Q1. What is a database? Differentiate between SQL and NoSQL databases.

A database is a structured collection of data that is organized and stored in a way that allows for efficient retrieval, manipulation, and management of data. It provides a means for storing, managing, and accessing large amounts of information. Databases are widely used in various applications and systems, ranging from simple personal data management tools to complex enterprise systems.

SQL and NoSQL are two different types of database management systems (DBMS) that differ in their data models, query languages, and design philosophies. Here's a brief comparison between SQL and NoSQL databases:

SQL Databases:

1. Structure: SQL databases are based on the relational model, where data is organized into tables with predefined schemas. These databases enforce strong data consistency and adhere to ACID (Atomicity, Consistency, Isolation, Durability) properties.
2. Query Language: SQL (Structured Query Language) is used to communicate with SQL databases. It provides a standardized way to query and manipulate data, making it easy to perform complex joins, aggregations, and transactions.
3. Scalability: SQL databases typically excel in vertical scalability, meaning they can handle a growing workload by adding more resources (such as CPU, memory) to a single server. Horizontal scalability can also be achieved through sharding or replication techniques, but it requires additional effort.
4. Schema Definition: SQL databases require a predefined schema, which defines the structure of the data and its relationships. Any changes to the schema usually involve altering the table structure, which can be complex and time-consuming.

NoSQL Databases:

1. Structure: NoSQL databases use various data models, such as key-value pairs, documents, column-family, or graph structures. They provide more flexibility in data representation and can handle unstructured or semi-structured data effectively.
2. Query Language: NoSQL databases may have their own query languages, but they often provide simpler APIs (Application Programming Interfaces) for data access and manipulation. Querying capabilities are usually more limited compared to SQL databases, especially for complex joins and aggregations.
3. Scalability: NoSQL databases are designed to scale horizontally, allowing them to distribute data across multiple servers easily. They can handle large volumes of data and high write loads by adding more nodes to the system.
4. Schema Definition: NoSQL databases are schema-less or have flexible schemas. They allow for dynamic changes to the data structure, making it easier to adapt to evolving requirements without requiring extensive schema modifications.

The choice between SQL and NoSQL databases depends on the specific needs of an application. SQL databases are typically favored when data consistency, complex querying, and transactions are critical. NoSQL databases are often used in scenarios that require scalability, high performance, and flexible data models, such as in web applications, real-time analytics, or handling large amounts of unstructured data.

Q2. What is DDL? Explain why CREATE, DROP, ALTER, and TRUNCATE are used with an example.

DDL stands for Data Definition Language. It is a subset of SQL (Structured Query Language) that is used to define and manage the structure of a database, including creating, modifying, and deleting database objects such as tables, indexes, and views.

Here's an explanation of the four commonly used DDL statements:

1. CREATE: The CREATE statement is used to create new database objects such as tables, views, indexes, or schemas. For example, to create a table named "Employees" with columns for employee ID, name, and salary, you can use the following SQL statement:

**CREATE TABLE Employees (**

**EmployeeID INT,**

**Name VARCHAR(50),**

**Salary DECIMAL(10,2)**

**);**

1. DROP: The DROP statement is used to delete existing database objects. It permanently removes the specified object and all its associated data from the database. For example, to delete the "Employees" table created in the previous example, you can use the following SQL statement:

DROP TABLE Employees;

1. ALTER: The ALTER statement is used to modify the structure of an existing database object. It allows you to add, modify, or drop columns in a table, change data types, or rename objects. For example, to add a new column named "Department" to the "Employees" table, you can use the following SQL statement:

ALTER TABLE Employees

ADD Department VARCHAR(50);

1. TRUNCATE: The TRUNCATE statement is used to delete all data from a table while keeping its structure intact. Unlike the DROP statement, TRUNCATE retains the table structure and can be used to quickly remove all rows from a table. For example, to remove all data from the "Employees" table without deleting the table itself, you can use the following SQL statement:

TRUNCATE TABLE Employees;

These DDL statements are essential for managing the structure of a database and allow you to create, modify, and delete database objects according to your requirements.

Q3. What is DML? Explain INSERT, UPDATE, and DELETE with an example.

DML stands for Data Manipulation Language, and it is a subset of SQL (Structured Query Language) that is used to manage data within a database. DML consists of three primary commands: INSERT, UPDATE, and DELETE.

1. INSERT: The INSERT command is used to add new rows of data into a table. It allows you to specify the values that should be inserted into the table's columns. Here's an example:

Let's say we have a table called "Employees" with columns "EmployeeID," "FirstName," and "LastName." To insert a new employee record, we would use the following INSERT statement:

INSERT INTO Employees (EmployeeID, FirstName, LastName)

VALUES (1, 'John', 'Doe');

This statement would add a new row to the "Employees" table with the values 1, 'John', and 'Doe' in the respective columns.

1. UPDATE: The UPDATE command is used to modify existing data in a table. It allows you to update specific columns with new values based on certain conditions. Here's an example:

we want to update the last name of an employee with EmployeeID 1 in the "Employees" table. We can use the following UPDATE statement:

UPDATE Employees

SET LastName = 'Smith'

WHERE EmployeeID = 1;

This statement would change the last name of the employee with EmployeeID 1 from 'Doe' to 'Smith'.

1. DELETE: The DELETE command is used to remove one or more rows from a table. It allows you to specify conditions that determine which rows should be deleted. Here's an example:

we want to delete all employees from the "Employees" table whose last name is 'Smith.' We can use the following DELETE statement:

DELETE FROM Employees

WHERE LastName = 'Smith';

This statement would remove all rows from the "Employees" table where the last name is 'Smith'.

In summary, INSERT is used to add new data, UPDATE is used to modify existing data, and DELETE is used to remove data from a table. These DML commands provide essential functionality for manipulating data within a database.

Q4. What is DQL? Explain SELECT with an example.

DQL stands for Data Query Language. It is a sublanguage of SQL (Structured Query Language) that is primarily used for querying and retrieving data from a relational database management system (RDBMS). DQL statements are used to specify the data that needs to be retrieved from the database.

One of the most commonly used DQL statements is the SELECT statement. It is used to retrieve data from one or more tables in a database. The SELECT statement allows you to specify the columns you want to retrieve, the table from which you want to retrieve data, and any conditions that need to be met for the data to be included in the result set.

Here's an example of a SELECT statement:

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

we have a table called "Employees" with columns like "EmployeeID," "FirstName," "LastName," and "Salary." We can use the SELECT statement to retrieve specific information from this table. For example, to retrieve the first and last names of all employees whose salary is above $50,000, the following query can be used:

SELECT FirstName, LastName

FROM Employees

WHERE Salary > 50000;

The result of this query will be a result set containing the first and last names of employees who meet the specified condition (salary above 50,000). The SELECT statement allows you to choose which columns to include in the result set and the WHERE clause provides the condition for filtering the data.

By using various clauses, functions, and operators within the SELECT statement, you can perform complex queries and retrieve specific data from a database based on your requirements.

Q5. Explain Primary Key and Foreign Key.

In database management, a primary key and a foreign key are both important concepts used to establish relationships between tables. They are key components of relational databases, which organize and structure data.

1. Primary Key:

A primary key is a column or a set of columns in a table that uniquely identifies each record (row) in that table. It ensures the uniqueness and integrity of the data within the table. Here are some key characteristics of a primary key:

\*Uniqueness: Each value in the primary key column(s) must be unique. No two records in the table can have the same primary key value.

\*Non-nullability: A primary key column cannot contain null (empty) values. It must have a value for each record.

\*Unchanging: The values in the primary key column(s) should generally be static and not change over time.

By designating a primary key for a table, you can enforce data integrity and provide a reference point for establishing relationships with other tables. Common examples of primary keys include social security numbers, unique identifiers, or auto-incrementing integers.

1. Foreign Key:

A foreign key is a column or a set of columns in a table that refers to the primary key of another table. It establishes a relationship between two tables, allowing data to be linked across tables. Here are some important aspects of foreign keys:

\*Referential Integrity: The foreign key references the primary key of another table, creating a link between the two. It ensures referential integrity, meaning that data stored in the foreign key column(s) must correspond to an existing value in the referenced primary key column(s) of the related table.

\*Relationship Establishment: The foreign key represents a relationship between two tables, typically indicating that the values in the foreign key column(s) of one table are related to the primary key column(s) of another table.

\*Multiple or Composite Foreign Keys: A table can have multiple foreign keys, each referencing a different primary key in another table. In some cases, a composite foreign key may be used, which consists of multiple columns that together reference the primary key of another table.

Foreign keys are used to establish relationships such as one-to-one, one-to-many, or many-to-many between tables. They enable the enforcement of referential integrity, help maintain data consistency, and allow for efficient retrieval of related data through joins.It's important to note that the primary key and foreign key concepts are specific to relational databases and may vary in other data storage systems.

Q6. Write a python code to connect MySQL to python. Explain the cursor() and execute() method.

To connect MySQL to Python, you can use the mysql-connector-python package. Make sure you have it installed by running pip install mysql-connector-python. Here's an example code snippet that connects to a MySQL database and explains the cursor() and execute() methods:

import mysql.connector

# Establishing a connection to the MySQL database

connection = mysql.connector.connect(

host="localhost",

user="your\_username",

password="your\_password",

database="your\_database"

)

# Creating a cursor object to interact with the database

cursor = connection.cursor()

# Executing SQL queries using the execute() method

cursor.execute("SELECT \* FROM your\_table")

# Fetching the results of the executed query

results = cursor.fetchall()

# Iterating over the results and printing each row

for row in results:

print(row)

# Closing the cursor and the database connection

cursor.close()

connection.close()

Explanation:

1. First, you import the mysql.connector module, which provides the necessary functionality to connect to a MySQL database.
2. You establish a connection to the MySQL database by providing the required connection details such as host, username, password, and database name. Modify the values accordingly for your setup.
3. Once the connection is established, you create a cursor object using the cursor() method. The cursor is used to execute SQL queries and fetch the results.
4. Using the execute() method of the cursor, you can execute SQL queries. In the provided example, a simple SELECT statement is executed to retrieve all rows from a table. You can modify the query as per your requirements.
5. After executing the query, you can retrieve the results using the fetchall() method, which fetches all the rows returned by the query.
6. Finally, you can iterate over the results and process them as needed. In the example, each row is printed to the console, but you can perform any required operations on the data.
7. Once you are done with the cursor and the database connection, it is essential to close them using the close() method to free up resources and ensure proper cleanup.

Note: Remember to replace the placeholders (your\_username, your\_password, your\_database, your\_table) in the code with the appropriate values for your MySQL setup.

Q7. Give the order of execution of SQL clauses in an SQL query.

In SQL, the order of execution of clauses in a query is as follows:

1. FROM: This clause specifies the table or tables from which the data will be retrieved.
2. WHERE: The WHERE clause is used to filter the rows based on a specific condition or set of conditions.
3. GROUP BY: This clause is used to group the rows based on one or more columns.
4. HAVING: The HAVING clause is used to filter the grouped rows based on a condition or set of conditions.
5. SELECT: The SELECT clause is used to specify the columns to be retrieved from the table(s).
6. DISTINCT: The DISTINCT keyword is used to eliminate duplicate rows from the result set.
7. ORDER BY: This clause is used to sort the result set based on one or more columns.
8. LIMIT/OFFSET: The LIMIT clause is used to restrict the number of rows returned by the query, while the OFFSET clause is used to skip a certain number of rows before starting to return the result set.

It's important to note that not all clauses are required in every SQL query. The basic structure of a query consists of the SELECT and FROM clauses, while the other clauses are optional and can be included as needed to refine the query results. Additionally, some clauses like GROUP BY, HAVING, DISTINCT, and ORDER BY may not always be present in a query depending on the specific requirements.