**Module 4 – Introduction to DBMS**

Name: Limbasiya Vishvash

Theory:-

Q.1:- What is SQL, and why is it essential in database management?

Ans:-

SQL is used to interact with relational databases. It works by understanding and analysing data of virtually any size, from small datasets to large stacks. It’s a powerful tool that enables you to perform many functions efficiently and quickly.

Q.2:- Explain the difference between DBMS and RDBMS.

Ans:-

| **DBMS** | **RDBMS** |
| --- | --- |
| [DBMS](https://www.geeksforgeeks.org/introduction-of-dbms-database-management-system-set-1/) stores data as file. | [RDBMS](https://www.geeksforgeeks.org/rdbms-architecture/) stores data in tabular form. |
| Data elements need to access individually. | Multiple data elements can be accessed at the same time. |
| No relationship between data. | Data is stored in the form of tables which are related to each other. |
| Normalization is not present. | Normalization is present. |
| DBMS does not support distributed database. | RDBMS supports distributed database. |
| It stores data in either a navigational or hierarchical form. | It uses a tabular structure where the headers are the column names, and the rows contain corresponding values. |
| It deals with small quantity of data. | It deals with large amount of data. |
| Data redundancy is common in this model. | Keys and indexes do not allow Data redundancy. |
| It is used for small organization and deal with small data. | It is used to handle large amount of data. |
| Not all Codd rules are satisfied. | All 12 Codd rules are satisfied. |
| Security is less | More security measures provided. |
| It supports single user. | It supports multiple users. |
| Data fetching is slower for the large amount of data. | Data fetching is fast because of relational approach. |
| The data in a DBMS is subject to low security levels with regards to data manipulation. | There exists multiple levels of data security in a RDBMS. |
| Low software and hardware necessities. | Higher software and hardware necessities. |
| Examples:[XML](https://www.geeksforgeeks.org/xml-basics/), Window Registry, Forxpro, dbaseIIIplus etc. | Examples: [MySQL](https://www.geeksforgeeks.org/architecture-of-mysql/), [PostgreSQL](https://www.geeksforgeeks.org/what-is-postgresql-introduction/), [SQL](https://www.geeksforgeeks.org/what-is-sql/) Server, Oracle, |

Q.3:- . Describe the role of SQL in managing relational databases.

Ans:-

 **Data Definition**: SQL allows users to define the structure of databases using commands like CREATE, ALTER, and DROP. These commands let users create tables, modify schemas, and delete structures as needed.

 **Data Querying**: SQL enables users to retrieve and filter data using the SELECT statement. Through this, users can query specific data from one or more tables, apply conditions with WHERE, and sort or group the results.

 **Data Manipulation**: With SQL, users can insert new records, update existing ones, and delete data using INSERT, UPDATE, and DELETE commands. This helps manage the actual data content within the database.

 **Data Control**: SQL provides commands for controlling access to data with GRANT and REVOKE, as well as managing transactions to ensure data integrity, consistency, and rollback capabilities through commands like BEGIN, COMMIT, and ROLLBACK.

 **Data Integrity**: SQL supports the creation of constraints (e.g., primary keys, foreign keys, and unique constraints) to ensure that data is consistent, accurate, and follows business rules.

 **Data Relationship Management**: SQL is also key in establishing and managing relationships between tables through foreign keys, ensuring that data is linked logically and that referential integrity is maintained.

Q.4:- What are the key features of SQL?

Ans:-

1). **Data Definition**: SQL allows users to define and modify the structure of database objects. You can create, modify, and delete tables, views, indexes, and schemas using commands like:

* CREATE: To create tables, databases, etc.
* ALTER: To modify the structure of existing database objects.
* DROP: To delete tables or databases.

2).**Data Manipulation**: SQL enables you to insert, update, and delete data within tables. This includes:

* INSERT: To add new rows of data into a table.
* UPDATE: To modify existing data in one or more rows.
* DELETE: To remove data from a table.

3).**Data Querying**: SQL’s querying capabilities allow users to retrieve data from one or more tables using:

* SELECT: To retrieve data with various options like filtering, sorting, grouping, and joining tables.
* JOIN: To combine data from multiple tables based on related columns.
* WHERE, HAVING: To filter data based on specific conditions.

4).**Data Integrity**: SQL helps maintain data accuracy and consistency with features like:

* **Constraints**: Primary keys, foreign keys, unique constraints, and check constraints ensure data integrity.
* **Normalization**: SQL encourages designing databases in a way that reduces data redundancy and improves data consistency.

5).**Transaction Control**: SQL provides transaction management capabilities to ensure that operations are completed correctly and safely. The key transaction commands are:

* BEGIN TRANSACTION: To start a transaction.
* COMMIT: To permanently save changes made during a transaction.
* ROLLBACK: To undo changes made during a transaction if an error occurs.

6).**Data Security**: SQL provides mechanisms to control access to data through:

* **User Authentication**: Defining user roles and access permissions.
* **GRANT and REVOKE**: These commands allow administrators to control who can access or modify certain data or database objects.

Q.5:- What are the basic components of SQL syntax?

Ans:-

1)DDL:- data denification language

2)DML:- data manupilcation language

3)DQL:- data query language

4)DCL:- data control language

5)TCL:- tranjection control language

Q.6:- Write the general structure of an SQL SELECT statement.

Ans:- SELECT *column1*,*column2, ...*  
FROM *table\_name*;

Q.7:- . Explain the role of clauses in SQL statements?

Ans:-

1 **SELECT**: Specifies the columns to be retrieved from the database. It's usually the first clause in a query.

* Example: SELECT name, age FROM employees;

2) **FROM**: Indicates the table (or tables) from which to retrieve the data.

* Example: FROM employees

3) **WHERE**: Filters the rows based on a condition. Only rows that satisfy the condition will be included in the result.

* Example: WHERE age > 30

4) **ORDER BY**: Specifies the order in which the rows are returned, usually in ascending (ASC) or descending (DESC) order.

* Example: ORDER BY name ASC

5) **GROUP BY**: Groups rows that have the same values into summary rows, often used with aggregate functions like COUNT, SUM, AVG, etc.

* Example: GROUP BY department

6) **HAVING**: Filters the results of a GROUP BY clause. It’s similar to WHERE, but it works with grouped data.

* Example: HAVING COUNT(\*) > 5

7) **JOIN**: Combines rows from two or more tables based on a related column between them. There are various types of joins, like INNER JOIN, LEFT JOIN, etc.

* Example: INNER JOIN departments ON employees.department\_id = departments.id

8) **LIMIT**: Specifies the maximum number of rows to return (commonly used in databases like MySQL and PostgreSQL).

* Example: LIMIT 10

9) **INSERT INTO**: Adds new records into a table.

* Example: INSERT INTO employees (name, age) VALUES ('John Doe', 28);

10) **UPDATE**: Modifies existing records in a table.

* Example: UPDATE employees SET age = 29 WHERE name = 'John Doe';

11) **DELETE**: Removes records from a table.

* Example: DELETE FROM employees WHERE name = 'John Doe';

Q.8:- What are constraints in SQL? List and explain the different types of constraints

**1. PRIMARY KEY**

* **Purpose**: Ensures that each row in a table has a unique identifier. It uniquely identifies a record in a table and cannot contain NULL values.
* **Example**: CREATE TABLE employees (id INT PRIMARY KEY, name VARCHAR(50));
* **Explanation**: The id column is a primary key, meaning every id must be unique and not NULL.

**2. FOREIGN KEY**

* **Purpose**: Ensures that the value in a column (or a combination of columns) corresponds to a valid value in another table's column(s). This creates a relationship between two tables.
* **Example**: CREATE TABLE orders (order\_id INT PRIMARY KEY, customer\_id INT, FOREIGN KEY (customer\_id) REFERENCES customers(id));
* **Explanation**: The customer\_id in the orders table is a foreign key that references the id in the customers table, ensuring referential integrity.

**3. UNIQUE**

* **Purpose**: Ensures that all values in a column are unique across the table, i.e., no two rows can have the same value for that column. Unlike a primary key, a unique constraint can allow NULL values.
* **Example**: CREATE TABLE employees (email VARCHAR(100) UNIQUE);
* **Explanation**: The email column must contain unique values, but it can allow NULLs.

**4. CHECK**

* **Purpose**: Ensures that values in a column meet a specific condition or rule. It's used to enforce domain integrity.
* **Example**: CREATE TABLE employees (age INT CHECK (age >= 18));
* **Explanation**: The age column must contain values that are 18 or older.

**5. NOT NULL**

* **Purpose**: Ensures that a column cannot contain NULL values. This is used when it's essential for a column to always have a valid value.
* **Example**: CREATE TABLE employees (name VARCHAR(50) NOT NULL);
* **Explanation**: The name column must always have a value, and cannot be left blank (NULL).

**6. DEFAULT**

* **Purpose**: Provides a default value for a column when no value is specified during an insert operation.
* **Example**: CREATE TABLE employees (id INT, status VARCHAR(20) DEFAULT 'Active');
* **Explanation**: If no value is provided for status, it will automatically be set to 'Active'.

**7. AUTO\_INCREMENT (or SERIAL)**

* **Purpose**: Automatically generates a unique number for a column (usually for a primary key). This is useful when you need a column to act as a counter that increments every time a new row is inserted.
* **Example** (MySQL): CREATE TABLE employees (id INT AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(50));
* **Explanation**: The id column will automatically increment with each new row added, ensuring unique values.

**8. INDEX (not always considered a constraint, but sometimes grouped)**

* **Purpose**: Improves the performance of query operations by creating a data structure (index) on a column or set of columns. This allows for faster searching and sorting.
* **Example**: CREATE INDEX idx\_name ON employees (name);
* **Explanation**: An index is created on the name column, which speeds up query execution that involves searching or sorting by name.

Q.9:- How do PRIMARY KEY and FOREIGN KEY constraints differ?

**Key Differences Summary:**

| **Attribute** | **PRIMARY KEY** | **FOREIGN KEY** |
| --- | --- | --- |
| **Purpose** | Uniquely identifies records in the table | Links records in one table to records in another |
| **Uniqueness** | Must have unique values | Can have duplicate values |
| **NULL Values** | Cannot contain NULL values | Can contain NULL values (depending on definition) |
| **Table** | Defined within the same table | Refers to another table's primary or unique key |
| **Number of Keys** | Only one primary key per table | Can have multiple foreign keys in a table |
| **Data Integrity** | Ensures entity integrity (uniqueness and non-null) | Ensures referential integrity (valid relationships) |

Q.10:- What is the role of NOT NULL and UNIQUE constraints?

Ans:-

### 1. ****NOT NULL Constraint****

* **Purpose**: The **NOT NULL** constraint ensures that a column **cannot have a NULL value**. It forces the user to always provide a value for that column when inserting or updating a record.
* **Role**: It enforces **data completeness** and **ensures that critical fields always have a value**. This is important for columns where missing data would cause problems, such as a required email address, name, or other essential pieces of information.

### 2. ****UNIQUE Constraint****

* **Purpose**: The **UNIQUE** constraint ensures that all values in a column (or a combination of columns) are **unique across all rows** in the table. This means that no two rows can have the same value for the specified column(s).
* **Role**: It enforces **data uniqueness** and ensures that there are no duplicate values in columns that must have distinct data, such as user emails, product codes, or social security numbers. However, **NULL values are allowed**, and they are treated as unique as well (depending on the database system).

Q.11:- Define the SQL Data Definition Language (DDL).

Ans:-

**Key Features of DDL:**

* **Schema Management**: DDL allows you to define the structure of a database, including tables and relationships between tables.
* **Permanent Changes**: DDL commands result in permanent changes to the database schema, and typically, these operations are **auto-committed** (i.e., they cannot be rolled back unless explicitly stated).
* **No Data Manipulation**: DDL is focused on the structure of the database objects, not on modifying the data within those objects. For manipulating data, Data Manipulation Language (DML) commands are used.

**Common DDL Commands:**

1. **CREATE**:
   * Used to create database objects like tables, views, indexes, or databases.

**2) ALTER**:

* Used to modify the structure of an existing database object (e.g., adding, deleting, or modifying columns).

3) **DROP**:

* Used to delete database objects such as tables, views, or entire databases.

4) **TRUNCATE**:

* Used to delete all rows from a table, but unlike DROP, it does not remove the table structure, only the data.

5) **RENAME**:

* Used to rename a database object (e.g., a table or a column).

Q.12:- Explain the CREATE command and its syntax.

Ans:-

In SQL, the "CREATE" command is a Data Definition Language (DDL) statement used to create new database objects, primarily tables, within a database, specifying their structure by defining column names and data types for each column; essentially, it allows you to design the layout of your data storage within a table.

Syntax:

Code

CREATE TABLE table\_name (

column1 datatype,

column2 datatype,

...

columnN datatype,

[PRIMARY KEY (column\_name)]

);

Q.13:- What is the purpose of specifying data types and constraints during table creation?

Ans:-

Key points about data types and constraints:

* **Data types:**

These define the kind of data a column can hold, like integers, text, dates, etc., which helps the database system allocate appropriate storage and perform operations correctly on the data.

* **Constraints:**

These are rules applied to columns or tables to enforce specific data validation criteria, like ensuring a field is not null, unique values are present, or that values fall within a specific range.

Benefits of using data types and constraints:

* **Data integrity:**

By defining appropriate data types and constraints, you can prevent incompatible data from being entered into a column, ensuring the data is consistent and reliable.

* **Improved query performance:**

Knowing the data type of a column allows the database to optimize queries by selecting the most efficient way to search and retrieve data.

* **Reduced errors:**

Constraints catch invalid data attempts at the database level, preventing applications from processing incorrect information.

Common types of constraints:

* **NOT NULL:** Ensures a column cannot contain a null value
* **UNIQUE:** Guarantees that all values in a column must be distinct
* **PRIMARY KEY:** Identifies a unique row in a table, usually composed of one or more columns with the UNIQUE and NOT NULL constraints
* **FOREIGN KEY:** Creates a relationship between two tables by linking a column in one table to the primary key of another table
* **CHECK:** Allows you to define a custom condition that data in a column must meet.

Q.14:- What is the use of the ALTER command in SQL?

Ans:- The ALTER command in SQL is used to modify the structure of an existing database object, such as a table, view, or index. It allows you to make changes to the schema of the database, including adding, deleting, or modifying columns in a table, or changing other properties of objects. Here are some common uses of the ALTER command.

Q.15:- How can you add, modify, and drop columns from a table using ALTER?

Ans:- ALTER TABLE is a DDL command in SQL that is used to change the structure of the existing table i.e. we can add/modify/drop/rename constraints and columns in the table or add another[primary key](https://www.shiksha.com/online-courses/articles/introduction-to-primary-key-in-sql/) to a table, and even can change the [data type](https://www.shiksha.com/online-courses/articles/sql-server-data-types/) of a particular column.

Q.16:- What is the function of the DROP command in SQL?

### Ans:- Dropping a Table

To permanently remove a table and all of its data from the database:

sql

Copy

DROP TABLE table\_name;

**Example:**

sql

Copy

DROP TABLE employees;

Q.17:- What are the implications of dropping a table from a database?

Ans:- Dropping a table from a database permanently deletes the entire table and all its data, including any associated indexes, constraints, triggers, and views that depend on it, making the data completely inaccessible and requiring the table to be recreated from scratch if needed again; essentially, it removes the table definition from the database completely, leaving no trace of its existence.

Key implications of dropping a table:

* **Data loss:**

All rows within the table are permanently deleted and cannot be recovered without a backup.

* **Invalidated dependent objects:**

Any views, stored procedures, or other database objects that reference the dropped table will become invalid and need to be updated.

* **Removed constraints and triggers:**

Any constraints (like foreign keys) or triggers associated with the table are also dropped.

* **Storage reclaim:**

The space occupied by the table on the database is freed up and can be used by other tables.

Q.18:- . Define the INSERT, UPDATE, and DELETE commands in SQL.

Ans:-

### 1. ****INSERT Command****

The INSERT command is used to add new rows of data into an existing table.

#### Syntax:

sql

Copy

INSERT INTO table\_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...);

### 2. ****UPDATE Command****

The UPDATE command is used to modify existing data within a table.

#### Syntax:

sql

Copy

UPDATE table\_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

### 3. ****DELETE Command****

The DELETE command is used to remove one or more rows from a table.

#### Syntax:

sql

Copy

DELETE FROM table\_name

WHERE condition;

Q.19:- . What is the importance of the WHERE clause in UPDATE and DELETE operations?

Ans:-

### 1. ****In UPDATE Operations****

The UPDATE command is used to modify existing records. If you don’t specify a WHERE clause, **every row** in the table will be updated with the new values you provide, potentially leading to unintended changes across the entire dataset.

#### Example Without WHERE Clause:

sql

Copy

UPDATE employees

SET salary = 50000;

This query would **set the salary of every employee** in the employees table to 50000, which is usually not the intended outcome.

#### Example With WHERE Clause:

sql

Copy

UPDATE employees

SET salary = 50000

WHERE department = 'Sales';

### 2. ****In DELETE Operations****

The DELETE command removes rows from a table. Similarly, if you omit the WHERE clause, **all rows** in the table will be deleted, which might result in the complete loss of data in that table.

#### Example Without WHERE Clause:

sql

Copy

DELETE FROM employees;

This query will **delete all rows** in the employees table, effectively removing all employee records.

#### Example With WHERE Clause:

sql

Copy

DELETE FROM employees

WHERE department = 'Sales';

Q.20:- . What is the SELECT statement, and how is it used to query data?

Ans:-

1. **SELECT**: Specifies the columns you want to retrieve.

2.**FROM**: Identifies the table(s) from which to retrieve data.

3. **WHERE**: Filters rows based on a condition.

4. **ORDER BY**: Sorts the result set.

5. **LIMIT / TOP**: Limits the number of rows returned.

6. **Aggregate Functions**: Performs calculations like COUNT(), SUM(), etc.

7. **DISTINCT**: Eliminates duplicate rows.

Q.21:- Explain the use of the ORDER BY and WHERE clauses in SQL queries.

Ans:-

### 1. ****WHERE Clause: Filtering Data****

The WHERE clause is used to filter rows based on a specified condition. It limits the result set to only those rows that meet the criteria defined in the condition. You can use various operators in the WHERE clause, such as =, >, <, IN, BETWEEN, LIKE, etc.

#### Syntax:

sql

Copy

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

* **condition**: Specifies the criteria used to filter the rows. Rows that do not meet the condition are excluded from the result.

#### Example 1: Basic Filtering

sql

Copy

SELECT first\_name, last\_name, department

FROM employees

WHERE department = 'Sales';

* **Purpose**: This query retrieves the first\_name, last\_name, and department of employees who work in the 'Sales' department.

#### Example 2: Using Comparison Operators

sql

Copy

SELECT \*

FROM employees

WHERE salary > 50000;

* **Purpose**: This query retrieves all columns for employees with a salary greater than 50,000.

#### Example 3: Using IN, BETWEEN, and LIKE

sql

Copy

SELECT \*

FROM employees

WHERE department IN ('Sales', 'Marketing')

AND hire\_date BETWEEN '2020-01-01' AND '2023-01-01';

* **Purpose**: This query retrieves employees from the 'Sales' and 'Marketing' departments, hired between January 1, 2020, and January 1, 2023.

sql

Copy

SELECT \*

FROM employees

WHERE first\_name LIKE 'J%';

### 2. ****ORDER BY Clause: Sorting Data****

The ORDER BY clause is used to sort the result set in either **ascending** (ASC) or **descending** (DESC) order. By default, the ORDER BY clause sorts in ascending order. You can specify one or more columns to sort by.

#### Syntax:

sql

Copy

SELECT column1, column2, ...

FROM table\_name

ORDER BY column1 [ASC|DESC], column2 [ASC|DESC], ...;

* **column1, column2,...**: Columns by which to sort the result set.
* **ASC**: Sorts the data in **ascending** order (default).
* **DESC**: Sorts the data in **descending** order.

#### Example 1: Sorting by a Single Column

sql

Copy

SELECT first\_name, last\_name, hire\_date

FROM employees

ORDER BY hire\_date ASC;

* **Purpose**: This query sorts employees by their hire\_date in ascending order (earliest hire date first).

#### Example 2: Sorting by Multiple Columns

sql

Copy

SELECT first\_name, last\_name, department, salary

FROM employees

ORDER BY department ASC, salary DESC;

* **Purpose**: This query sorts employees first by their department in ascending order, and for employees in the same department, it sorts them by salary in descending order.

#### Example 3: Sorting in Descending Order

sql

Copy

SELECT \*

FROM employees

ORDER BY salary DESC;

* **Purpose**: This query sorts the employees by salary in descending order, showing the highest-paid employees first.

### Combining WHERE and ORDER BY Clauses

You can combine both WHERE and ORDER BY clauses in a single query. First, the WHERE clause filters the rows based on a condition, and then the ORDER BY clause sorts the filtered rows.

#### Example: Filtering and Sorting

sql

Copy

SELECT first\_name, last\_name, department, salary

FROM employees

WHERE department = 'Sales'

ORDER BY salary DESC;

* **Purpose**: This query retrieves employees from the 'Sales' department and sorts them by salary in descending order (highest salary first).

### Key Points:

* **WHERE clause**: Filters the rows returned by the query based on a condition.
  + It is essential for narrowing down the result set to only the relevant data.
  + It can use various comparison operators (e.g., =, >, LIKE, IN, etc.) to apply conditions.
* **ORDER BY clause**: Sorts the result set by one or more columns.
  + It helps organize data in a meaningful way, either in ascending (ASC) or descending (DESC) order.
  + Multiple columns can be sorted together, and sorting is done sequentially by each column listed.

### Conclusion:

* Use **WHERE** to filter rows based on specific conditions.
* Use **ORDER BY** to sort the filtered result set based on one or more columns.
* Both clauses are often used together to get a precise and well-organized result set from a database query.

Q.22:- What is the purpose of GRANT and REVOKE in SQL?

Ans:- **1. GRANT Statement**

The **GRANT** statement is used to give privileges to users or roles. These privileges allow them to perform specific actions on database objects.

**Syntax:**

sql

CopyEdit

GRANT privilege\_name ON object\_name TO user\_name [WITH GRANT OPTION];

* **privilege\_name**: Specifies the type of access (e.g., SELECT, INSERT, UPDATE, DELETE, EXECUTE, etc.).
* **object\_name**: The database object (e.g., table, view, procedure) on which the privilege is granted.
* **user\_name**: The user or role to whom the privilege is being granted.
* **WITH GRANT OPTION**: Allows the recipient to grant the same privileges to other users.

**2. REVOKE Statement**

The **REVOKE** statement is used to remove previously granted privileges from users or roles. It ensures that users lose access to specific database objects or actions.

**Syntax:**

sql

CopyEdit

REVOKE privilege\_name ON object\_name FROM user\_name;

* **privilege\_name**: Specifies the type of privilege to revoke.
* **object\_name**: The database object for which the privilege is being revoked.
* **user\_name**: The user or role from whom the privilege is being revoked.

Q.23:- How do you manage privileges using these commands?

Ans:- **SQL Databases (e.g., MySQL, PostgreSQL)**

* **GRANT**: Grants specific privileges to a user or role.

sql

GRANT SELECT, INSERT ON database.table TO 'username'@'host';

* **REVOKE**: Revokes specific privileges from a user or role.

sql

REVOKE INSERT ON database.table FROM 'username'@'host';

Q.24:- What is the purpose of the COMMIT and ROLLBACK commands in SQL?

1. Ans:- **COMMIT**:
   * The COMMIT command is used to permanently save all changes made during the current transaction to the database.
   * Once a COMMIT is executed, the changes become permanent and cannot be undone.
   * Example:

sql

CopyEdit

START TRANSACTION;

UPDATE accounts SET balance = balance - 100 WHERE account\_id = 1;

UPDATE accounts SET balance = balance + 100 WHERE account\_id = 2;

COMMIT;

1. **ROLLBACK**:
   * The ROLLBACK command is used to undo all changes made during the current transaction.
   * It reverts the database to the state it was in before the transaction began.
   * Example:

sql

CopyEdit

START TRANSACTION;

UPDATE accounts SET balance = balance - 100 WHERE account\_id = 1;

UPDATE accounts SET balance = balance + 100 WHERE account\_id = 2;

ROLLBACK;

Q.25:- Explain how transactions are managed in SQL databases.

Ans:- The most common commands used for managing transactions are: BEGIN TRANSACTION: Starts a transaction. COMMIT: Commits the transaction, saving all changes made. ROLLBACK: Rolls back the transaction, undoing all changes made since the BEGIN TRANSACTION

Q.26:- Explain the concept of JOIN in SQL. What is the difference between INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN?

Ans:- **Understanding JOIN in SQL**

In SQL, a **JOIN** is used to combine rows from two or more tables based on a related column between them. This allows for retrieving data from multiple tables in a structured manner.

**Types of Joins in SQL**

There are several types of JOINs in SQL, each serving a specific purpose:

| **JOIN Type** | **Includes Matching Rows** | **Includes Non-Matching Rows** |
| --- | --- | --- |
| **INNER JOIN** | ✅ Yes | ❌ No |
| **LEFT JOIN** | ✅ Yes | ✅ From Left Table |
| **RIGHT JOIN** | ✅ Yes | ✅ From Right Table |
| **FULL OUTER JOIN** | ✅ Yes | ✅ From Both Tables |

**1. INNER JOIN (Default JOIN)**

* Retrieves only the rows that have matching values in both tables.
* Rows without a match are **excluded**.

**Example:**

Let's assume we have two tables:

**Customers Table (customers)**

| **customer\_id** | **name** |
| --- | --- |
| 1 | Alice |
| 2 | Bob |
| 3 | Charlie |

**Orders Table (orders)**

| **order\_id** | **customer\_id** | **amount** |
| --- | --- | --- |
| 101 | 1 | 500 |
| 102 | 2 | 700 |
| 103 | 4 | 900 |

SELECT customers.customer\_id, customers.name, orders.order\_id, orders.amount

FROM customers

INNER JOIN orders ON customers.customer\_id = orders.customer\_id;

**Result:**

| **customer\_id** | **name** | **order\_id** | **amount** |
| --- | --- | --- | --- |
| 1 | Alice | 101 | 500 |
| 2 | Bob | 102 | 700 |

**Note:**

* Customer Charlie (ID 3) is missing because they don’t have an order.
* Order ID 103 is missing because no customer exists with customer\_id = 4.

**2. LEFT JOIN (LEFT OUTER JOIN)**

* Returns all rows from the **left table** (customers), and matching rows from the **right table** (orders).
* If there is **no match**, NULL values appear for columns from the right table.

SELECT customers.customer\_id, customers.name, orders.order\_id, orders.amount

FROM customers

LEFT JOIN orders ON customers.customer\_id = orders.customer\_id;

**Result:**

| **customer\_id** | **name** | **order\_id** | **amount** |
| --- | --- | --- | --- |
| 1 | Alice | 101 | 500 |
| 2 | Bob | 102 | 700 |
| 3 | Charlie | NULL | NULL |

**Note:**

* Charlie appears with NULL values because they have no orders.

**3. RIGHT JOIN (RIGHT OUTER JOIN)**

* Returns all rows from the **right table** (orders), and matching rows from the **left table** (customers).
* If there is **no match**, NULL values appear for columns from the left table.

SELECT customers.customer\_id, customers.name, orders.order\_id, orders.amount

FROM customers

RIGHT JOIN orders ON customers.customer\_id = orders.customer\_id;

**Result:**

| **customer\_id** | **name** | **order\_id** | **amount** |
| --- | --- | --- | --- |
| 1 | Alice | 101 | 500 |
| 2 | Bob | 102 | 700 |
| NULL | NULL | 103 | 900 |

**Note:**

* Order ID 103 appears even though there's no matching customer.

**4. FULL OUTER JOIN**

* Returns all rows from **both tables**.
* If there's a match, the row is combined. Otherwise, NULL values appear where data is missing.

SELECT customers.customer\_id, customers.name, orders.order\_id, orders.amount

FROM customers

FULL OUTER JOIN orders ON customers.customer\_id = orders.customer\_id;

**Result:**

| **customer\_id** | **name** | **order\_id** | **amount** |
| --- | --- | --- | --- |
| 1 | Alice | 101 | 500 |
| 2 | Bob | 102 | 700 |
| 3 | Charlie | NULL | NULL |
| NULL | NULL | 103 | 900 |

**Note:**

* Includes all rows from both tables.
* Charlie appears with NULL for order\_id and amount (no orders).
* Order ID 103 appears with NULL for customer\_id and name (no matching customer).

**Conclusion**

| **JOIN Type** | **Includes Matching Rows** | **Includes Non-Matching Rows** |
| --- | --- | --- |
| **INNER JOIN** | ✅ Yes | ❌ No |
| **LEFT JOIN** | ✅ Yes | ✅ From Left Table |
| **RIGHT JOIN** | ✅ Yes | ✅ From Right Table |
| **FULL OUTER JOIN** | ✅ Yes | ✅ From Both Tables |

**Q.28:- How are joins used to combine data from multiple tables?**

**Ans:- a) INNER JOIN (Matches Only)**

* **Returns only the rows where there is a match in both tables.**

**sql**

**CopyEdit**

**SELECT customers.customer\_id, customers.name, orders.order\_id, orders.amount**

**FROM customers**

**INNER JOIN orders ON customers.customer\_id = orders.customer\_id;**

**Result:**

| **customer\_id** | **name** | **order\_id** | **amount** |
| --- | --- | --- | --- |
| **1** | **Alice** | **101** | **500** |
| **2** | **Bob** | **102** | **700** |

**🔹 Why?**

* **Charlie (ID 3) is missing because they have no orders.**
* **Order 103 (ID 4) is missing because no customer has that ID.**

**b) LEFT JOIN (All from Left, Matches from Right)**

* **Returns all rows from the left table (customers).**
* **If no match in the right table (orders), NULL values appear.**

**sql**

**CopyEdit**

**SELECT customers.customer\_id, customers.name, orders.order\_id, orders.amount**

**FROM customers**

**LEFT JOIN orders ON customers.customer\_id = orders.customer\_id;**

**Result:**

| **customer\_id** | **name** | **order\_id** | **amount** |
| --- | --- | --- | --- |
| **1** | **Alice** | **101** | **500** |
| **2** | **Bob** | **102** | **700** |
| **3** | **Charlie** | **NULL** | **NULL** |

**🔹 Why?**

* **Charlie appears with NULL values because they have no orders.**

**c) RIGHT JOIN (All from Right, Matches from Left)**

* **Returns all rows from the right table (orders).**
* **If no match in the left table (customers), NULL values appear.**

**sql**

**CopyEdit**

**SELECT customers.customer\_id, customers.name, orders.order\_id, orders.amount**

**FROM customers**

**RIGHT JOIN orders ON customers.customer\_id = orders.customer\_id;**

**Result:**

| **customer\_id** | **name** | **order\_id** | **amount** |
| --- | --- | --- | --- |
| **1** | **Alice** | **101** | **500** |
| **2** | **Bob** | **102** | **700** |
| **NULL** | **NULL** | **103** | **900** |

**🔹 Why?**

* **Order 103 appears with NULL values because there is no matching customer.**

**d) FULL OUTER JOIN (All Data from Both Tables)**

* **Returns all rows from both tables.**
* **If there is no match, NULL values appear in the missing columns.**

**sql**

**CopyEdit**

**SELECT customers.customer\_id, customers.name, orders.order\_id, orders.amount**

**FROM customers**

**FULL OUTER JOIN orders ON customers.customer\_id = orders.customer\_id;**

**Result:**

| **customer\_id** | **name** | **order\_id** | **amount** |
| --- | --- | --- | --- |
| **1** | **Alice** | **101** | **500** |
| **2** | **Bob** | **102** | **700** |
| **3** | **Charlie** | **NULL** | **NULL** |
| **NULL** | **NULL** | **103** | **900** |
|  |  |  |  |

Q.29:- What is the GROUP BY clause in SQL? How is it used with aggregate functions?

Ans:- Purpose of GROUP BY

The GROUP BY clause groups rows that have the same values in the specified columns. This allows you to perform aggregate calculations on each group rather than on the entire table.

Syntax

sql

Copy

SELECT column1, column2, aggregate\_function(column3)

FROM table\_name

WHERE condition

GROUP BY column1, column2;

column1, column2: Columns used to group the rows.

aggregate\_function(column3): An aggregate function (e.g., SUM, COUNT, AVG) applied to each group.

WHERE: Optional condition to filter rows before grouping.

GROUP BY: Specifies the columns to group by.

Common Aggregate Functions

COUNT(): Counts the number of rows in each group.

SUM(): Calculates the total sum of a numeric column in each group.

AVG(): Calculates the average value of a numeric column in each group.

MAX(): Finds the maximum value in a column for each group.

MIN(): Finds the minimum value in a column for each group.

Q.30:- Explain the difference between GROUP BY and ORDER BY.

Ans:-

| **GROUP BY** | **ORDER BY** |
| --- | --- |
| Group by statement is used to group the rows that have the same value. | Whereas Order by statement sort the result-set either in ascending or descending order. |
| It may be allowed in CREATE VIEW statement. | While it does not use in CREATE VIEW statement. |
| In select statements, it is always used before the order by keyword. | While in the select statement, it is always used after the group by keyword. |
| An attribute cannot be in the group by a statement under the aggregate function. | Whereas in order by statement, the attribute can be under aggregate function. |
| In group by clause, the tuples are grouped based on the similarity between the attribute values of tuples. | Whereas in order by clause, the result set is sorted based on ascending or descending order. |
| Group by controls the presentation of [tuples(rows).](https://www.geeksforgeeks.org/tuple-in-dbms/) | While order by clause controls the presentation of columns. |

Q.31:- What is a stored procedure in SQL, and how does it differ from a standard SQL query?

Ans:- **Key Differences:**

| **Feature** | **Stored Procedure** | **Standard SQL Query** |
| --- | --- | --- |
| **Definition** | Predefined SQL block stored in the database | One-time SQL command executed by the user |
| **Reusability** | Can be executed multiple times | Must be rewritten/executed each time |
| **Performance** | Precompiled, faster execution | Parsed and executed every time |
| **Logic Handling** | Supports loops, conditions (IF, WHILE) | Single query execution |
| **Security** | Can restrict direct table access | Requires user permissions for each execution |

Q.32:- Explain the advantages of using stored procedures.

Ans:-

**1. Improved Performance**

* Precompiled and cached in the database, reducing execution time.

**2. Reusability**

* Can be executed multiple times without rewriting queries.

**3. Security**

* Restricts direct table access and allows controlled execution via permissions.

**4. Reduced Network Traffic**

* Only procedure calls are sent, not entire queries, reducing data transfer.

**5. Maintainability**

* Centralized logic simplifies updates and debugging.

**6. Supports Complex Logic**

* Allows loops, conditions (IF, WHILE), and error handling within SQL.

**7. Transaction Management**

* Can include COMMIT and ROLLBACK to ensure data integrity.

Q.33:- What is a view in SQL, and how is it different from a table?

Ans:- **What Is a View?**

A **view** is a **virtual table** based on a SQL query. It does not store data but dynamically presents results from one or more tables.

**Key Differences:**

| **Feature** | **View** | **Table** |
| --- | --- | --- |
| **Storage** | No physical storage, just a query result | Stores actual data |
| **Data Updates** | Reflects changes in underlying tables | Stores fixed data until modified |
| **Performance** | Faster for read operations, but may slow down with complex queries | Optimized for direct data access |
| **Security** | Can restrict column access (e.g., hiding sensitive data) | Provides full access to stored data |

**Example of a View:**

sql

CopyEdit

CREATE VIEW CustomerOrders AS

SELECT customers.name, orders.order\_id, orders.amount

FROM customers

JOIN orders ON customers.customer\_id = orders.customer\_id;

**Use it like a table:**

sql

CopyEdit

SELECT \* FROM CustomerOrders;

Q.34:- Explain the advantages of using views in SQL databases.

Ans:-

**1. Simplifies Complex Queries**

* Stores reusable SQL queries, making data retrieval easier.

**2. Enhances Security**

* Restricts access to specific columns or rows, hiding sensitive data.

**3. Improves Data Abstraction**

* Provides a logical representation of data without exposing table structures.

**4. Ensures Data Consistency**

* Always reflects the latest data from underlying tables.

**5. Reduces Code Duplication**

* Eliminates the need to rewrite complex joins and filters.

**6. Supports Read-Only Access**

* Can be used to prevent unwanted data modifications.

Q.35:- What is a trigger in SQL? Describe its types and when they are used.

Ans:- 1) **Before Trigger**

* Executes before the specified event occurs.
* Used for data validation and enforcing business rules.

2) **After Trigger**

* Executes after the specified event occurs.
* Used for auditing, logging, or updating related tables.

3) **Instead of Trigger**

* Executes instead of the triggering event.
* Used in views to handle modifications that are otherwise not directly possible.

4) **Row-Level Trigger**

* Executes once for each affected row.
* Used when individual row processing is required.

5) **Statement-Level Trigger**

* Executes once per SQL statement, regardless of the number of rows affected.
* Used for logging or enforcing complex business rules at the statement level.

Q.36:- Explain the difference between INSERT, UPDATE, and DELETE triggers.

Ans:-

| **Trigger Type** | **Executed On** | **Purpose** | **Example Use Case** |
| --- | --- | --- | --- |

|  |  |  |  |
| --- | --- | --- | --- |
| **INSERT Trigger** | When a new row is added to a table | Validates or modifies inserted data before insertion, logs insertions, or maintains audit tables | Automatically logs new employee records in an audit table |

|  |  |  |  |
| --- | --- | --- | --- |
| **UPDATE Trigger** | When an existing row is modified | Tracks changes, prevents unauthorized updates, or maintains history of data modifications | Logs salary changes in an employee table |

|  |  |  |  |
| --- | --- | --- | --- |
| **DELETE Trigger** | When a row is removed from a table | Prevents accidental deletions, archives deleted records, or enforces referential integrity | Moves deleted customer records to an archive table |

Q.37:- What is PL/SQL, and how does it extend SQL's capabilities?

Ans:-

1) **Procedural Logic** – Allows use of loops (FOR, WHILE), conditions (IF-ELSE), and error handling.

2) **Block Structure** – Organizes code into blocks (DECLARE, BEGIN, EXCEPTION, END).

3) **Stored Procedures & Functions** – Enables reusable, modular programming.

\* **Triggers** – Automates actions on database events.

\* **Cursors** – Handles row-by-row query processing.

\* **Exception Handling** – Manages runtime errors gracefully.

Q.38:- List and explain the benefits of using PL/SQL.

Ans:-

Procedural Capabilities – Supports loops, conditions, and functions, making SQL more powerful.

Improved Performance – Reduces network traffic by executing multiple SQL statements as a single block.

Modularity & Reusability – Supports stored procedures, functions, and packages for code reuse.

Better Security – Allows restricted access to database objects using stored procedures.

Exception Handling – Provides robust error handling with EXCEPTION blocks.

Portability – Can run on any Oracle database without modification.

Scalability – Supports complex business logic and large applications efficiently.

Triggers & Event Handling – Automates database actions like logging and auditing

Q.40:- What are control structures in PL/SQL? Explain the IF-THEN and LOOP control structures.

Ans:- **1. IF-THEN Control Structure**

* Used for conditional execution.
* Syntax:

plsql

CopyEdit

IF condition THEN

-- Execute statements

END IF;

**2. LOOP Control Structure**

* Used for repeating a set of statements until a condition is met.
* Types: LOOP, WHILE LOOP, FOR LOOP.

**Basic LOOP Example**

plsql

CopyEdit

LOOP

counter := counter + 1;

EXIT WHEN counter > 5;

END LOOP;

Q.41:- How do control structures in PL/SQL help in writing complex queries?

Ans:- 1) **Conditional Execution (IF-THEN-ELSE, CASE)** – Allows decision-making within queries.

2) **Loops (LOOP, WHILE, FOR)** – Automates repetitive tasks like processing multiple rows.

3) **Exception Handling (EXCEPTION)** – Manages runtime errors smoothly.

4) **Performance Optimization** – Reduces redundant queries and improves efficiency.

5) **Modularity** – Organizes complex logic into reusable procedures and functions.

Q.42:- What is a cursor in PL/SQL? Explain the difference between implicit and explicit cursors.

Ans:-

| **Type** | **Implicit Cursor** | **Explicit Cursor** |
| --- | --- | --- |
| **Definition** | Automatically created by PL/SQL for SELECT, INSERT, UPDATE, or DELETE statements. | Defined explicitly by the user for handling complex queries. |
| **Usage** | Used internally by PL/SQL, no declaration needed. | Requires explicit declaration, opening, fetching, and closing. |
| **Control** | PL/SQL handles it automatically. | User has full control over execution. |
| **When to Use** | For simple queries that return a single row. | For queries returning multiple rows that need row-by-row processing. |

Q.43:- When would you use an explicit cursor over an implicit one?

Ans:-

**1)Handling Multiple Rows** – For queries that return more than one row, explicit cursors allow you to process each row individually.

2) **Complex Queries** – When you need more control over the query execution, like customizing the fetch process, opening/closing the cursor, or handling different result sets.

3) **Performance Optimization** – For large result sets, explicit cursors can help manage memory and improve performance by processing rows in batches.

Q.44:- Explain the concept of SAVEPOINT in transaction management. How do ROLLBACK and COMMIT interact with savepoints?

Ans:-

* **Creating a savepoint:**

You use a "SAVEPOINT" statement to define a named marker within a transaction.

* **Rolling back to a savepoint:**

If an error occurs after a savepoint is created, you can use "ROLLBACK TO SAVEPOINT [name]" to undo all changes made after that savepoint, allowing you to continue the transaction from that point.

* **COMMIT and all savepoints:**

When you execute a "COMMIT" statement, all changes made in the current transaction, including any savepoints, become permanent and are committed to the database.

* **ROLLBACK and savepoints:**

If you use a "ROLLBACK" without specifying a savepoint, the entire transaction is rolled back, effectively discarding all changes made within the transaction, including any savepoints.

Q.45:- When is it useful to use savepoints in a database transaction?

Ans:- Savepoints are useful in database transactions when you want to set a point within a transaction that you can roll back to, without rolling back the entire transaction. This allows for partial rollback, improving flexibility and reducing the risk of losing all progress in case of an error. It's particularly helpful in complex transactions with multiple operations, where only certain parts may need to be undone.

**Practical:-**

Q.1:- Create a new database named school\_db and a table called students with the following columns: student\_id, student\_name, age, class, and address.

Ans:-

CREATE DATABASE school\_db;

USE school\_db;

CREATE TABLE students (student\_id INT AUTO\_INCREMENT PRIMARY KEY,

student\_name VARCHAR(100) ,

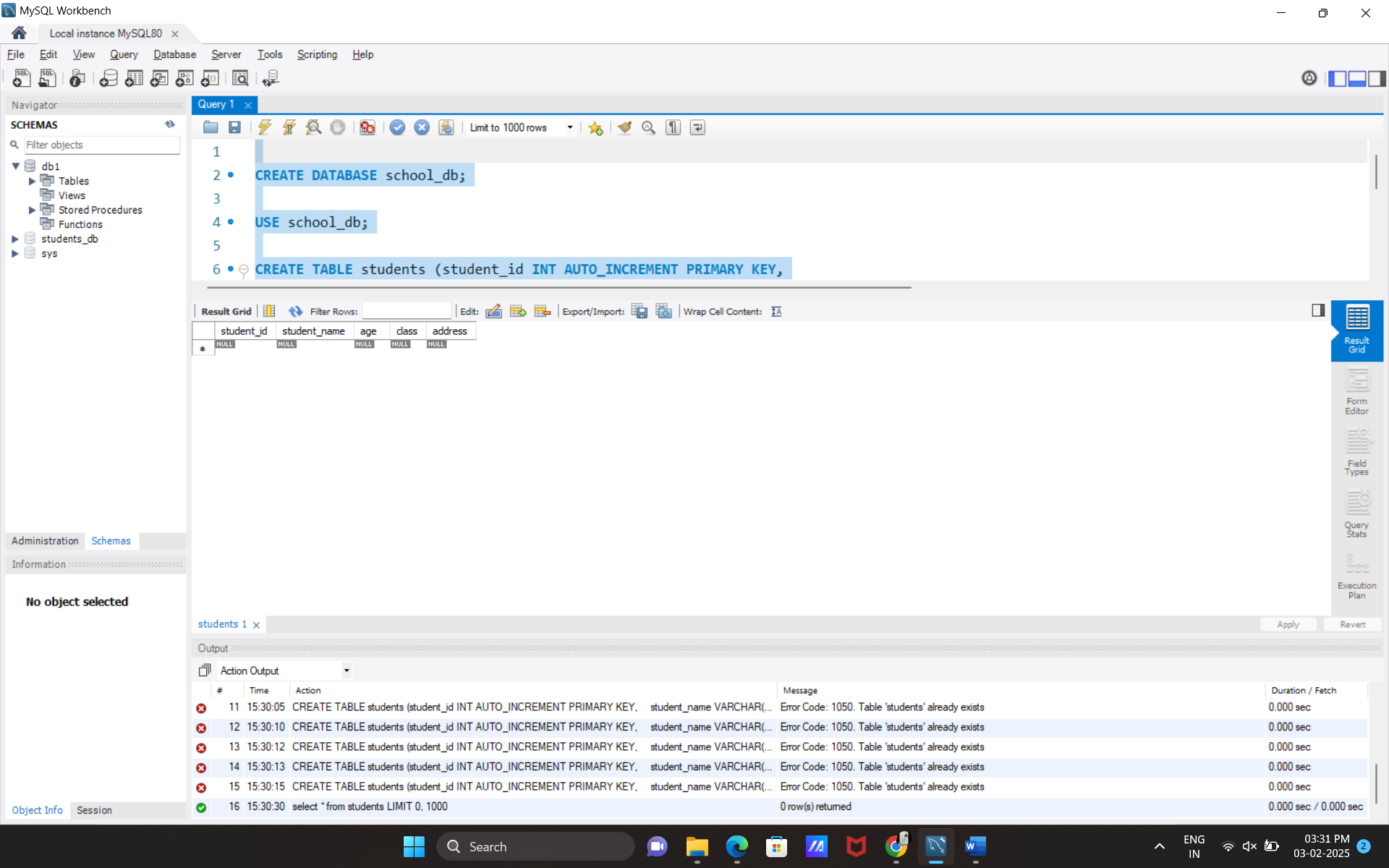
age INT ,

class VARCHAR(50),

address TEXT

);

select \* from students;



Q.2:- : Insert five records into the students table and retrieve all records using the SELECT statement.

Ans:-

INSERT INTO students (student\_name, age, class, address)

VALUES

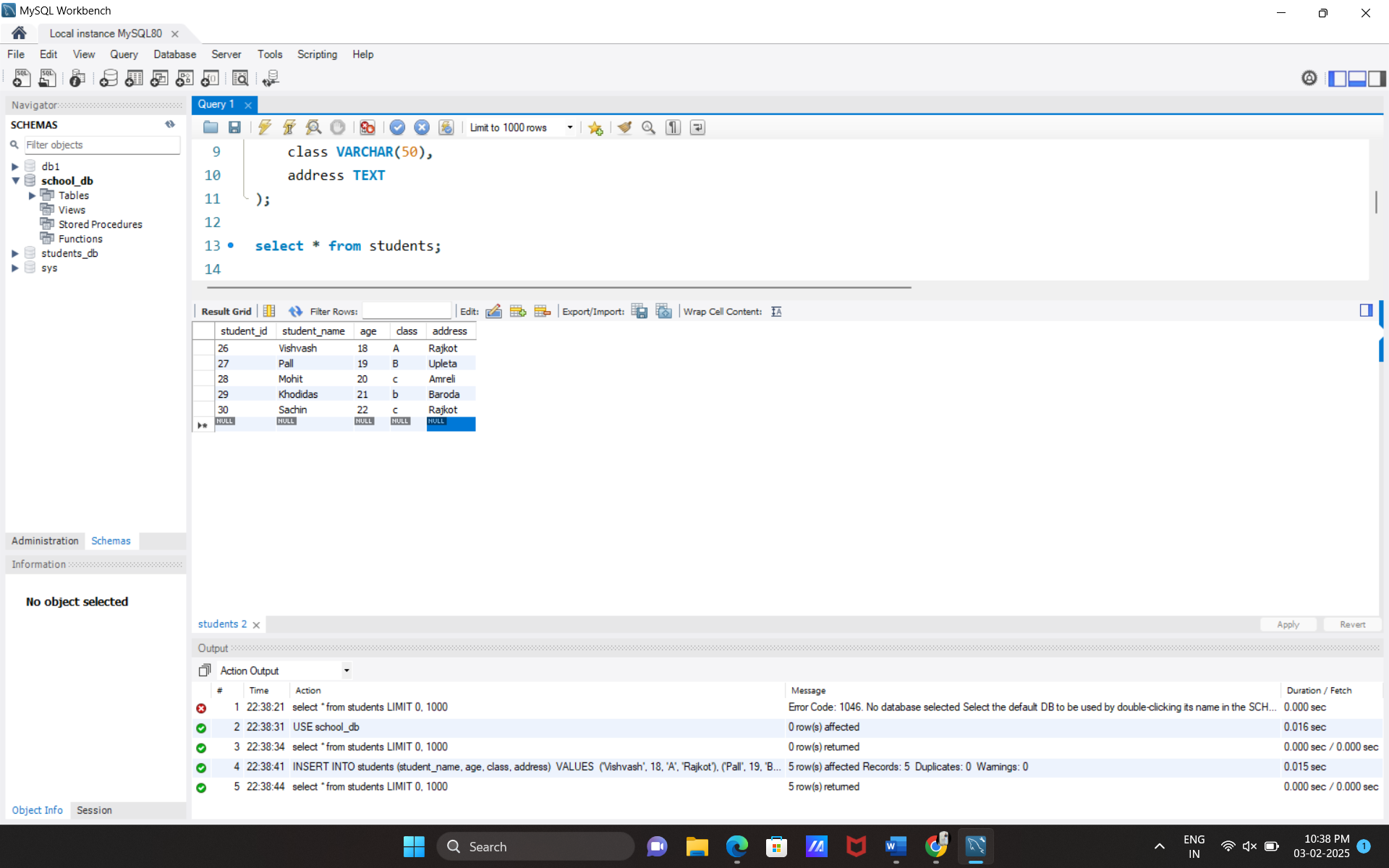
('Vishvash', 18, 'A', 'Rajkot'),

('Pall', 19, 'B', 'Upleta'),

('Mohit', 20, 'c', 'Amreli'),

('Khodidas', 21, 'b', 'Baroda'),

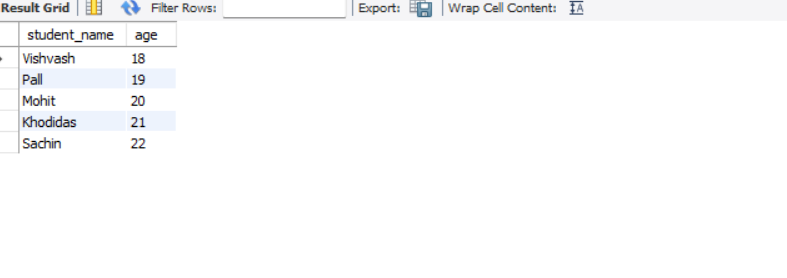
('Sachin', 22, 'c', 'Rajkot');



Q.3:- Write SQL queries to retrieve specific columns (student\_name and age) from the students table.

Ans:-

SELECT student\_name, age FROM students;



Q.4:- Write SQL queries to retrieve all students whose age is greater than 10.

Ans:-

SELECT \* FROM students

WHERE age > 10;

Q.5:- Create a table teachers with the following columns: teacher\_id (Primary Key), teacher\_name (NOT NULL), subject (NOT NULL), and email (UNIQUE).

Ans:-

CREATE TABLE teachers (

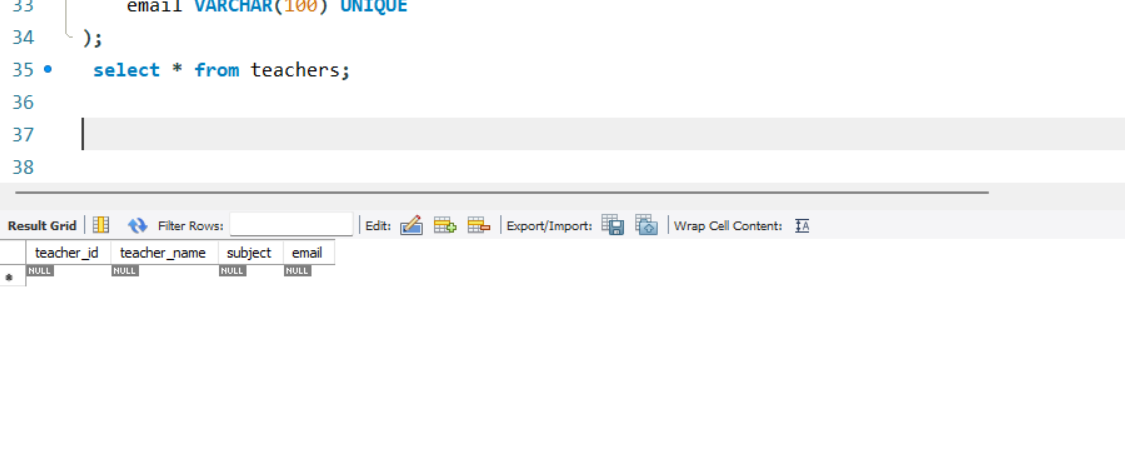
teacher\_id INT AUTO\_INCREMENT PRIMARY KEY,

teacher\_name VARCHAR(100) NOT NULL,

subject VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE

);



Q.6:- Implement a FOREIGN KEY constraint to relate the teacher\_id from the teachers table with the students table.

Ans:-

INSERT INTO teachers (teacher\_name, subject, email) VALUES

('Ramesh', 'Mathematics', 'ramesh01@gamil.com'),

('suresh', 'Science', 'suresh02@gmail.com');

INSERT INTO students (student\_name, age, class, address, teacher\_id) VALUES

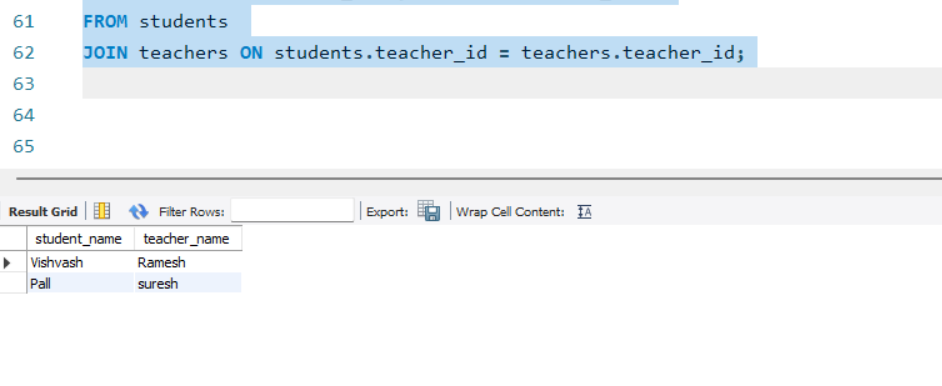
('Vishvash', 18, 'A', 'Rajkot',1),

('Pall', 19, 'B', 'Upleta',2);

SELECT students.student\_name, teachers.teacher\_name

FROM students

JOIN teachers ON students.teacher\_id = teachers.teacher\_id;



Q.7:- Create a table courses with columns: course\_id, course\_name, and course\_credits. Set the course\_id as the primary key.

Ans:-

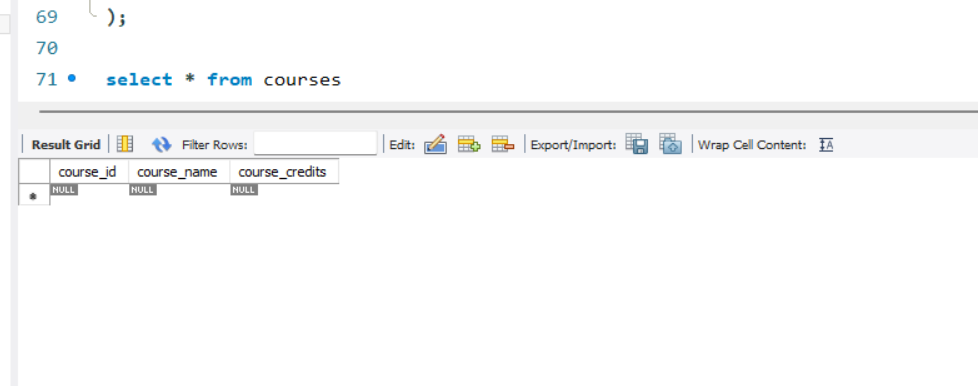
CREATE TABLE courses (

course\_id INT AUTO\_INCREMENT PRIMARY KEY,

course\_name VARCHAR(100) ,

course\_credits INT

);

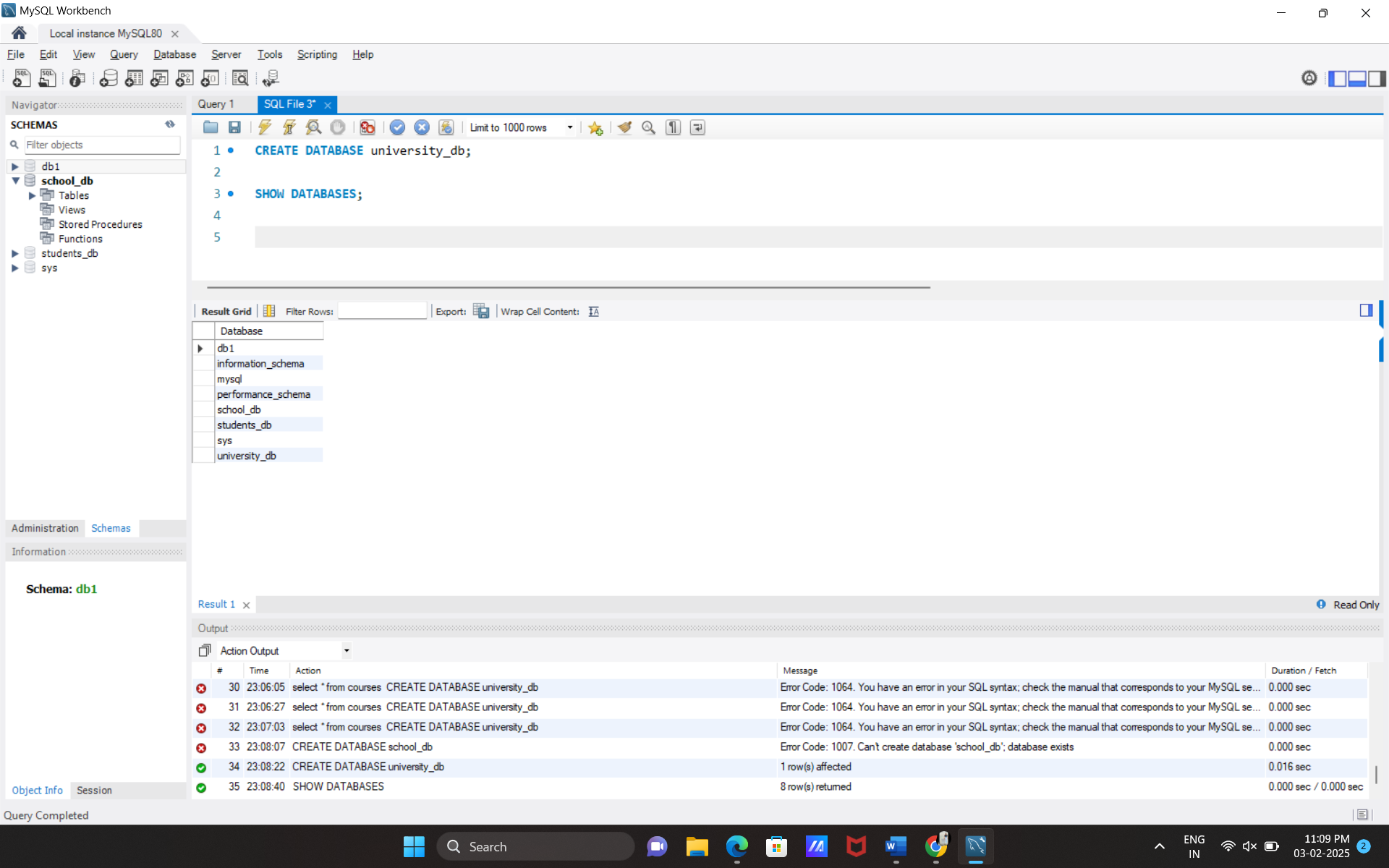


Q.8:- : Use the CREATE command to create a database university\_db.

Ans:-

CREATE DATABASE university\_db;

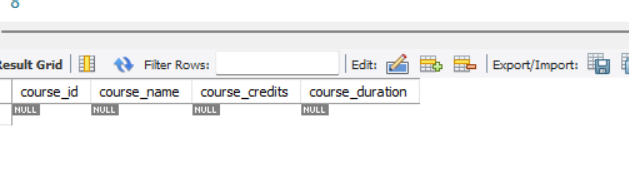
SHOW DATABASES;



Q.9:- Modify the courses table by adding a column course\_duration using the ALTER command

Ans:- ALTER TABLE courses

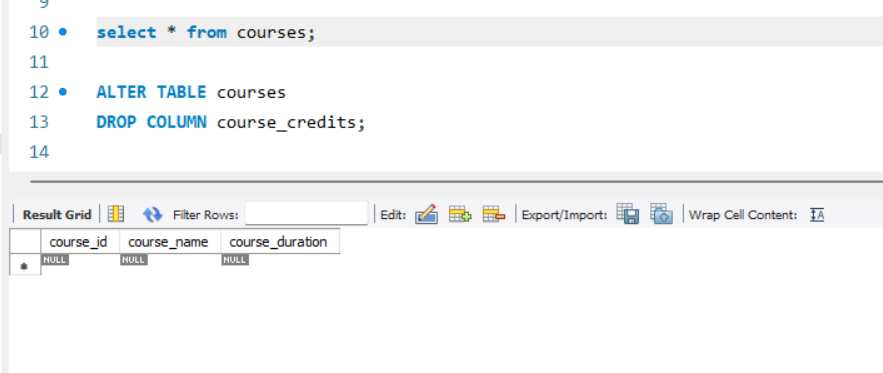
ADD COLUMN course\_duration VARCHAR(50);



Q.10:- Drop the course\_credits column from the courses table.

Ans:- ALTER TABLE courses

DROP COLUMN course\_credits;



Q.11:- : Drop the teachers table from the school\_db database.

Ans:-

use school\_db;

DROP TABLE teachers;

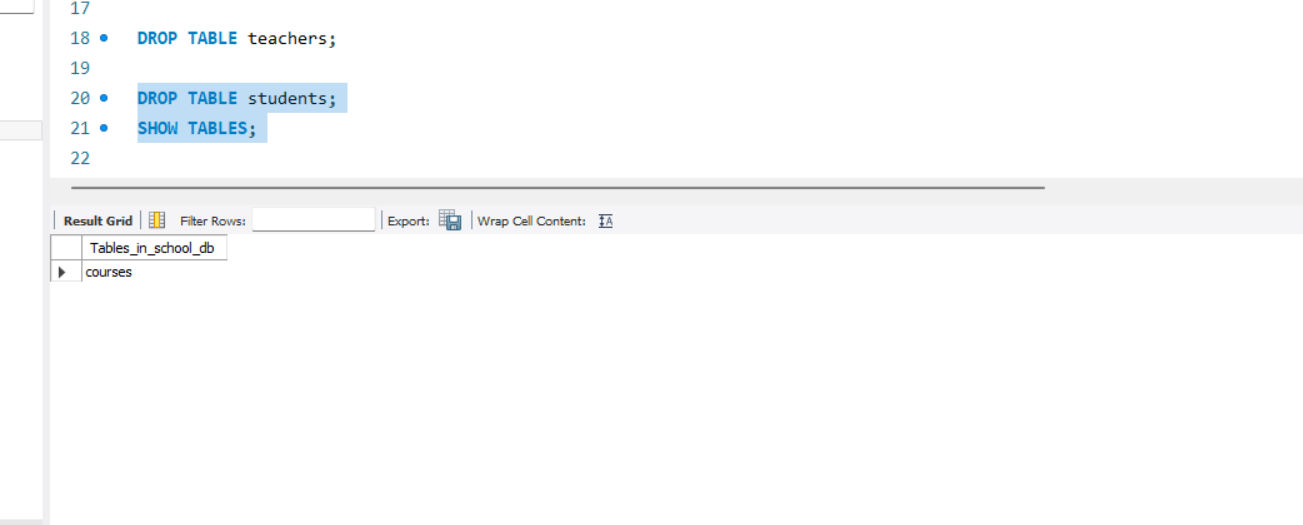
Q.12:- Drop the students table from the school\_db database a

nd verify that the table has been removed.

Ans:-

DROP TABLE students;

SHOW TABLES;



Q.13:- Insert three records into the courses table using the INSERT command.

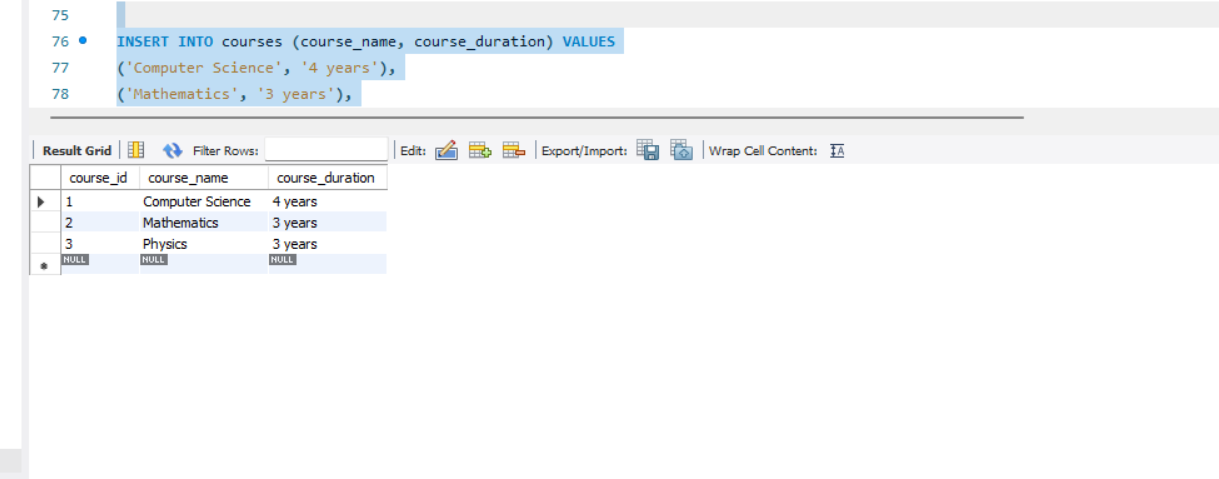
Ans:-

INSERT INTO courses (course\_name, course\_duration) VALUES

('Computer Science', '4 years'),

('Mathematics', '3 years'),

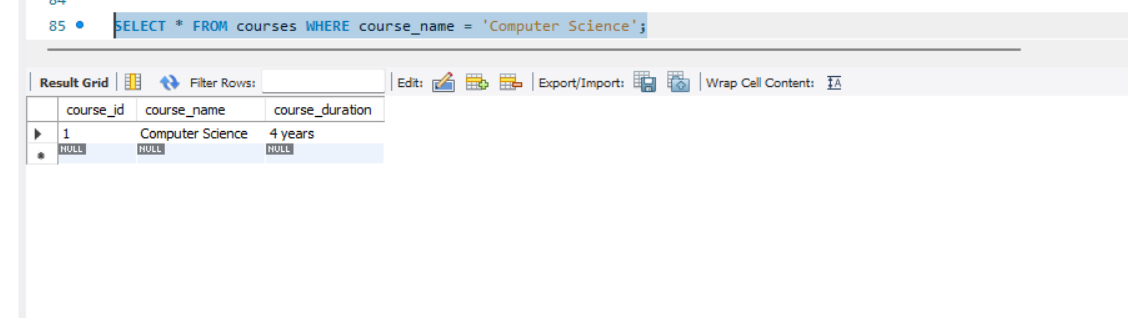
('Physics', '3 years');



Q.14:- Update the course duration of a specific course using the UPDATE command.

Ans:-

SELECT \* FROM courses WHERE course\_name = 'Computer Science';

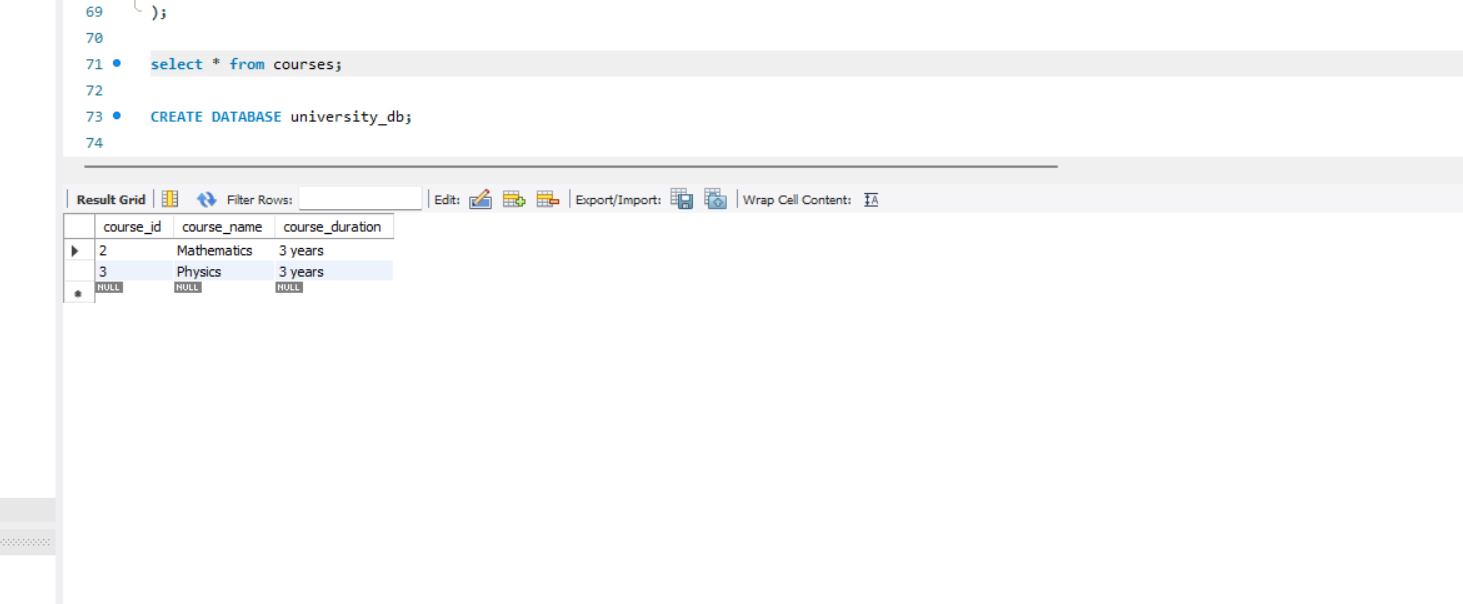


Q.15:- Delete a course with a specific course\_id from the courses table using the DELETE command.

Ans:-

DELETE FROM courses

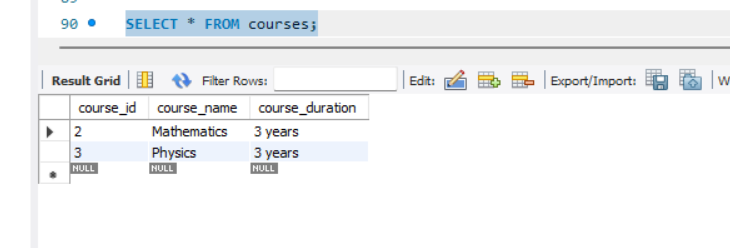
WHERE course\_id = 1;



Q.16:- Retrieve all courses from the courses table using the SELECT statement.

Ans:-

SELECT \* FROM courses;

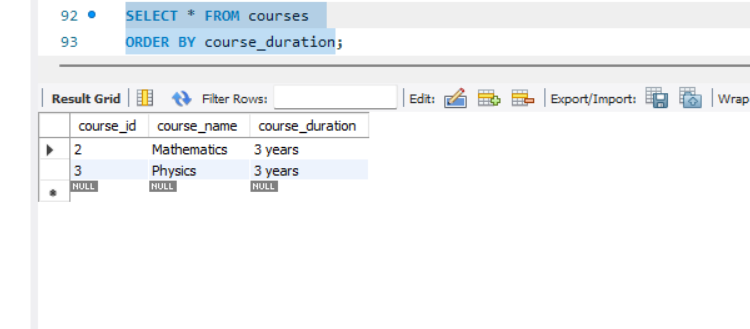


Q.17:- : Sort the courses based on course\_duration in descending order using ORDER BY.

Ans:-

SELECT \* FROM courses

ORDER BY course\_duration



Q.18:- Create two new users user1 and user2 and grant user1 permission to SELECT from the courses table.

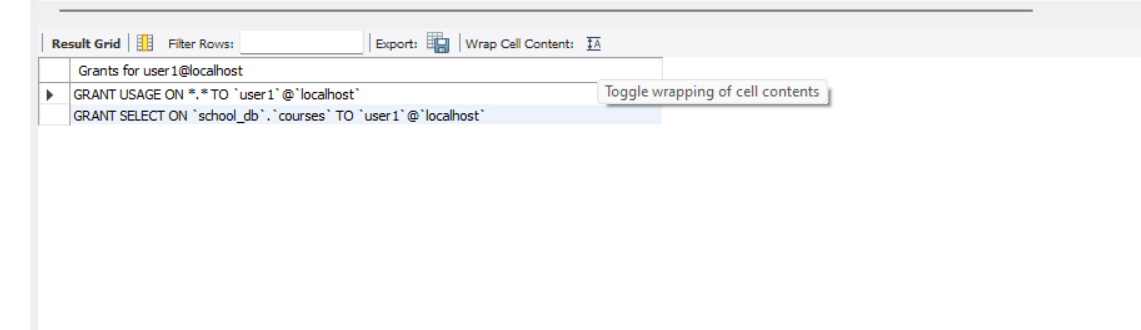
Ans:-

CREATE USER 'user1'@'localhost' IDENTIFIED BY 'password1';

CREATE USER 'user2'@'localhost' IDENTIFIED BY 'password2';

GRANT SELECT ON school\_db.courses TO 'user1'@'localhost';

SHOW GRANTS FOR 'user1'@'localhost';



Q.19:- Revoke the INSERT permission from user1 and give it to user2.

Ans:-

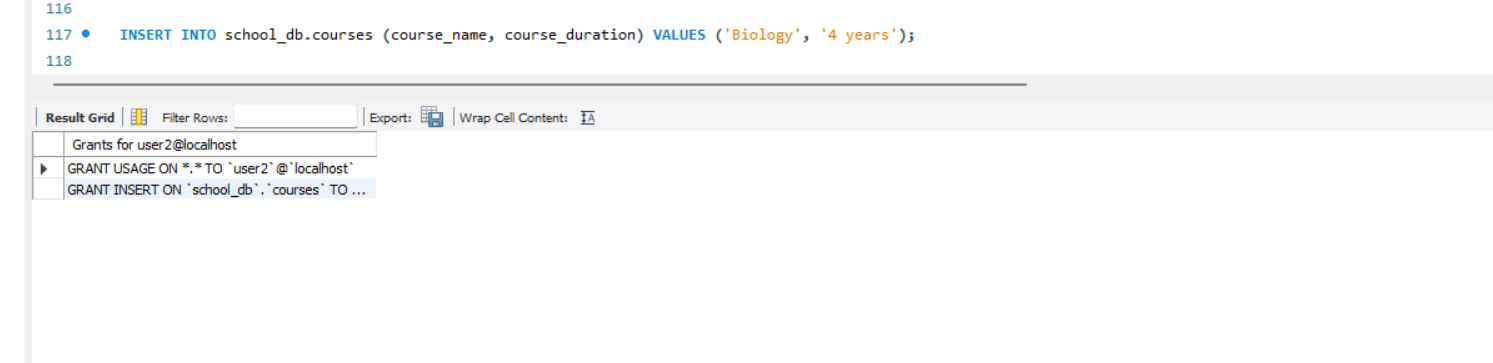
REVOKE INSERT ON school\_db.courses FROM 'user1'@'localhost';

GRANT INSERT ON school\_db.courses TO 'user2'@'localhost';

SHOW GRANTS FOR 'user1'@'localhost';

SHOW GRANTS FOR 'user2'@'localhost';

INSERT INTO school\_db.courses (course\_name, course\_duration) VALUES ('Biology', '4 years');



Q.20:- Insert a few rows into the courses table and use COMMIT to save the changes.

Ans:-

INSERT INTO courses (course\_id, course\_name, course\_duration) VALUES

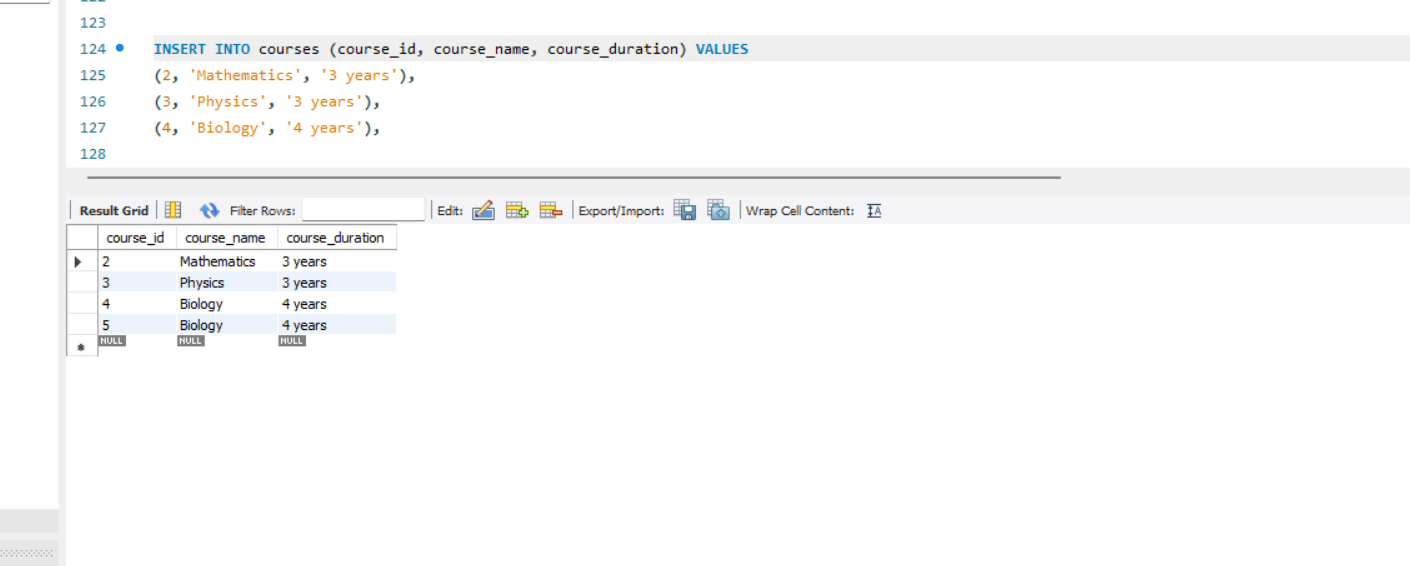
(1, 'Computer Science', '4 years'),

(2, 'Mathematics', '3 years'),

(3, 'Physics', '3 years'),

(4, 'Biology', '4 years'),

COMMIT;



Q.21:- Insert additional rows, then use ROLLBACK to undo the last insert operation.

Ans:-

INSERT INTO courses (course\_id, course\_name, course\_duration) VALUES

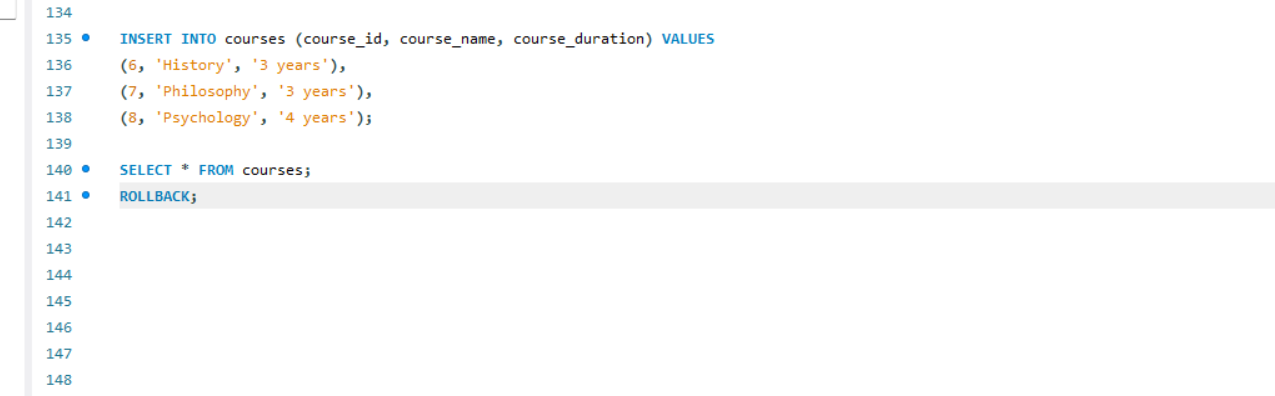
(6, 'History', '3 years'),

(7, 'Philosophy', '3 years'),

(8, 'Psychology', '4 years');

SELECT \* FROM courses;

ROLLBACK;



Q.22:- : Create two tables: departments and employees. Perform an INNER JOIN to display employees along with their respective departments.

Ans:-

CREATE TABLE departments (

department\_id INT PRIMARY KEY AUTO\_INCREMENT,

department\_name VARCHAR(100) NOT NULL

);

CREATE TABLE employees (

employee\_id INT PRIMARY KEY AUTO\_INCREMENT,

employee\_name VARCHAR(100) NOT NULL,

department\_id INT,

FOREIGN KEY (department\_id) REFERENCES departments(department\_id)

);

INSERT INTO departments (department\_name) VALUES

('HR'),

('Finance'),

('IT'),

('Marketing');

INSERT INTO employees (employee\_name, department\_id) VALUES

('vishvash', 1),

('Bob', 2),

('mohit', 3),

('rano', 4),

('kamo', 3);

SELECT

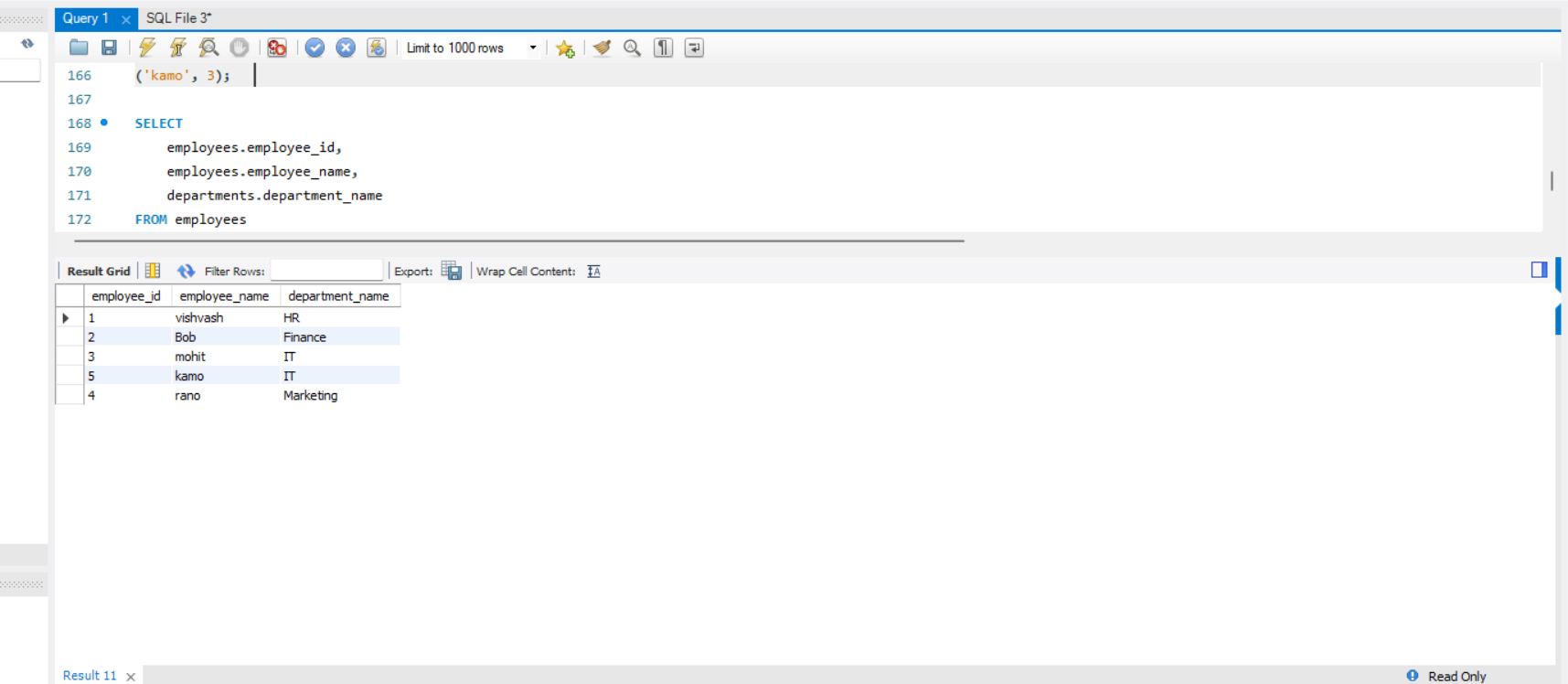
employees.employee\_id,

employees.employee\_name,

departments.department\_name

FROM employees

INNER JOIN departments ON employees.department\_id = departments.department\_id;



Q.23:- Use a LEFT JOIN to show all departments, even those without employees.

Ans:-

SELECT

departments.department\_id,

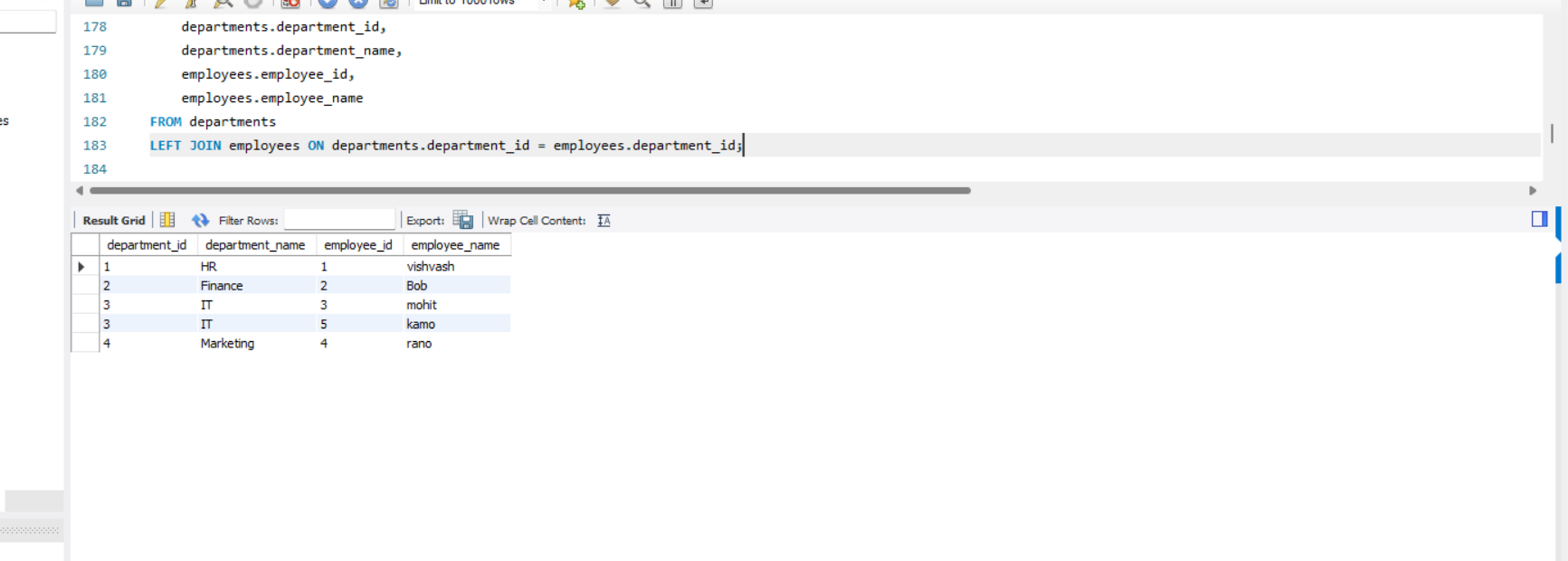
departments.department\_name,

employees.employee\_id,

employees.employee\_name

FROM departments

LEFT JOIN employees ON departments.department\_id = employees.department\_id;



Q.24:- : Group employees by department and count the number of employees in each department using GROUP BY.

Ans:-

SELECT

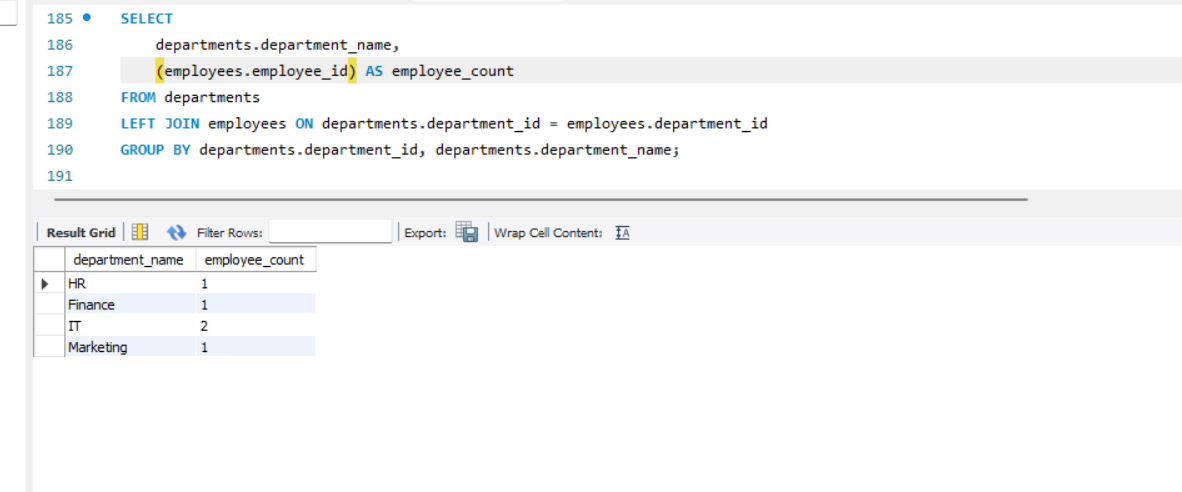
departments.department\_name,

(employees.employee\_id) AS employee\_count

FROM departments

LEFT JOIN employees ON departments.department\_id = employees.department\_id

GROUP BY departments.department\_id, departments.department\_name;



Q.25:- Write a stored procedure to retrieve all employees from the employees table based on department.

Ans:-

DELIMITER $$

CREATE PROCEDURE GetEmployeesByDepartment(IN dept\_name VARCHAR(100))

BEGIN

SELECT

employees.employee\_id,

employees.employee\_name,

employees.salary,

departments.department\_name

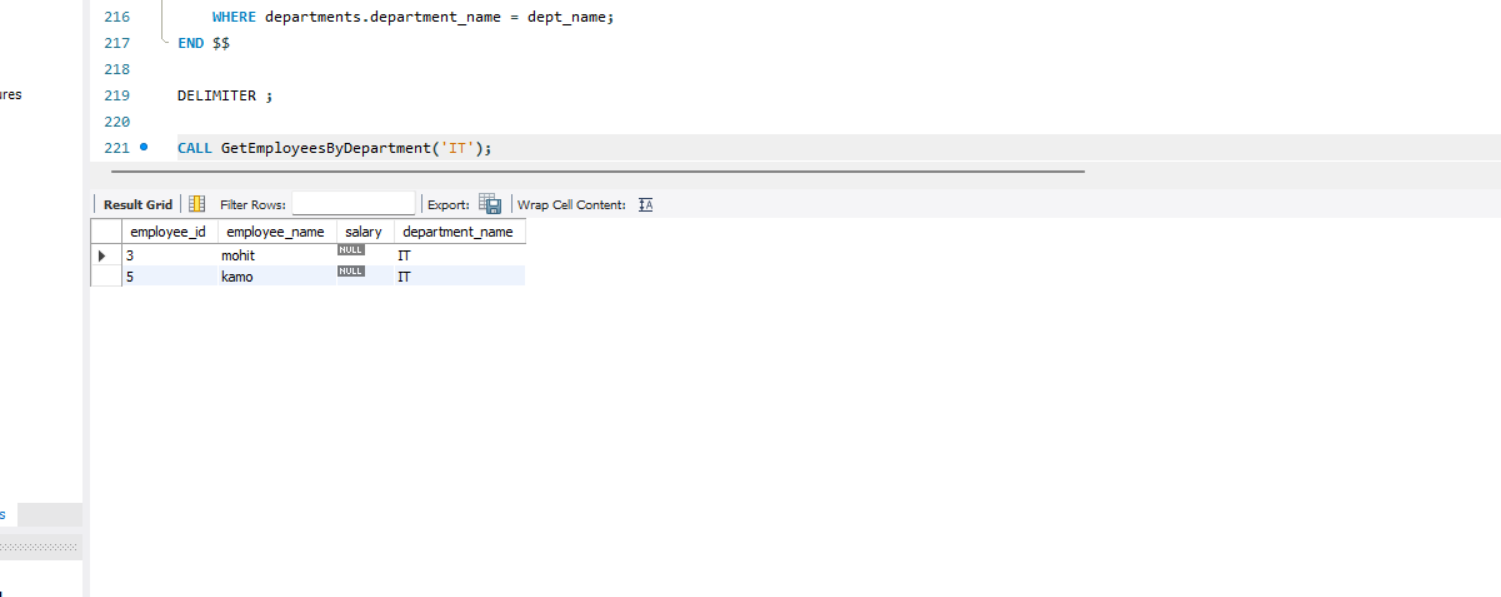
FROM employees

INNER JOIN departments ON employees.department\_id = departments.department\_id

WHERE departments.department\_name = dept\_name;

END $$

DELIMITER ;



Q.26:- Write a stored procedure that accepts course\_id as input and returns the course details.

Ans:-

CREATE PROCEDURE GetCourseDetails(IN input\_course\_id INT)

BEGIN

SELECT

course\_id,

course\_name,

course\_duration

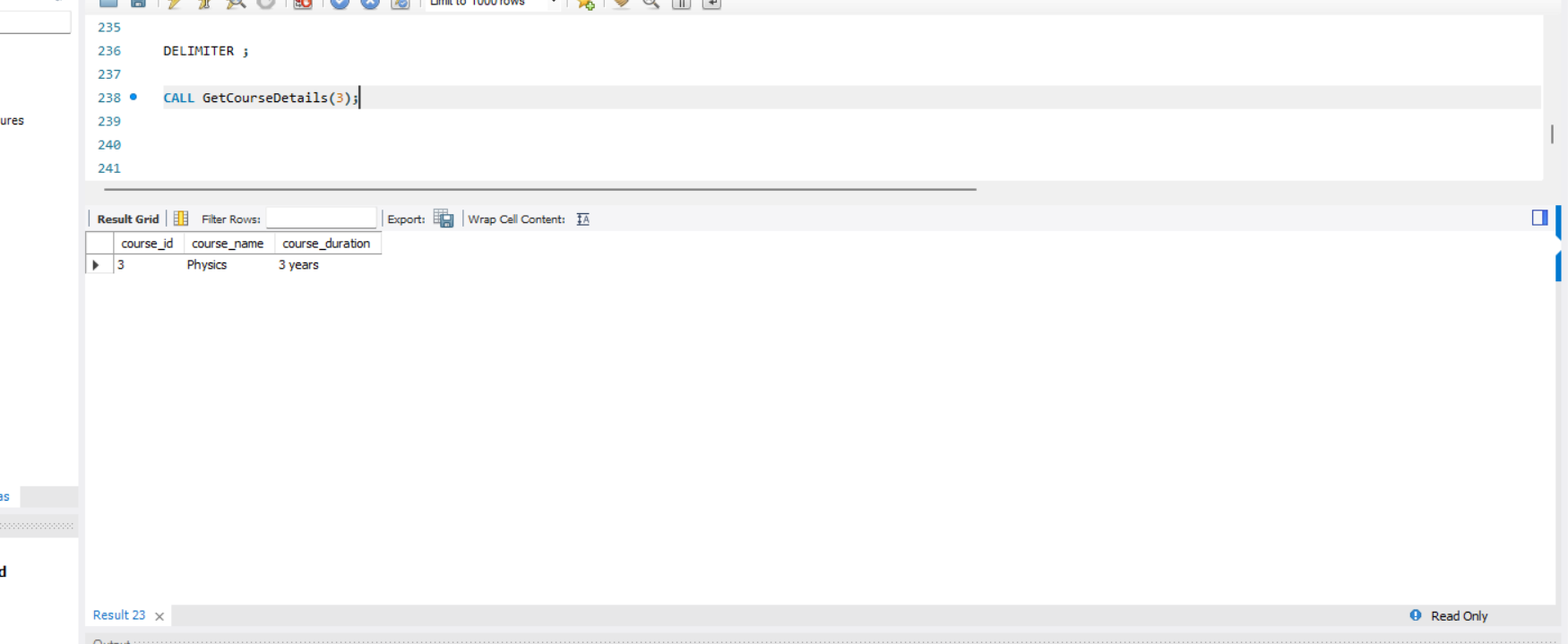
FROM courses

WHERE course\_id = input\_course\_id;

END $$

DELIMITER ;

CALL GetCourseDetails(3



Q.27:- Create a view to show all employees along with their department names.

Ans:-

CREATE VIEW EmployeeDepartmentView AS

SELECT

employees.employee\_id,

employees.employee\_name,

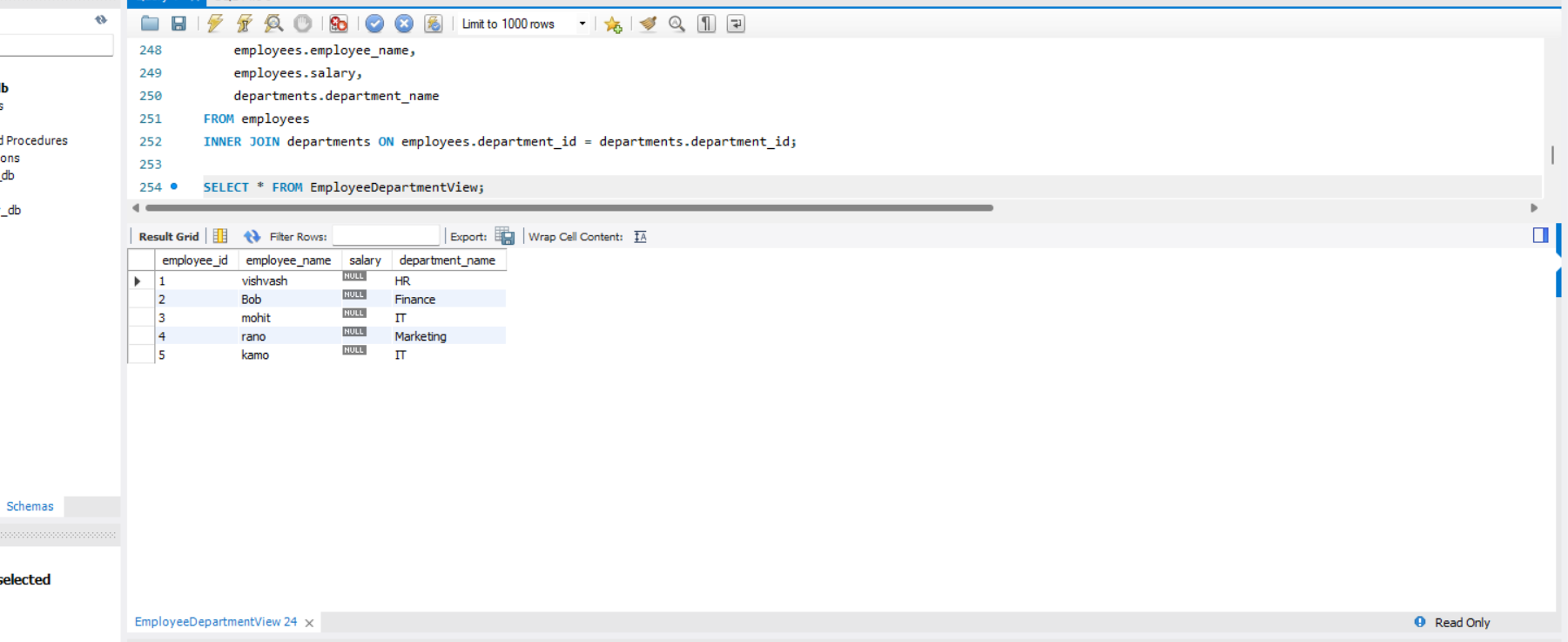
employees.salary,

departments.department\_name

FROM employees

INNER JOIN departments ON employees.department\_id = departments.department\_id;

SELECT \* FROM EmployeeDepartmentView;



Q.28:- Modify the view to exclude employees whose salaries are below $50,000.

Ans:-

CREATE VIEW EmployeeDepartmentView AS

SELECT

employees.employee\_id,

employees.employee\_name,

employees.salary,

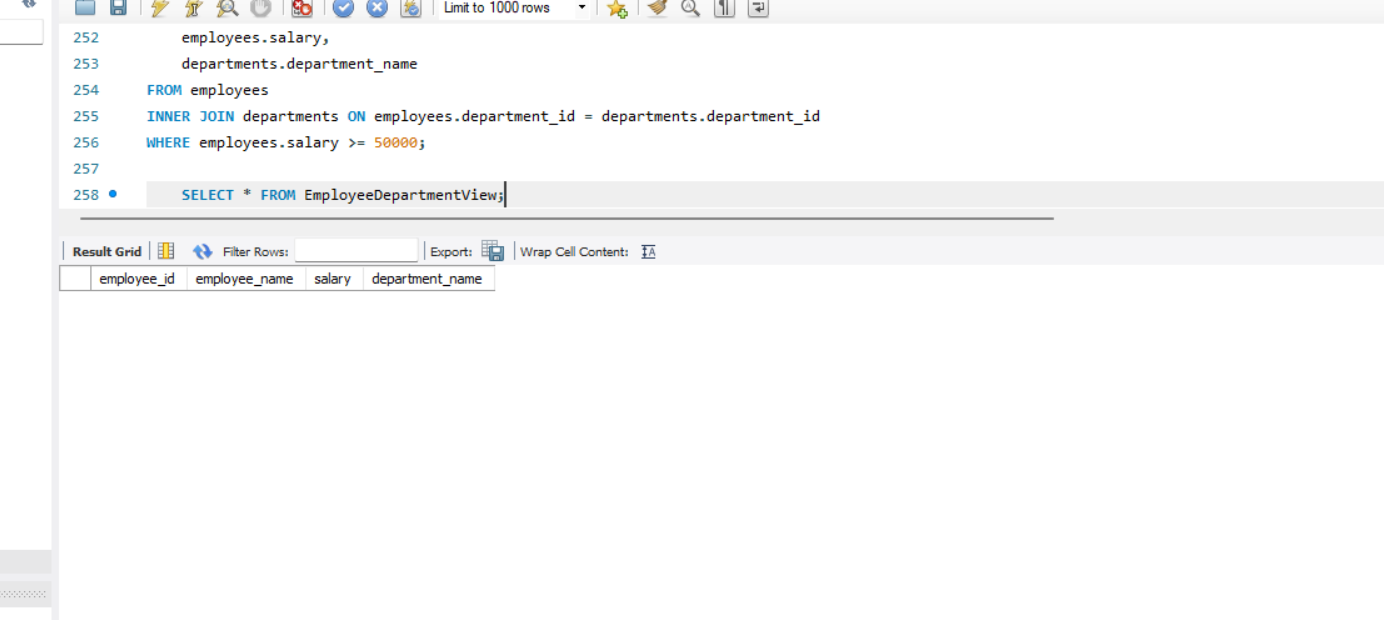
departments.department\_name

FROM employees

INNER JOIN departments ON employees.department\_id = departments.department\_id

WHERE employees.salary >= 50000;

SELECT \* FROM EmployeeDepartmentView;



Q.29:- Create a trigger to automatically log changes to the employees table when a new employee is added.

Ans:- CREATE TABLE employee\_log (

log\_id INT AUTO\_INCREMENT PRIMARY KEY,

employee\_id INT,

employee\_name VARCHAR(255),

salary DECIMAL(10,2),

department\_id INT,

);

DELIMITER $$

CREATE TRIGGER after\_employee\_insert

AFTER INSERT ON employees

FOR EACH ROW

BEGIN

INSERT INTO employee\_log (employee\_id, employee\_name, salary, department\_id, action\_type)

VALUES (NEW.employee\_id, NEW.employee\_name, NEW.salary, NEW.department\_id, 'INSERT');

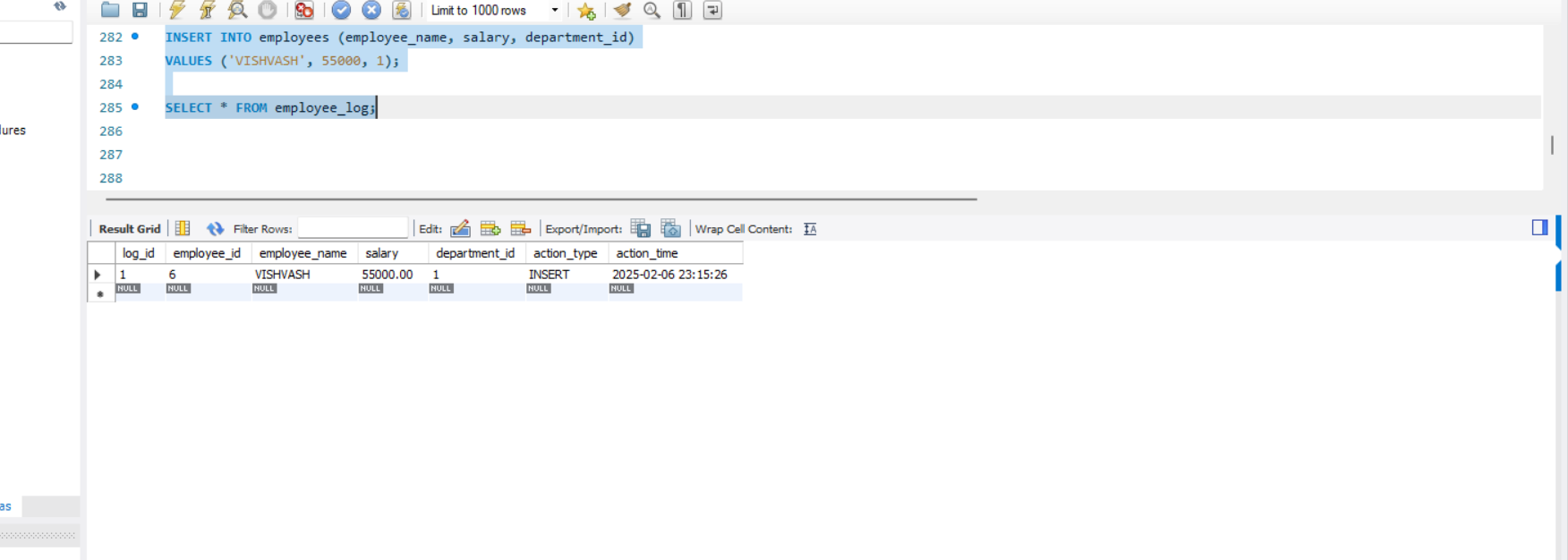
END $$

DELIMITER ;

INSERT INTO employees (employee\_name, salary, department\_id)

VALUES ('VISHVASH', 55000, 1);

SELECT \* FROM employee\_log;



Q.30:- Create a PL/SQL block that calculates the total sales from an orders table.

Ans:-

CREATE TABLE orders (

order\_id INT AUTO\_INCREMENT PRIMARY KEY,

customer\_name VARCHAR(255),

order\_date DATE,

total\_amount DECIMAL(10,2)

);

INSERT INTO orders (customer\_name, order\_date, total\_amount)

VALUES

('vishvash', '2024-02-01', 100.50),

('pal', '2024-02-02', 250.75),

('mohit', '2024-02-03', 300.00);

DELIMITER $$

CREATE PROCEDURE Calculate()

BEGIN

DECLARE total\_sales DECIMAL(10,2);

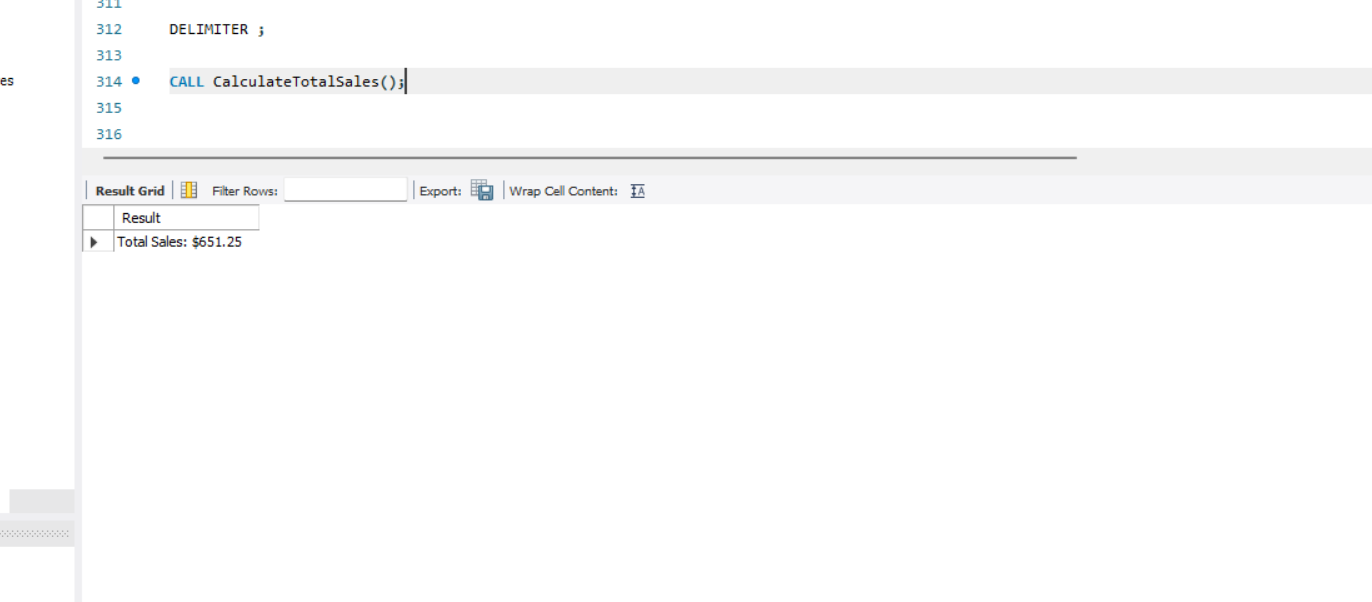
SELECT SUM(total\_amount) INTO total\_sales FROM orders;

SELECT CONCAT('Total Sales: $', total\_sales) AS Result;

END $$

DELIMITER ;

CALL Calculate();



Q.31:- Explain the concept of SAVEPOINT in transaction management. How do ROLLBACK and COMMIT interact with savepoints?

Ans:-

CREATE TABLE products (

product\_id INT PRIMARY KEY AUTO\_INCREMENT,

product\_name VARCHAR(100) NOT NULL,

price DECIMAL(10,2) NOT NULL

);

INSERT INTO products (product\_name, price) VALUES ('Laptop', 1500);

SAVEPOINT sp1;

INSERT INTO products (product\_name, price) VALUES ('Smartphone', 800);

SAVEPOINT sp2;

INSERT INTO products (product\_name, price) VALUES ('Tablet', 500);

SAVEPOINT sp3;

ROLLBACK TO SAVEPOINT sp3;

COMMIT;

SELECT \* FROM products;

