Database Concepts and Interview Questions

What is a Database?

- A database is an organized collection of structured information, or data, typically stored electronically in a computer system.
- A database is usually controlled by a database management system (DBMS).
- Most databases use structured query language (SQL) for writing and querying data.

What are Anomalies in db?

Anomalies in a database typically refer to inconsistencies, errors, or unexpected patterns within the data stored in the database.

- **Insertion Anomaly:** This occurs when certain attributes cannot be inserted into the database without the presence of other unrelated attributes. For example, if a database requires a customer to have at least one order to be entered, it would create an insertion anomaly because a customer may exist without placing an order.

- **Deletion Anomaly:** This happens when deleting a certain piece of data inadvertently removes other necessary data. For instance, if deleting an order removes the associated customer information, it would lead to a deletion anomaly.

- **Update Anomaly:** An update anomaly occurs when modifying data in a database leads to inconsistencies or unexpected changes. For example, if a product's price is stored in multiple places and updating it in one location while forgetting to update it in the others results in inconsistent pricing.

SQL JOIN:

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Different Types of SQL JOINs:

Here are the different types of the JOINs in SQL:

- **(INNER) JOIN:** Returns records that have matching values in both tables.

Query: SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders, Customers Where Orders.CustomerID=Customers.CustomerID;

- **LEFT (OUTER) JOIN:** Returns all records from the left table, and the matched records from the right table.

Query: SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate FROM Orders, Customers Where Orders.CustomerID=Customers.CustomerID(+);

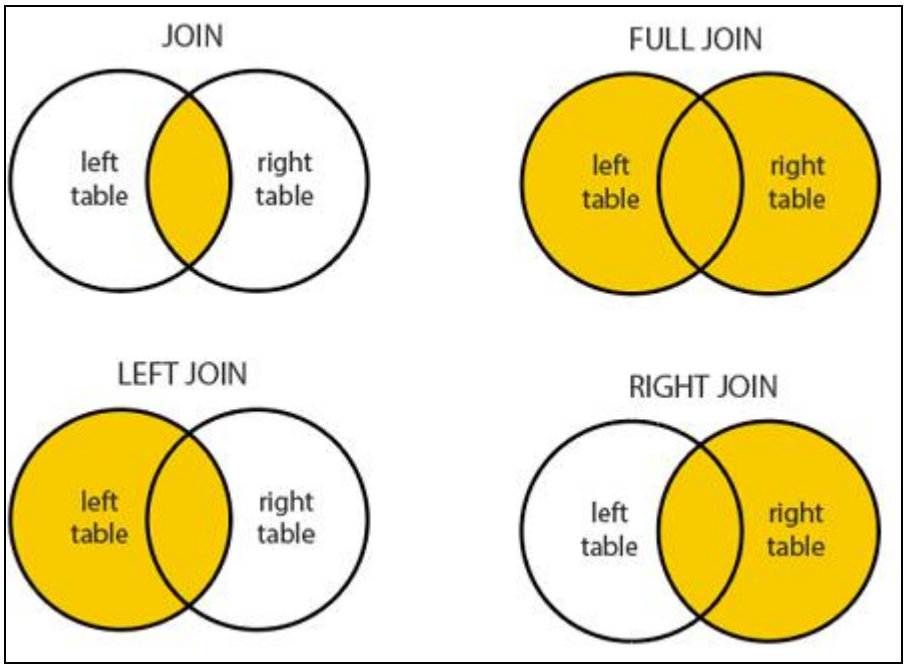
- **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table.

Query: SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate FROM Orders, Customers Where Orders.CustomerID(+)=Customers.CustomerID;

- **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table

Query: SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate FROM Orders, Customers;

- **SELF JOIN**: A join in which a table is joined with itself (which is also called Unary relationships).



Normalization:

Normalization is a process used in database design to eliminate redundancy and dependency issues within the data, resulting in a more organized and efficient database structure. It

involves breaking down a larger table into smaller, related tables and establishing relationships between them.

Normalization is used for mainly two purposes,

1) Eliminating redundant(useless) data.

2) Ensuring data dependencies make sense i.e data is logically stored.

Normalization Rules:

Normalization rules are divided into the following normal forms:

1. First Normal Form
2. Second Normal Form
3. Third Normal Form
4. BCNF
5. Fourth Normal Form

First Normal Form (1NF)

For a table to be in the First Normal Form, it should follow the following 4 rules:

1. It should only have single(atomic) valued attributes/columns.
2. Values stored in a column should be of the same domain.
3. All the columns in a table should have unique names.
4. And the order in which data is stored, does not matter.

Second Normal Form (2NF)

For a table to be in the Second Normal Form,

1. It should be in the First Normal form.
2. And, it should not have Partial Dependency.

Partial Dependency: Partial dependency in database normalization happens when a non-key attribute relies on only a portion of the primary key, rather than the entire key.

Example: Suppose we have a table called "**Orders**" with the following attributes:

OrderID (Primary Key)
ProductID (Primary Key)
ProductName
CustomerID
CustomerName
In this case, the primary key consists of both **OrderID** and **ProductID**. However, if we observe that the attribute **CustomerName** is functionally dependent only on **CustomerID** and not on the entire composite primary key, we have a partial dependency.

To resolve this partial dependency, we would need to decompose the table into two separate tables.

Table 1: Orders
OrderID (Primary Key)
ProductID (Primary Key)
CustomerID (Foreign Key)

Table 2: Customers
CustomerID (Primary Key)
CustomerName
By splitting the original table, we remove the partial dependency issue, as **CustomerName** is now fully dependent on the primary key of the Customers table.

Third Normal Form (3NF)

A table is said to be in the Third Normal Form when,

1. It is in the Second Normal form.

2. And, it doesn't have Transitive Dependency.

Transitive dependency: Transitive dependency is a concept in database normalization that occurs when an attribute depends on another non-key attribute rather than directly depending on the primary key.

Example: Consider a table called "Employees" with the following attributes:

EmployeeID (Primary Key)

DepartmentID

DepartmentName

ManagerID

ManagerName

In this case, the **DepartmentName** attribute is functionally dependent on the **DepartmentID** attribute, while the **ManagerName** attribute is functionally dependent on the **ManagerID** attribute. However, the **ManagerID** itself depends on the **DepartmentID**. This creates a transitive dependency, as the value of **ManagerName** indirectly relies on the **DepartmentID**.

To resolve this transitive dependency, we would need to decompose the table into separate tables.

Table 1: Employees

EmployeeID (Primary Key)

DepartmentID (Foreign Key)

ManagerID (Foreign Key)

Table 2: Departments

DepartmentID (Primary Key)

DepartmentName

Table 3: Managers

ManagerID (Primary Key)

ManagerName

Boyce and Codd Normal Form (BCNF)

BCNF ensures that no non-key attribute is functionally dependent on any other non-key attribute. If such dependencies exist, it indicates that the table can be further decomposed into multiple tables to eliminate the violation of BCNF.

Achieving BCNF involves decomposing the table by splitting it into multiple smaller tables based on the functional dependencies. Each new table will have a primary key and include attributes that are fully functionally dependent on that key.

What is a Relational Database?

Relational database means the data is stored as well as retrieved in the form of relations (tables).

These are some important terminologies that are used in terms of relation.

Attribute: A characteristic or property of a database entity or table column.

Tuple: A single row or record in a database table that contains a collection of related attributes.

Degree: The number of attributes or columns in a table.

Cardinality: The number of tuples or rows in a table, indicating the size or extent of the table's data.

SQL:

SQL(Structured Query Language) is the programming language for relational databases like MySQL, Oracle, Sybase, SQL Server, Postgre, etc.

SQL vs NoSQL Databases:

1. **SQL** databases are table based databases whereas **NoSQL** databases can be document based, key-value pairs, graph databases.
2. **SQL** databases are vertically scalable while **NoSQL** databases are horizontally scalable.
3. **SQL** databases have a predefined schema whereas **NoSQL** databases use dynamic schema for unstructured data.

What are Keys?

A DBMS key is an attribute or set of an attribute which helps you to identify a row(tuple) in a relation(table). They allow you to find the relation between two tables. Keys help you uniquely identify a row in a table by a combination of one or more columns in that table.

Types of Keys:

Super key
Primary key
Foreign key
Compound key

What is the Super key?

A super key is a set of one or more attributes (columns) in a database table that

can uniquely identify each tuple (row) in that table. It is a broader concept than the

primary key, as it may include additional attributes beyond what is necessary for uniqueness.

Example: Consider a table called "**Students**" with the following attributes:

StudentID

StudentName

Email

Phone

In this case, a super key could be the combination of **StudentID** and **Email**. However, it's worth noting that other combinations of attributes, such as **StudentID** and **Phone** or **StudentID**, **Email**, and **Phone**, could also serve as super keys for this table.

What is a Primary Key?

Primary key is a column or group of columns in a table that uniquely identify every row in that table.

Rules for defining Primary key:

- Two rows can't have the same primary key value
- It must for every row to have a primary key value.
- The primary key field cannot be null.
- The value in a primary key column can never be modified or updated if any foreign key refers to that primary key.

What is the Foreign key?

A foreign key is an attribute or set of attributes in a database table that refers to the primary key of another table.

What is the Compound key?

Compound key has two or more attributes that allow you to uniquely recognize a specific record.It is possible that each column may not be unique by itself within the database. However, when combined with the other column or columns the combination of composite keys become unique. The purpose of compound key is to uniquely identify each record in the table.

SQL commands

SQL commands are mainly categorized into five categories as:

1) DDL – Data Definition Language
2) DQL – Data Query Language
3) DML – Data Manipulation Language
4) DCL – Data Control Language
5) TCL – Transaction Control Language.

DDL(Data Definition Language) : DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema.

Examples:

CREATE – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).

DROP – is used to delete objects from the database.

ALTER – is used to alter the structure of the database.

TRUNCATE – is used to remove all records from a table, including all spaces allocated for the records are removed.

COMMENT –is used to add comments to the data dictionary.

RENAME – is used to rename an object existing in the database.

DQL (Data Query Language) :

DML statements are used for performing queries on the data.

Example:

SELECT – is used to retrieve data from the a database.

DML(Data Manipulation Language) : The SQL commands that deals with the manipulation of data.

Examples:

INSERT – is used to insert data into a table.

UPDATE – is used to update existing data within a table.

DELETE – is used to delete records from a database table.

Truncate vs Delete vs DROP?

Delete: SQL statement used to remove specific rows from a table while preserving the table structure and other data.

Drop: SQL statement used to remove an entire table from the database, including its structure and all data contained within it.

Truncate: SQL statement used to remove all rows from a table, effectively resetting the table's data, but preserving the table structure and associated indexes.

DCL(Data Control Language): DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

Examples:

GRANT - gives user's access privileges to the database.

REVOKE - withdraw user's access privileges given by using the

TCL(transaction Control Language) : TCL commands deals with the transaction within the database.

Examples:

COMMIT – commits a Transaction.

ROLLBACK – rollbacks a transaction in case of any error occurs.

SAVEPOINT – sets a savepoint within a transaction.

What are ACID properties?

Atomicity: Ensures that a transaction is treated as a single, indivisible unit of work, either fully completed or fully rolled back.

Consistency: Ensures that a transaction brings the database from one valid state to another, preserving the defined integrity constraints.

Isolation: Ensures that concurrent transactions are isolated from each other, maintaining data integrity and preventing interference.

Durability: Ensures that once a transaction is committed, its changes are permanent and will survive any subsequent failures or system restarts.

SQL Views

view is a virtual table derived from one or more tables or other views. It does not store data itself but rather presents the data from the underlying tables in a structured and customized way.

Advantages of Views:

- **Simplified Data Access:** Views provide a simplified and customized way to access data by presenting a specific subset of columns or rows from one or more tables. This can make querying and retrieving data more convenient and efficient.

- **Data Abstraction:** Views act as an abstraction layer, allowing users to interact with the data without directly accessing the underlying table structures. This provides a level of separation between the logical and physical representation of data.

- **Security and Access Control:** Views can be used to enforce data security by limiting access to specific columns or rows. It allows granting users permission to access only the relevant data they need while hiding sensitive information.

Disadvantages of Views:

- The view becomes irrelevant if we drop a table related to that view.
- Much memory space is occupied when the view is created for large tables.

Example:

```
CREATE VIEW DetailsView AS SELECT NAME, ADDRESS FROM StudentDetails WHERE S_ID < 5;
```

To see the data in the View,

```
SELECT * FROM DetailsView;
```

To drop a view,

```
DROP INDEX index_name;
```

What are Triggers?

A trigger is a stored procedure in a database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

Syntax:

```
create trigger [trigger_name]
[before | after]
{insert | update | delete}
on [table_name]
[for each row]
[trigger_body]
```

Example:

```
CREATE TRIGGER InsertOrderTrigger AFTER INSERT ON Orders
FOR EACH ROW
BEGIN
  INSERT INTO OrderAuditLog (OrderID, AuditDate, Action) VALUES (NEW.OrderID, NOW(), 'Inserted');
END;
```

BEFORE and AFTER of Trigger:

- BEFORE triggers run the trigger action before the triggering statement is run.
- AFTER triggers run the trigger action after the triggering statement is run.

Procedure:

A procedure is a named collection of SQL statements that are stored and can be executed as a unit.

Reusability: Procedures allow you to define a set of SQL statements that can be reused multiple times throughout an application or database system. This promotes code reuse and avoids redundancy.

Example:

```
CREATE PROCEDURE GetProductsByPrice(IN minPrice DECIMAL(10, 2))

BEGIN

  SELECT ProductName, Price FROM Products WHERE Price > minPrice;

END;
```