1. What does SQL stand for and what are its primary functions?

   SQL : Structured Query Language

   Functions :

   - DDL : Data Defination LAngauge : CREATE, ALTER, DROP, TRUNCATE
     - To create, alter, drop and truncate tables
   - DML: Data Manipulation Language : Insert, Update Delete
   - DQL: Data Query Language : select command
     - Select command uses clause like where, group by, having, and sub clauses like in, not in, between, not between, is null, is not null.
   - TCL : Transaction Control Language :to control the database transaction with Rollback and Commit.
   - DCL : Data Control Laguage: Grant and Revoke command to to give or take back the rights.

2. Explain the difference between a primary key and a foreign key.

Primary key: Uniquely identifies rows within a table.
Foreign key: links to the primary key in another table, establishing relationships between them.

3. What is a subquery in SQL and when would you use it?

A subquery (also known as an inner query or nested query) is a query nested inside another query. It's a powerful feature that allows you to retrieve data based on the results of another query. Subqueries can be used in the SELECT, FROM, and WHERE clauses.

We can use them into

1. Filtering the data : to retrieve data based on a condition that involves another table
   a. SELECT * FROM employees WHERE department_id IN (SELECT department_id FROM departments WHERE location = 'New York');
2. Comparing against a list of data :
   a. SELECT duration FROM films WHERE duration> (SELECT AVG(duration) FROM products)

4. Can you describe what a view is in SQL and how it differs from a table?

In SQL, a view functions as a virtual table generated by the outcome of a SELECT query, differing from a table in that it lacks physical data storage. Unlike tables, views don't store data directly; rather, they offer a means to depict query results as if they were a table. Views are constructed by selecting columns from one or more tables and may involve computations, aggregations, or other operations. Once established, a view is queryable like a standard table. Noteworthy distinctions between tables and views encompass data storage, updatability, structural attributes, usage scenarios, index handling, and the existence of materialized views. While tables store actual data and allow direct updates, views lack physical storage, and the feasibility of updates depends on their complexity. Tables have a defined structure with columns, data types, constraints, and indexes, while views derive their structure from underlying tables or views. Tables are primarily used for data storage, while views serve purposes like simplifying complex queries, restricting access, or presenting data differently. Unlike tables, views don't possess independent indexes and rely on those of underlying tables. Additionally, there's a concept of materialized views in some databases, which physically store result sets for faster query performance, contrasting with traditional views, sometimes referred to as "virtual views." As an illustration, consider creating a view called EmployeeView that selects specific columns from an Employees table in the SQL language. This virtual table includes only the designated columns from the original table.

What is a stored procedure in SQL and what are its advantages?

The piece of SQL sentences which is prepared to do a specific task to avoid redundancy in the scripts .

Advantages :
Code reusability : reusable codes
Security : less errors due to decrease in code sentences.
Easy maintenance. : to change a pattern only one change is required. No need to update the whole code

Section 2

6. Write a SQL statement to create a table employees with columns id, name, role, and salary.

```
CREATE TABLE employees (id INT PRIMARY KEY,name VARCHAR(255),role VARCHAR(255),salary DECIMAL(10, 2));
```

7. How would you update the salary of a single employee in the employees table?

```
UPDATE employees SET salary = 80000 WHERE id = 45; -- Replace 45 with the actual upload
```

8. Construct a SQL query to find the third highest salary in the `employees` table.

```
SELECT DISTINCT salary FROM employees ORDER BY salary DESC LIMIT 1 OFFSET 2;
```

9. Write a SQL statement to delete all employees who have a role of 'Intern'.

```
DELETE FROM employees WHERE role = 'Intern';
```

10. Write a SQL query to list employees and their roles, but only include roles that have more than five employees.

SELECT name, role FROM employees GROUP BY name, role HAVING COUNT(*) > 5;

Section 3

1. Given a sales table, write a SQL query to list the top 3 best-selling products in the last month.

If the sales table has columns like ProductID, ProductName , QuantitySold, and SaleDate,

```
SELECT ProductID, ProductName, SUM(QuantitySold) AS TotalSold
FROM Sales
WHERE SaleDate >= DATEADD(MONTH, -1, GETDATE())
GROUP BY ProductID, ProductName
ORDER BY TotalSold DESC
LIMIT 3;
```

2. Describe the SQL process you would use to normalize a table with redundant data.

Step 1: Identify duplicate Data
Examine the table to pinpoint columns or groups of columns containing repetitive information. Redundancy occurs when the same data is stored in multiple rows or columns.

Step 2: Establish New tables
Create additional tables to store the distinct values identified in step 1. Each new table should represent a unique entity or concept within your data model.

Step 3: Assign primary keys
Allocate a primary key to each new table, ensuring that it uniquely identifies each record in the table.

Step 4: Formulate relationships
Create connections between the original table and the new tables using foreign keys. These keys link the tables, maintaining referential integrity.

Step 5: Update existing Data
Populate the new tables with the unique values identified in step 1. Ensure that the foreign keys in the original table point to the corresponding primary keys in the new tables.

Step 6: Adjust queries
Modify any queries or applications interacting with the original table to accommodate the changes. Make necessary adjustments to facilitate correct table joins.

Step 7: validate
Thoroughly test the revised database structure to ensure proper functionality for data retrieval and modification operations. Confirm that relationships are enforced, and data integrity is upheld.

Step 8: Repeat as needed
Repeat the normalization process for other tables in the database exhibiting redundancy. The ultimate objective is to establish a normalized database schema, minimizing redundancy, and ensuring consistent data.

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(255),
    -- other columns);
```

Joining other table to this one using foreign key

```
ALTER TABLE Orders
ADD COLUMN CustomerID INT,
ADD FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID);
```

13 How would you write a SQL statement to merge data from two tables `orders` and `products`, where each order includes the product name, not just the ID?

```
SELECT orders.OrderID, orders.OrderDate, products.ProductName, orders.Quantity
FROM orders
JOIN products ON orders.ProductID = products.ProductID;
```

14 If you wanted to provide a roll-up report of sales by quarter and by region, which SQL clause would you use?

```
SELECT Region, Quarter, SUM(SalesAmount) AS TotalSales
FROM Sales GROUP BY ROLLUP (Region, Quarter);
```

Where Region and Quarter are columns in your table representing the region and quarter information. SalesAmount is the column representing the sales amount.

15. Imagine you have a database that is running slow. What SQL commands might you use to diagnose and fix the performance issues?

Indexing:

Problem Addressed: Lack of proper indexing can lead to slow query performance, especially for large dataset.
Solution: Identify the frequently queried columns and create appropriate indexes using CREATE INDEX statement. Regular analyze and optimize indexes to ensure their effectiveness.

Query Optimization:

Problem Addressed: Inefficient or poorly constructed queries can contribute to slow performance.
Solution: Review and optimize queries by ensuring proper use of indexes, minimizing unnecessary subqueries, and structuring joins efficiently. Use tools like EXPLAIN or EXPLAIN ANALYZE to analyze query plans.

Database Caching:

Problem Addressed: Frequently executing the same queries can strain the database.
Solution: Implement caching mechanisms to store and reuse query results or cache frequently accessed data. This reduces load on database by serving precomputed results when applicable.

Resource Monitoring and Optimization:

Problem Addressed: Inadequate resource allocation or high resource usage can impact database performance.
Solution: Regularly monitor resource usage using database-specific tools. Adjust memory configurations, optimize queries, and consider partitioning large tables to distribute load. Update statistics to help query optimizer make better decisions.