# Database Management Systems - Queries

**Basic SQL Queries:**

1. Retrieve all columns from a table named "employees."
2. Display the distinct values in a column named "department" from the "employees" table.
3. Count the total number of records in a table named "customers."
4. Select the first 10 records from a table named "orders."
5. Find the average salary from the "salary" column in the "employees" table.
6. List the employee names and their corresponding department names from two tables, "employees" and "departments."
7. Update the salary of an employee with ID 101 to 50000 in the "employees" table.
8. Delete all records from a table named "products" where the quantity is less than 10.

**Joins and Relationships:**

1. Perform an INNER JOIN between the "orders" and "customers" tables.
2. List all customers who have not placed any orders using a LEFT JOIN.
3. Display the product names and their corresponding category names using an INNER JOIN.
4. Retrieve the employee names along with their manager names from the "employees" table (self-join).
5. Find the total quantity of products ordered by each customer using a GROUP BY clause.
6. List all employees and their projects, even if they are not assigned to any project (use OUTER JOIN).

**Subqueries and Nested Queries:**

1. Retrieve the employee names who have a salary greater than the average salary.
2. Find the customers who have placed more than two orders.
3. Display the products with prices greater than the average price in their category.
4. List the employees who do not have a manager.
5. Find the second-highest salary in the "employees" table using a subquery.
6. Show the customers who have placed orders in the last 30 days.

**Aggregation and Grouping:**

1. Calculate the total sales amount for each product.
2. Find the maximum salary for each department.
3. Count the number of orders placed by each customer.
4. Calculate the average order quantity.
5. Determine the number of employees in each department.
6. Retrieve the department with the highest average salary.

**Advanced Query Techniques:**

1. Use the CASE statement to categorize employees into salary ranges.
2. Fetch the top 5 highest-paid employees.
3. Retrieve the oldest and youngest employees.
4. Rank employees based on their salaries using the RANK() function.
5. Get the 3rd highest salary from the "employees" table.
6. Use the EXISTS keyword to find employees with specific conditions.

**Constraints and Indexing:**

1. Add a primary key to a table named "students" on the "student\_id" column.
2. Create a unique constraint on the "email" column in the "users" table.
3. Explain the purpose of an index and how it improves query performance.
4. Drop an existing index on a table named "books."
5. Add a foreign key constraint to the "orders" table referencing the "customers" table.

**Data Modification:**

1. Insert a new record into the "products" table.
2. Update the quantity of a product with ID 102 to 50 in the "inventory" table.
3. Delete all records from the "employees" table where the department is 'HR.'

**Views and Stored Procedures:**

1. Create a view that displays the product names and their prices.
2. Write a stored procedure to calculate the total order amount for a given customer.
3. Alter an existing view to include an additional column from another table.
4. Execute a stored procedure that inserts a new record into the "employees" table.
5. Drop a view named "monthly\_sales."

**Transactions and Concurrency:**

1. Begin a transaction and update the prices of all products by 10%.
2. Rollback a transaction and undo the changes made in the previous step.
3. Commit a transaction that adds a new customer to the "customers" table.
4. Set the isolation level to SERIALIZABLE for a specific transaction.
5. Explain how to handle concurrent updates using transactions.

**Data Security and Permissions:**

1. Grant SELECT permission on the "employees" table to a new user.
2. Revoke INSERT permission on the "orders" table from a specific role.
3. Create a new user with limited access to only the "products" table.
4. Grant EXECUTE permission on a stored procedure to a user.
5. Show how to use the GRANT statement to assign multiple permissions to a user.

**Indexing and Query Optimization:**

1. Explain the benefits of indexing and how it impacts query performance.
2. Identify and remove duplicate records from a table using indexing.
3. Optimize a slow-performing query by adding appropriate indexes.
4. Demonstrate the use of the EXPLAIN statement to analyze query execution.
5. Discuss the scenarios where indexing might not be beneficial.

**Constraints and Triggers:**

1. Create a trigger that automatically updates a "last\_modified" column on each update.
2. Explain the purpose of a CHECK constraint and provide an example.
3. Implement a BEFORE INSERT trigger to validate data before insertion.
4. Discuss how constraints and triggers contribute to data integrity.
5. Show how to disable a trigger temporarily for a specific operation.

**Data Backup and Recovery:**

1. Perform a full database backup using the appropriate SQL command.
2. Restore a specific table from a backup file.
3. Explain the importance of regular database backups for data recovery.
4. Create a database snapshot for point-in-time recovery.
5. Describe the steps to recover from a database failure.

**NoSQL and JSON:**

1. Query a JSON document to retrieve specific elements.
2. Explain the structure of a JSON document and how it can be stored in a NoSQL database.
3. Retrieve data from a MongoDB collection using various query operators.
4. Discuss the advantages of using a NoSQL database over a traditional relational database.
5. Write a query to filter data based on nested fields in a JSON document.

**Cloud Databases:**

1. Connect to a cloud database using the appropriate credentials.
2. Discuss the benefits and challenges of migrating to a cloud-based database solution.
3. Show how to scale a database in a cloud environment.
4. Explain the concept of database as a service (DBaaS) and its advantages.
5. Describe the security measures implemented in a cloud database.

**Data Warehousing:**

1. Design a star schema for a data warehousing scenario.
2. Write a query to aggregate data in a data warehouse for business intelligence.
3. Discuss the differences between OLAP and OLTP in the context of data warehousing.
4. Explain the purpose of an ETL process in data warehousing.
5. Describe the role of dimensions and measures in a data cube.

**Advanced Concepts:**

1. Explain the concept of blockchain and its applications in databases.
2. Discuss how machine learning algorithms can be integrated into database systems.
3. Describe the impact of IoT on database architecture and data management.
4. Discuss the role of edge computing in enhancing database performance.
5. Explain the challenges and considerations in implementing a graph database.

**Real-world Scenarios:**

1. Given a business requirement, design a set of SQL queries to extract the necessary information.
2. Discuss the considerations for choosing between a relational and NoSQL database for a specific application.
3. Write a query to identify and resolve data inconsistencies in a database.
4. Demonstrate how to handle data migration when upgrading to a new database version.
5. Design a database schema for a social media platform with user profiles, posts, and comments.

**Case-Based Questions:**

1. Given a complex business scenario, design SQL queries to extract meaningful insights.
2. Discuss the steps involved in optimizing the performance of a large database.
3. Analyze a situation where database transactions and concurrency control are crucial.
4. Propose a solution to address security concerns in a database with sensitive customer information.
5. Describe the process of implementing a backup and recovery strategy for a mission-critical database.