SQL INTERVIEW QUESTIONS

40

questions

A collection of SQL questions for technical interview preparation

sql-academy.org

TABLE OF CONTENTS

Question 1

What is SQL and What is it Used For?

SQL (Structured Query Language) is the standard language for interacting with relational databases. It is used to define, manage, and retrieve data from databases. With SQL, you can perform the following operations:

- Create new databases and tables.
- Insert new data into tables.
- Read data using queries.
- Update existing data.
- Delete data.
- Manage access and permissions.

Question 2

Explain the differences between DDL, DML, and DCL in SQL?

SQL commands are divided into three main categories:

DDL (Data Definition Language):

- Used for defining the structure of a database (schema).
- Key commands: `CREATE`, `ALTER`, `DROP`, `TRUNCATE`, `RENAME`.

DML (Data Manipulation Language):

- Used for working with data within tables.
- Key commands: `SELECT`, `INSERT`, `UPDATE`, `DELETE`.

DCL (Data Control Language):

- Used for managing database access rights.
- Key commands: `GRANT`, `REVOKE`.

Question 3

What is a Primary Key and a Foreign Key?

Primary Key (PRIMARY KEY):

- A unique identifier for a record in a table.
- Does not allow duplicates or NULL values.
- Can consist of one or more columns (composite key).

```
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    name VARCHAR(100),
    age INT
);
```

Foreign Key (FOREIGN KEY)

- A column or set of columns that refer to the primary key of another table.
- Ensures referential integrity between tables.
- Allows linking records from different tables.

```
CREATE TABLE enrollments (
    enrollment_id INT PRIMARY KEY,
    student_id INT,
    course_id INT,
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
);
```

Question 4

Explain the Concept of Normalization and Its Benefits

> *Normalization is the process of organizing the structure of a database to reduce data redundancy and ensure data integrity.*

Main goals of normalization:

- Elimination of Redundancy: Prevents data duplication.
- Improved Data Integrity: Minimizes the chance of data inconsistency.
- Easier Maintenance and Updates: Makes the database structure more flexible and understandable.

Key normal forms

- First Normal Form (1NF): All columns contain atomic (indivisible) values.
- Second Normal Form (2NF): Satisfies 1NF and all non-key columns depend on the entire primary key.
- Third Normal Form (3NF): Satisfies 2NF and there are no transitive dependencies between non-key columns.

Example

Before normalization:

StudentID	StudentName	CourseID	CourseName
1	John	101	Math
1	John	102	Music

After normalization:

Students table

StudentID	StudentName
1	John

Courses table

CourseID	CourseName
101	Math
102	Music

Enrollments table

StudentID	CourseID
1	101
1	102

Question 5

What is a JOIN and what types of JOINs do you know

> *JOIN is an operation in SQL that allows combining rows from two or more tables based on related columns between them.*

Types of JOINs:

- `INNER JOIN`:

Returns records that have matching values in both tables..

- `LEFT JOIN (or LEFT OUTER JOIN)`:

Returns all records from the left table and the matching records from the right table. If there is no match, it returns NULL for the right table.

- `RIGHT JOIN (or RIGHT OUTER JOIN)`:

Returns all records from the right table and the matching records from the left table. If there is no match, it returns NULL for the left table.

- `FULL OUTER JOIN`:

Returns all records where there is a match in one of the tables.

- `CROSS JOIN`:

Performs a Cartesian product of two tables, combining each row of the first table with each row of the second table.

Question 6

What is a Subquery and When is it Used?

> ***A subquery*** *is an SQL query nested inside another query. It is used to perform operations whose results are needed for the main query.*

Use cases for subqueries:

- **Data Filtering**: Using the results of a subquery in `WHERE` or `HAVING` conditions.
- **Data Selection**: Using a subquery in the list of selected columns.
- **Creating Virtual Tables**: Using a subquery in the `FROM` clause.

Examples

1. Subquery in `WHERE`:

```
SELECT name
FROM employees
WHERE department_id = (SELECT id FROM departments WHERE name = 'IT');
```

2. Subquery in `FROM`:

```
SELECT sub.department, COUNT(*)
FROM (
    SELECT department_id AS department
    FROM employees
) sub
GROUP BY sub.department;
```

Question 7

How to remove duplicates in the result of an SQL query?

Use the `DISTINCT` keyword in the `SELECT` statement to return only unique records.

Пример:

```
SELECT DISTINCT position
FROM employees;
```

Question 8

Explain the difference between WHERE and HAVING

WHERE

- Filters rows before data is grouped.
- Cannot use aggregate functions (`SUM()` , `COUNT()` , `AVG()` , etc.).
- Applies to individual records in the table.

Example:

```
SELECT department_id, COUNT()
FROM employees
WHERE salary > 50000
GROUP BY department_id;
```

HAVING

- *Filters groups of rows after the data is grouped.*
- *Can use aggregate functions.*
- *Applies to the results of* `GROUP BY` *.*

Example:

```
SELECT department_id, COUNT() AS num_employees
FROM employees
GROUP BY department_id
HAVING COUNT(*) > 5;
```

Question 9

What is an index and how does it affect performance?

> ***An Index*** *is a special data structure that improves the speed of data retrieval operations from a table by creating pointers to the data.*

Indexes speed up read operations but can slow down write operations (inserts, updates, deletes) since the indexes need to be updated when the data changes.

Advantages of Indexes:

- Faster execution of `SELECT` operations.
- Improved performance when sorting and searching data.

Disadvantages of Indexes:

- Additional disk space required.
- Slower performance for `INSERT`, `UPDATE`, and `DELETE` operations.

Example of creating an index:

```
-- Creating an index on the name column in the employees table
CREATE INDEX idx_employees_name ON employees(name);
```

Question 10

What are the key differences between DELETE and TRUNCATE?

DELETE:

- Removes selected records from a table.
- A `WHERE` condition can be used to delete specific records.
- The operation is logged row by row in the transaction log.
- `ON DELETE` triggers are activated.
- Slower compared to `TRUNCATE`.

Example:

```
DELETE FROM employees WHERE salary < 30000;
```

TRUNCATE:

- Removes all records from a table with no possibility of recovery via `ROLLBACK` (in most DBMS).
- Cannot use `WHERE`.
- Faster since it is not logged row by row.
- Resets identity values (if auto-increment is used).
- Triggers are not activated.

Example:

```
TRUNCATE TABLE employees;
```

Question 11

What is a transaction and what are the ACID properties of a transaction?

> ***A Transaction*** *is a sequence of operations performed as a single logical unit, which must be fully completed or fully rolled back.*

Transaction Properties (ACID):

Atomicity:

- A transaction is fully completed or not executed at all.
- If a failure occurs, all changes are rolled back.

Consistency:

- A transaction transitions the database from one consistent state to another.
- All database rules and constraints are maintained.

Isolation:

- The results of a transaction are invisible to other transactions until it is completed.
- Prevents interference between concurrent transactions.

Durability:

- Once a transaction is successfully completed, its results are permanently saved, even in the event of system failures.
- Changes are written to persistent storage.

Question 12

What are triggers in SQL?

> ***A Trigger*** *r is a stored procedure that is automatically executed when a specific event occurs in the database, such as an `INSERT` , `UPDATE` , or `DELETE` on a particular table.*

Types of Triggers:

- DML Triggers: Respond to `INSERT` , `UPDATE` , or `DELETE` operations.
- DDL Triggers: Respond to `CREATE` , `ALTER` , or `DROP` operations.
- Row-level or Statement-level Triggers.

Advantages of triggers:

- Automates checks and constraints.
- Logs changes.
- Maintains data integrity.

Example of creating a trigger:

```sql
CREATE TRIGGER trg_after_insert_employee
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    INSERT INTO audit_log (employee_id, action, action_time)
    VALUES (NEW.id, 'INSERT', NOW());
END;
```

Question 13

Explain what a VIEW is and its benefits

> ***A VIEW*** *is a virtual table based on the result of an SQL query. A view does not store data itself but provides a specific way to view data from one or more tables.*

Benefits of a VIEW:

- **Simplifies complex queries**: Allows saving a complex query and using it as a simple table.
- **Security**: Grants users access only to specific data, hiding the rest.
- **Updatability**: In some cases, data can be updated through the view.
- **Maintains data integrity**: Can combine data from multiple tables in a specific way.

Example of creating a VIEW:

```
CREATE VIEW employee_details AS
SELECT e.id, e.name, d.name AS department, e.salary
FROM employees e
JOIN departments d ON e.department_id = d.id;
```

Using a VIEW:

```
SELECT * FROM employee_details WHERE salary > 50000;
```

Question 14

How to use the LIKE operator and what is it used for?

The `LIKE` operator is used in `WHERE` clauses to search for rows that match a specific pattern. Wildcards are used in the patterns:

- `%` — matches any sequence of characters (including an empty sequence).
- `_` — matches any single character.

Examples of usage:

1. Searching for rows that start with “A”:

```
SELECT  FROM employees WHERE name LIKE 'A%';
```

2. Searching for rows that end with “e”:

```
SELECT  FROM employees WHERE name LIKE '%e';
```

3. Searching for rows where the second character is “a”:

```
SELECT * FROM employees WHERE name LIKE '_a%';
```

Question 15

What are aggregate functions? Provide examples

Aggregate functions perform calculations on a set of values and return a single value. They are often used in combination with the `GROUP BY` clause.

Key Aggregate Functions

- `COUNT()` — counts the number of rows.
- `SUM()` — calculates the sum of values.
- `AVG()` — calculates the average value.
- `MAX()` — finds the maximum value.
- `MIN()` — finds the minimum value.

Examples of usage

1. Counting the number of employees:

```
SELECT COUNT(*) FROM employees;
```

2. Average salary by department:

```
SELECT department_id, AVG(salary) AS average_salary
FROM employees
GROUP BY department_id;
```

3. Maximum salary in the company:

```
SELECT MAX(salary) FROM employees;
```

4. Total sales for the month:

```
SELECT SUM(amount) FROM sales WHERE sale_date BETWEEN '2023-01-01' AND '2023-01-31';
```

Question 16

Explain the difference between UNION and UNION ALL

UNION:

- Combines the results of two or more `SELECT` queries.
- Removes duplicates from the combined result.

Syntax

```
SELECT column_list FROM table1
UNION
SELECT column_list FROM table2;
```

UNION ALL:

- Combines the results of two or more SELECT queries.
- Retains duplicates in the combined result.
- Performs faster as it doesn’t perform the additional operation of removing duplicates.

Syntax

```
SELECT column_list FROM table1
UNION ALL
SELECT column_list FROM table2;
```

Question 17

What is a stored procedure and how does it differ from a function?

Stored Procedure:

- A set of SQL commands stored on the server for reuse.
- Can perform `SELECT`, `INSERT`, `UPDATE`, `DELETE` operations.
- May return multiple result sets or return nothing.
- Can have input and output parameters.
- Cannot be called within an SQL query.

Example of a stored procedure:

```
CREATE PROCEDURE GetEmployeeByID(IN emp_id INT)
BEGIN
    SELECT * FROM employees WHERE id = emp_id;
END;
```

Calling the procedure:

```
CALL GetEmployeeByID(1);
```

Function:

- Returns a single value (scalar function) or a table (table-valued function).
- Can be used in SQL expressions (e.g., in SELECT or WHERE clauses).
- Must return a value.
- Typically used for calculations and returns a deterministic result.

Example of a function:

```
CREATE FUNCTION GetEmployeeSalary(emp_id INT) RETURNS DECIMAL(10,2)
BEGIN
    DECLARE salary DECIMAL(10,2);
    SELECT e.salary INTO salary FROM employees e WHERE e.id = emp_id;
    RETURN salary;
END;
```

Using the function:

```sql
SELECT name, GetEmployeeSalary(id) FROM employees;
```

Question 18

How to optimize SQL query performance?

Use Indexes:

- Create indexes on columns frequently used in `WHERE`, `JOIN`, and `ORDER BY` clauses.
- Avoid redundant indexes.

Avoid `SELECT`:

- **Select only the necessary columns.**
- **Reduces the amount of data transferred.**

Optimize `JOIN` and `WHERE` Conditions:

- **Use equality (=) instead of inequality where possible.**
- **Avoid functions and calculations on indexed columns in conditions.**

Use Result Limits (`LIMIT`):

- **Limit the number of returned rows if you don’t need all the data.**

Avoid Subqueries Where Joins Are Possible::

- **Replace correlated subqueries with `JOIN` or `EXISTS`.**

Cache frequently used data:

- **Use materialized views or caching at the application level.**

Profiling and Query Analysis:**

- Use tools (`EXPLAIN`, `EXPLAIN PLAN`) to analyze query execution plans.

Question 19

What are constraints and what types exist?

Constraints ensure data integrity and reliability in a table by defining rules for the data in columns.

Types of constraints:

NOT NULL:

- Prohibits storing `NULL` values in a column.

Example:

```
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL
);
```

UNIQUE:

- Ensures uniqueness of values in a column or group of columns.

Example:

```
CREATE TABLE users (
    user_id INT PRIMARY KEY,
    email VARCHAR(100) UNIQUE
);
```

PRIMARY KEY:

- A combination of `NOT NULL` and `UNIQUE`.
- Identifies each record in the table.

FOREIGN KEY:

- Ensures referential integrity between tables.
- The value must match an existing primary key value in the related table.

Example:

```sql
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    user_id INT,
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);
```

CHECK:

- Defines a condition that the values in a column must meet.

Example:

```sql
CREATE TABLE employees (
    id INT PRIMARY KEY,
    age INT CHECK (age >= 18)
);
```

DEFAULT:

- Sets a default value for a column if no value is provided during insertion.

```sql
CREATE TABLE tasks (
    task_id INT PRIMARY KEY,
    status VARCHAR(20) DEFAULT 'Pending'
);
```

Question 20

What is SQL injection and how to protect against it?

> ***SQL Injection*** *is a method of attacking a database where an attacker inserts malicious SQL code through input fields, allowing unauthorized SQL queries to be executed.*

Consequences of SQL Injection:

- Data theft.
- Deletion or modification of data.
- Gaining administrative access.

Methods of protection against SQL injection

1. Parameterized queries (Prepared Statements):

- Use parameters instead of string concatenation.
- The DBMS automatically escapes special characters.

Example (in Java using JDBC):

```
String sql = "SELECT * FROM users WHERE username = ? AND password = ?";
PreparedStatement stmt = connection.prepareStatement(sql);
stmt.setString(1, username);
stmt.setString(2, password);
ResultSet rs = stmt.executeQuery();
```

2. Using ORM (Object-Relational Mapping):

- ORM libraries often include built-in protection mechanisms against SQL injection.

3. Input Validation and Filtering:

- Validate input data to match the expected format.
- Use validation both on the server and client side.

4. Restricting access rights:

- Grant only the minimum necessary privileges to database users.
- Restrict access to system tables and operations.

5. Using stored procedures:

- Encapsulates data logic within a procedure.

- Users have access only to the procedures, not directly to the tables.

Question 21

What is a relational database?

> ***A relational database*** *is a database based on the relational model of data. In such a database, data is stored in tables, and relationships between data are defined using keys.*

Key characteristics:

- Tables (relations): Data is organized into tables consisting of rows and columns.
- Rows (records): Each row represents an individual record.
- Columns (attributes): Each column contains data of a specific type.
- Keys: Used to identify records and establish relationships between tables.

Advantages of relational databases:

- Flexibility: New tables and columns can be easily added.
- Data Integrity: Use of constraints to maintain data integrity.
- SQL: A standard language for managing data.

Question 22

Explain the Difference Between INNER JOIN and OUTER JOIN

INNER JOIN:

- Returns only records that have matching entries in both joined tables.
- If there is no match, the record is not included in the result.

Example:

```
SELECT
FROM employees e
INNER JOIN departments d ON e.department_id = d.id;
```

OUTER JOIN:

- *Returns matching records, as well as records from one table that do not have a match in the other table.*

Types of OUTER JOIN:

LEFT OUTER JOIN (LEFT JOIN):

- *Returns all records from the left table and the matching records from the right table.*
- *If there is no match, the columns from the right table will be `NULL`.*

Example:

```
SELECT
FROM employees e
LEFT JOIN departments d ON e.department_id = d.id;
```

RIGHT OUTER JOIN (RIGHT JOIN):

- Returns all records from the right table and the matching records from the left table.
- If there is no match, the columns from the left table will be `NULL`.

Example:

```
SELECT
FROM employees e
RIGHT JOIN departments d ON e.department_id = d.id;
```

FULL OUTER JOIN (FULL JOIN):

- *Returns all records where there is a match in either of the tables.*
- *If there is no match, the corresponding columns will be `NULL`.*

```
SELECT
FROM employees e
FULL OUTER JOIN departments d ON e.department_id = d.id;
```

Question 23

What is NULL and How to Work with It in SQL?

> ***NULL** is a special value in SQL that represents the absence of data or an unknown value.*

Characteristics of NULL:

- `NULL` is not equivalent to an empty string or zero.
- Operations with `NULL` return `NULL`.
- Comparing `NULL = NULL` returns `FALSE`.

Working with NULL

To check for `NULL`, use the `IS NULL` or `IS NOT NULL` operator.

```
-- Finding records with an unknown birth date
SELECT  FROM employees WHERE birth_date IS NULL;

-- Finding records with a known birth date
SELECT  FROM employees WHERE birth_date IS NOT NULL;
```

Functions for working with NULL

COALESCE Returns the first non-`NULL` value from the list.

```
COALESCE(val1[, val2, ...., val_n])
```

ISNULL Returns `1` or `0` depending on whether the expression is `NULL`.

```
ISNULL(value)
```

IFNULL Returns the first argument if it is not `NULL`. Otherwise, it returns the second argument.

```
IFNULL(value, alternative_value)
```

Question 24

How to use the CASE operator in SQL?

The `CASE` operator is used to implement conditional logic in SQL queries. It allows you to return values based on conditions, similar to the `IF-ELSE` statement.

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    ...
    ELSE default_result
END
```

Examples

- Assigning categories based on salary:

```
SELECT name,
    salary,
    CASE
        WHEN salary >= 80000 THEN 'High'
        WHEN salary >= 50000 THEN 'Medium'
        ELSE 'Low'
    END AS salary_category
FROM employees;
```

Question 25

Explain Transactional Commands COMMIT and ROLLBACK.

COMMIT

- Commits the current transaction.
- All changes made during the transaction become permanent and visible to other users.
- After a `COMMIT`, changes cannot be undone.

```
BEGIN TRANSACTION;

UPDATE accounts SET balance = balance - 100 WHERE account_id = 1;
UPDATE accounts SET balance = balance + 100 WHERE account_id = 2;

COMMIT;
```

ROLLBACK

- Reverts the current transaction.
- All changes made during the transaction are undone.
- The database is returned to the state it was in before the transaction began.

```
BEGIN TRANSACTION;

DELETE FROM orders WHERE order_date < '2022-01-01';

-- If you change your mind
ROLLBACK;
```

Usage in transaction management:

- `BEGIN TRANSACTION` or `START TRANSACTION`: begins a transaction.
- `COMMIT`: commits the transaction.
- `ROLLBACK`: rolls back the transaction.

Question 26

Explain the differences between CHAR and VARCHAR

CHAR(n):

- Stores fixed-length strings of length `n`.
- If the entered string is shorter than `n`, it is padded with spaces to reach the length of `n`.
- Used for storing data of uniform length (e.g., country codes, postal codes).

VARCHAR(n):

- Stores variable-length strings up to `n` characters.
- Actually occupies as much space as the number of characters in the string plus a small overhead for storing the length.
- Used for storing string data of variable length.

Key differences:

Memory and performance:

- `CHAR` always occupies a fixed amount of memory.
- `VARCHAR` is more memory-efficient but may be slightly slower in access.

Usage:

- `CHAR` is suitable for data with predictable length.
- `VARCHAR` is suitable for data with variable length.

Example:

```
CREATE TABLE products (
    code CHAR(10),      -- Fixed-length product code
    name VARCHAR(100)   -- Variable-length product name
);
```

Inserting data:

```
INSERT INTO products (code, name)
VALUES ('A123', 'Lenovo Laptop');
```

In the `code` column, the value will be padded with spaces up to 10 characters.

Question 27

What is a temporary table in SQL?

> ***A temporary table*** *is a table that exists only for the duration of the current session or connection and is automatically deleted when the session ends or the connection is closed.*

Creating a temporary table

```
CREATE TEMPORARY TABLE TempTable (
    id INT,
    name VARCHAR(100)
);
```

Using a temporary table

```
-- inserting data into the temporary table
INSERT INTO #TempTable (id, name)
VALUES (1, 'Иван'), (2, 'Петр');

-- selecting data from the temporary table
SELECT * FROM #TempTable;

-- the temporary table will be automatically deleted after the session ends
```

Applications of temporary tables

- Storing intermediate results in complex queries.
- Handling large datasets in batch operations.
- Avoiding conflicts when multiple users are working simultaneously.

Question 28

What are window functions in SQL?

> ***Window functions** are functions that perform calculations over a set of rows (a window) related to the current row and return a result for each row without grouping the data.*

Main window functions:

Aggregate functions:

- `SUM` — calculates the total sum of values
- `COUNT` — counts the total number of records in a column
- `AVG` — calculates the arithmetic mean
- `MAX` — finds the maximum value
- `MIN` — determines the minimum value

Ranking functions:

- `ROW_NUMBER`: assigns a sequential number to a row within the window
- `RANK`: assigns a rank to a row within the window, with gaps when values tie
- `DENSE_RANK`: assigns a rank to a row without gaps

Offset functions:

- `LAG`: returns the value from the previous row
- `LEAD`: returns the value from the next row
- `FIRST_VALUE`: returns the first value in the window
- `LAST_VALUE`: returns the last value in the window

A detailed explanation of how window functions work can be found in our course.

Question 29

Explain the concept of CTE (common table expression)

> ***CTE (common table expression)*** *is a temporary named result set defined in an SQL query using the* `WITH` *keyword.*

It enhances the readability and structure of complex queries.

Syntax

```
WITH CTEName (column1, column2, ...)
AS (
     -- your query
    SELECT ...
)
SELECT * FROM CTEName;
```

Advantages of CTE

- Improves code readability.
- Allows breaking down complex queries into logical parts.
- Supports recursive queries (recursive CTEs).

A detailed explanation of CTEs can be found in our course.

Question 30

How to delete a table along with its data?

The `DROP TABLE` command is used to delete a table and all its data from the database.

Syntax

```
DROP TABLE table_name;
```

Features:

- All data, indexes, triggers, and permissions associated with the table are deleted.
- The action is irreversible unless backup or recovery mechanisms are in place.

Question 31

What is a FOREIGN KEY and how does it ensure data integrity?

> ***A FOREIGN KEY*** *is a constraint that establishes a relationship between a column or set of columns in one table and a column or columns in another table (typically a primary key).*

It ensures referential integrity by guaranteeing that values in the foreign key column correspond to existing values in the related table.

How data integrity is ensured:

- **Prevents insertion of invalid data**

It is impossible to insert a value into the foreign key column if that value does not exist in the related table.

- **Prevents deletion of related records**

It is impossible to delete a record from the parent table if records in the child table reference it, unless additional actions are taken.

Example of a foreign key:

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(100),
    DepartmentID INT,
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);
```

Question 32

How to add a new column to an existing table

The `ALTER TABLE` command with the `ADD` operator is used to add a new column to an existing table.

Syntax

```
ALTER TABLE table_name
ADD column_name data_type [constraints];
```

Example

Suppose we have a table `employees`, and we want to add a column `email` of type `VARCHAR(255)`.

```
ALTER TABLE employees
ADD email VARCHAR(255);
```

Adding a column with a `NOT NULL` constraint and a default value:

```
ALTER TABLE employees
ADD date_of_birth DATE NOT NULL DEFAULT '1900-01-01';
```

Note When adding a column with a `NOT NULL` constraint, if there are already existing rows in the table, you must specify a default value; otherwise, an error will occur.

Question 33

What is a correlated subquery?

> ***A correlated subquery*** *is a subquery that depends on the outer query. It is executed for each row of the outer query, using values from that row.*

Features

- The subquery references columns from the outer query.
- It can be less efficient due to multiple executions.

Example

Suppose there are tables `employees` and `departments`.

```
SELECT e.name, e.salary
FROM employees e
WHERE e.salary > (
    SELECT AVG(salary)
    FROM employees
    WHERE department_id = e.department_id
);
```

- Here, the subquery calculates the average salary for the department of each employee.
- The main query selects employees whose salary is higher than the department average.

More information about correlated subqueries can be found in our course.

Question 34

Explain the difference between DELETE, TRUNCATE, and DROP

DELETE

- Removes selected records from a table.
- A `WHERE` condition can be used to delete specific records.
- The operation is logged in the transaction log row by row.
- `ON DELETE` triggers are activated.

Syntax

```
DELETE FROM table_name [WHERE condition];
```

TRUNCATE

- Removes all data from the table without the possibility of recovery.
- `WHERE` cannot be used.
- Faster because it does not log the deletion of each row.
- Resets auto-increment identifiers.
- Triggers are not activated.

Syntax

```
TRUNCATE TABLE table_name;
```

DROP

- Deletes the entire table along with its data, structure, indexes, and constraints.
- The action is irreversible.

Syntax

```
DROP TABLE table_name;
```

Choosing between commands

- Use `DELETE` when you need to remove specific records.

- Use `TRUNCATE` to quickly remove all data from the table while keeping its structure.
- Use `DROP` to completely remove the table from the database.

Question 35

What is a Self-Join and when is it used?

> ***A Self-Join*** *is a type of join in SQL where a table is joined with itself. It is useful when you need to compare rows within the same table or handle hierarchical data.*

When to use a Self-Join:

- **Hierarchical structures**

For example, in an employee table where each employee may have a manager who is also an employee in the same table.

- **Comparing records**

To find duplicates or compare values between different rows in the same table.

Example

Consider the following employee data:

employeeId	name	managerId
1	John	
2	Michail	1
3	Alisa	1
4	Max	2

To get a list of employees and their managers:

```
SELECT e.name AS Employee, m.name AS Manager
FROM Employee e
LEFT JOIN Employee m ON e.managerId = m.employeeId;
```

In this query, we join the `Employee` table with itself to match each employee with their manager.

Question 36

How to perform database backup and recovery

Backing up and restoring a database are critical operations to ensure data safety and the ability to recover it in case of failure or loss.

How to back up a database:

The method of backup depends on the database management system (DBMS) you are using.

Below are examples for some popular DBMS.

MySQL

Backup using the `mysqldump` utility:

```
mysqldump -u username -p mydatabase > backup.sql
```

- `username` — the database username.
- `mydatabase` — the name of the database to back up.
- `backup.sql` — the file where the backup will be saved.

PostgreSQL

Backup using the `pg_dump` utility:

```
pg_dump -U username mydatabase > backup.sql
```

- `-U username` — the database username.
- `mydatabase` — the name of the database.
- `backup.sql` — the output file for the backup.

How to restore a database from a backup:

MySQL

Restoring a database from the `backup.sql` file:

```
mysql -u username -p mydatabase < backup.sql
```

PostgreSQL

Restoring a database using the `psql` utility:

```
psql -U username mydatabase < backup.sql
```

Recommendations

- **Permissions**

Ensure that you have the necessary permissions to perform backup and restore operations.

- **Regularity**

Set up regular automatic backups to minimize the risk of data loss.

- **Backup storage**

Store backups in a secure and reliable location, preferably off the main server.

- **Testing**

Periodically test backups by restoring them to a test server to verify their integrity and functionality.

Question 37

How to implement many-to-many relationships in SQL?

A many-to-many relationship in relational databases occurs when one record in the first table can be associated with multiple records in the second table, and vice versa.

In SQL, such relationships are implemented using a junction table (also known as a linking or associative table) that connects the two main tables via foreign keys.

How to implement many-to-many relationships in SQL

- **Create two main tables**, that need to be linked.
- **Create a junction table**, containing foreign keys that reference the primary keys of both main tables.
- **Define foreign keys and a composite primary key** in the junction table to ensure referential integrity and uniqueness of relationship pairs.

Example implementation

Consider a scenario with `Student` and `Course` tables where one student can enroll in multiple courses, and one course can be taken by multiple students.

- Creating the students table:

```sql
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(100)
);
```

- Creating the courses table:

```sql
CREATE TABLE Course (
    CourseID INT PRIMARY KEY,
    Title VARCHAR(100)
);
```

- Creating a junction table to establish the many-to-many relationship:

```
CREATE TABLE StudentCourse (
    StudentID INT,
    CourseID INT,
    PRIMARY KEY (StudentID, CourseID),
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
);
```

As a result:

- The **junction table** `StudentCourse` contains pairs of `StudentID` and `CourseID`, representing the relationships between students and courses.
- The **composite primary key** `(StudentID, CourseID)` ensures that each pair is unique, preventing duplicate relationships.
- The **foreign keys** ensure data integrity by referencing the corresponding records in the `Student` and `Course` tables.

Question 38

How do the REVOKE and GRANT commands work?

The `GRANT` and `REVOKE` commands in SQL are used to manage user access rights to database objects.

They allow granting or revoking specific privileges to or from users or roles, ensuring security and control over who can perform what actions in the database.

GRANT

The `GRANT` command provides users or roles with specific privileges on database objects.

Syntax:

```
GRANT privileges ON object TO user [WITH GRANT OPTION];
```

- `privileges`: the actions allowed (e.g., `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `ALL PRIVILEGES`).
- `object`: the database, table, view, procedure, etc.
- `user`: the username or role to which privileges are granted.
- `WITH GRANT OPTION` (optional): allows the recipient of the privileges to grant them to others.

Example: To grant user `user1` the privilege to select data from the `employees` table:

```
GRANT SELECT ON employees TO user1;
```

REVOKE

The `REVOKE` command removes previously granted privileges from users or roles.

Syntax:

```
REVOKE privileges ON object FROM user;
```

Example: To revoke the `SELECT` privilege from user `user1` on the `employees`

```
REVOKE SELECT ON employees FROM user1;
```

Question 39

Explain the use of hashing functions in SQL

Hash functions in SQL are used to transform input data of arbitrary length into a fixed-length string. This transformation is called **hashing**, and it is widely used to ensure security, data integrity, and optimize comparison and search operations.

Key use cases for hash functions in SQL:

1. Storing passwords

- Instead of storing passwords in plain text, they are stored as hashes. This increases security since, even if the database is compromised, attackers cannot recover the original passwords.
- Cryptographic hash functions like `SHA-256`, `SHA-512`, and others are commonly used.

2. Data integrity verification

Hash functions allow detecting whether data has been altered. By creating a hash of the original data and comparing it with the current hash, changes can be identified.

3. Indexing and search optimization

Hash values can be used for quick comparison of large amounts of data or to create indexes for faster searches.

4. Creating unique identifiers

Hashing helps generate unique identifiers for records based on their content.

Example of using hash functions in SQL:

```sql
-- Create a users table
CREATE TABLE Users (
    UserID INT PRIMARY KEY,
    Username VARCHAR(50) NOT NULL,
    PasswordHash VARBINARY(512) NOT NULL
);

-- When adding a new user, hash the password
INSERT INTO Users (UserID, Username, PasswordHash)
VALUES (1, 'user1', HASHBYTES('SHA2_512', 'password123'));

-- Verify that the entered password matches the hash in the database
SELECT UserID FROM Users
WHERE Username = 'user1' AND PasswordHash = HASHBYTES('SHA2_512', 'password123');
```

Recommendations and considerations

Choosing a hash function

- Prefer modern and robust algorithms like `SHA-256` or `SHA-512`.
- Avoid using outdated functions like `MD5` or `SHA1` due to known vulnerabilities.

Secure storage

Store hashes in binary format (`VARBINARY`) instead of string format (`VARCHAR`) to preserve accuracy and save space.

Irreversibility of hashes

Remember that hashing is a one-way function; the original data cannot be recovered from the hash.

Question 40

How does an SQL trigger work?

What is an SQL trigger

> ***An SQL trigger*** *is a database object representing a special type of stored procedure that automatically executes when a specific event occurs in the database. These events may include `INSERT`, `UPDATE`, or `DELETE` operations on tables or views.*

How an SQL trigger works

A trigger is automatically activated in response to a specified event (e.g., adding a new record to a table).

Types of triggers

- `BEFORE`: Works before the operation is executed.

Used to validate or modify data before it is saved.

- `AFTER`: Works after the operation is executed.

Often used for logging or updating related tables.

- `INSTEAD OF`: Replaces the default behavior of the operation.

Used when you need to override the default action, such as when working with views.

Within a trigger, you can access old and new data values through special pseudo-tables: `OLD` and `NEW`.

Example of using a trigger

Suppose we have an `Employees` table, and we want to automatically save a history of salary changes in the `SalaryHistory` table whenever employee data is updated.

Creating tables:

```
-- Employees table
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(100),
    Salary DECIMAL(10, 2)
);

-- Salary history table
CREATE TABLE SalaryHistory (
    EmployeeID INT,
    OldSalary DECIMAL(10, 2),
    NewSalary DECIMAL(10, 2),
    ChangeDate DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Creating a trigger::

```
DELIMITER $$

CREATE TRIGGER trg_AfterSalaryUpdate
AFTER UPDATE ON Employees
FOR EACH ROW
BEGIN
    IF OLD.Salary <> NEW.Salary THEN
        INSERT INTO SalaryHistory (EmployeeID, OldSalary, NewSalary)
        VALUES (NEW.EmployeeID, OLD.Salary, NEW.Salary);
    END IF;
END$$

DELIMITER ;
```

When to use triggers

- **Logging and auditing**:

Automatically logging data changes to track user actions.

- **Maintaining data integrity**:

Enforcing complex business rules that cannot be achieved through constraints.

- **Data synchronization**:

Automatically updating or modifying related tables when data changes.

- **Calculating values**:

Automatically calculating and updating aggregated or derived data.