with examples. Triggers in SQL are concise snippets of code that automatically execute when specific events occur on a table. These triggers play a vital role in maintaining data integrity, ensuring the accuracy and consistency of information. Similar to their real-world counterparts, SQL triggers act as a safeguard, responding to events and safeguarding data integrity. For example, when the gun trigger is pulled a bullet is fired. John is the marketing officer in a company. When a new customer data is entered into the company’s database he has to send the welcome message to each new customer. If it is one or two customers John can do it manually, but what if the count is more than a thousand? Well in such scenario triggers come in handy. Thus, now John can easily create a trigger which will automatically send a welcome email to the new customers once their data is entered into the database. So I hope you are clear with the introduction of Triggers in SQL. Always remember that there cannot be two triggers with similar action time and event for one table. For example, we cannot have two BEFORE UPDATE triggers for a table. But we can have a BEFORE UPDATE and a BEFORE INSERT trigger, or a BEFORE UPDATE and an AFTER UPDATE trigger. Before we dive further into the fundamentals of triggers I would suggest you to understand the concepts of SQL Basics and Normalization so that you get a better grip on Triggers in SQL.

Syntax

Lets now look at the syntax of a trigger.

**Create Trigger Trigger\_Name**

**(Before | After) [ Insert | Update | Delete]**

**on [Table\_Name]**

**[ for each row | for each column ]**

**[ trigger\_body ]**

Now let me break down this syntax and explain each and every part in detail.

**Create Trigger**

These two keywords are used to specify that a trigger block is going to be declared.

**Trigger\_Name**

It specifies the name of the trigger. Trigger name has to be unique and shouldn’t repeat.

**( Before | After )**

This specifies when the trigger will be executed. It tells us the time at which the trigger is initiated, i.e, either before the ongoing event or after.

Before Triggers are used to update or validate record values before they’re saved to the database. After Triggers are used to access field values that are set by the system and to effect changes in other records. The records that activate the after trigger are read-only. We cannot use After trigger if we want to update a record because it will lead to read-only error.

**[ Insert | Update | Delete ]**

These are the DML operations and we can use either of them in a given trigger.

**on [ Table\_Name ]**

We need to mention the table name on which the trigger is being applied. Don’t forget to use on keyword and also make sure the selected table is present in the database.

**[ for each row | for each column ]**

Row-level trigger gets executed before or after any column value of a row changes Column Level Trigger gets executed before or after the specified column changes

**[ trigger\_body]**

It consists of queries that need to be executed when the trigger is called.