Assignment 12

October 6, 2023

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 for efficient retrieval, updating, and management of that data. Databases are used to store, manage, and manipulate data in various applications and systems, from simple address books to complex enterprise-level systems.

SQL (Structured Query Language) and NoSQL (Not Only SQL) are two broad categories of database management systems, each with its own characteristics and use cases. Here's a differentiation between the two:

Data Model:

SQL databases use a relational data model. Data is organized into tables with rows and columns, and relationships between tables are defined by keys (primary keys and foreign keys). NoSQL databases use various data models, including document-oriented, key-value, column-family, and graph-based models. These models are more flexible and can handle unstructured or semi-structured data.

Schema:

SQL databases have a fixed schema, which means that the structure of the data (table schema) must be defined before data can be inserted. Any changes to the schema may require altering existing data. NoSQL databases typically have a dynamic or schema-less approach. They allow you to insert data without a predefined schema, which can be advantageous when dealing with evolving or unpredictable data structures.

Query Language:

SQL databases use the SQL language for querying and manipulating data. SQL provides powerful and standardized querying capabilities, making it well-suited for complex queries. NoSQL databases have their own query languages or APIs tailored to their specific data models. These query languages may not be as standardized or feature-rich as SQL but are designed for efficient access to data in their respective models.

Scalability:

SQL databases are traditionally scaled vertically by adding more resources (CPU, RAM) to a single server. They can be challenging to scale horizontally (across multiple servers) for handling high traffic loads. NoSQL databases are often designed for horizontal scalability. They can distribute data across multiple servers or nodes, making it easier to handle large amounts of data and high throughput.

Use Cases:

SQL databases are well-suited for applications where data structure is stable and the need for complex querying is high, such as traditional relational database applications, financial systems, and reporting. NoSQL databases are preferred for applications with rapidly changing data, high scalability requirements, and where flexibility in data modeling is crucial, such as social media platforms, real-time analytics, and IoT (Internet of Things) applications.

[ ]:
```

```

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

ANS:

DDL stands for Data Definition Language, which is a subset of SQL (Structured Query Language) used for defining and managing the structure of a database. DDL statements are responsible for creating, modifying, and deleting database objects such as tables, indexes, and constraints. Here are explanations of the commonly used DDL statements:

CREATE:

The CREATE statement is used to create new database objects, such as tables, views, indexes, and schemas. Example: Creating a new table named "employees" in an SQL database:

[ ]:
```
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    hire_date DATE
);
```

DROP:

The DROP statement is used to delete existing database objects, such as tables, views, or indexes. Example: Deleting the "employees" table:

[ ]:
```
DROP TABLE employees;
```

ALTER:

The ALTER statement is used to modify the structure of an existing database object, such as adding, modifying, or dropping columns from a table. Example: Adding a new column "email" to the "employees" table:

[ ]:
```
ALTER TABLE employees
ADD COLUMN email VARCHAR(100);
```

TRUNCATE:

The TRUNCATE statement is used to delete all the rows from a table but retains the table structure. Example: Truncating the "employees" table:

[ ]:
```
TRUNCATE TABLE employees;
```

[ ]:
```

```

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

ANS:

DML stands for Data Manipulation Language, which is a subset of SQL (Structured Query Language) used for manipulating data stored within a database. DML statements are responsible for inserting, updating, and deleting data in database tables. Here are explanations of the commonly used DML statements:

INSERT:

The INSERT statement is used to add new rows (records) of data into a table. Example: Inserting a new employee record into the "employees" table:

[ ]:
```
INSERT INTO employees (emp_id, first_name, last_name, hire_date)
VALUES (101, 'John', 'Doe', '2023-10-06');
```

UPDATE:

The UPDATE statement is used to modify existing data in a table. Example: Updating the hire date of an employee with the ID 101 in the "employees" table:

[ ]:
```
UPDATE employees
SET hire_date = '2023-11-15'
WHERE emp_id = 101;
```

DELETE:

The DELETE statement is used to remove one or more rows from a table based on a specified condition. Example: Deleting an employee record with the ID 101 from the "employees" table:

[ ]:
```
DELETE FROM employees
WHERE emp_id = 101;
```

[ ]:
```

```

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

ANS:

DQL stands for Data Query Language, which is a subset of SQL (Structured Query Language) used for retrieving and querying data from a database. The primary DQL statement is SELECT, which allows you to specify the data you want to retrieve from one or more database tables. Here's an explanation of the SELECT statement with an example:

SELECT Statement:

The SELECT statement is used to retrieve data from one or more tables in a database. It can specify which columns to include in the result, filter rows based on conditions, and even join multiple tables to combine data. The basic syntax of the SELECT statement is as follows:

You can select specific columns by listing their names after the SELECT keyword, and you specify the table(s) from which to retrieve data after the FROM keyword. The WHERE clause is optional but allows you to filter rows based on specified conditions.

[ ]:
```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Q5. Explain Primary Key and Foreign Key.

ANS:

Primary Key and Foreign Key are two important concepts in relational databases that define relationships between tables and help maintain data integrity. Here's an explanation of each:

Primary Key (PK):

A Primary Key is a column or a set of columns in a relational database table that uniquely identifies each row or record in that table. It enforces the entity integrity constraint, ensuring that there are no duplicate or null values in the primary key column(s). Each table in a database can have only one primary key. The primary key is used to establish relationships with other tables, typically through Foreign Keys. Example: In an "employees" table, the "emp_id" column can be designated as the primary key, ensuring that each employee has a unique identifier.

[ ]:
```
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    hire_date DATE
);
```

Foreign Key (FK):

A Foreign Key is a column or a set of columns in a table that establishes a link between the data in two related tables. It creates referential integrity, ensuring that relationships between tables are consistent. The Foreign Key in one table typically references the Primary Key in another table. Foreign Keys help maintain data consistency and enforce data integrity rules by ensuring that values in the Foreign Key column(s) match values in the Primary Key column(s) of the referenced table. Foreign Keys are used to create relationships between tables, such as one-to-many or many-to-one relationships. Example: In a database that includes an "orders" table and a "customers" table, the "customer_id" column in the "orders" table can be a Foreign Key that references the "customer_id" Primary Key in the "customers" table, indicating which customer placed each order.

[ ]:
```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    order_date DATE,
    customer_id INT,
    -- Other order-related columns
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

[ ]:

```

```

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

ANS:

To connect Python to a MySQL database, you can use the mysql-connector-python library, which is a Python driver for MySQL. You can install this library using pip if you haven't already:

[2]:

```
pip install mysql-connector-python
```

```
Collecting mysql-connector-python
  Downloading mysql_connector_python-8.1.0-cp310-cp310-manylinux_2_17_x86_64.whl
(27.5 MB)
                           27.5/27.5 MB
39.9 MB/s eta 0:00:0000:0100:01
Requirement already satisfied: protobuf<=4.21.12,>=4.21.1 in
/opt/conda/lib/python3.10/site-packages (from mysql-connector-python) (4.21.11)
Installing collected packages: mysql-connector-python
Successfully installed mysql-connector-python-8.1.0
Note: you may need to restart the kernel to use updated packages.
```

Here's a Python code example to connect to a MySQL database and perform a simple SQL query using the cursor() and execute() methods:

[ ]:

```
import mysql.connector

# Define the database connection parameters
db_config = {
    "host": "your_host",
    "user": "your_username",
    "password": "your_password",
    "database": "your_database",
}

try:
    # Create a connection to the MySQL database
    connection = mysql.connector.connect(**db_config)

    # Create a cursor object to interact with the database
    cursor = connection.cursor()

    # Execute an SQL query
    sql_query = "SELECT * FROM your_table"
    cursor.execute(sql_query)

    # Fetch and print the results
    for row in cursor.fetchall():
        print(row)

except mysql.connector.Error as error:
    print(f"Error: {error}")
finally:
    # Close the cursor and connection
    if cursor:
        cursor.close()
    if connection:
        connection.close()
```

cursor() method:

The cursor() method is called on the database connection object (in this case, connection) to create a cursor object. A cursor is like a handle or pointer to the result set of a SQL query. It allows you to execute SQL statements and fetch data from the database.

execute() method:

The execute() method is called on the cursor object to execute an SQL query. It takes an SQL query as a parameter and sends it to the database for execution. After executing the query, the cursor is positioned at the beginning of the result set (if applicable).

[ ]:

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

ANS:

In an SQL query, SQL clauses are typically executed in a specific order, known as the logical order of execution. Understanding this order is crucial for writing SQL queries correctly. The general order of execution for SQL clauses is as follows:

FROM: The FROM clause specifies the table(s) from which you are retrieving data. It identifies the data source(s) for the query.

JOIN: If your query involves multiple tables and you're using JOIN operations, the JOIN clauses are executed next. They combine rows from different tables based on specified join conditions.

WHERE: The WHERE clause is used to filter rows from the result set. It specifies a condition that rows must meet to be included in the result.

GROUP BY: If you want to group rows by one or more columns, you use the GROUP BY clause. This is typically used with aggregate functions like COUNT, SUM, AVG, etc., to perform calculations on groups of rows.

HAVING: The HAVING clause filters groups of rows created by the GROUP BY clause. It's used to filter groups based on aggregate functions' results.

SELECT: The SELECT clause determines which columns or expressions are included in the result set. It specifies the data to be retrieved from the tables.

DISTINCT: If you use the DISTINCT keyword in your query, duplicate rows in the result set are removed at this stage.

ORDER BY: The ORDER BY clause specifies the order in which the result set should be sorted. It sorts the rows based on one or more columns in ascending (ASC) or descending (DESC) order.

LIMIT/OFFSET: If your database supports it (e.g., MySQL, PostgreSQL), you can use the LIMIT clause to restrict the number of rows returned in the result set. The OFFSET clause can be used in conjunction with LIMIT for pagination.

UNION/INTERSECT/EXCEPT: If your query involves set operations like UNION (combining results of multiple SELECT statements), INTERSECT (returns common rows), or EXCEPT (returns rows in the first result set but not in the second), these operations are executed at this stage.

[ ]:

```

```