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

Ans: A database is a structured collection of data that is organized and stored in a way that allows efficient retrieval, management, and manipulation of the data. It serves as a central repository for storing, managing, and querying large volumes of data.

SQL and NoSQL are two different types of database management systems (DBMS) that differ in their data models, querying languages, and usage scenarios.

1. SQL Databases:

SQL (Structured Query Language) databases are based on the relational data model. They use tables to store data, and these tables consist of rows and columns. SQL databases adhere to a predefined schema that defines the structure of the data and the relationships between tables. They use SQL as the standard language for querying and manipulating the data.

Key characteristics of SQL databases include:

- Structure: Data is organized into tables with a predefined schema.

- Schema enforcement: The schema defines the structure and constraints on the data, ensuring data integrity.

- Relationships: SQL databases support relationships between tables using primary and foreign keys.

- ACID compliance: ACID stands for Atomicity, Consistency, Isolation, and Durability. SQL databases ensure transactional consistency and reliability.

- Scalability: SQL databases can handle structured data and are well-suited for complex queries involving multiple tables.

Some popular SQL database systems include MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server, and SQLite.

2. NoSQL Databases:

NoSQL (Not Only SQL) databases offer a more flexible approach to data storage and retrieval. They are designed to handle large volumes of unstructured or semi-structured data. NoSQL databases can store data in various formats, such as key-value pairs, documents, column-family, and graphs.

Key characteristics of NoSQL databases include:

- Schema flexibility: NoSQL databases do not enforce a fixed schema, allowing for dynamic and evolving data structures.

- Scalability: They are designed to scale horizontally, meaning they can handle large amounts of data across multiple servers.

- High performance: NoSQL databases are optimized for high-speed read and write operations.

- Distributed architecture: They can distribute data across multiple servers to ensure fault tolerance and high availability.

- Non-relational: NoSQL databases do not rely on SQL as the primary querying language. Each type of NoSQL database has its own query language or API.

There are several types of NoSQL databases, including key-value stores (e.g., Redis), document databases (e.g., MongoDB), columnar databases (e.g., Apache Cassandra), and graph databases (e.g., Neo4j).

The choice between SQL and NoSQL databases depends on factors such as the nature of the data, scalability requirements, performance needs, and the complexity of the queries and relationships involved.

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

Ans: 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 and its objects. DDL statements are responsible for creating, modifying, and deleting database objects such as tables, views, indexes, and constraints.

Let's explain the commonly used DDL statements with examples:

1. CREATE:

The CREATE statement is used to create new database objects, such as tables, views, indexes, or constraints. Here's an example of creating a table called "employees" with some columns:

```

CREATE TABLE employees (

id INT PRIMARY KEY,

name VARCHAR(50),

age INT,

salary DECIMAL(10,2)

);

```

In the above example, the CREATE TABLE statement is used to create a new table named "employees" with columns for ID, name, age, and salary. The data types and constraints for each column are specified.

2. DROP:

The DROP statement is used to remove an existing database object, such as a table or view. It permanently deletes the object and all associated data. Here's an example of dropping a table:

```

DROP TABLE employees;

In the above example, the DROP TABLE statement is used to remove the "employees" table from the database.

3. ALTER:

The ALTER statement is used to modify the structure of an existing database object, such as a table. It allows you to add, modify, or delete columns, constraints, or indexes. Here's an example of adding a new column to a table:

```

ALTER TABLE employees

ADD COLUMN department VARCHAR(50);

```

In the above example, the ALTER TABLE statement is used to add a new column called "department" to the "employees" table with the VARCHAR data type.

4. TRUNCATE:

The TRUNCATE statement is used to remove all data from a table, but it keeps the structure intact. It is faster than deleting all rows using the DELETE statement because TRUNCATE doesn't generate individual row-level delete operations. Here's an example of truncating a table:

```

TRUNCATE TABLE employees;

```

In the above example, the TRUNCATE TABLE statement is used to remove all data from the "employees" table, but the table structure remains.

These DDL statements are essential for managing the structure and definition of database objects, allowing you to create, modify, or remove tables, views, indexes, and other database components.

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

Ans: DML stands for Data Manipulation Language. It is a subset of SQL (Structured Query Language) that is used to manipulate and modify data within a database. DML statements are responsible for inserting, updating, and deleting data in database tables.

Let's explain the commonly used DML statements with examples:

1. INSERT:

The INSERT statement is used to insert new rows of data into a table. It allows you to specify the values for each column or insert data from another table. Here's an example of inserting a new row into a table named "employees":

```

INSERT INTO employees (id, name, age, salary)

VALUES (1, 'John Doe', 30, 50000);

```

In the above example, the INSERT INTO statement is used to insert a new row into the "employees" table. The values for the id, name, age, and salary columns are specified.

2. UPDATE:

The UPDATE statement is used to modify existing data in a table. It allows you to update one or more columns with new values based on specified conditions. Here's an example of updating the salary for an employee with a specific ID:

```

UPDATE employees

SET salary = 55000

WHERE id = 1;

```

In the above example, the UPDATE statement is used to modify the salary column for the employee with id = 1 in the "employees" table. The SET clause sets the new value for the salary column, and the WHERE clause specifies the condition for which rows to update.

3. DELETE:

The DELETE statement is used to remove one or more rows from a table. It allows you to specify conditions to determine which rows to delete. Here's an example of deleting an employee with a specific ID:

```

DELETE FROM employees

WHERE id = 1;

```

In the above example, the DELETE FROM statement is used to remove the row(s) from the "employees" table where the id is equal to 1. The WHERE clause specifies the condition for which rows to delete.

These DML statements are essential for manipulating and modifying data within database tables. INSERT allows you to add new rows, UPDATE modifies existing data, and DELETE removes rows from tables based on specified conditions.

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

Ans: DQL stands for Data Query Language. It is a subset of SQL (Structured Query Language) that is used to retrieve and query data from a database. DQL statements are primarily focused on selecting and fetching data from one or more tables.

The most commonly used DQL statement is the SELECT statement, which allows you to retrieve data from one or more tables based on specified criteria. Here's an example of using the SELECT statement:

```

SELECT \* FROM employees;

```

In the above example, the SELECT statement is used to retrieve all columns (indicated by the asterisk \*) from the "employees" table. This query will return all rows and columns from the table.

You can also select specific columns instead of all using a comma-separated list of column names:

```

SELECT name, age, salary FROM employees;

```

In this example, the SELECT statement retrieves only the "name," "age," and "salary" columns from the "employees" table.

The SELECT statement can also include conditions using the WHERE clause to filter the retrieved data based on specific criteria. For example:

```

SELECT name, age, salary FROM employees

WHERE age > 30;

```

In this case, the query selects the "name," "age," and "salary" columns from the "employees" table but only includes rows where the age is greater than 30.

Additionally, the SELECT statement can include various other clauses, such as ORDER BY to sort the results, GROUP BY to group data, JOIN to combine data from multiple tables, and more. These clauses provide powerful capabilities to retrieve and manipulate data based on specific requirements.

The SELECT statement is a fundamental tool for querying data from a database. It allows you to specify the columns to retrieve, the tables to query, conditions to filter the data, and additional clauses to control the result set.

Q5. Explain Primary Key and Foreign Key.

Ans: Primary Key:

A primary key is a column or a set of columns in a database table that uniquely identifies each record or row in that table. It provides a way to ensure the uniqueness and integrity of the data within the table. Some key characteristics of a primary key are:

1. Uniqueness: Each value in the primary key column(s) must be unique, ensuring that no two rows in the table have the same primary key value.

2. Non-nullability: A primary key column cannot contain null values. Every row must have a non-null value in the primary key column(s).

3. Immutable: The values in the primary key column(s) should not change once they are assigned to a record. This helps maintain the integrity of the relationships with other tables.

4. Single-value or Composite: A primary key can be a single column or a combination of multiple columns. In the case of a composite primary key, the combination of values from multiple columns must be unique.

Primary keys are used to uniquely identify records in a table and establish relationships between tables using foreign keys. They also facilitate efficient indexing and retrieval of data.

Foreign Key:

A foreign key is a column or a set of columns in a database table that refers to the primary key of another table. It establishes a relationship between two tables, known as a parent-child relationship. The foreign key in the child table refers to the primary key in the parent table. Here are some key aspects of foreign keys:

1. Referential Integrity: The foreign key ensures referential integrity, meaning that the values in the foreign key column(s) must exist in the corresponding primary key column(s) of the referenced table. This enforces data consistency and prevents orphaned or inconsistent data.

2. Relationship establishment: By using foreign keys, tables can be related to each other based on the primary key and foreign key relationship. This allows for data retrieval and manipulation across multiple tables using joins.

3. Cascading Actions: Foreign keys can define cascading actions, such as ON DELETE and ON UPDATE, which specify what should happen when referenced data is deleted or updated. Actions can include cascading deletes or updates, setting null values, or restricting the operation based on the referential integrity rules.

Foreign keys are crucial for maintaining data integrity and establishing relationships between tables in a relational database. They enable efficient querying and ensure consistency in data references between related tables.

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

Ans: To connect Python with MySQL, you need to install the MySQL Connector Python module. You can install it using pip with the following command:

```

pip install mysql-connector-python

```

Once you have installed the module, you can use the following Python code to connect to MySQL:

```python

import mysql.connector

# Establishing the connection

cnx = mysql.connector.connect(

user='your\_username',

password='your\_password',

host='your\_host',

database='your\_database'

)

# Creating a cursor object

cursor = cnx.cursor()

# Executing a SQL query

query = "SELECT \* FROM your\_table"

cursor.execute(query)

# Fetching the results

results = cursor.fetchall()

# Printing the results

for row in results:

print(row)

# Closing the cursor and connection

cursor.close()

cnx.close()

```

Explanation of cursor() and execute() method:

1. cursor():

The `cursor()` method is used to create a cursor object in MySQL Connector Python. The cursor object allows you to execute SQL queries and fetch the results. It provides various methods to interact with the database, such as `execute()`, `fetchone()`, `fetchall()`, etc. In the code example, `cursor = cnx.cursor()` creates a cursor object.

2. execute():

The `execute()` method is used to execute a SQL query or statement. It takes the SQL query as a parameter and executes it on the MySQL server. In the code example, `cursor.execute(query)` executes the SQL query stored in the `query` variable. The result of the execution can be fetched using methods like `fetchone()` or `fetchall()`.

After executing the query, you can fetch the results using methods like `fetchone()` (to fetch a single row) or `fetchall()` (to fetch all rows). In the code example, `results = cursor.fetchall()` retrieves all the rows returned by the query.

Once you have fetched the results or performed any necessary operations, it is important to close the cursor and the connection using `cursor.close()` and `cnx.close()` to release resources and close the connection to the MySQL server.

Remember to replace `'your\_username'`, `'your\_password'`, `'your\_host'`, `'your\_database'`, and `'your\_table'` with the actual credentials and table details specific to your MySQL setup.

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

Ans: The order of execution of SQL clauses in an SQL query typically follows the sequence outlined below:

1. FROM: The FROM clause specifies the table(s) from which the data will be retrieved. It identifies the table(s) involved in the query.

2. WHERE: The WHERE clause filters the rows from the specified table(s) based on the specified conditions. It narrows down the result set by applying logical conditions to the data.

3. GROUP BY: The GROUP BY clause is used to group rows based on one or more columns. It creates groups based on common values in the specified column(s).

4. HAVING: The HAVING clause is used to filter the grouped rows based on conditions. It works similarly to the WHERE clause but operates on the grouped data.

5. SELECT: The SELECT clause selects the columns that should be included in the result set. It determines the specific data to be retrieved from the table(s).

6. DISTINCT: The DISTINCT keyword, if used, eliminates duplicate rows from the result set. It ensures that only unique rows are returned.

7. ORDER BY: The ORDER BY clause sorts the result set based on one or more columns in ascending (ASC) or descending (DESC) order. It arranges the rows based on the specified column(s).

8. LIMIT/OFFSET: The LIMIT clause, optionally combined with the OFFSET clause, limits the number of rows returned by the query. It is used for pagination or to restrict the result set to a specific number of rows.

It is important to note that not all clauses are required in every SQL query. The presence or absence of certain clauses depends on the specific requirements of the query. The given order represents a typical sequence, but there may be cases where the clauses are rearranged or certain clauses are not included at all.