**Q-1 What do you understand By Database**

A database is information that is set up for easy access, management and updating. Computer databases typically store aggregations of [data](https://www.techtarget.com/searchdatamanagement/definition/data) records or [files](https://searchsqlserver.techtarget.com/definition/flat-file) that contain information, such as sales transactions, customer data, financials and product information.

Databases are used for storing, maintaining and accessing any sort of data. They collect information on people, places or things. That information is gathered in one place so that it can be observed and analyzed. Databases can be thought of as an organized collection of information.

**Q-2 What is Normalization?**

**Normalization** is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization rules divides larger tables into smaller tables and links them using relationships. The purpose of Normalisation in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.

The inventor of the [relational model](https://www.guru99.com/relational-data-model-dbms.html) Edgar Codd proposed the theory of normalization of data with the introduction of the First Normal Form, and he continued to extend theory with Second and Third Normal Form. Later he joined Raymond F. Boyce to develop the theory of Boyce-Codd Normal Form.

Here is a list of Normal Forms in SQL:

* 1NF (First Normal Form)
* 2NF (Second Normal Form)
* 3NF (Third Normal Form)
* BCNF (Boyce-Codd Normal Form)
* 4NF (Fourth Normal Form)
* 5NF (Fifth Normal Form)
* 6NF (Sixth Normal Form)

**Q-3 What is Difference between DBMS and RDBMS?**

|  |  |
| --- | --- |
| **RDBMS** | **DBMS** |
| Data stored is in table format | Data stored is in the file format |
| Multiple data elements are accessible together | Individual access of data elements |
| Data in the form of a table are linked together | No connection between data |
| Normalisation is not achievable | There is normalisation |
| Support distributed database | No support for distributed database |
| Data is stored in a large amount | Data stored is a small quantity |
| Here, redundancy of data is reduced with the help of key and indexes in RDBMS | Data redundancy is common |
| RDBMS supports multiple users | DBMS supports a single user |
| It features multiple layers of security while handling data | There is only low security while handling data |
| The software and hardware requirements are higher | The software and hardware requirements are low |
| Oracle, SQL Server. | XML, Microsoft Access. |

**Q-4 What is MF Cod Rule of RDBMS Systems?**

Codd's twelve rules are a set of thirteen rules (numbered zero to twelve) proposed by Edgar F. Codd, a pioneer of the relational model for databases, designed to define what is required from a database management system in order for it to be considered relational, i.e., a relational database management system (RDBMS).

**Rule 0:** The *foundation rule*:

For any system that is advertised as, or claimed to be, a relational data base management system, that system must be able to manage data bases entirely through its relational capabilities.

**Rule 1:** The *information rule*:

All information in a relational data base is represented explicitly at the logical level and in exactly one way – by values in tables.

**Rule 2:** The *guaranteed access rule*:

Each and every datum (atomic value) in a relational data base is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name.

**Rule 3:** *Systematic treatment of null values*:

Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type.

**Rule 4:** *Dynamic*[*online*](https://en.wikipedia.org/wiki/Online)*[catalog](https://en.wikipedia.org/wiki/Database_catalog" \o "Database catalog) based on the relational model*:

The data base description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data.

**Rule 5:** The *comprehensive data sublanguage rule*:

A relational system may support several languages and various modes of terminal use (for example, the fill-in-the-blanks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and that is comprehensive in supporting all of the following items:

1. Data definition.
2. View definition.
3. Data manipulation (interactive and by program).
4. Integrity constraints.
5. Authorization.
6. Transaction boundaries (begin, commit and rollback).

**Rule 6:** The [*view*](https://en.wikipedia.org/wiki/View_(SQL))*updating rule*:

All views that are theoretically updatable are also updatable by the system.

**Rule 7:** Relational Operations Rule / *Possible for high-level insert, update, and delete*:

The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update and deletion of data.

**Rule 8:** *Physical data independence*:

Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representations or access methods.

**Rule 9:** *Logical data independence*:

Application programs and terminal activities remain logically unimpaired when information-preserving changes of any kind that theoretically permit unimpairment are made to the base tables.

**Rule 10:** *Integrity independence*:

Integrity constraints specific to a particular relational data base must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.

**Rule 11:** *Distribution independence*:

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only.

**Rule 12:** The *nonsubversion rule*:

If a relational system has a low-level (single-record-at-a-time) language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher level relational language (multiple-records-at-a-time).

**Q-5 What do you understand By Data Redundancy?**

Data redundancy **occurs when the same piece of data exists in multiple places**, whereas data inconsistency is when the same data exists in different formats in multiple tables. Unfortunately, data redundancy can cause data inconsistency, which can provide a company with unreliable and/or meaningless information.

**Q-6 What is DDL Interpreter?**

DDL Interpreter: **It processes the DDL statements into a set of table containing meta data (data about data)**. Embedded DML Pre-compiler: It processes DML statements embedded in an application program into procedural calls. Query Optimizer: It executes the instruction generated by DML Compiler.

**Q-7 What is DML Compiler in SQL?**

DML Compiler: **It processes the DML statements into low level instruction (machine language), so that they can be executed**. DDL Interpreter: It processes the DDL statements into a set of table containing meta data (data about data).

**Q-8 What is 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 following constraints are commonly used in SQL:

* [NOT NULL](https://www.w3schools.com/sql/sql_notnull.asp) - Ensures that a column cannot have a NULL value
* [UNIQUE](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different
* [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
* [FOREIGN KEY](https://www.w3schools.com/sql/sql_foreignkey.asp) - Prevents actions that would destroy links between tables
* [CHECK](https://www.w3schools.com/sql/sql_check.asp) - Ensures that the values in a column satisfies a specific condition
* [DEFAULT](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column if no value is specified
* [CREATE INDEX](https://www.w3schools.com/sql/sql_create_index.asp) - Used to create and retrieve data from the database very quickly

**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.

To create a table in the database, first, we need to select the database in which we want to create a table.

* mysql> USE dbs;

Then we will write a query to create a table named student in the selected database 'dbs'.

* mysql> **CREATE** **TABLE** student(ID **INT**, **Name** **VARCHAR**(20), Percentage **INT**, Location **VARCHAR**(20), DateOfBirth **DATE**);

Now, we will write a single query to insert multiple records in the student table:

mysql> **INSERT** **INTO** student(ID, **Name**, Percentage, Location, DateOfBirth)

**VALUES**

(1, "Manthan Koli", 79, "Delhi", "2003-08-20"),

(2, "Dev Dixit", 75, "Pune", "1999-06-17"),

(3, "Aakash Deshmukh", 87, "Mumbai", "1997-09-12"),

(4, "Aaryan Jaiswal", 90, "Chennai", "2005-10-02"),

(5, "Rahul Khanna", 92, "Ambala", "1996-03-04"),

(6, "Pankaj Deshmukh", 67, "Kanpur", "2000-02-02"),

(7, "Gaurav Kumar", 84, "Chandigarh", "1998-07-06"),

(8, "Sanket Jain", 61, "Shimla", "1990-09-08"),

Q-10 What is trigger and how to create a Trigger in SQL?

Creates a DML, DDL, or logon trigger. A trigger is a special type of stored procedure that automatically runs when an event occurs in the database server. DML triggers run when a user tries to modify data through a data manipulation language (DML) event. DML events are INSERT, UPDATE, or DELETE statements on a table or view. These triggers fire when any valid event fires, whether table rows are affected or not. For more information, see [DML Triggers](https://learn.microsoft.com/en-us/sql/relational-databases/triggers/dml-triggers?view=sql-server-ver16).

DDL triggers run in response to a variety of data definition language (DDL) events. These events primarily correspond to Transact-SQL CREATE, ALTER, and DROP statements, and certain system stored procedures that perform DDL-like operations.

Logon triggers fire in response to the LOGON event that's raised when a user's session is being established. You can create triggers directly from Transact-SQL statements or from methods of assemblies that are created in the Microsoft .NET Framework common language runtime (CLR) and uploaded to an instance of SQL Server. SQL Server lets you create multiple triggers for any specific statement.

**SQL Server Syntax**

Syntaxsql

-- SQL Server Syntax

-- Trigger on an INSERT, UPDATE, or DELETE statement to a table or view (DML Trigger)

CREATE [ OR ALTER ] TRIGGER [ schema\_name . ]trigger\_name

ON { table | view }

[ WITH <dml\_trigger\_option> [ ,...n ] ]

{ FOR | AFTER | INSTEAD OF }

{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }

[ WITH APPEND ]

[ NOT FOR REPLICATION ]

AS { sql\_statement [ ; ] [ ,...n ] | EXTERNAL NAME <method specifier [ ; ] > }

<dml\_trigger\_option> ::=

[ ENCRYPTION ]

[ EXECUTE AS Clause ]

<method\_specifier> ::=

assembly\_name.class\_name.method\_name

Syntaxsql

-- SQL Server Syntax

-- Trigger on an INSERT, UPDATE, or DELETE statement to a

-- table (DML Trigger on memory-optimized tables)

CREATE [ OR ALTER ] TRIGGER [ schema\_name . ]trigger\_name

ON { table }

[ WITH <dml\_trigger\_option> [ ,...n ] ]

{ FOR | AFTER }

{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }

AS { sql\_statement [ ; ] [ ,...n ] }

<dml\_trigger\_option> ::=

[ NATIVE\_COMPILATION ]

[ SCHEMABINDING ]

[ EXECUTE AS Clause ]

Syntaxsql

-- Trigger on a CREATE, ALTER, DROP, GRANT, DENY,

-- REVOKE or UPDATE statement (DDL Trigger)

CREATE [ OR ALTER ] TRIGGER trigger\_name

ON { ALL SERVER | DATABASE }

[ WITH <ddl\_trigger\_option> [ ,...n ] ]

{ FOR | AFTER } { event\_type | event\_group } [ ,...n ]

AS { sql\_statement [ ; ] [ ,...n ] | EXTERNAL NAME < method specifier > [ ; ] }

<ddl\_trigger\_option> ::=

[ ENCRYPTION ]

[ EXECUTE AS Clause ]

Syntaxsql

-- Trigger on a LOGON event (Logon Trigger)

CREATE [ OR ALTER ] TRIGGER trigger\_name

ON ALL SERVER

[ WITH <logon\_trigger\_option> [ ,...n ] ]

{ FOR| AFTER } LOGON

AS { sql\_statement [ ; ] [ ,...n ] | EXTERNAL NAME < method specifier > [ ; ] }

<logon\_trigger\_option> ::=

[ ENCRYPTION ]

[ EXECUTE AS Clause ]

**Azure SQL Database Syntax**

Syntaxsql

-- Azure SQL Database Syntax

-- Trigger on an INSERT, UPDATE, or DELETE statement to a table or view (DML Trigger)

CREATE [ OR ALTER ] TRIGGER [ schema\_name . ]trigger\_name

ON { table | view }

[ WITH <dml\_trigger\_option> [ ,...n ] ]

{ FOR | AFTER | INSTEAD OF }

{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }

AS { sql\_statement [ ; ] [ ,...n ] [ ; ] > }

<dml\_trigger\_option> ::=

[ EXECUTE AS Clause ]

Syntaxsql

-- Azure SQL Database Syntax

-- Trigger on a CREATE, ALTER, DROP, GRANT, DENY,

-- REVOKE, or UPDATE STATISTICS statement (DDL Trigger)

CREATE [ OR ALTER ] TRIGGER trigger\_name

ON { DATABASE }

[ WITH <ddl\_trigger\_option> [ ,...n ] ]

{ FOR | AFTER } { event\_type | event\_group } [ ,...n ]

AS { sql\_statement [ ; ] [ ,...n ] [ ; ] }

<ddl\_trigger\_option> ::=

[ EXECUTE AS Clause ]