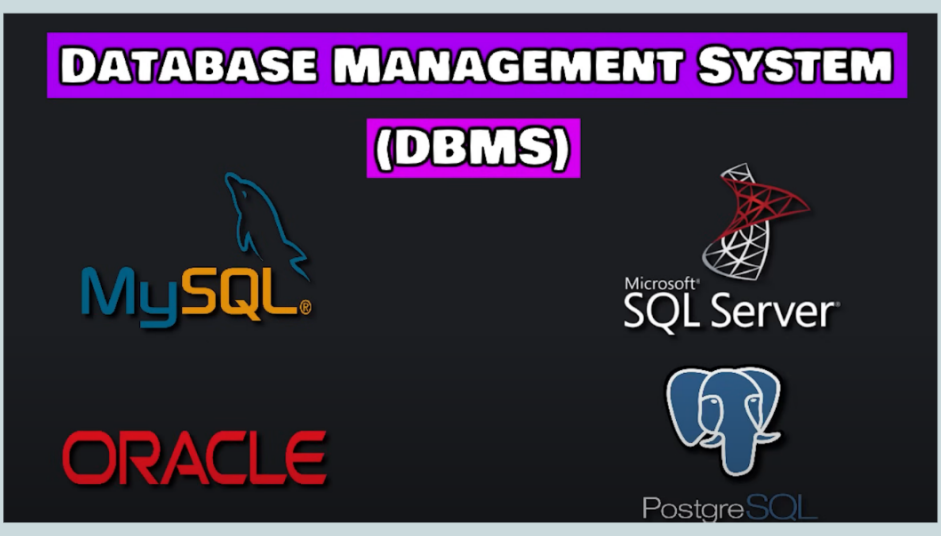
SQL

→ Structured Query Language

→

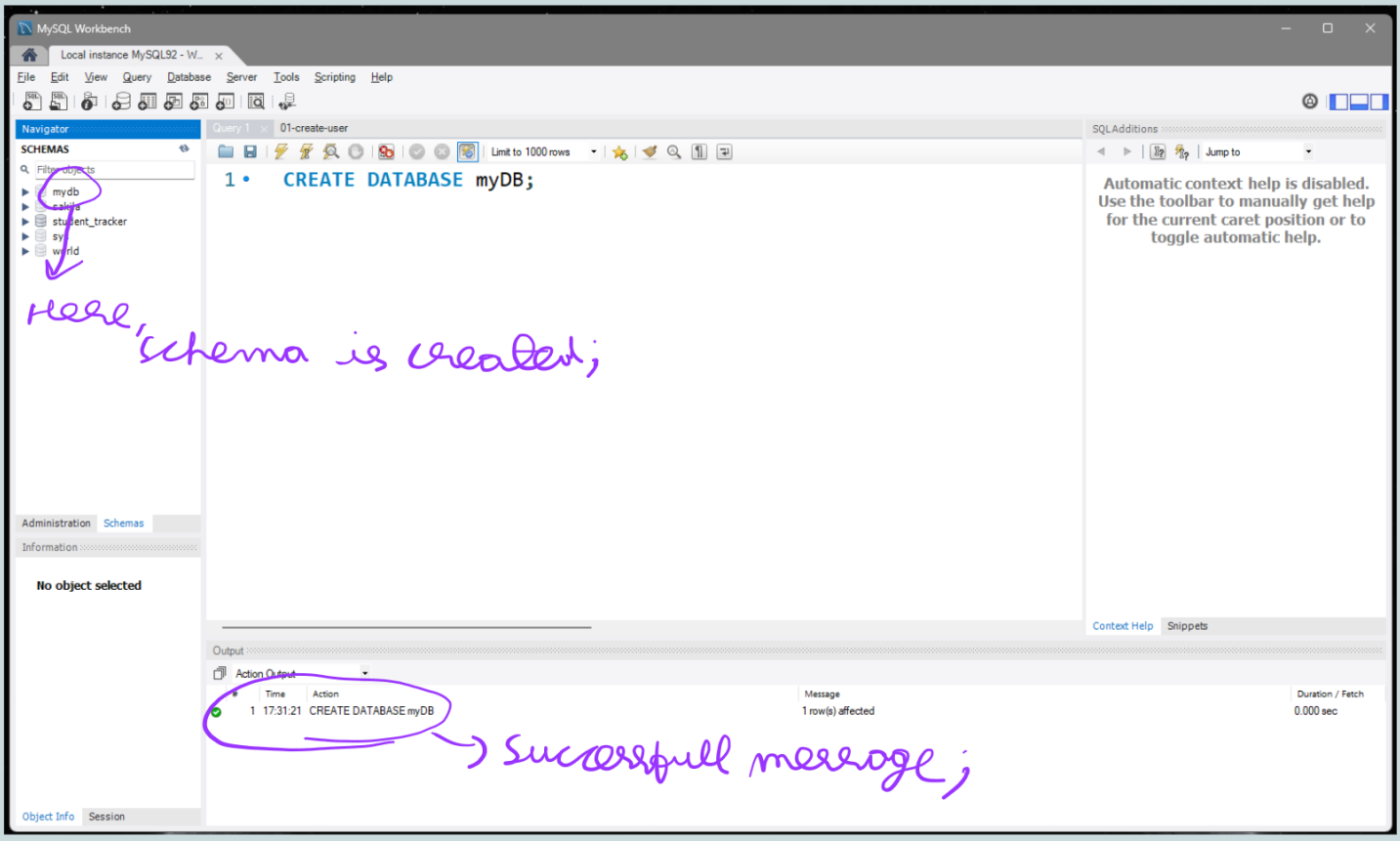
→ It is a relational databases having tables

→ DBMS (where we write code)

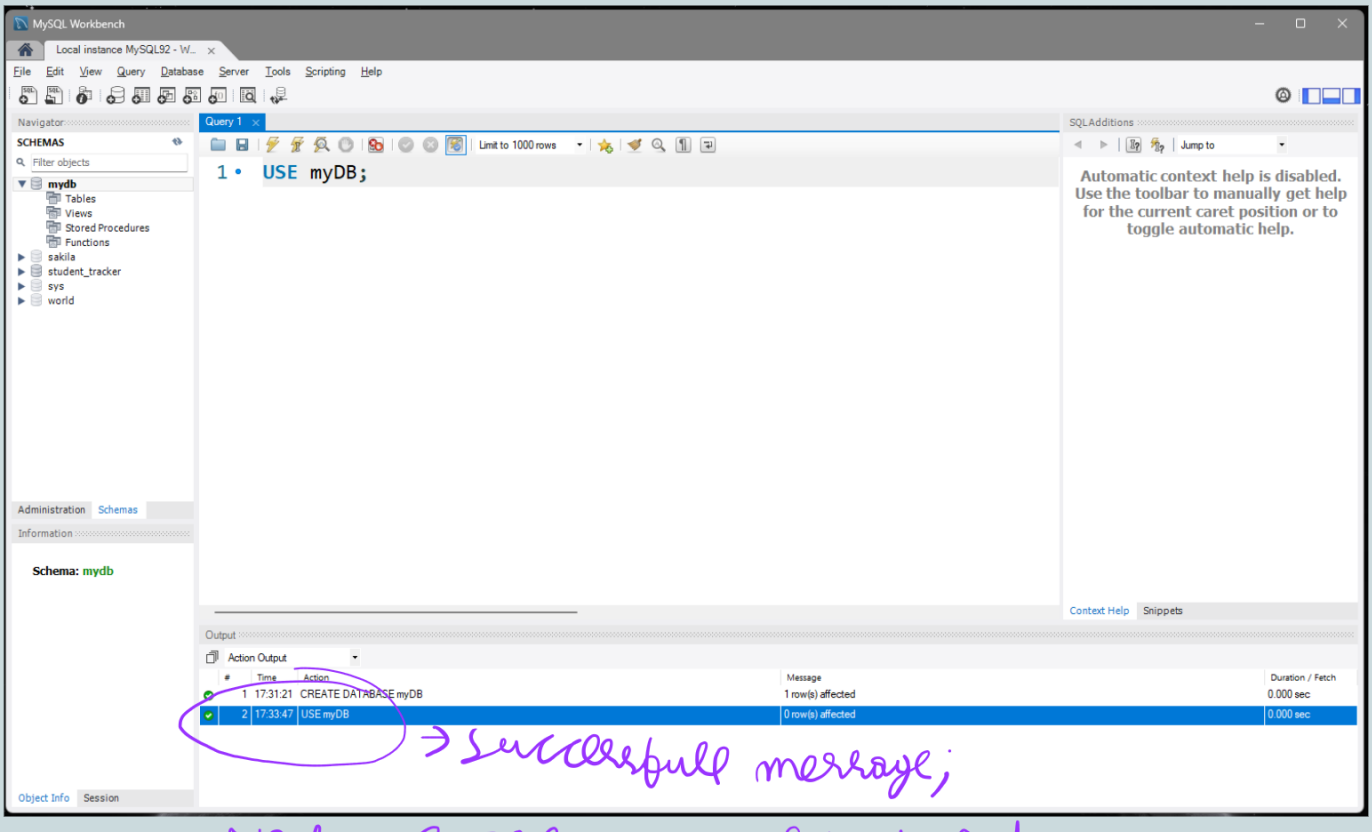


→ SQL Keyword and commands are generally not case-sensitive, meaning SELECT, select, and SeLeCt will all execute the same query

Creating Database

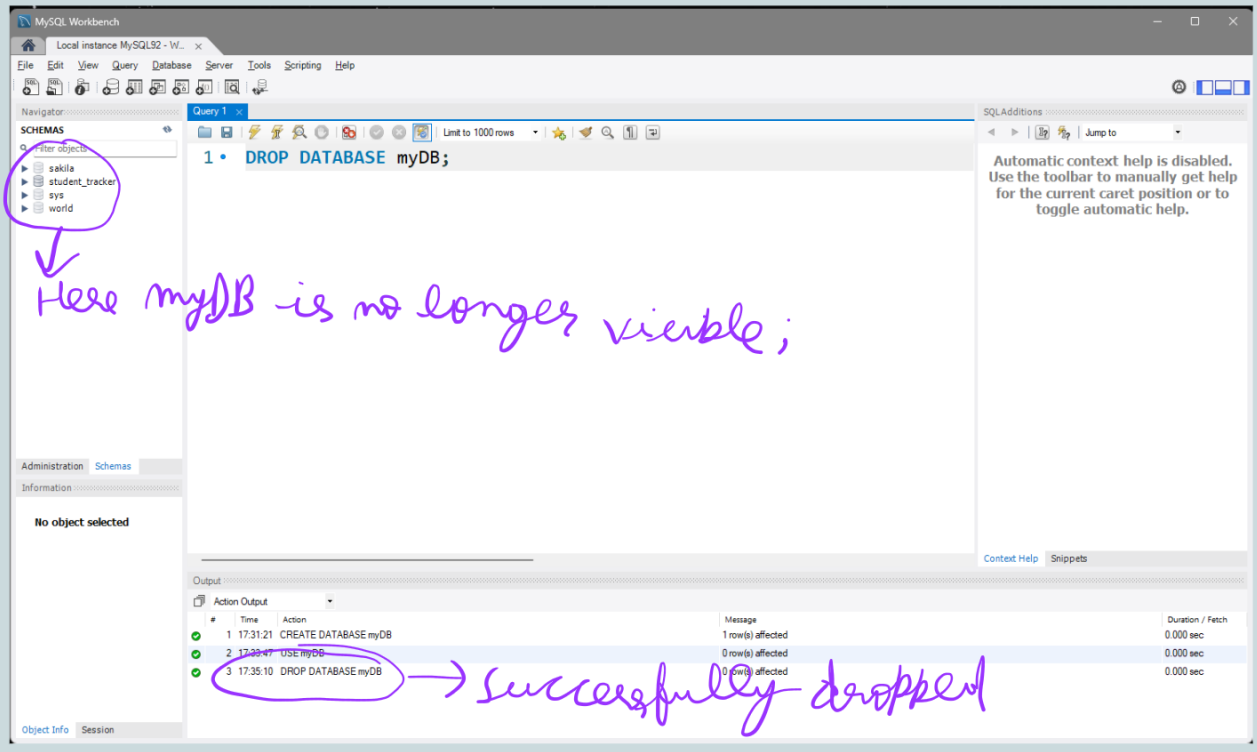


- To use the currently made Database;



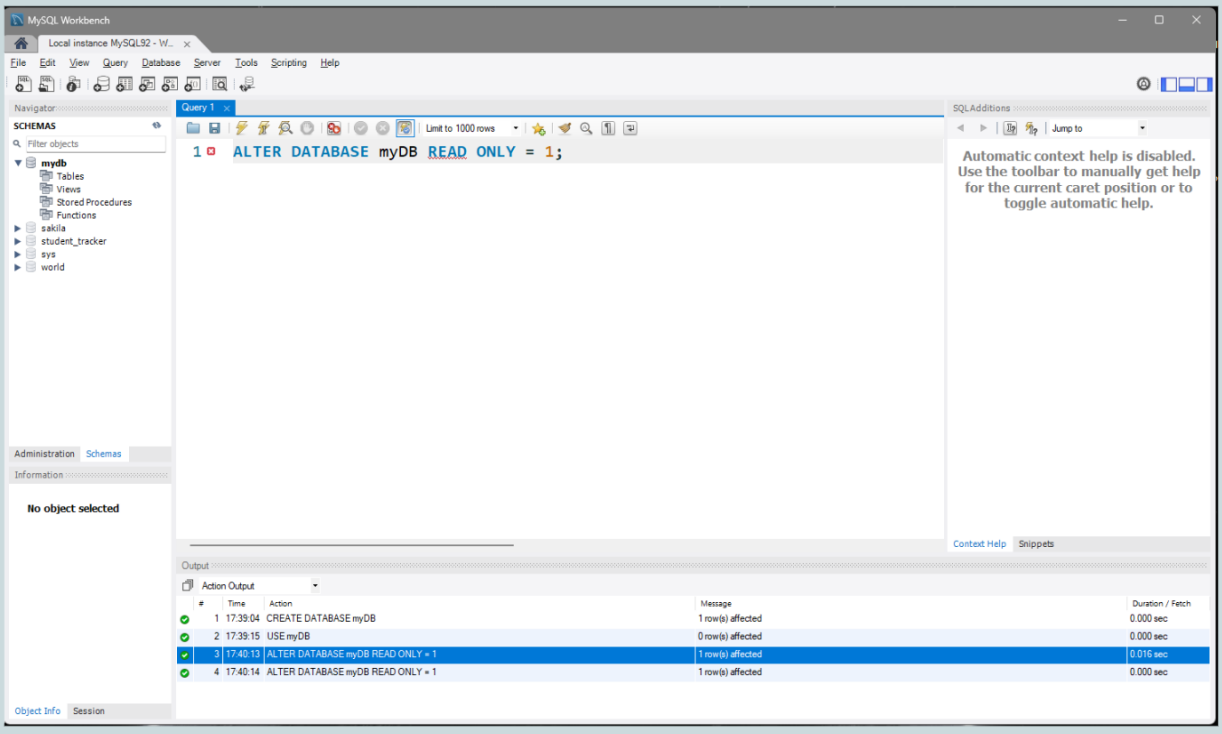
Now, we are using this database

- To drop (delete) Database

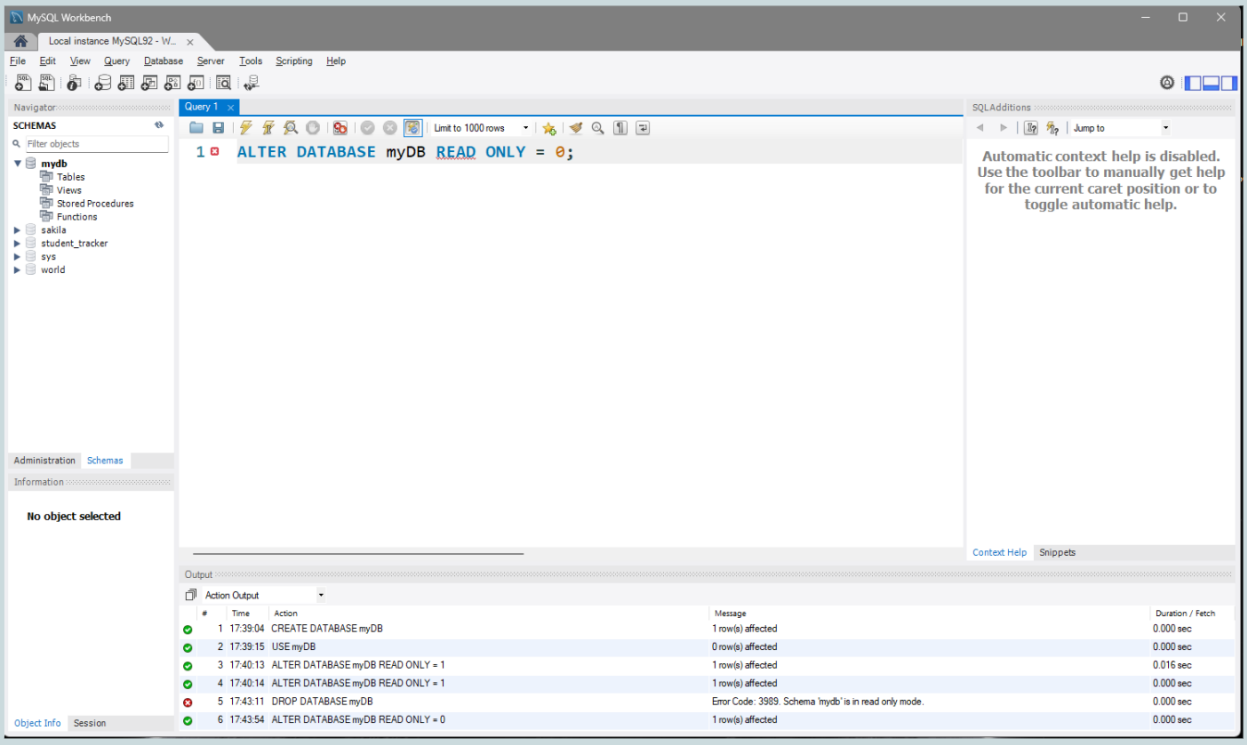


- To make our Database Read Only;

In SQL, "read-only" refers to a database or a cursor (a pointer to data) that allows only retrieval and viewing of data, preventing any modifications like updates, inserts, or deletes

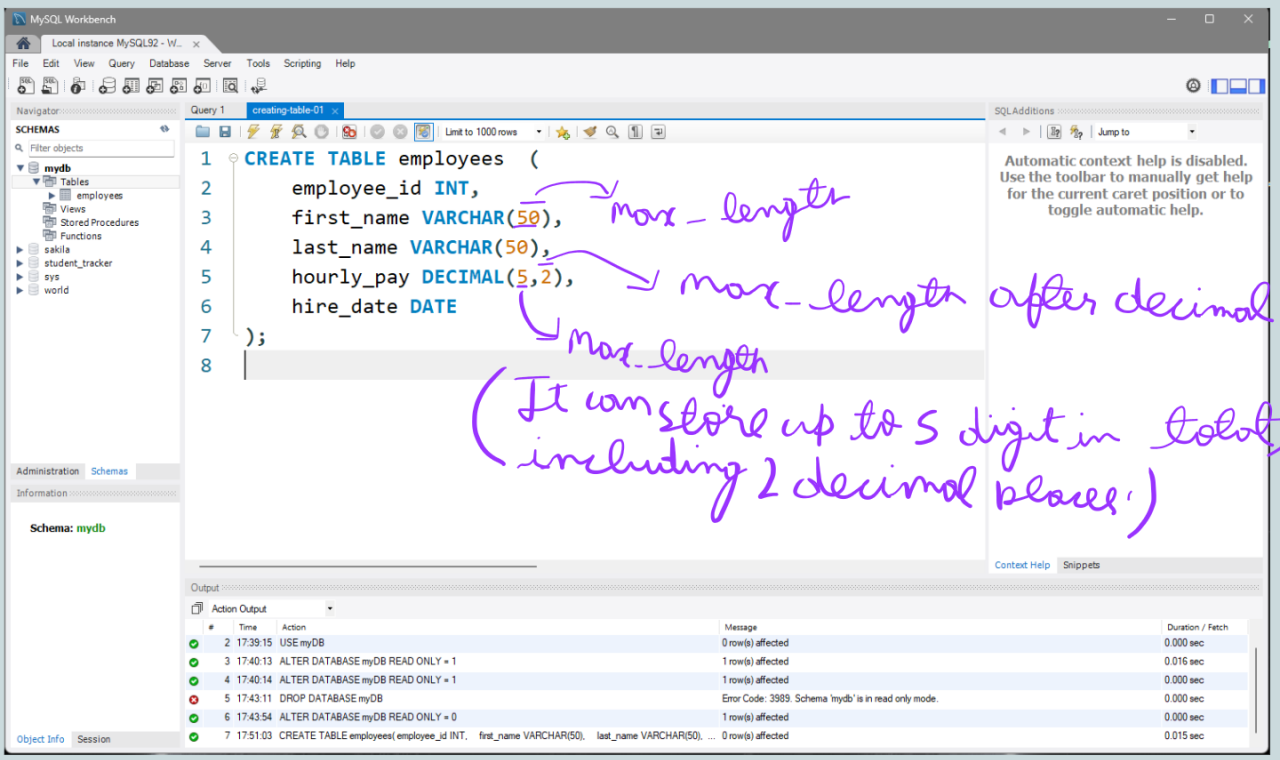


- To disable Read Only mode

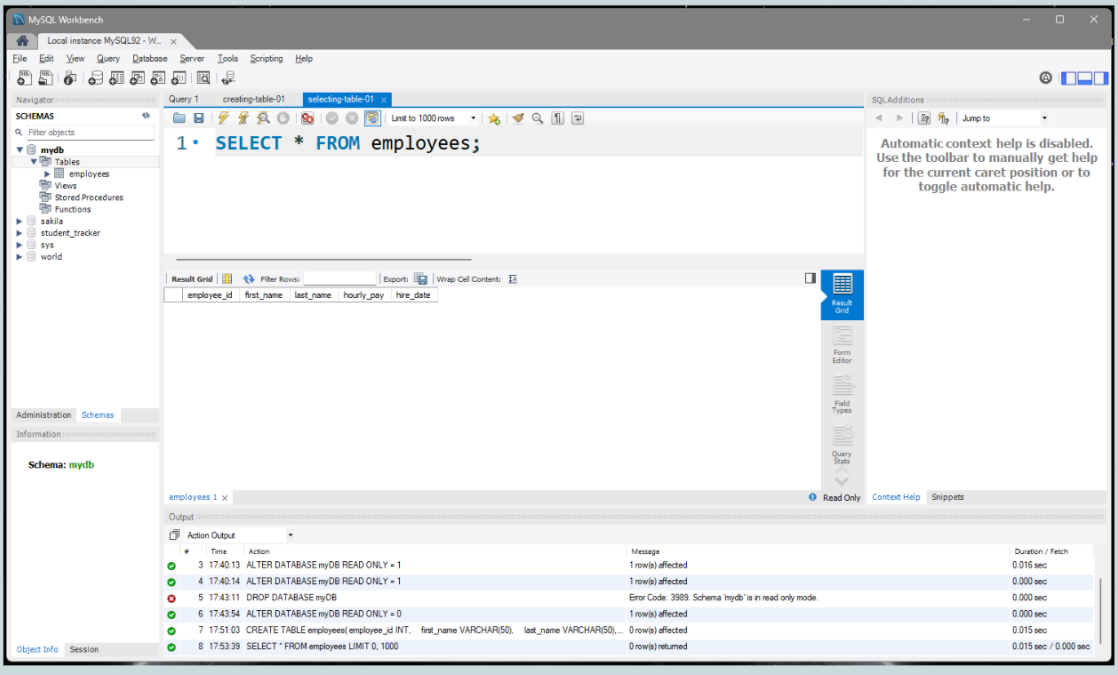


TABLE

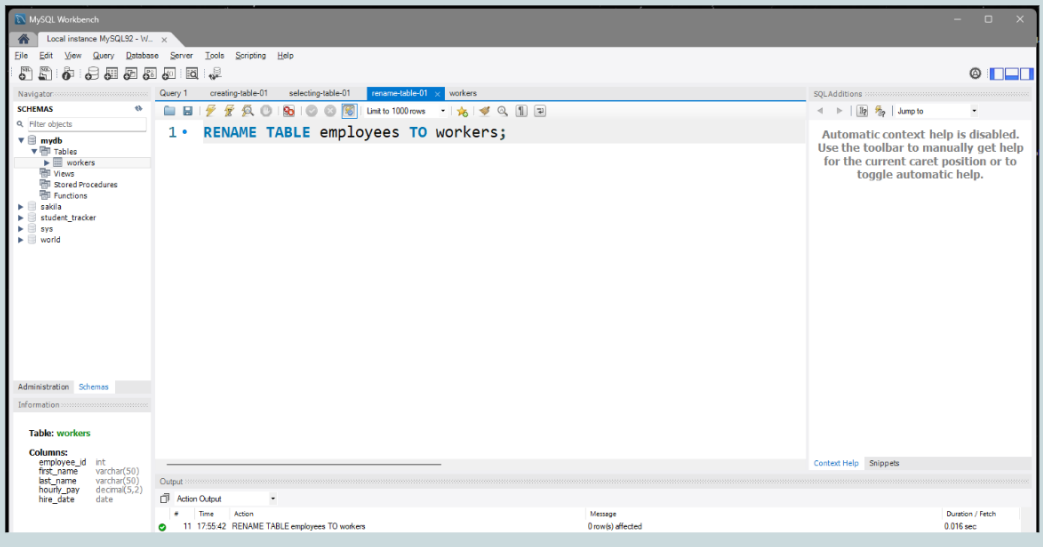
- Creating Table



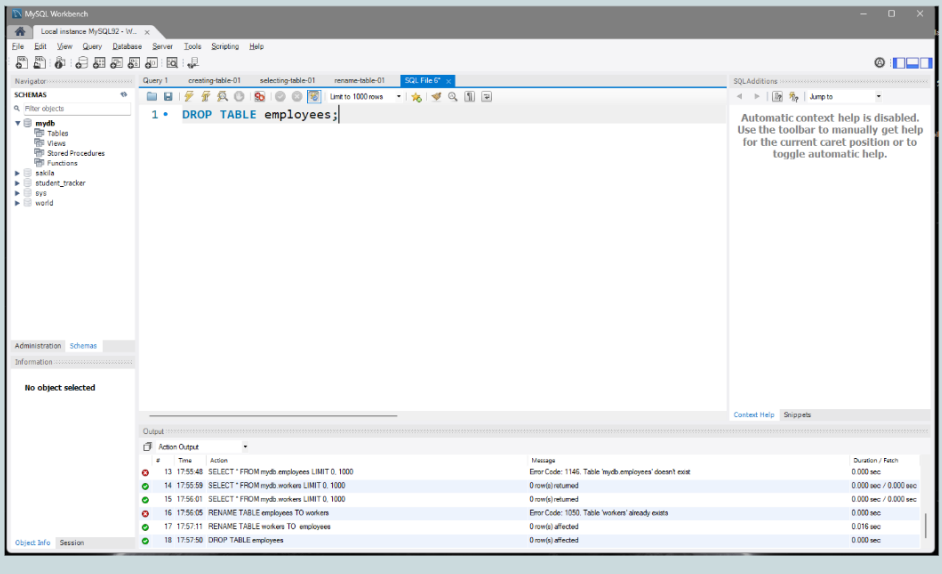
- Select Table



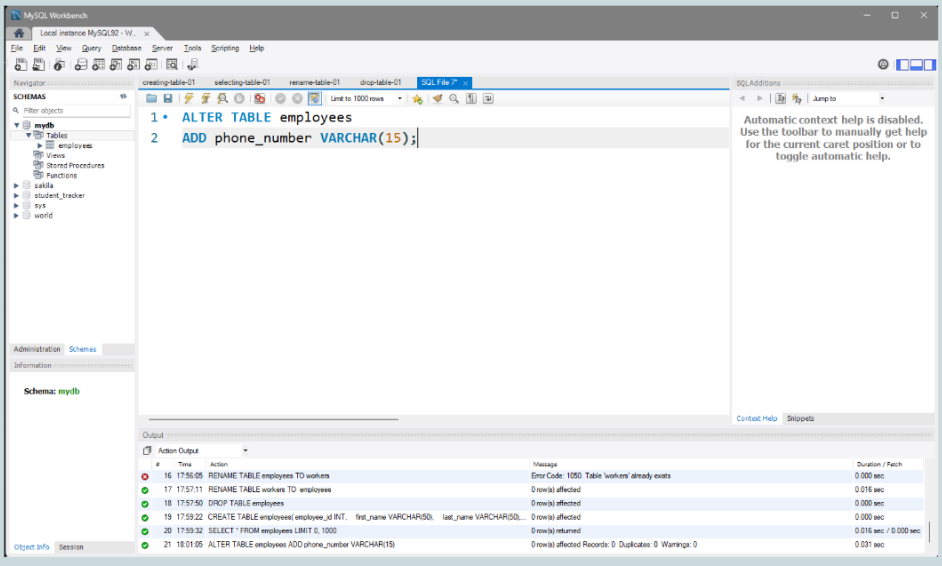
- Rename Table

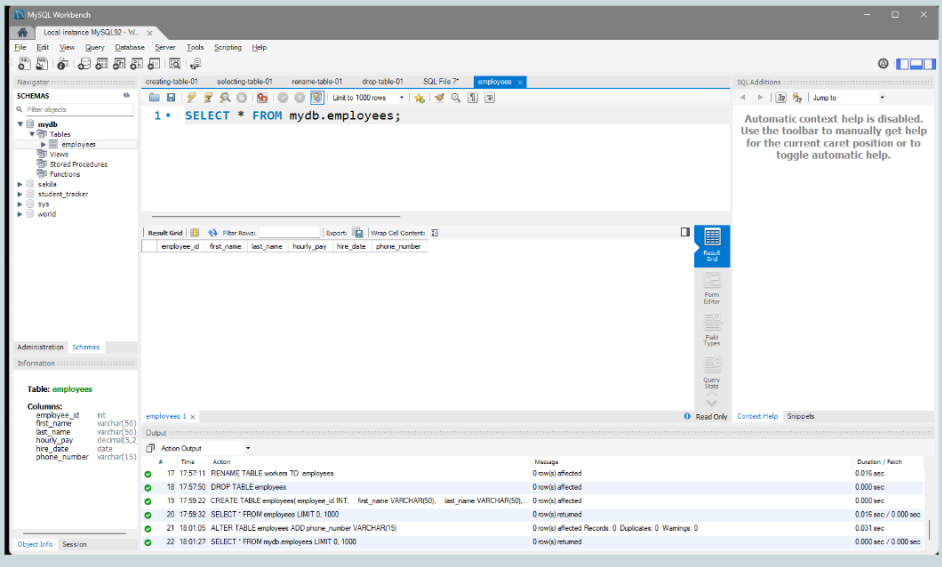


- Drop Table

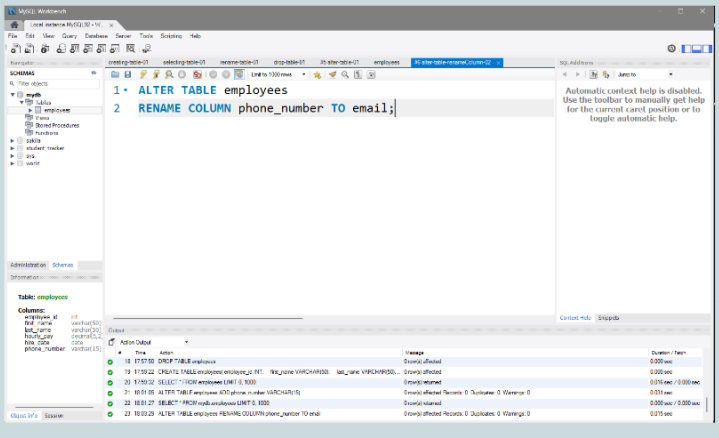


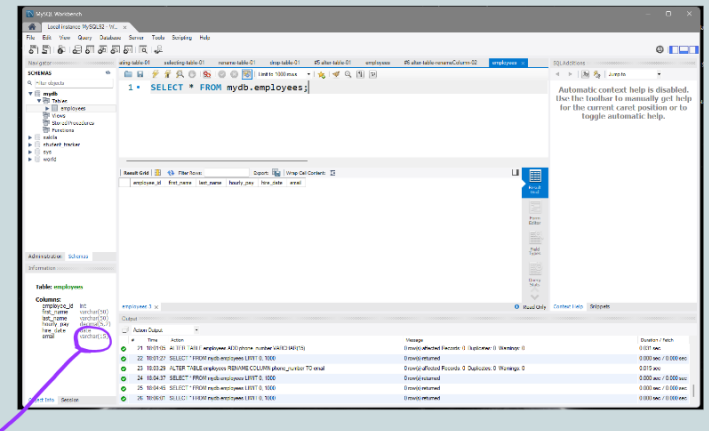
- ALTER Table (Add Column)





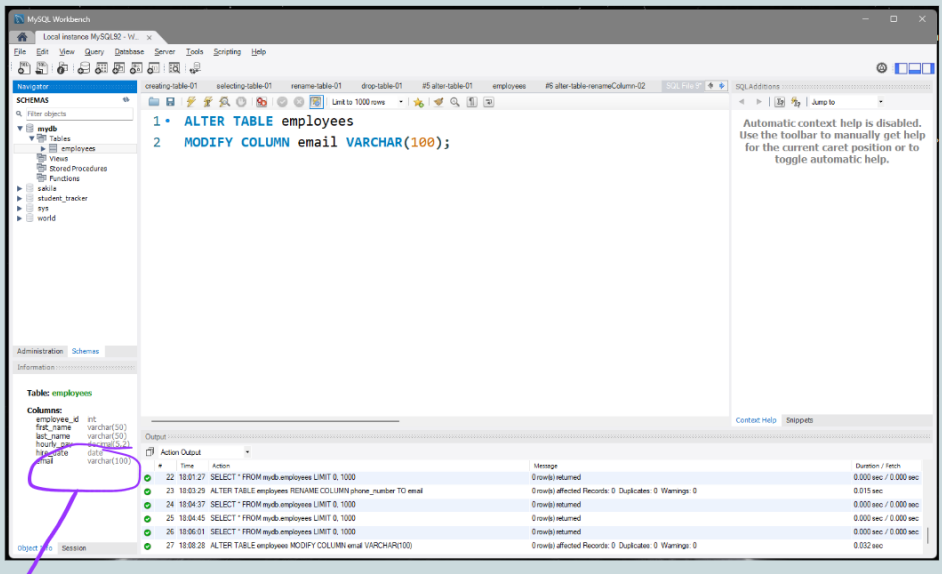
- ALTER Table (Rename Column)





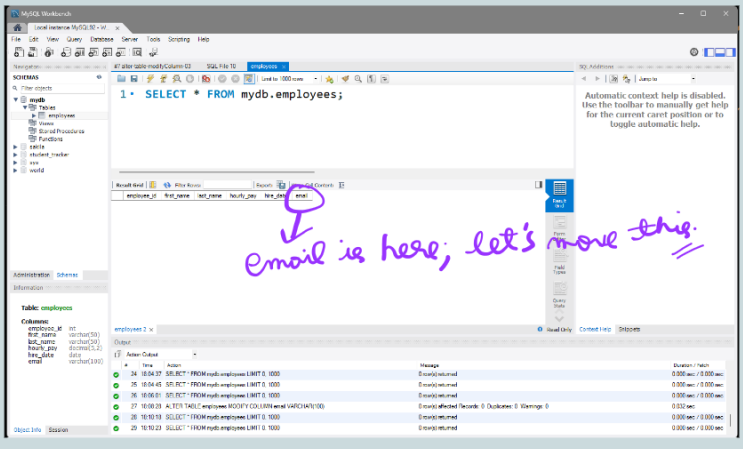
Here, email is of max_size 15, let's change this

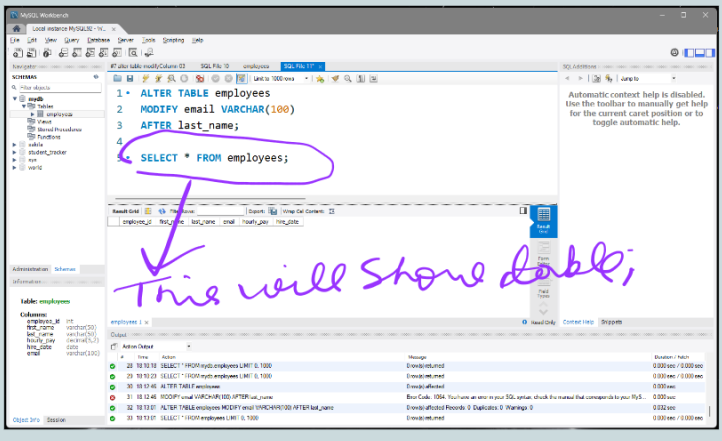
- ALTER Table (modify column properties)



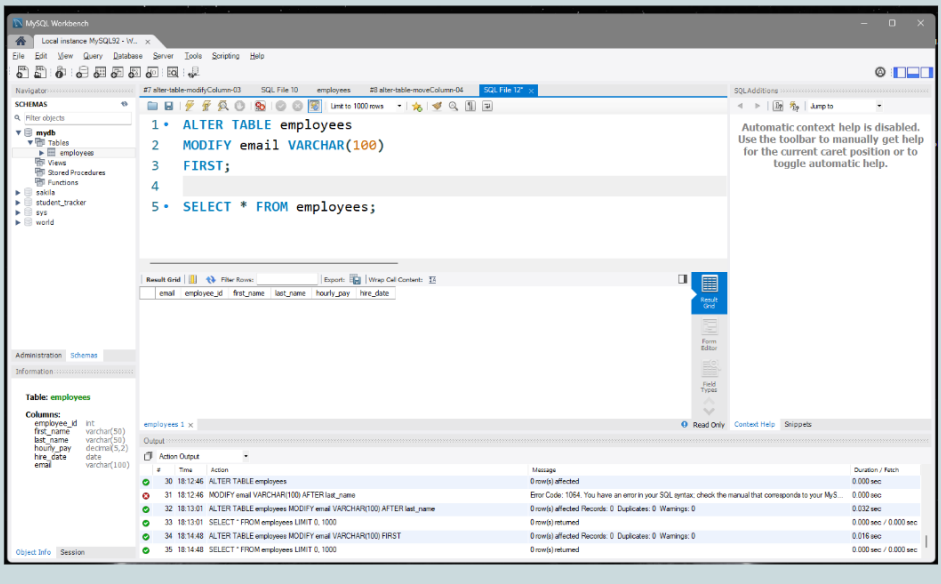
email max-size is changed.

- ALTER Table (Move Column)

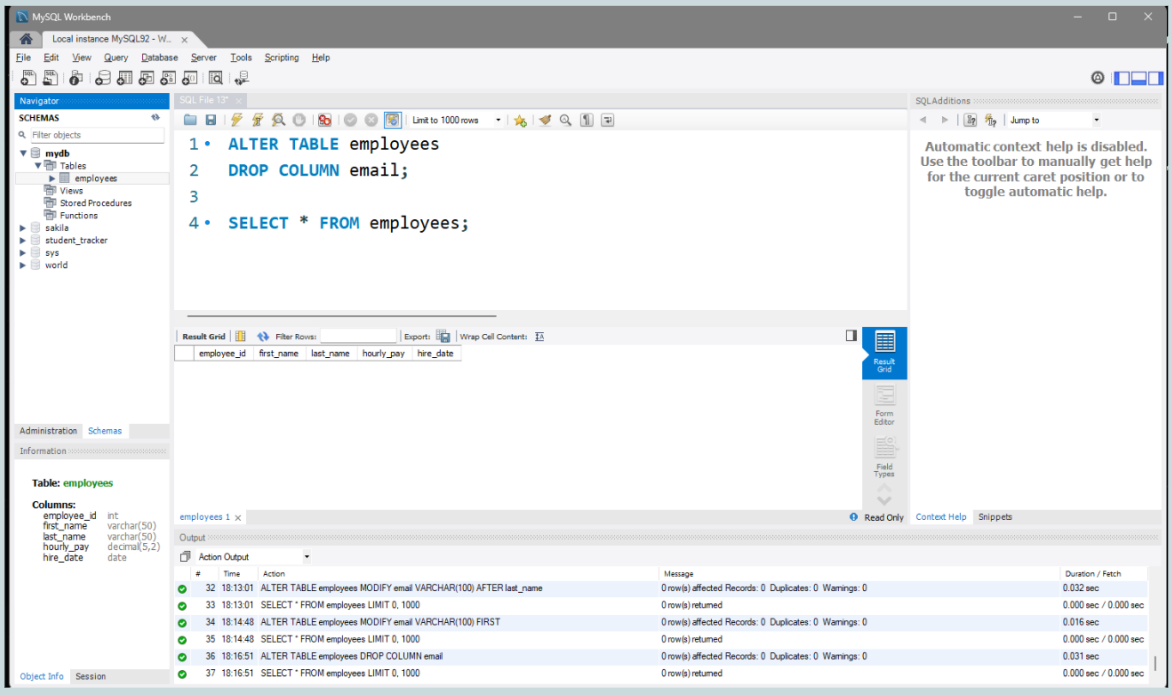




- ALTER Table (move column to first)

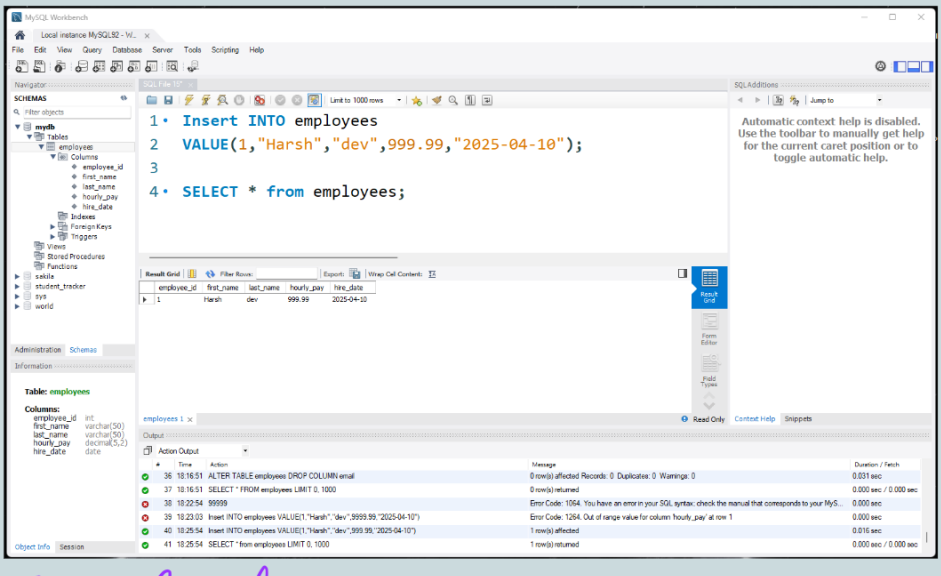


- ALTER Table (Drop Column)

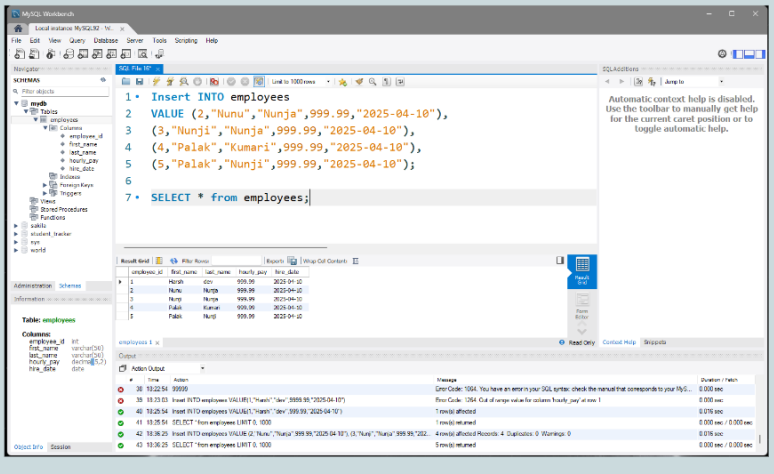


#ROWS

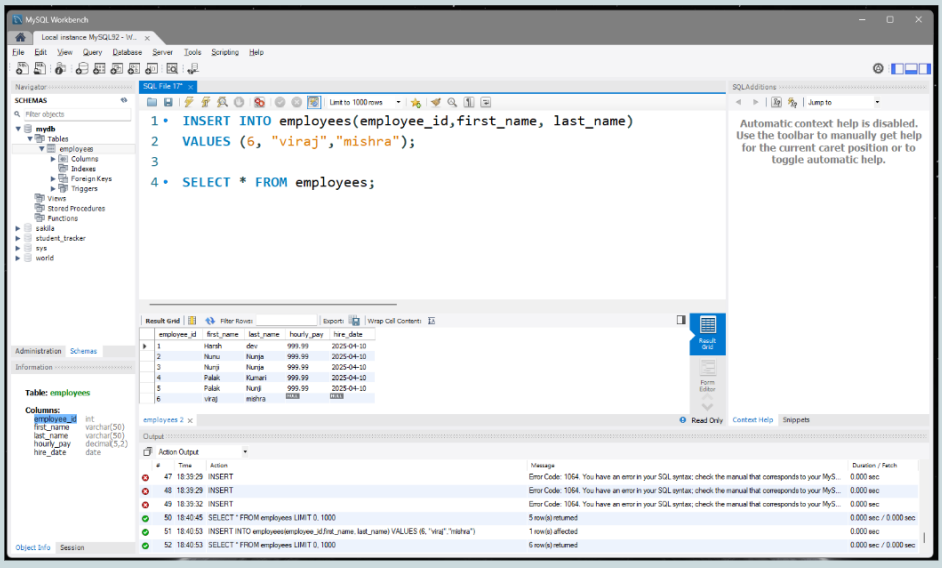
- Insert Rows



- Insert Multiple Rows

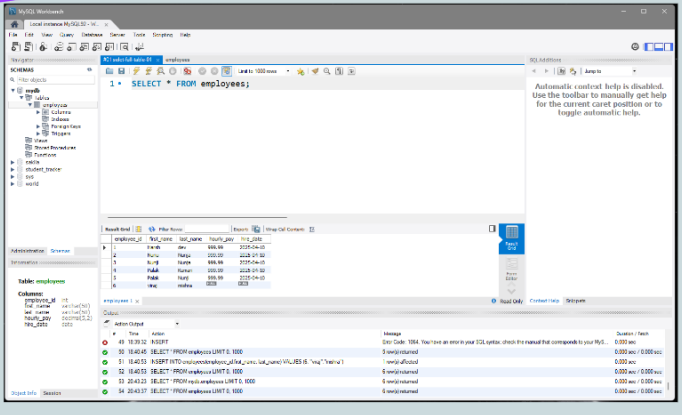


- Insert at desired Column in Table

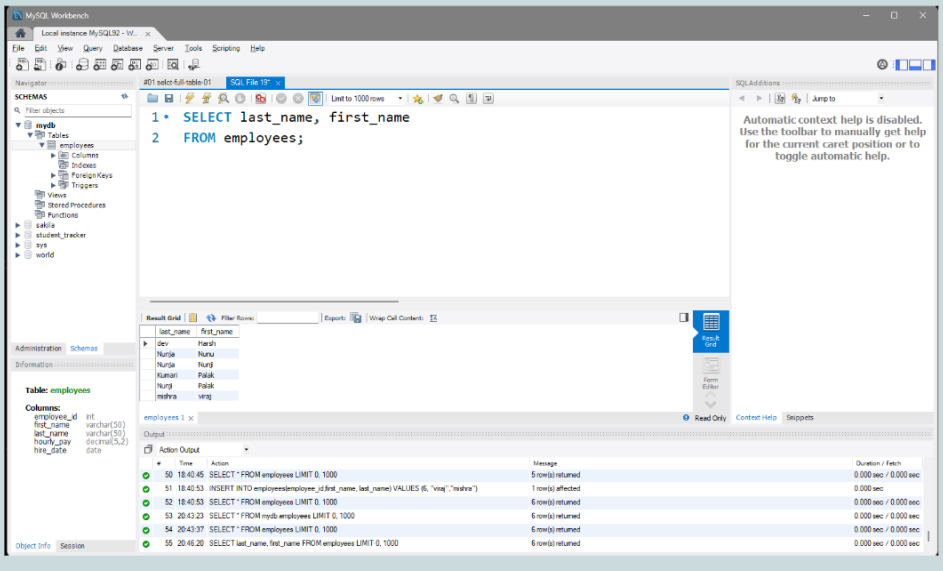


4. Select

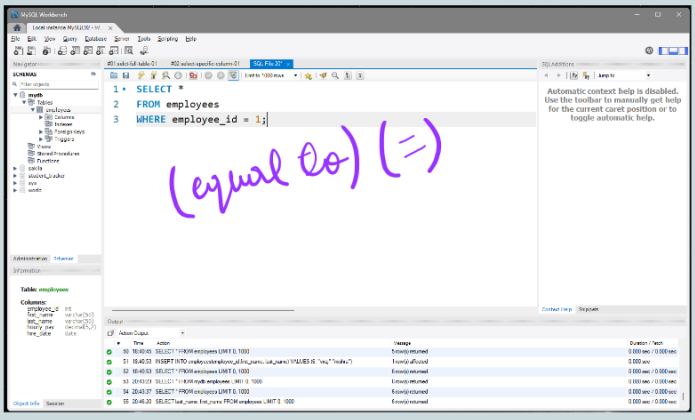
- Select full table

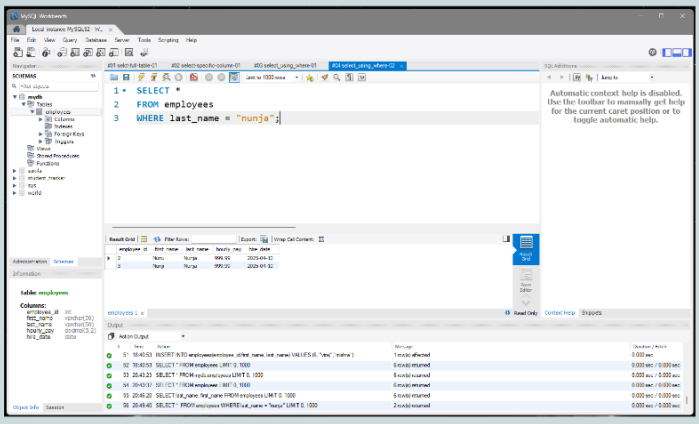


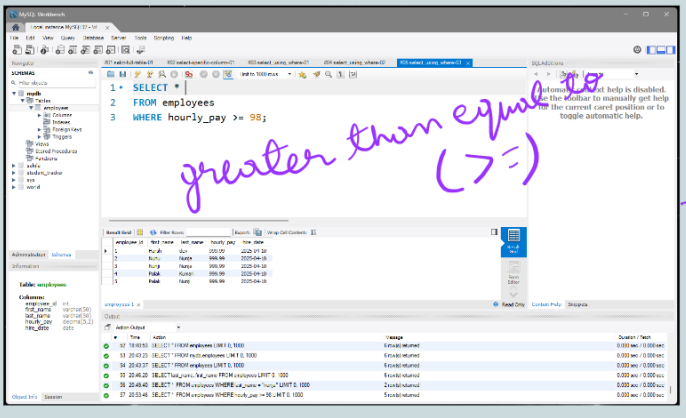
- Select specific Column

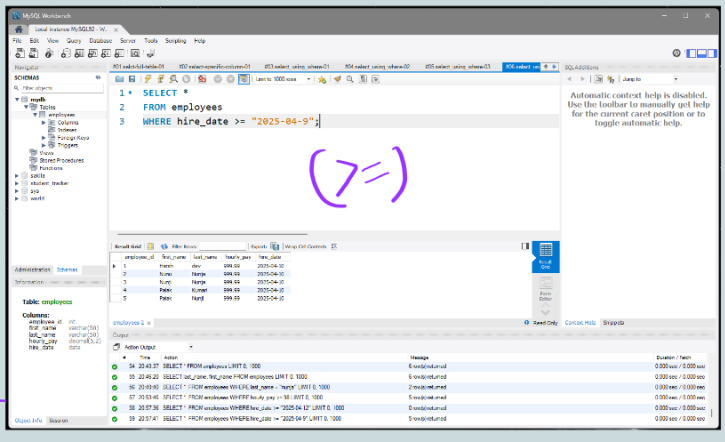


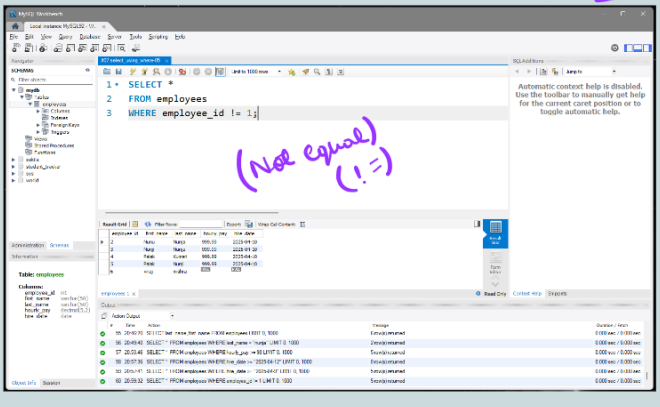
Select with where keyword

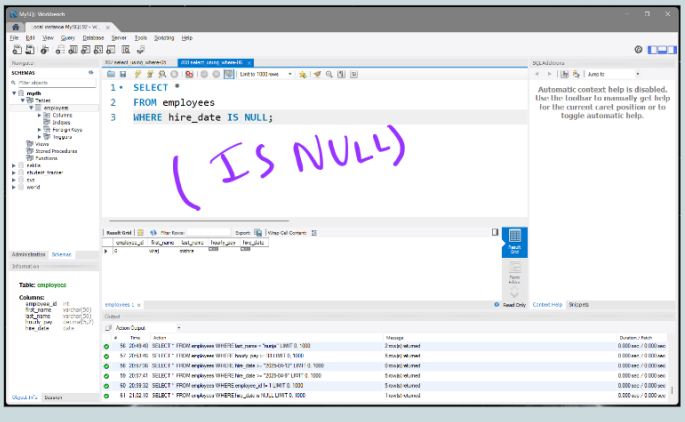


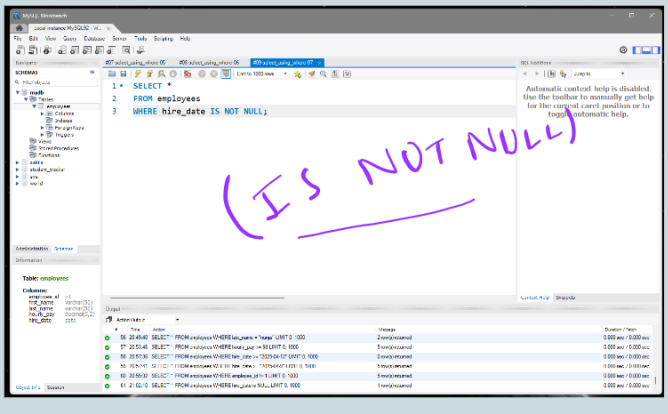






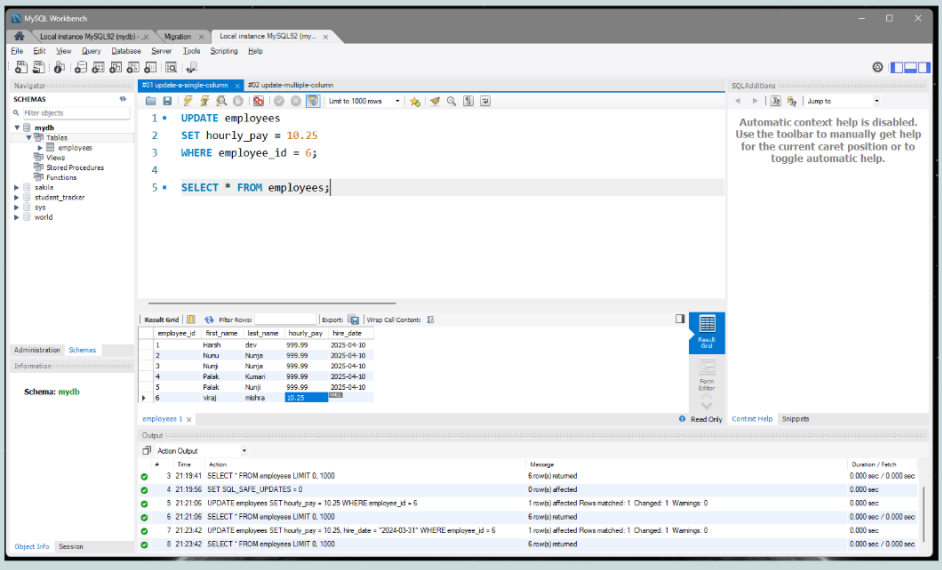




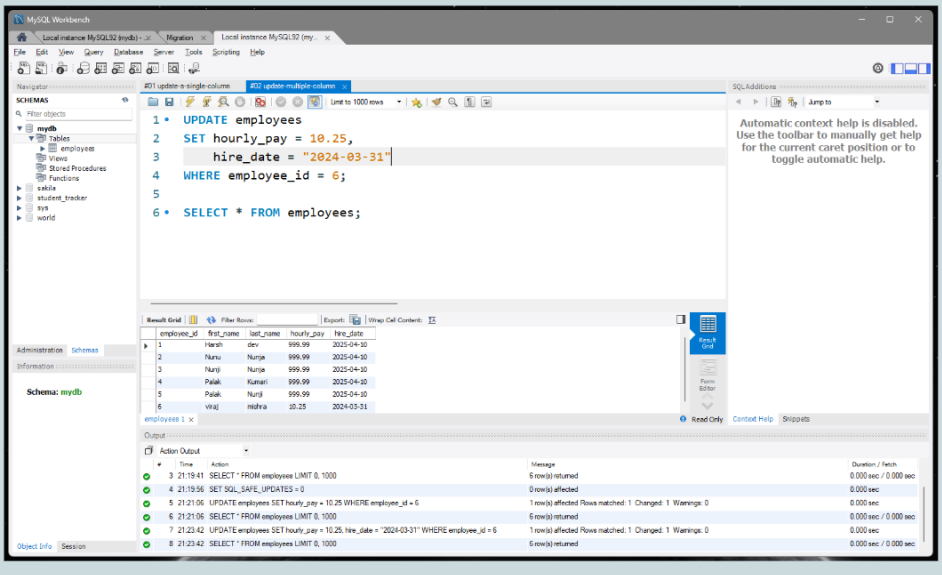


Update and Delete

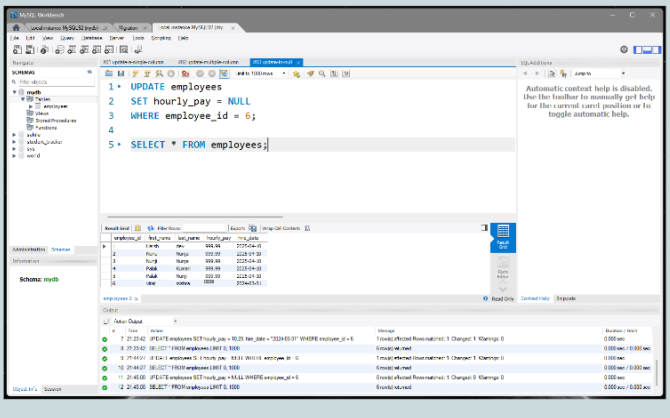
- Update a single column



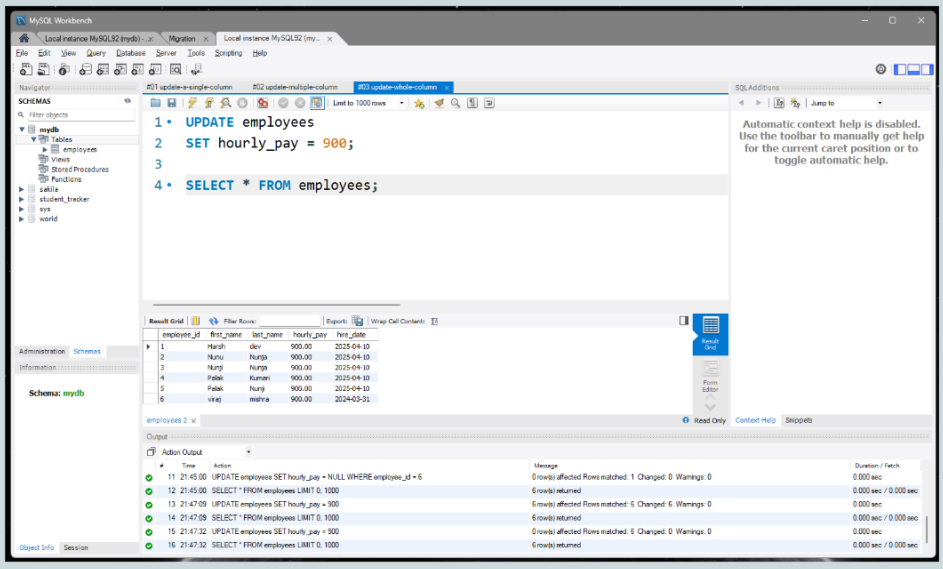
- Update multiple column



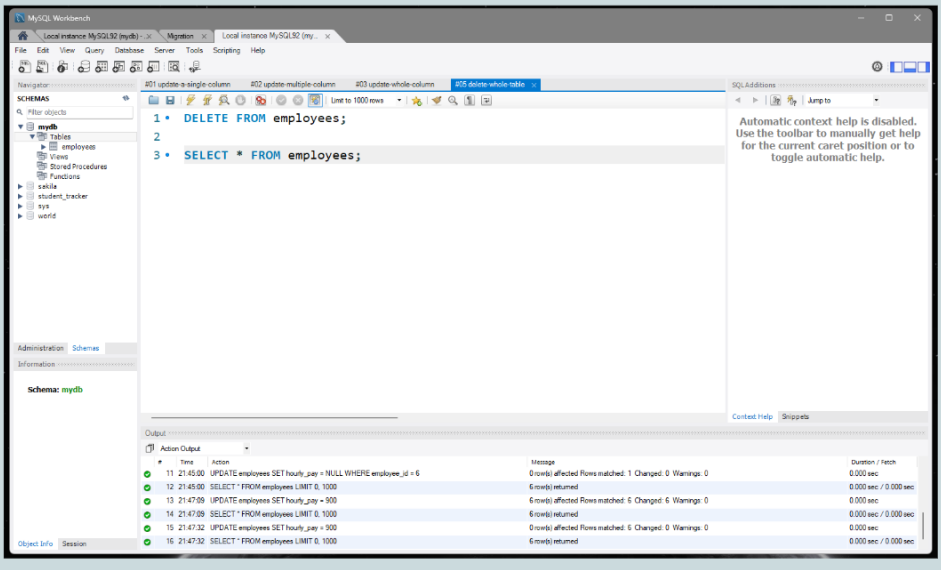
- Update to Null value



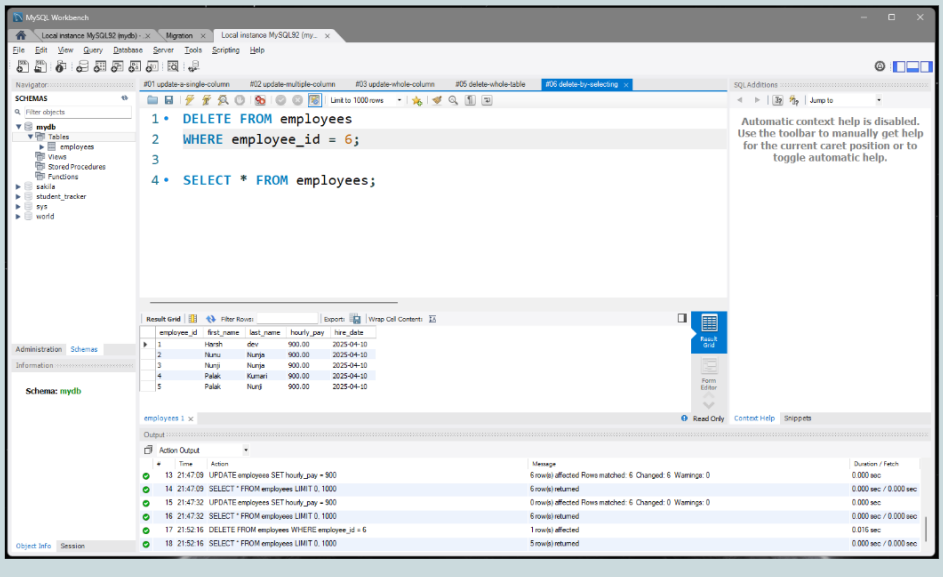
- Update whole Column



- Delete whole Table



- Delete By selecting



Autocommit, commit and Rollback

In SQL, autocommit manages how transactions are handled, commit makes changes permanent, and rollback undoes changes within a transaction.

Autocommit:

- **Default Behavior:** → By default, autocommit is set to on

Many database systems, by default, operate in autocommit mode, meaning each SQL statement is treated as a separate transaction and automatically committed.

- **No Explicit Commit/Rollback:**

In autocommit mode, you don't need to use COMMIT or ROLLBACK statements for each SQL statement, as they are implicitly handled.

- **Disabling Autocommit:**

You can disable autocommit to manage transactions explicitly, allowing you to group multiple statements into a single transaction that can be committed or rolled back together.

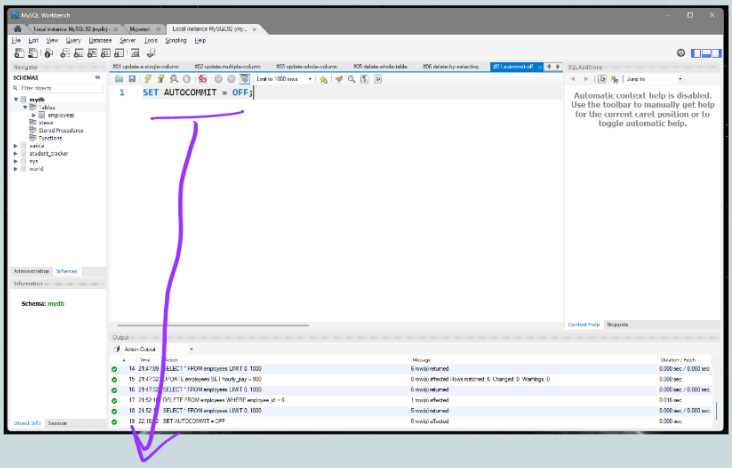
Commit:

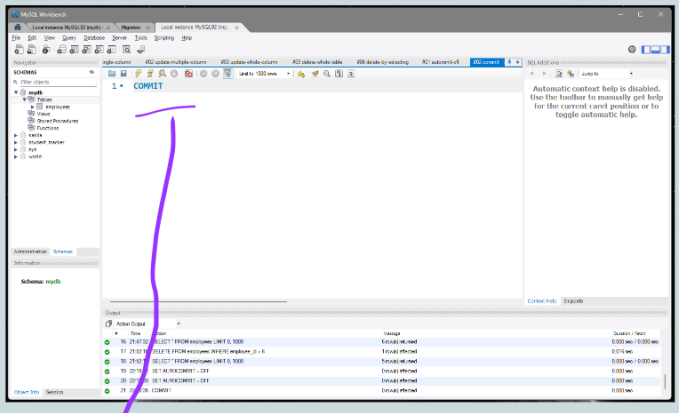
- **Making Changes Permanent:** The COMMIT statement makes all changes within the current transaction permanent and visible to other users or sessions.
- **After Successful Transaction:** Use COMMIT after a transaction has completed successfully to save the changes.
- **Releasing Locks:** COMMIT also releases any locks that were acquired during the transaction.

Rollback:

- **Undoing Changes:**

The ROLLBACK statement cancels all changes made within the current transaction, reverting the database to its state before the transaction started.





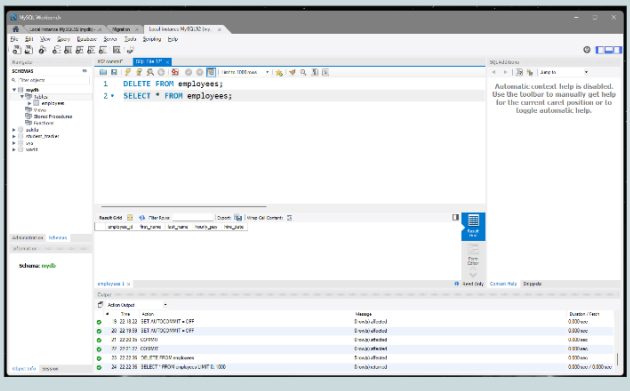
With this setting turned OFF, our transaction will not save automatically.

We have to run this command explicitly to save transaction

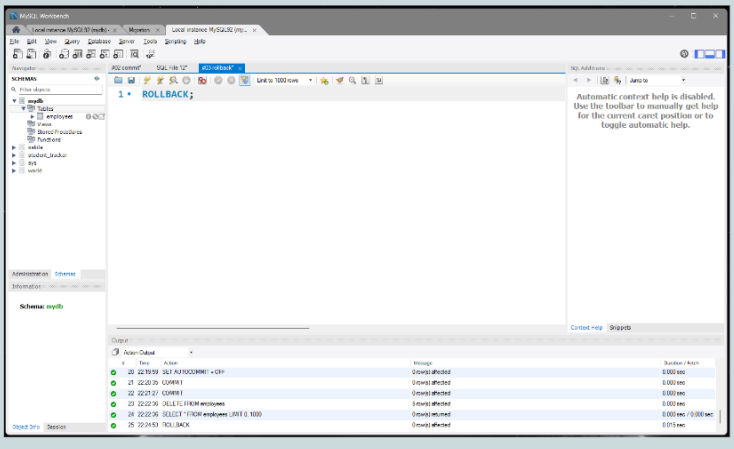
Step - By - Step

To delete whole table :→

①

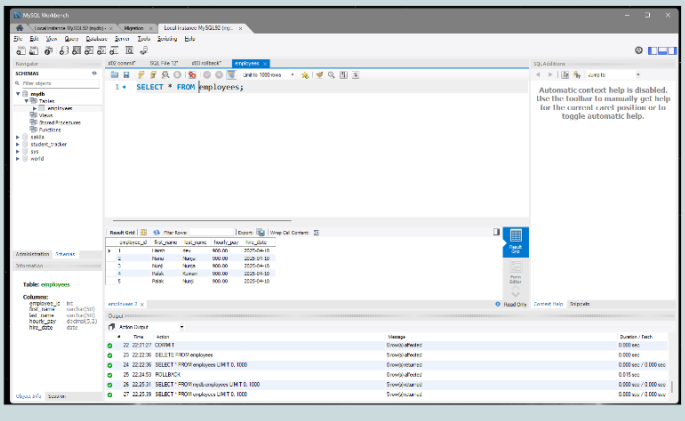


② :→ But if you want to restore previous table.



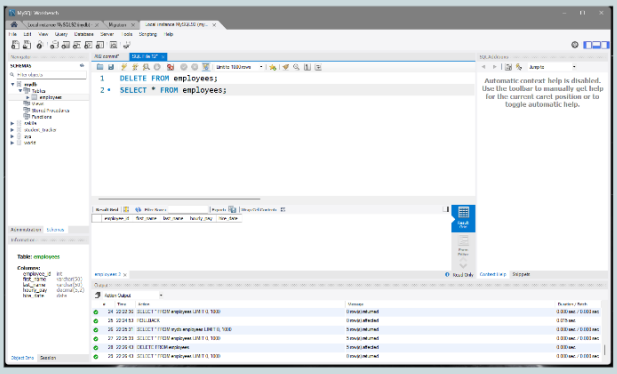
③

our table is restored

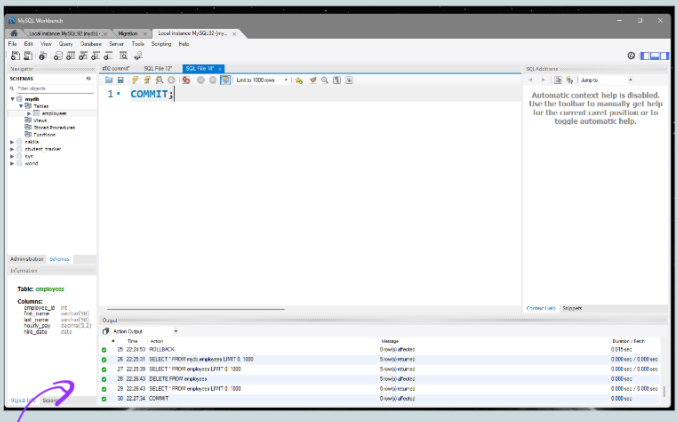


But if want to delete whole table & save it.

(1)

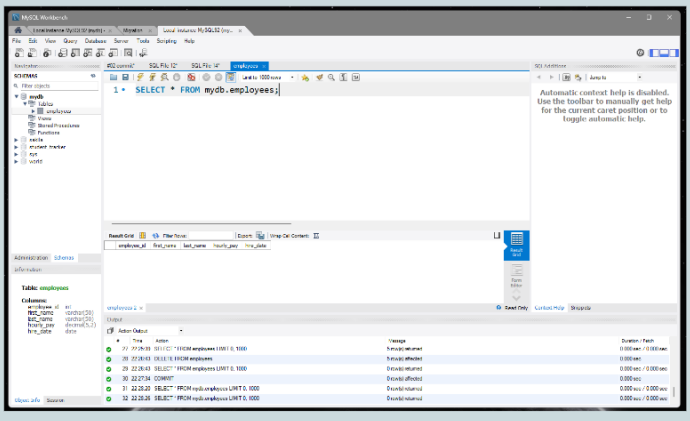


(2)



It will finally save in database

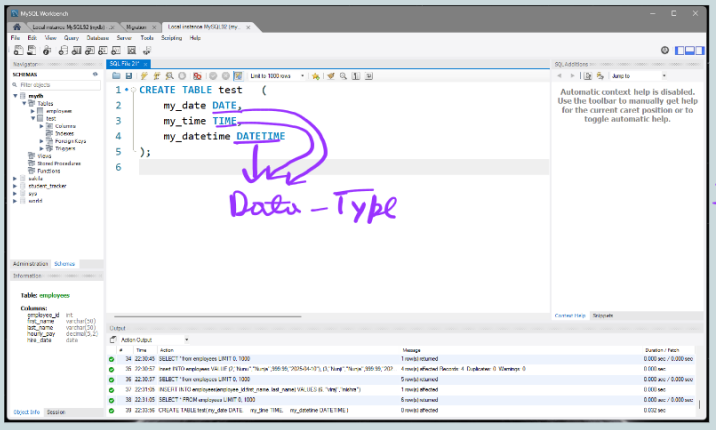
(3)

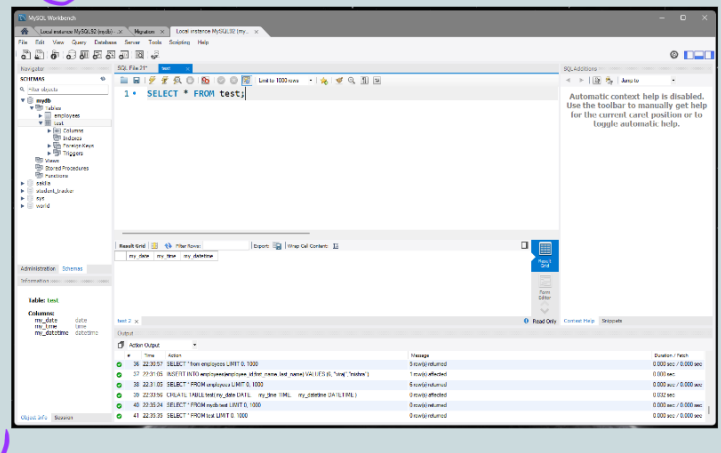


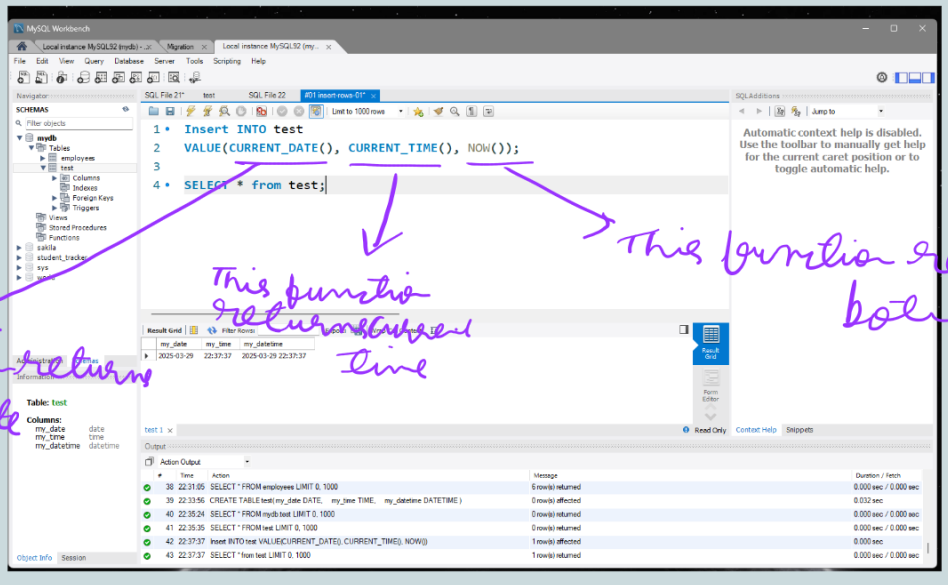
Current Date and Time

- Create a test table

① ② ③



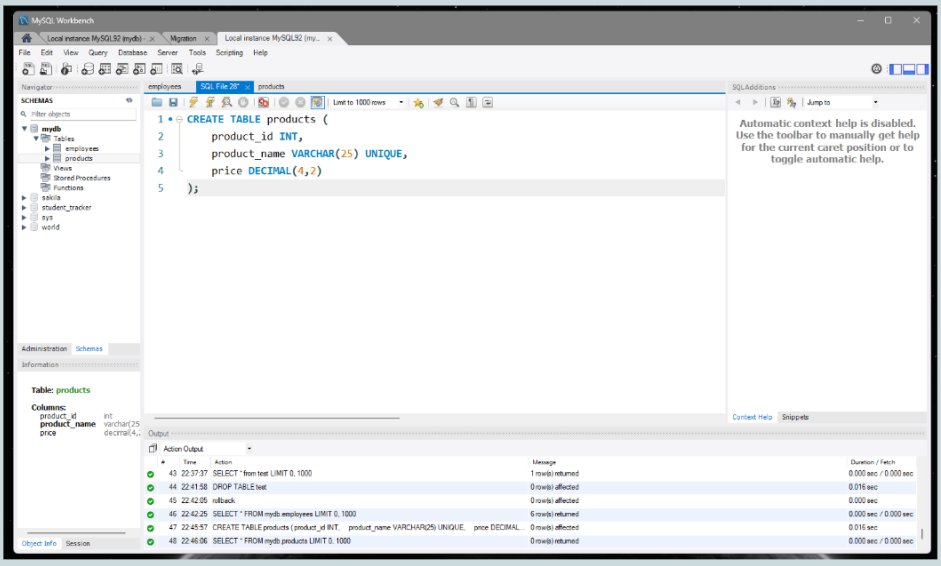




UNIQUE

In SQL, the UNIQUE keyword, used as a constraint, ensures that all values in a specified column (or set of columns) are distinct, meaning no two rows can have the same value(s) in those columns.

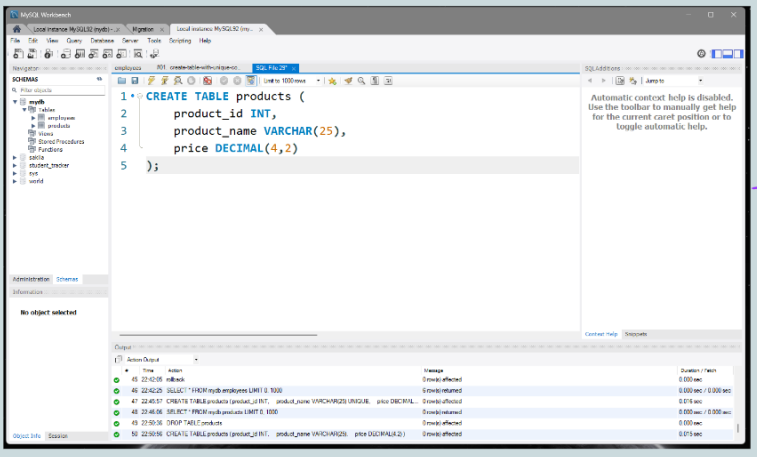
- Create Table with Unique Column :-

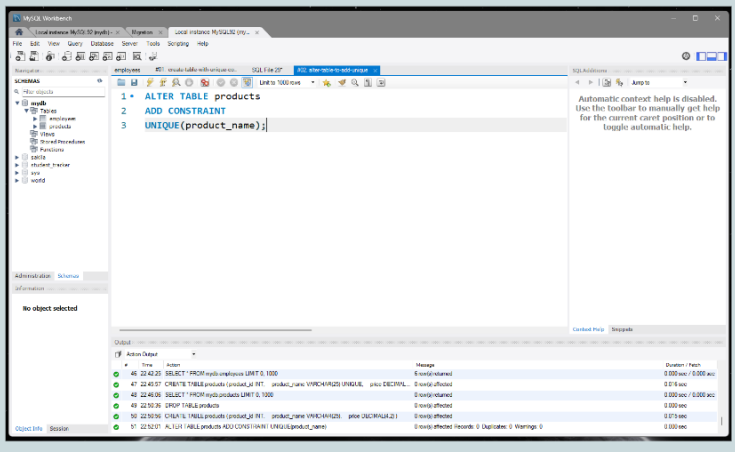


- Alter Table and then add Unique feature after making table

(1) Make table without any Unique column:

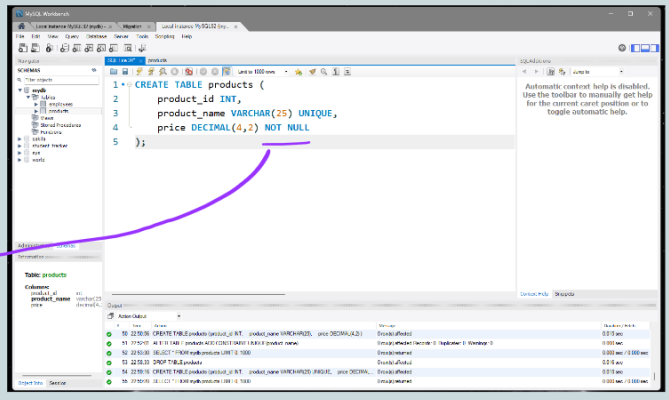
(2) Make product_name UNIQUE





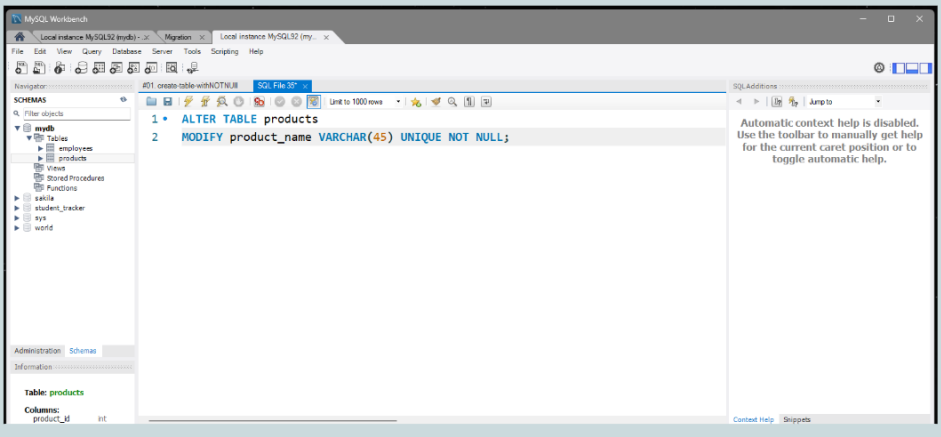
NOT NULL

- Create table with NOT NULL keyword



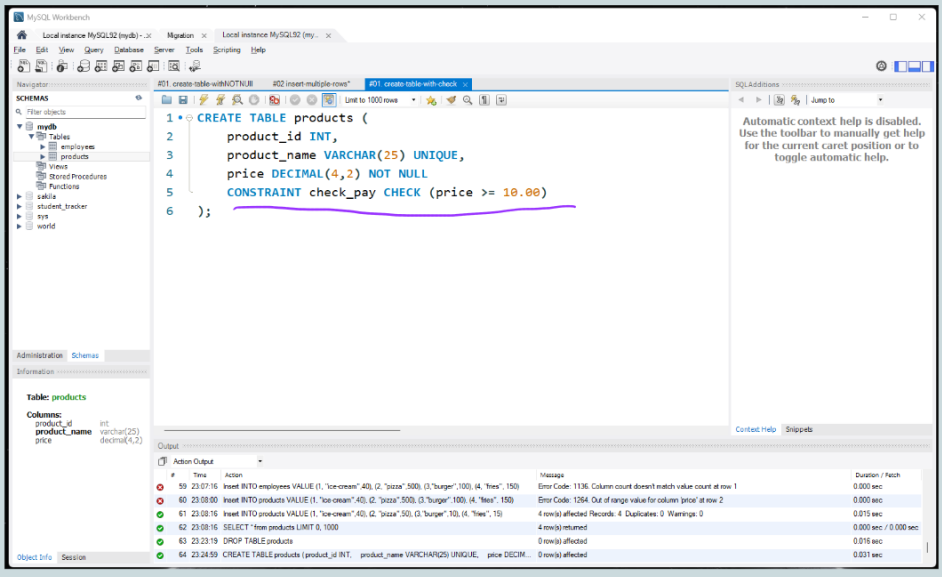
so, price can not be null.

- Alter table and add NOT NULL



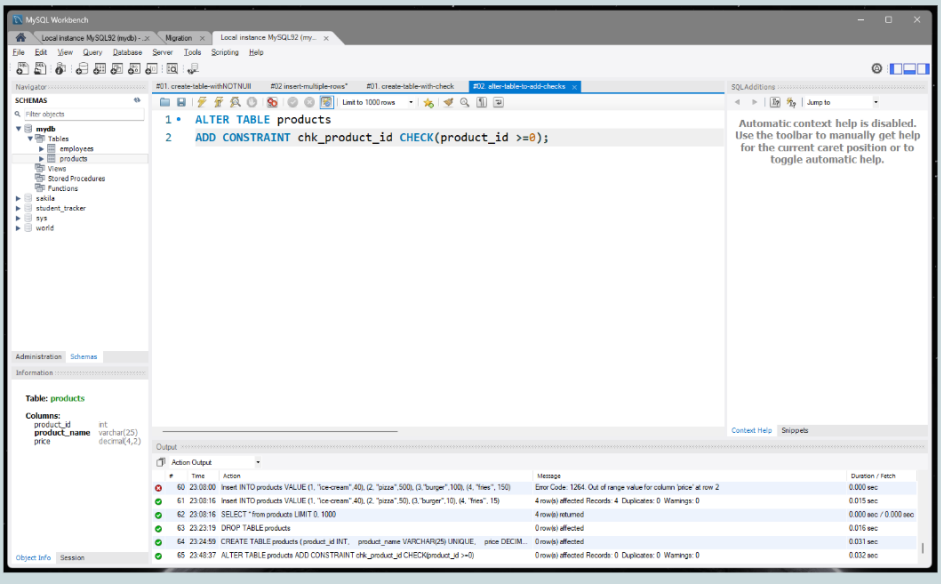
Check Constraints

- Create table with check.

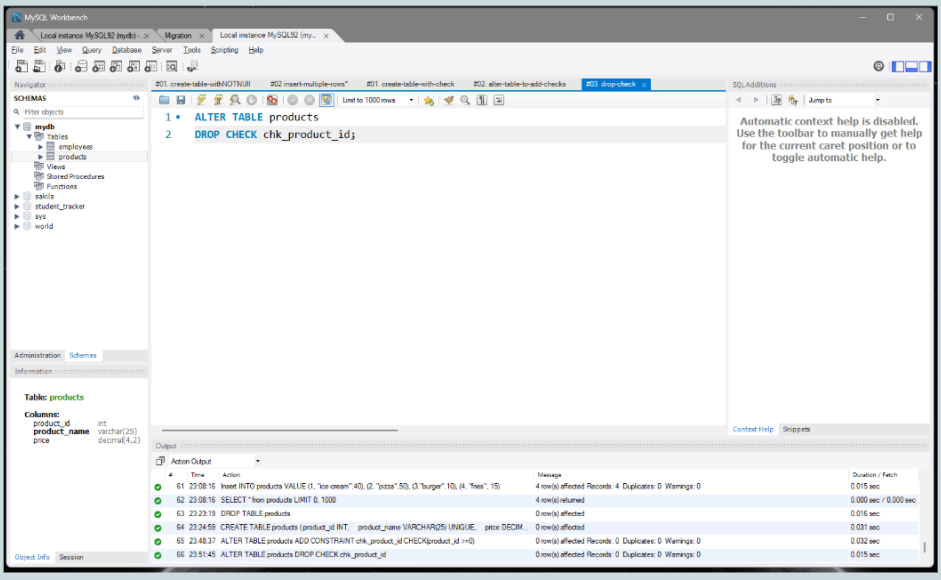


In SQL, a CHECK constraint is a database rule that enforces data integrity by specifying a condition that must be met for data inserted or updated in a table. It ensures that values in one or more columns adhere to a specific criterion, preventing invalid data from being entered.

- ALTER Table to add checks



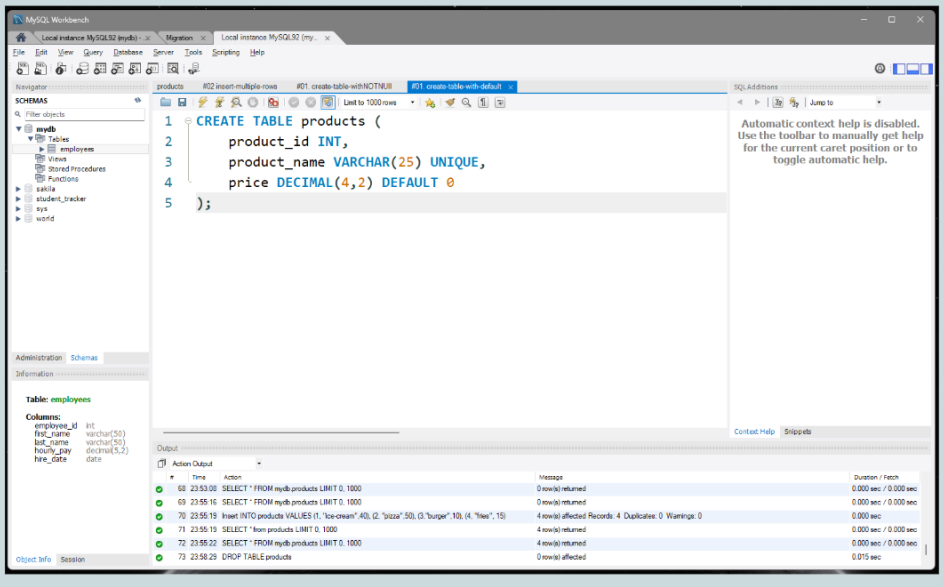
- Drop Checks



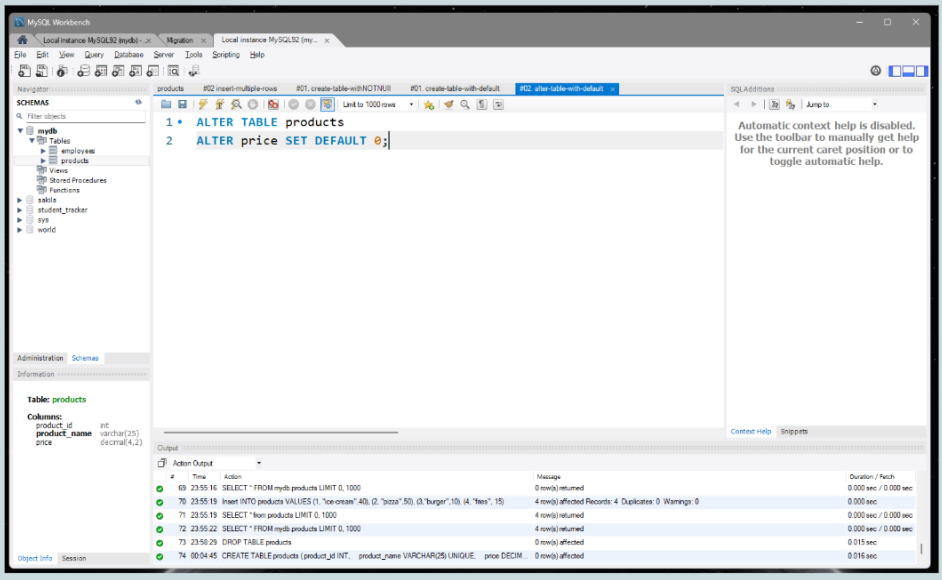
#DEFAULT

In SQL, a DEFAULT constraint specifies a default value for a column when no value is provided during an INSERT operation, ensuring that the column always has a value, even if no value is explicitly inserted.

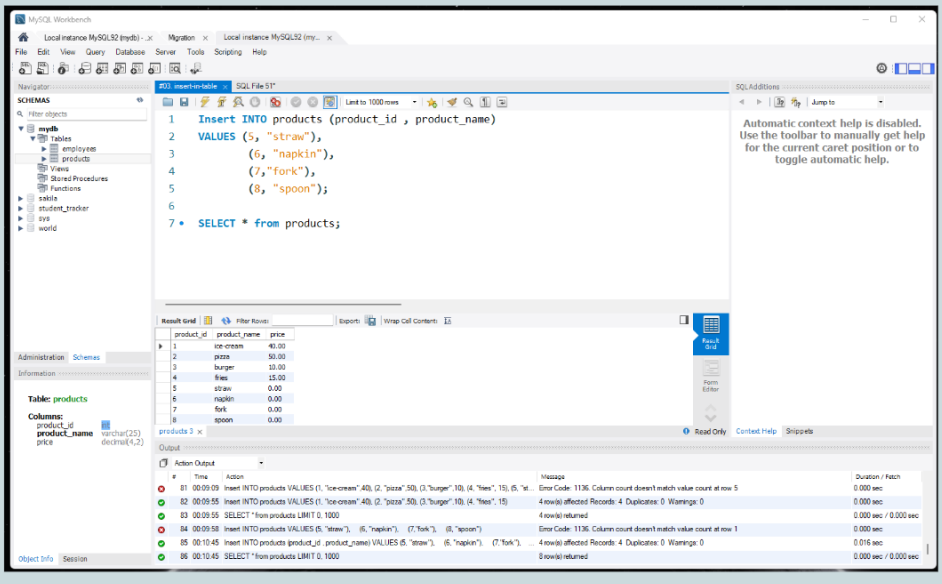
- Create Table with Default value



- Alter table to have Default value.

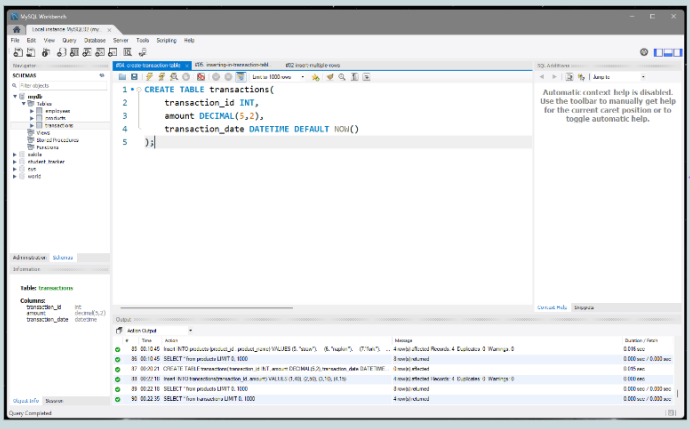


Now, add rows

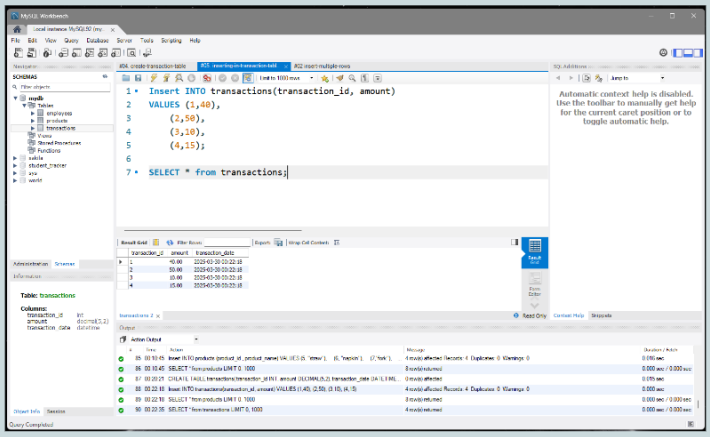


Let's See Step By step In Transaction table :>

①



②



Primary Key

In SQL, a primary key constraint uniquely identifies each record in a table, ensuring data integrity by preventing duplicate or null values in the specified column(s)

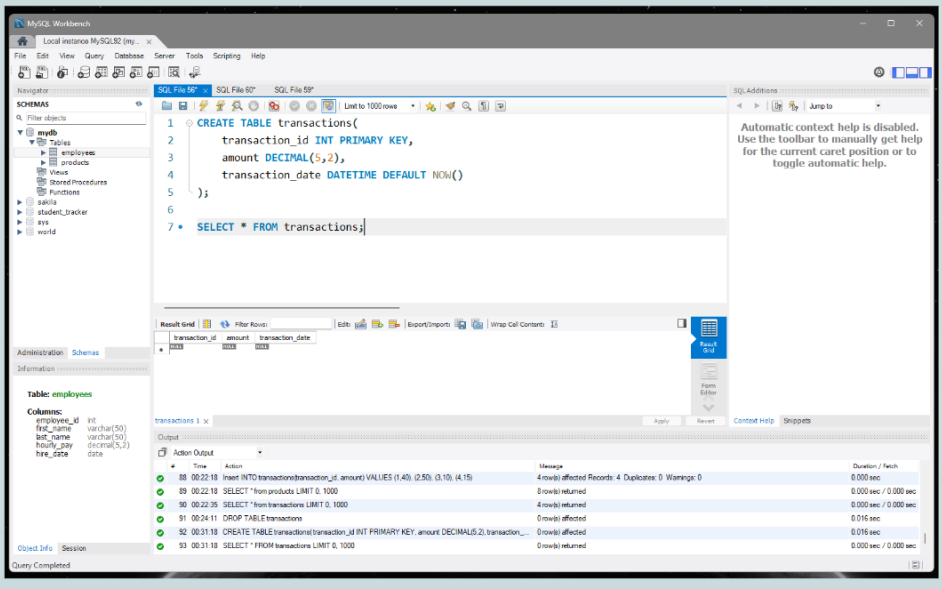
→ Primary Key constraint can be applied to the column where each value in that column has both unique & Not Null.

Primary Key = UNIQUE + Not Null

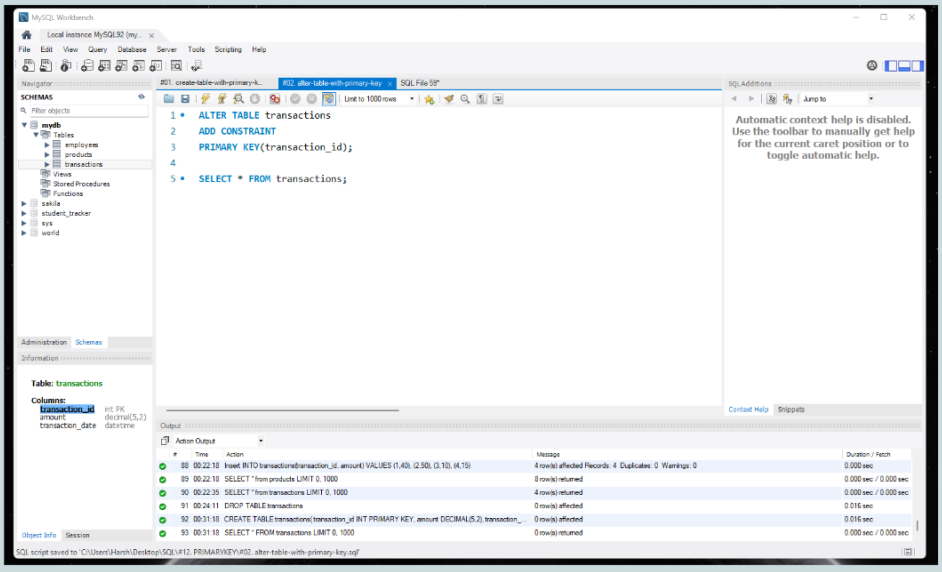
Characteristics:

- A table can have only one primary key.
- The primary key column(s) must contain unique values and cannot contain null values.

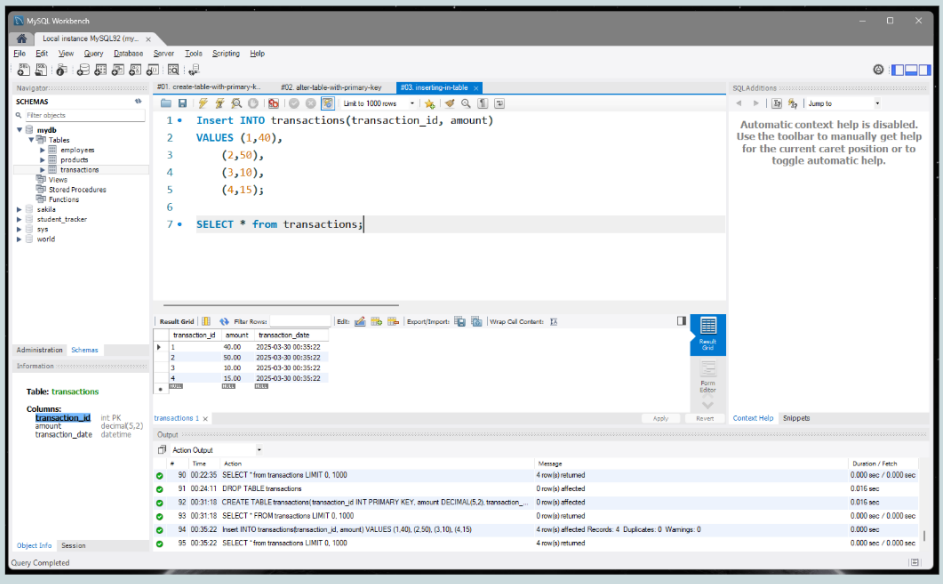
1. Create Table with Primary key



Alter table with Primary Key



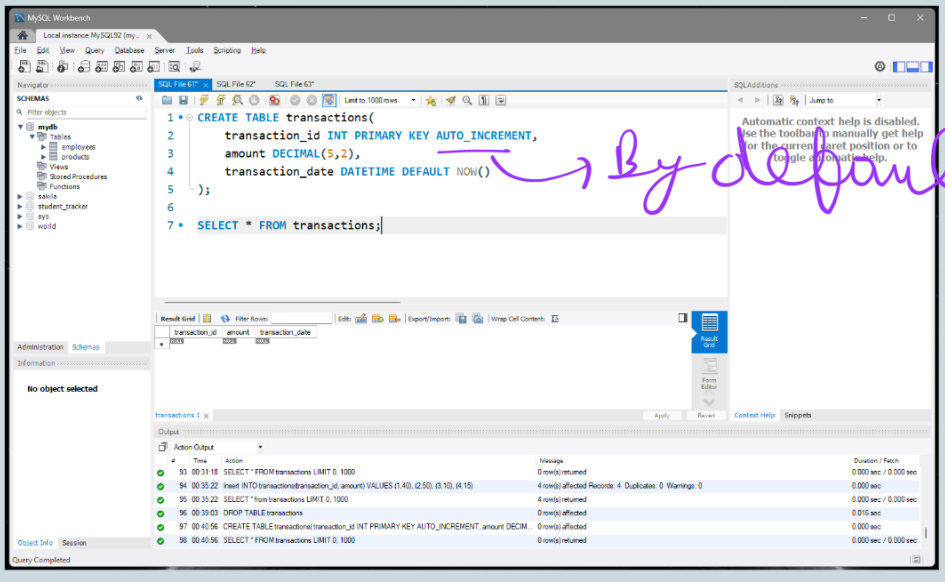
Inserting rows



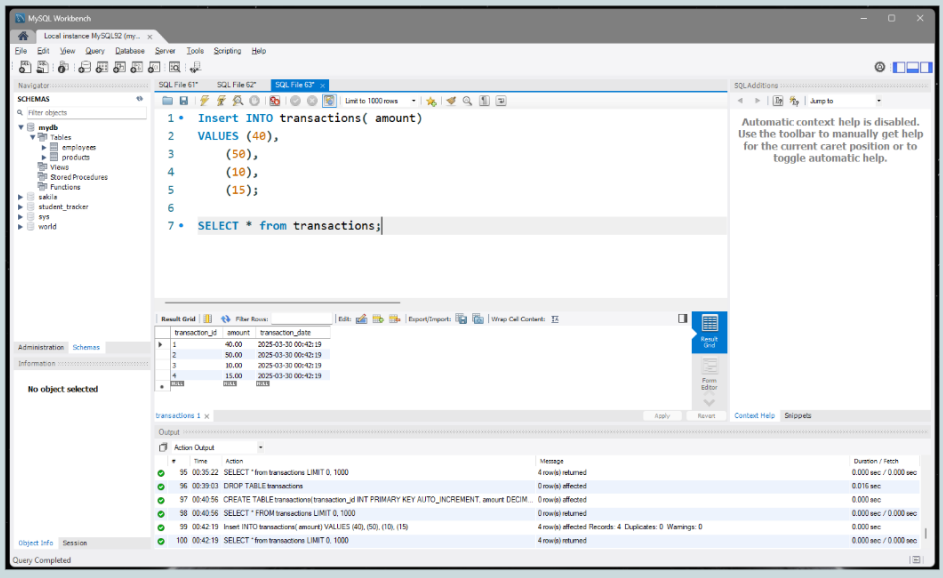
Auto Increment

In SQL, the AUTO_INCREMENT constraint (or similar mechanisms like IDENTITY in SQL Server or SERIAL in PostgreSQL) automatically generates unique, sequential values for a column when a new row is inserted, commonly used for primary keys.

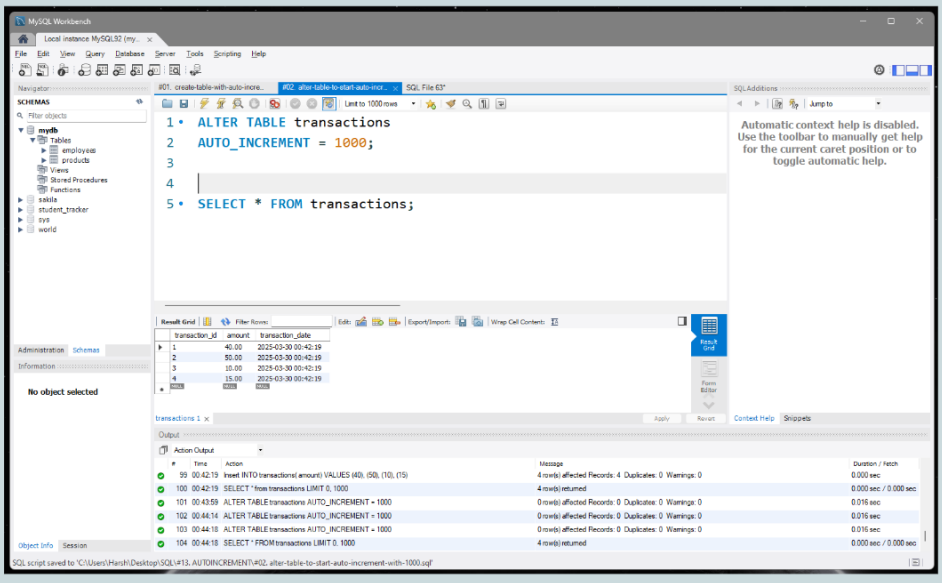
- Create table with Auto Increment



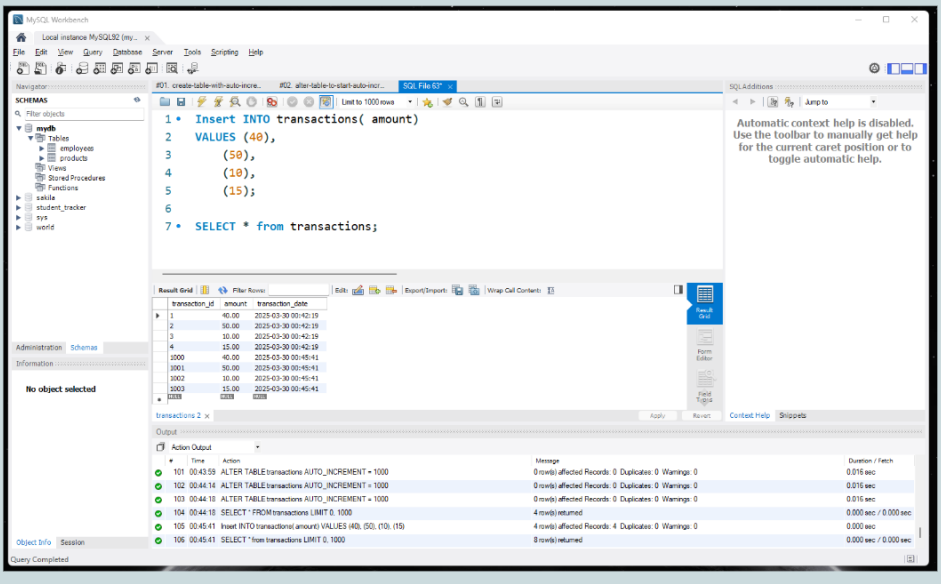
Inserting rows in transaction.



- Alter table to start auto increment with 1000

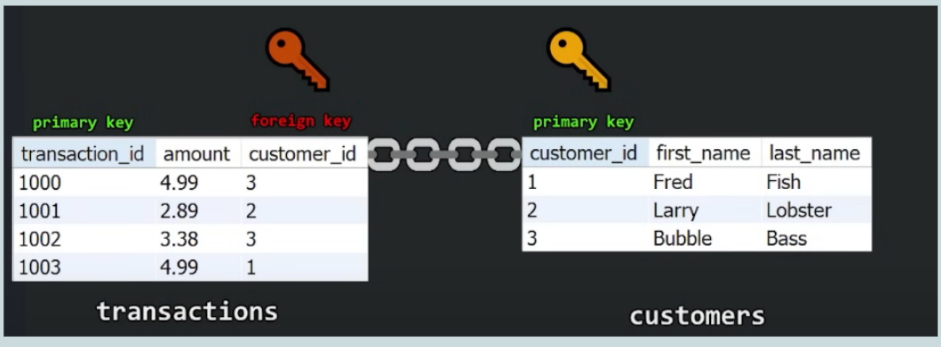


Inserting rows



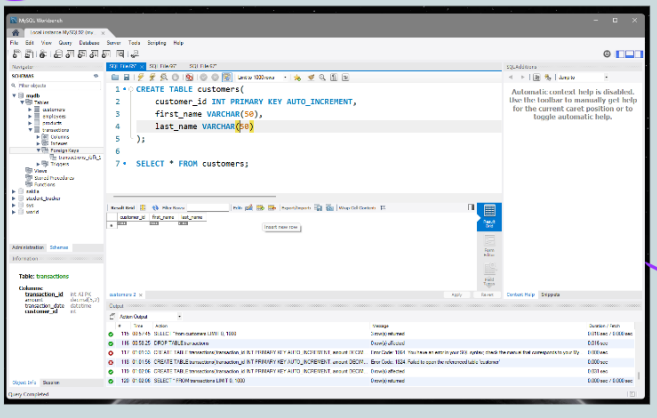
Foreign Key

In SQL, a foreign key constraint establishes a link between two tables, ensuring that values in a specific column (or columns) in one table (the "child" table) exist as primary key values in another table (the "parent" table), maintaining data integrity and consistency across related tables

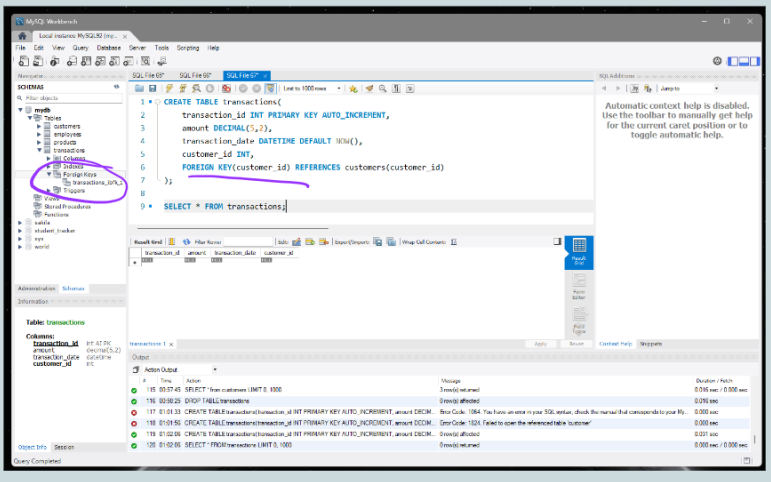


Customers Table

1

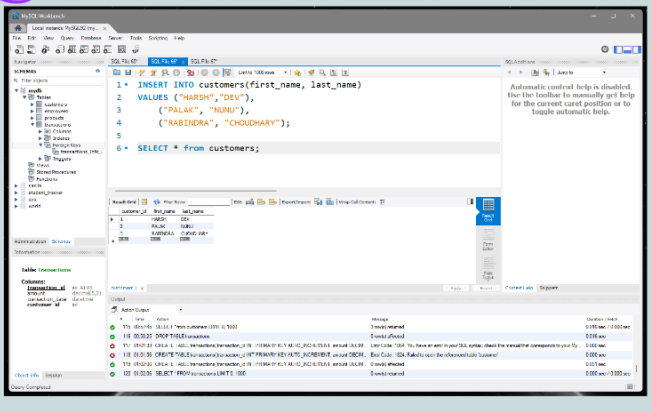


Transactions Table

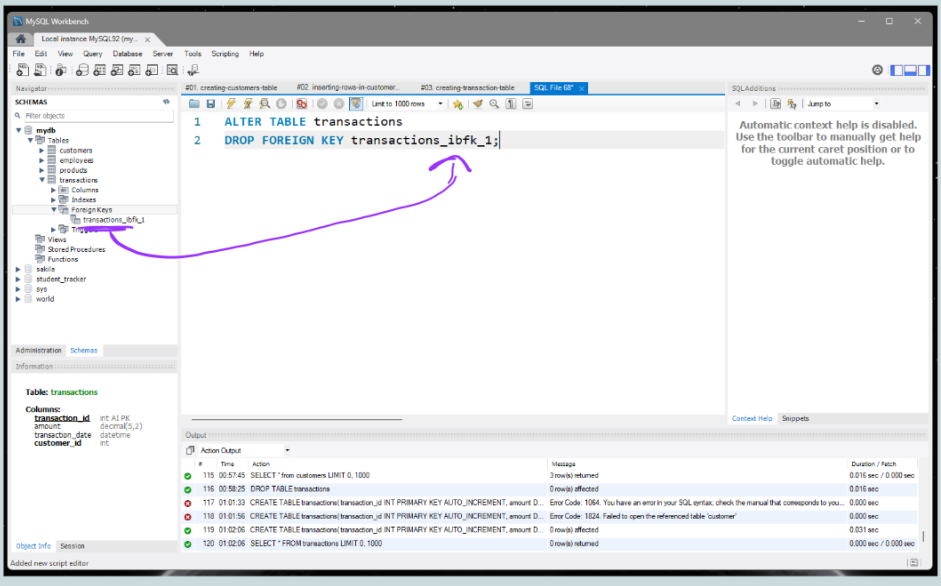


Inserting rows

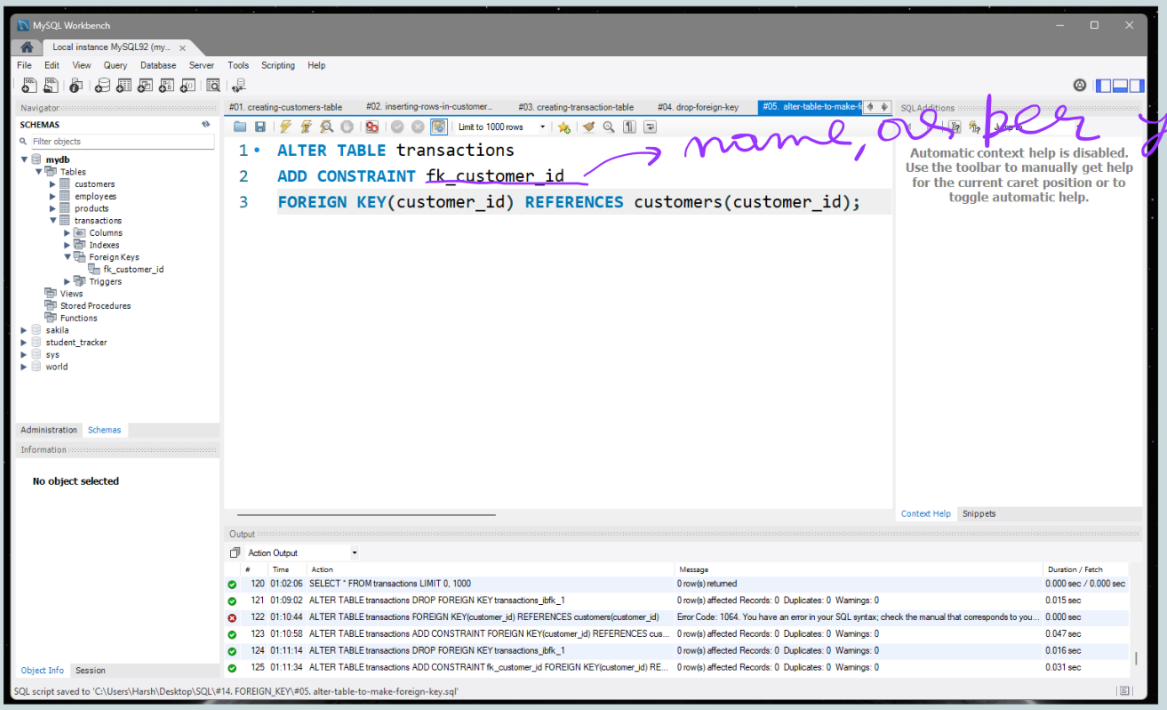
2



- Drop foreign key.

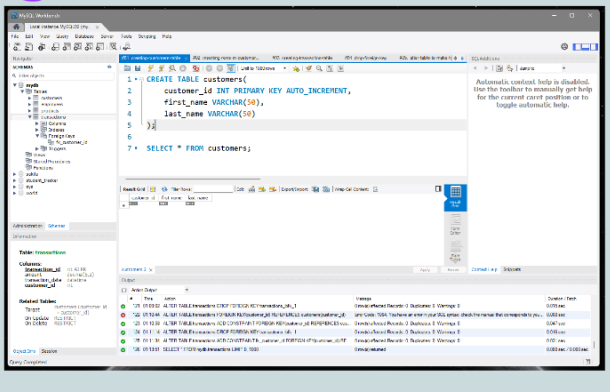


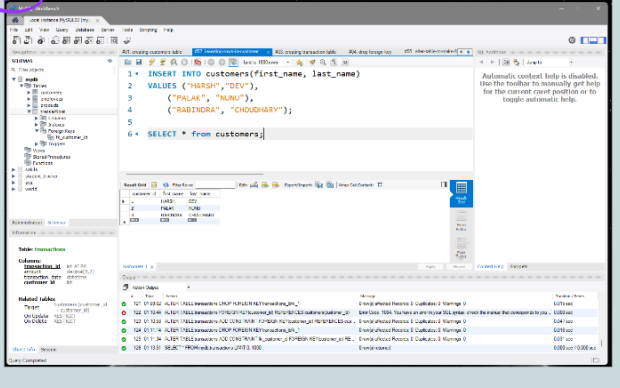
- Alter table to make foreign key



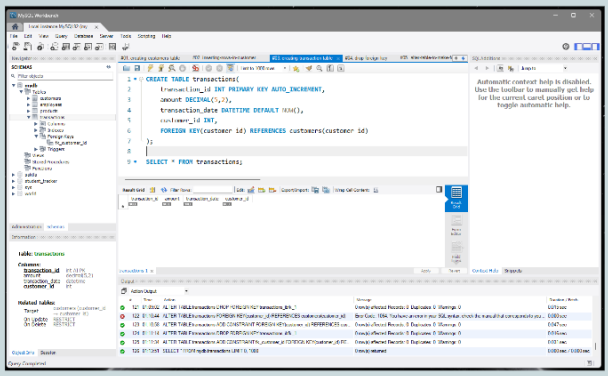
- Now, step by step Insertion →

① ②

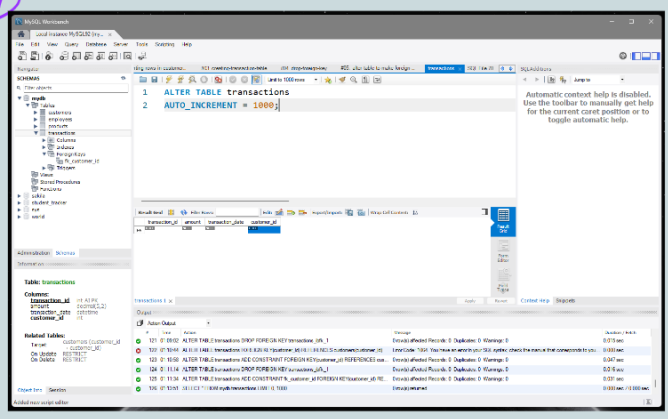




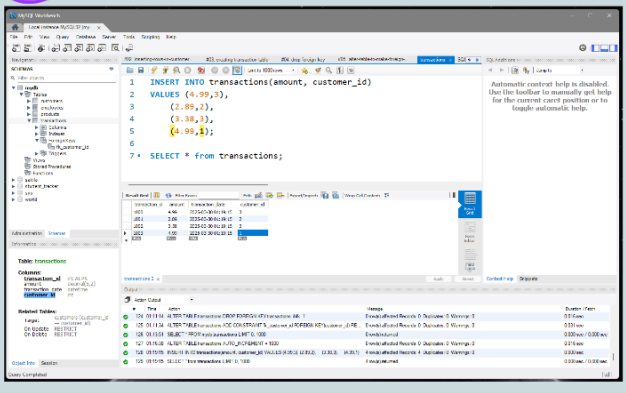
3



4



5



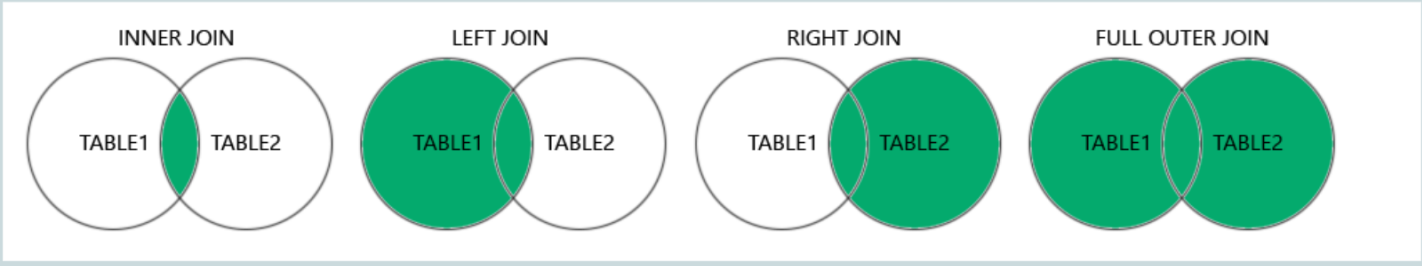
#JOINS

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- (INNER) JOIN: Returns records that have matching values in both tables
- LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
- RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
- FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table



INNER JOIN

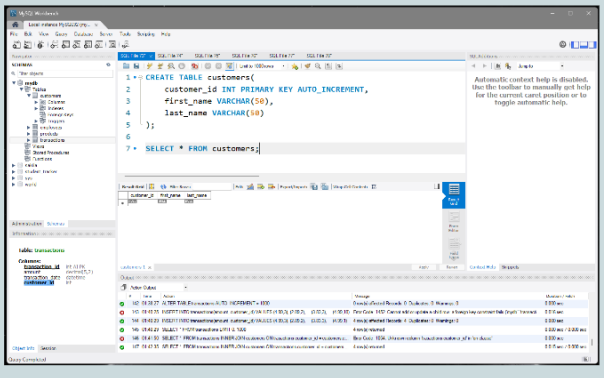
The INNER JOIN keyword selects records that have matching values in both tables.

JOIN or INNER JOIN

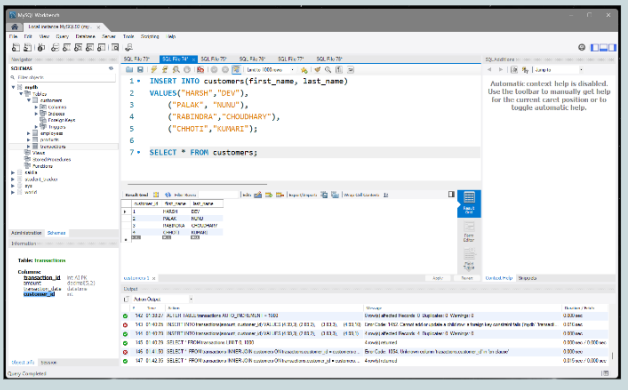
JOIN and INNER JOIN will return the same result.
INNER is the default join type for JOIN, so when you write JOIN the parser actually writes INNER JOIN.

Steps :→

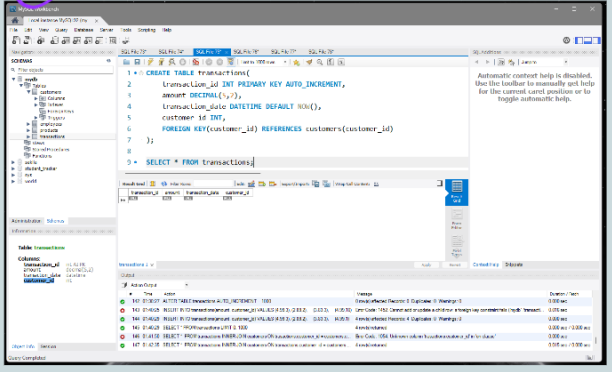
①



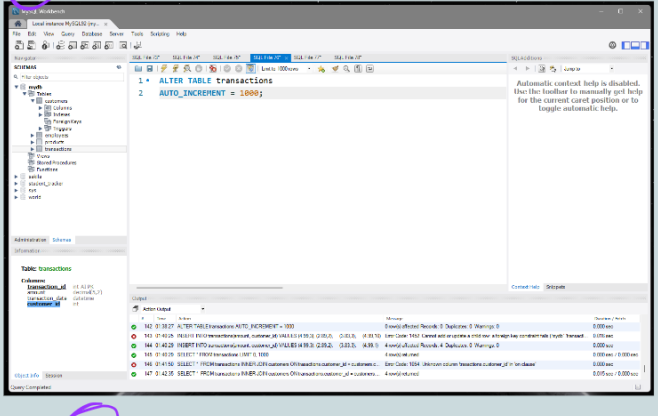
②



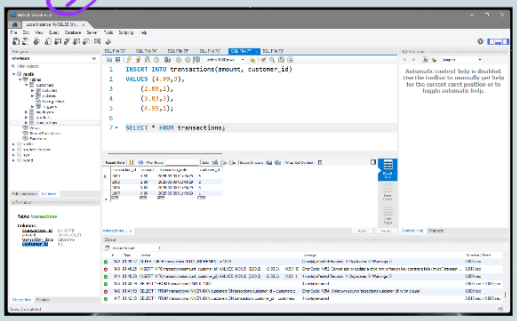
③



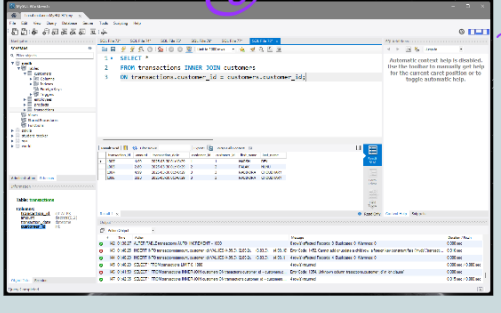
④



⑤

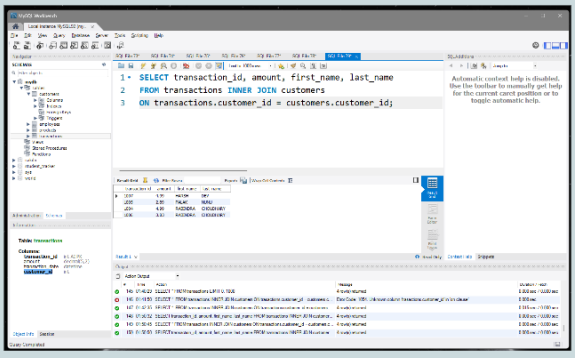


⑥



→ This will show all common Column.

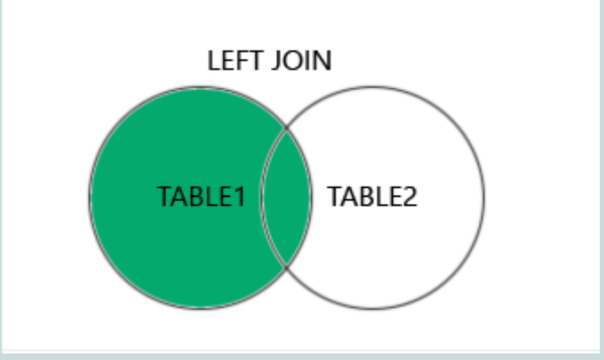
7

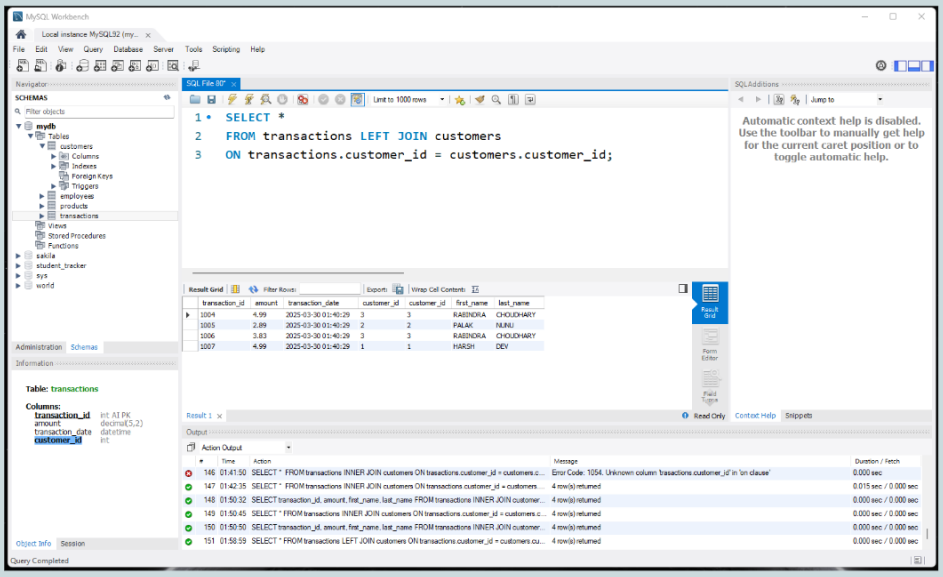


→ This will show first_name, last_name, amount of transaction_id;

SQL LEFT JOIN Keyword

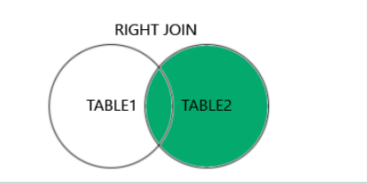
The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

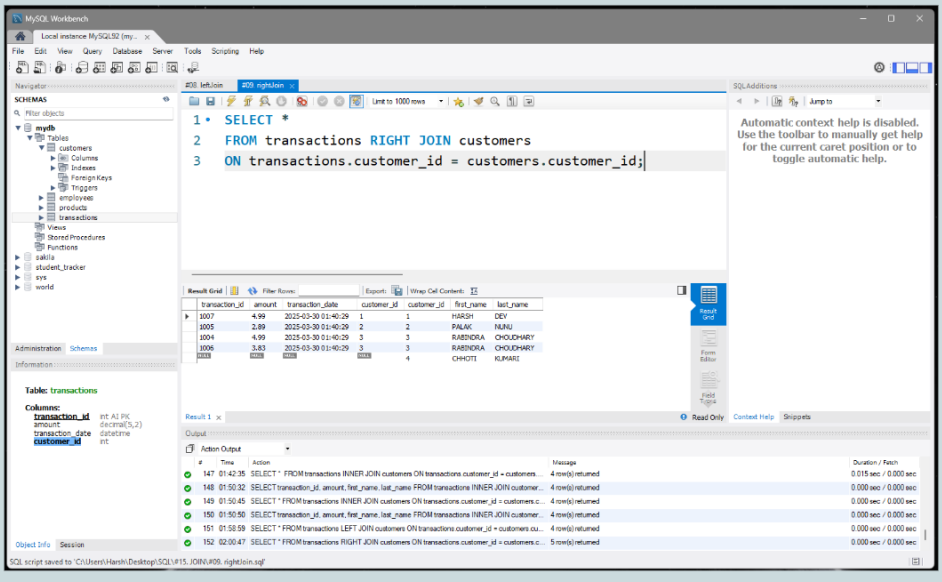




SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

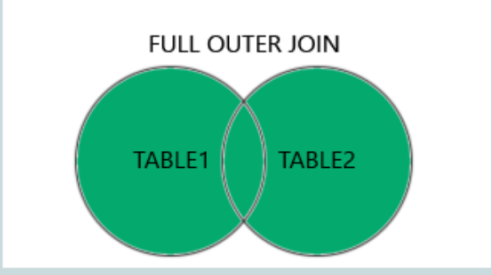




SQL FULL OUTER JOIN Keyword

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

Tip: FULL OUTER JOIN and FULL JOIN are the same.



SQL Self Join

A self join is a regular join, but the table is joined with itself.

SQL Functio

In SQL, functions are pre-written code blocks that perform specific actions on data, enabling users to manipulate, extract, and analyze information efficiently. They can be built-in (like SUM, COUNT, UPPER) or user-defined, and can be used on individual cells, records, or entire datasets.

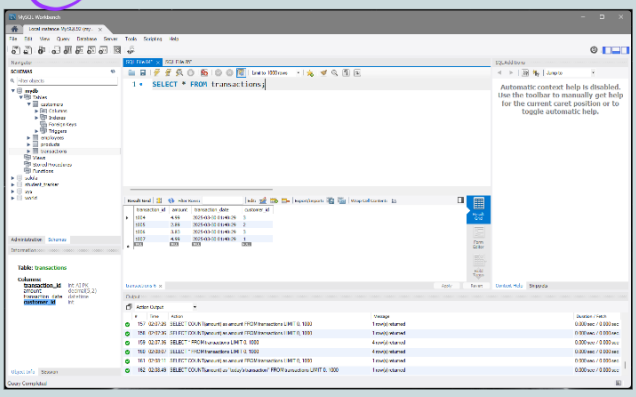
Types of SQL Functions:

- **Built-in Functions:**

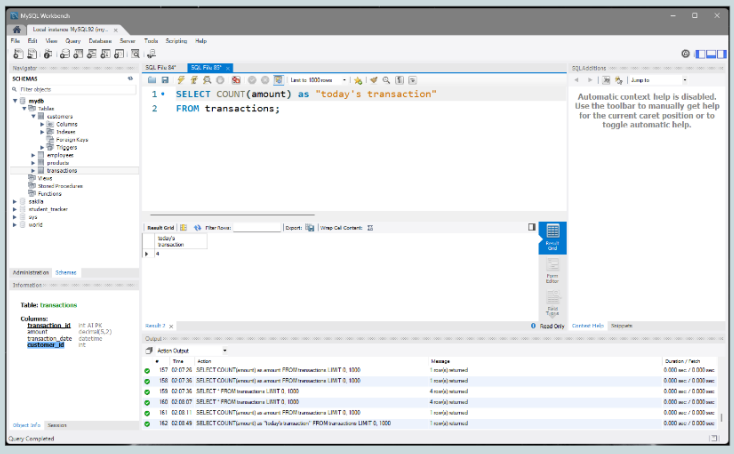
These are functions that are part of the SQL language itself and are readily available for use.

- **Aggregate Functions:** These functions perform calculations on a set of values and return a single value, such as SUM, COUNT, AVG, MIN, and MAX.
- **Scalar Functions:** These functions operate on a single value and return a single value, like UPPER, LOWER, SUBSTRING, and DATE functions.
- **String Functions:** These functions are used to manipulate strings, such as extracting substrings, converting case, or finding the length of a string.
- **Numeric/Math Functions:** These functions perform mathematical operations, such as ABS (absolute value), ROUND, and POWER.
- **Date Functions:** These functions are used to work with dates and times, such as DATE, TIME, DATEDIFF, and NOW.

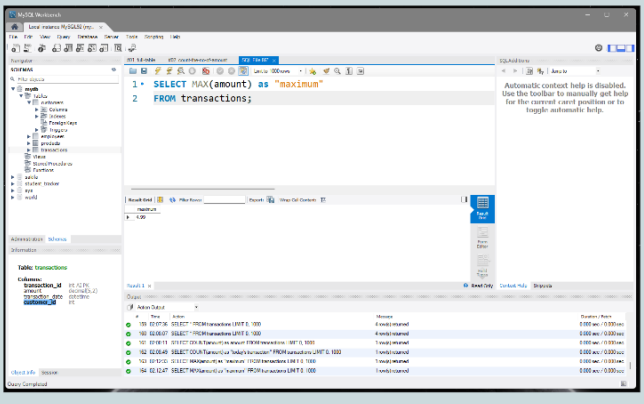
1



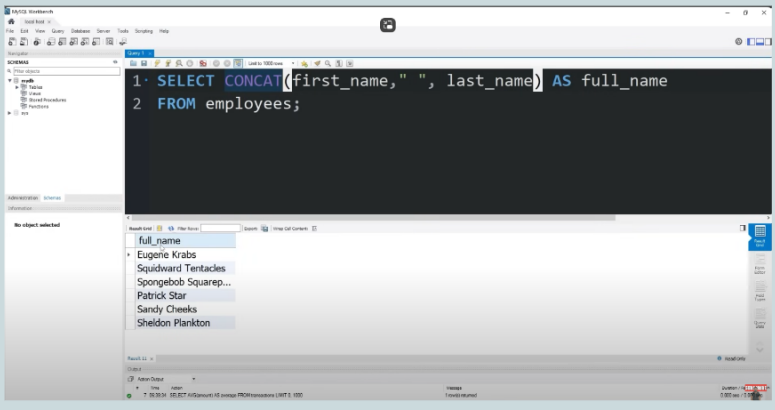
2



3



4



1. String Functions (Text Manipulation)

These functions work on string (VARCHAR, TEXT) data types.

Function	Description	Example
UPPER()	Converts text to uppercase.	`SELECT UPPER('hello');` → `HELLO`
LOWER()	Converts text to lowercase.	`SELECT LOWER('HELLO');` → `hello`
LENGTH()	Returns length of a string.	`SELECT LENGTH('SQL');` → `3`
CONCAT()	Joins two or more strings.	`SELECT CONCAT('Hello', ' ', 'World');` → `Hello World`
SUBSTRING()	Extracts part of a string.	`SELECT SUBSTRING('database', 1, 4);` → `data`
TRIM()	Removes spaces from both ends.	`SELECT TRIM(' SQL ');` → `SQL`
LTRIM()	Removes spaces from the left.	`SELECT LTRIM(' SQL');` → `SQL`
RTRIM()	Removes spaces from the right.	`SELECT RTRIM('SQL ');` → `SQL`
REPLACE()	Replaces part of a string.	`SELECT REPLACE('SQL is great', 'great', 'awesome');` → `SQL is awesome`
LEFT()	Extracts left portion of a string.	`SELECT LEFT('SQL', 2);` → `SQ`
RIGHT()	Extracts right portion of a string.	`SELECT RIGHT('SQL', 2);` → `QL`
LOCATE()	Finds the position of a substring.	`SELECT LOCATE('l', 'Hello');` → `3`
ASCII()	Returns ASCII code of a character.	`SELECT ASCII('A');` → `65`
CHAR()	Converts ASCII code to a character.	`SELECT CHAR(65);` → `A`

2. Scalar Functions (Return Single Value)

These functions return a single value based on input.

Function	Description	Example
IFNULL()	Returns alternative value if NULL.	`SELECT IFNULL(NULL, 'No Value');` → `No Value`
COALESCE()	Returns first non-null value.	`SELECT COALESCE(NULL, NULL, 'SQL', NULL);` → `SQL`
NULLIF()	Returns NULL if values are equal.	`SELECT NULLIF(10, 10);` → `NULL`
CAST()	Converts data type.	`SELECT CAST(123 AS CHAR);` → `'123'`
CONVERT()	Converts data type.	`SELECT CONVERT(123, CHAR);` → `'123'`

3. Mathematical Functions (Numeric Operations)

Used for performing calculations on numeric values.

Function	Description	Example
ABS()	Returns absolute value.	`SELECT ABS(-10);` → `10`
CEIL()	Rounds up to next integer.	`SELECT CEIL(4.2);` → `5`
FLOOR()	Rounds down to previous integer.	`SELECT FLOOR(4.9);` → `4`
ROUND()	Rounds to nearest integer.	`SELECT ROUND(4.6);` → `5`
MOD()	Returns remainder of division.	`SELECT MOD(10, 3);` → `1`
POWER()	Returns power of a number.	`SELECT POWER(2, 3);` → `8`
SQRT()	Returns square root.	`SELECT SQRT(16);` → `4`
RAND()	Generates a random number between 0 and 1.	`SELECT RAND();` → `0.7463`
EXP()	Returns e^x.	`SELECT EXP(1);` → `2.718`
LOG()	Returns natural logarithm.	`SELECT LOG(10);` → `2.302`
SIN()	Returns sine of angle (radians).	`SELECT SIN(PI()/2);` → `1`
COS()	Returns cosine of angle.	`SELECT COS(0);` → `1`
TAN()	Returns tangent of angle.	`SELECT TAN(PI()/4);` → `1`

4. Date and Time Functions

Used for manipulating and formatting date values.

Function	Description	Example
NOW()	Returns current date & time.	`SELECT NOW();` → `2025-03-29 14:00:00`
CURDATE()	Returns current date.	`SELECT CURDATE();` → `2025-03-29`
CURTIME()	Returns current time.	`SELECT CURTIME();` → `14:00:00`
DATE()	Extracts date from datetime.	`SELECT DATE('2025-03-29 14:00:00');` → `2025-03-29`
TIME()	Extracts time from datetime.	`SELECT TIME('2025-03-29 14:00:00');` → `14:00:00`
DAY()	Returns day of the month.	`SELECT DAY('2025-03-29');` → `29`
MONTH()	Returns month number.	`SELECT MONTH('2025-03-29');` → `3`
YEAR()	Returns year from date.	`SELECT YEAR('2025-03-29');` → `2025`
HOUR()	Returns hour from time.	`SELECT HOUR('14:30:00');` → `14`
MINUTE()	Returns minute from time.	`SELECT MINUTE('14:30:00');` → `30`
SECOND()	Returns second from time.	`SELECT SECOND('14:30:59');` → `59`
DATEDIFF()	Returns difference between dates.	`SELECT DATEDIFF('2025-04-10', '2025-03-29');` → `12`
ADDDATE()	Adds days to a date.	`SELECT ADDDATE('2025-03-29', INTERVAL 10 DAY);` → `2025-04-08`
SUBDATE()	Subtracts days from a date.	`SELECT SUBDATE('2025-03-29', INTERVAL 10 DAY);` → `2025-03-19`

5. Aggregate Functions (Grouping Data)

Used for calculations on groups of rows.

Function	Description	Example
COUNT()	Counts number of rows.	`SELECT COUNT(*) FROM employees;`
SUM()	Returns sum of a column.	`SELECT SUM(salary) FROM employees;`
AVG()	Returns average value.	`SELECT AVG(salary) FROM employees;`
MAX()	Returns maximum value.	`SELECT MAX(salary) FROM employees;`
MIN()	Returns minimum value.	`SELECT MIN(salary) FROM employees;`
GROUP_CONCAT()	Joins multiple values into one.	`SELECT GROUP_CONCAT(name) FROM employees;`

AND, OR, NOT

In SQL, AND, OR, and NOT are logical operators used to combine or manipulate conditions in WHERE or HAVING clauses to filter data based on specific criteria.

Here's a breakdown:

- AND:

Returns TRUE only if all conditions separated by AND are TRUE.

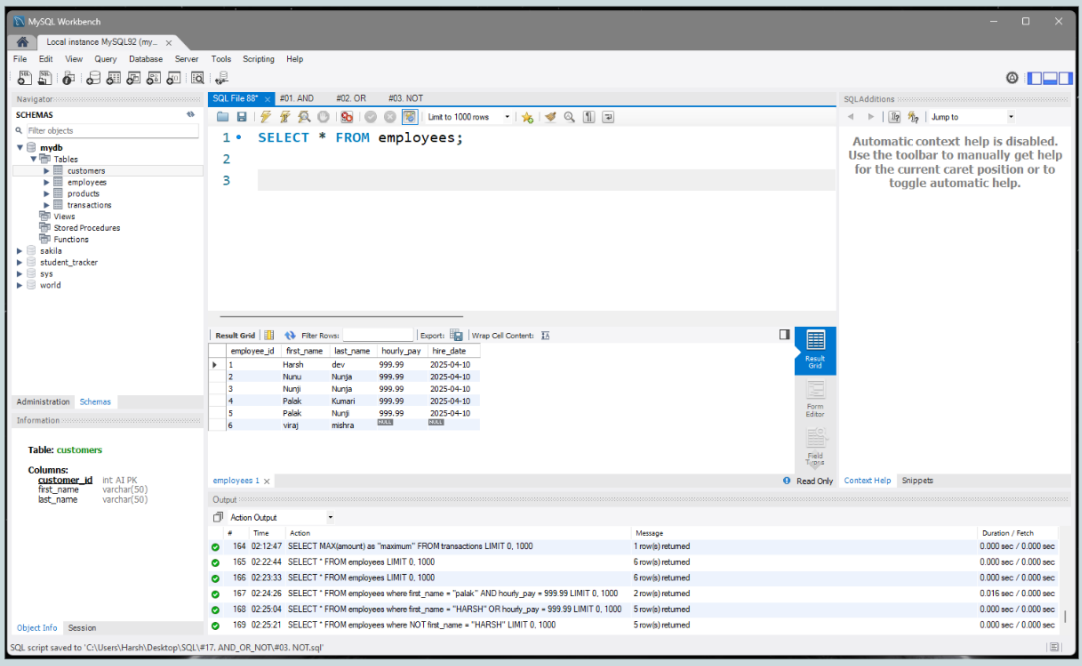
- **Example:** WHERE age > 18 AND city = 'New York' (This selects records where both conditions are met)
- OR:

Returns TRUE if at least one of the conditions separated by OR is TRUE.

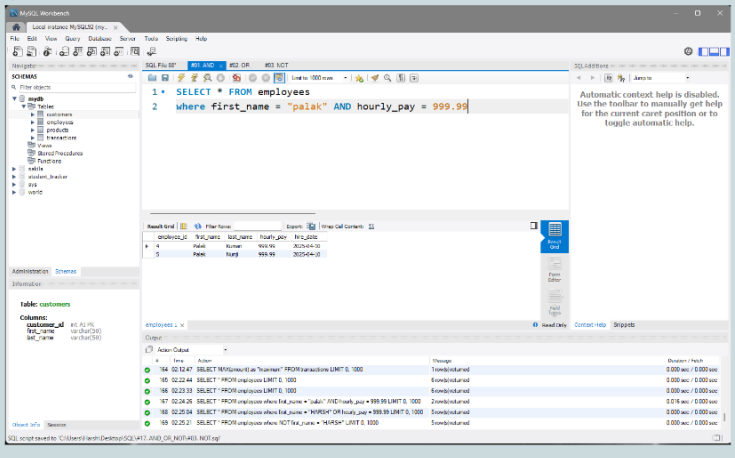
- **Example:** WHERE age > 65 OR disability = 'Yes' (This selects records where either condition is met)
- NOT:

Negates a condition, returning TRUE if the condition is FALSE.

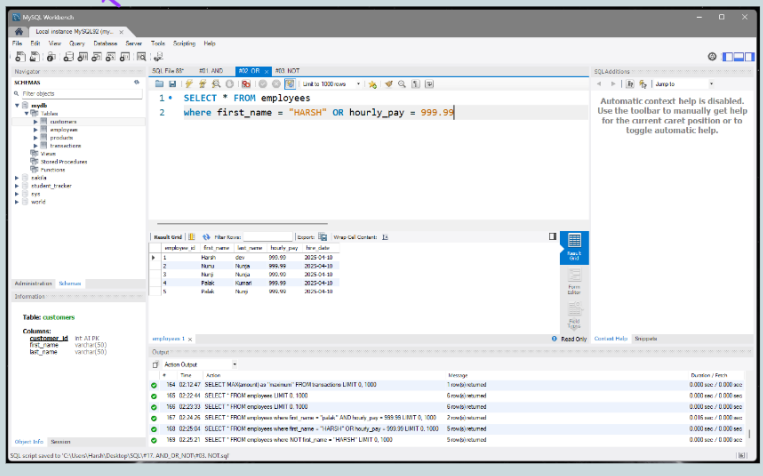
- **Example:** WHERE NOT (age < 18) (This selects records where the age is not less than 18)



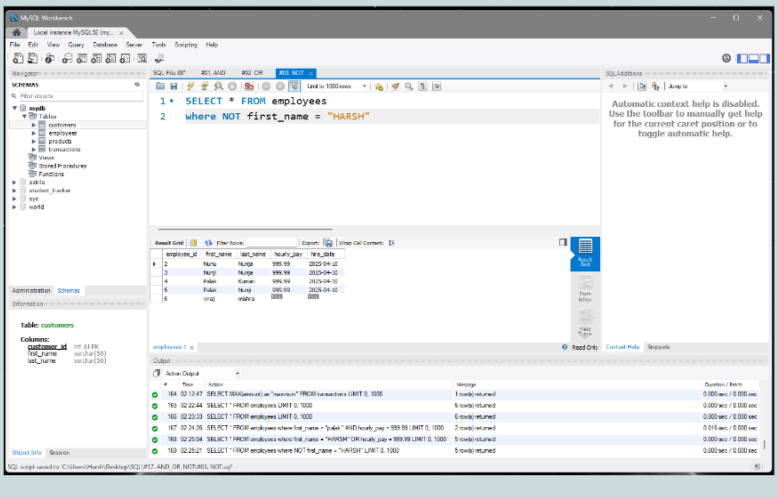
AND



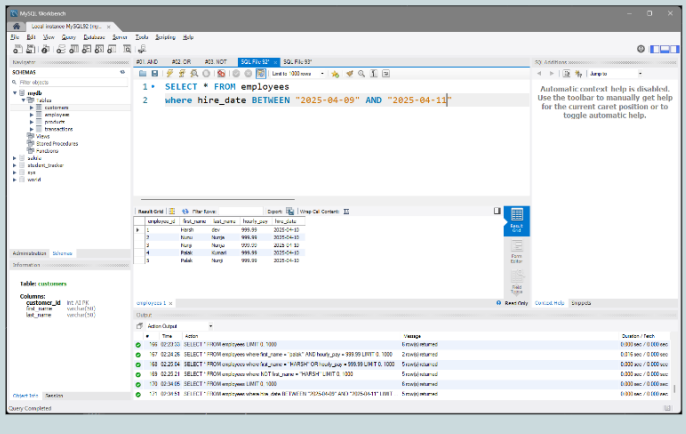
OR



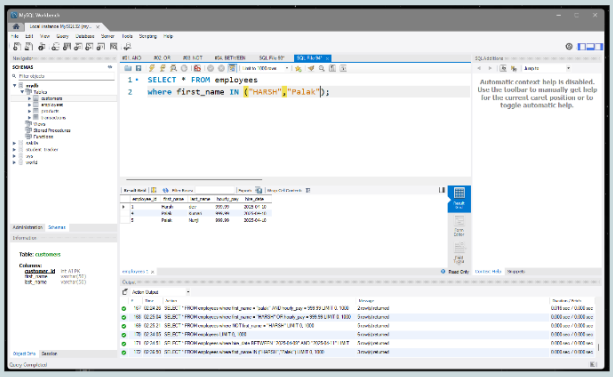
NOT



between



IN



#. Wild Cards

SQL Wildcard Characters

A wildcard character is used to substitute one or more characters in a string.

Wildcard characters are used with the LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

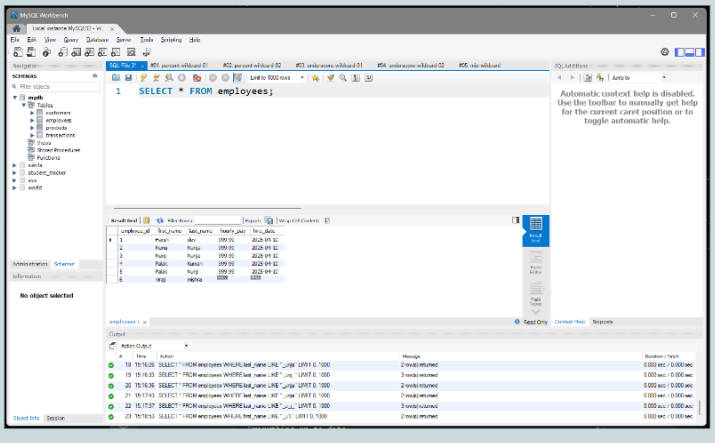
Wildcard Characters

Symbol	Description
%	Represents zero or more characters
_	Represents a single character
[]	Represents any single character within the brackets *
^	Represents any character not in the brackets *
-	Represents any single character within the specified range *
{}	Represents any escaped character **

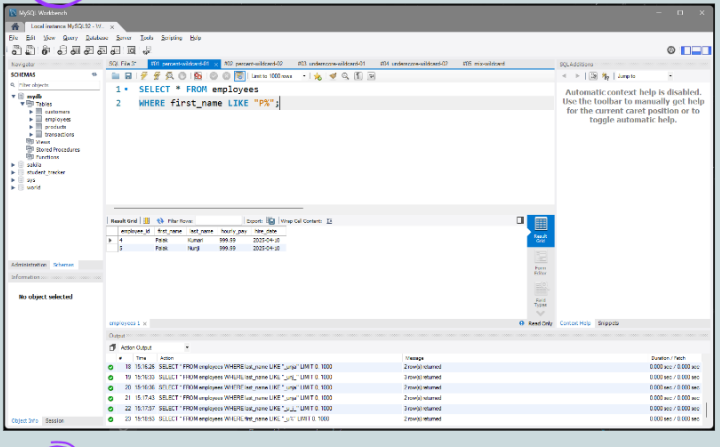
* Not supported in PostgreSQL and MySQL databases.

** Supported only in Oracle databases.

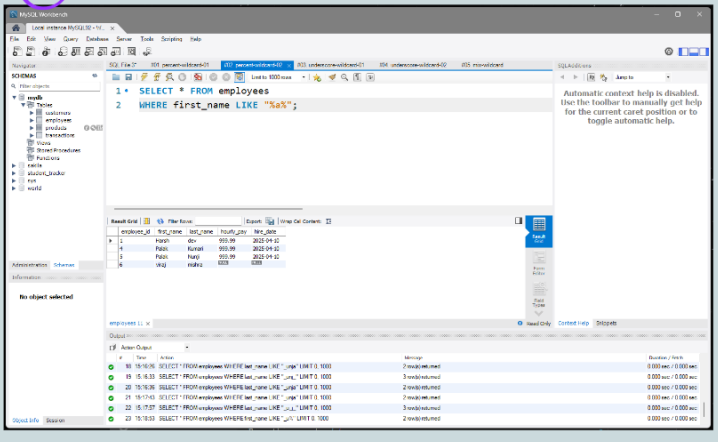
1



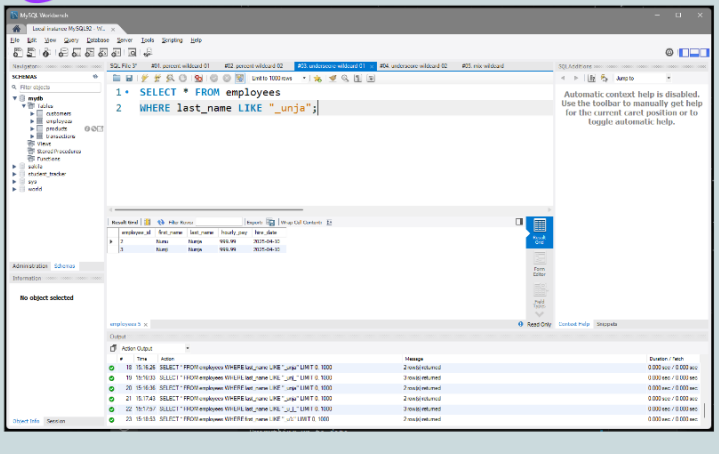
2



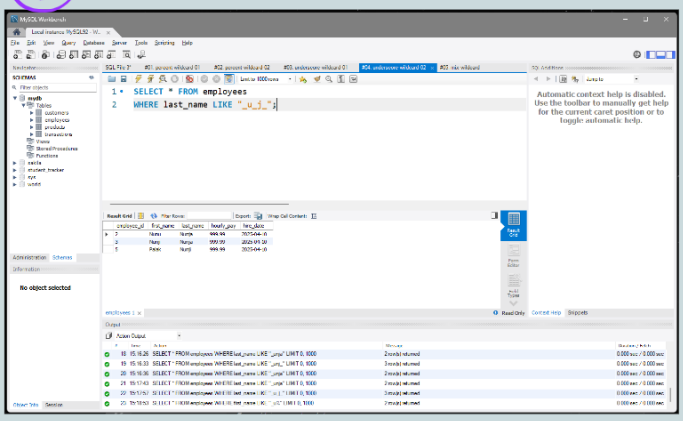
3



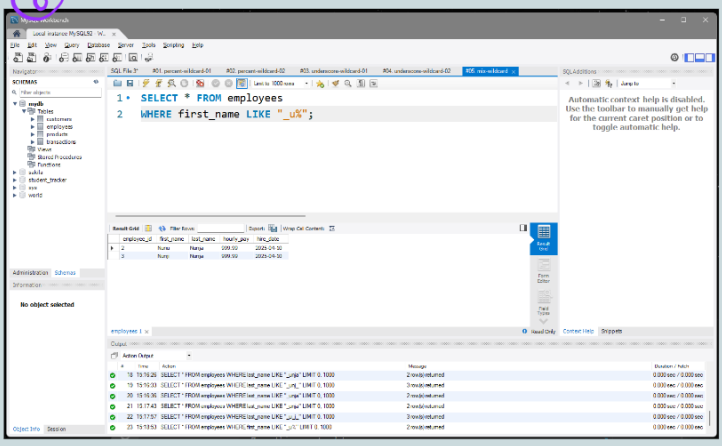
4



5



6



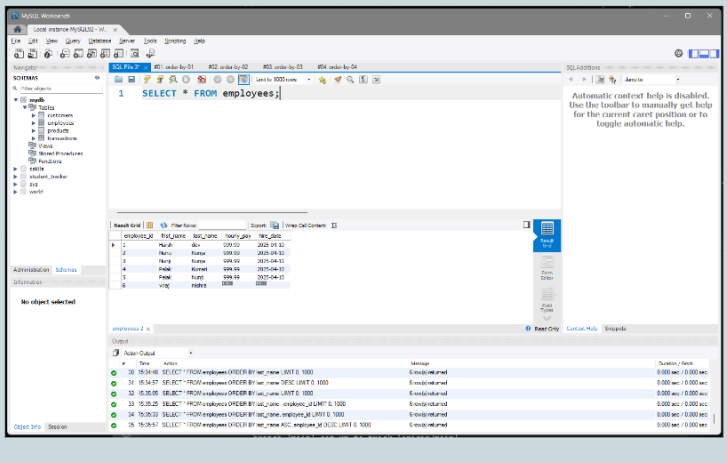
#ORDER BY

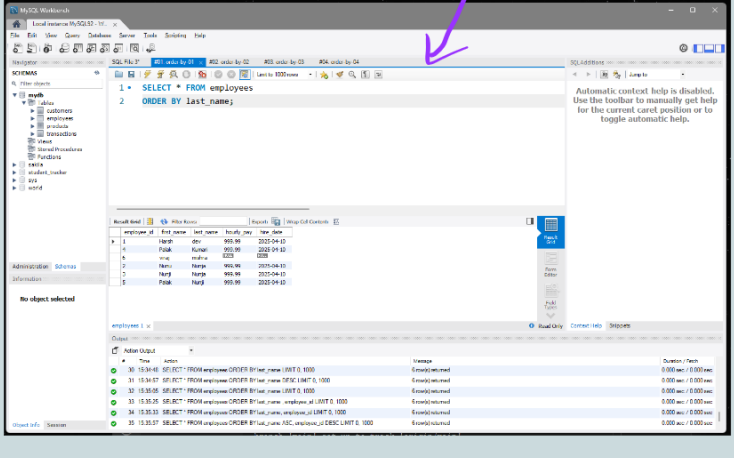
The ORDER BY clause in SQL sorts the result set of a SELECT statement, allowing you to arrange data in ascending (default) or descending order based on one or more columns.

- **Purpose:** The primary function of ORDER BY is to organize the data in a way that makes it easier to interpret and analyze.

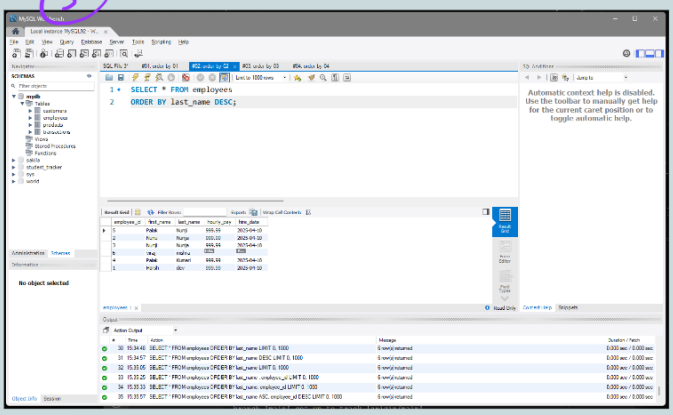
- **Key Points:**
- ORDER BY is used after the WHERE clause, if present.
- You can sort by column names, aliases, or column positions in the SELECT statement.
- The ORDER BY clause does not affect the data in the database table itself; it only affects the order of the rows returned in the query result.

→ By, default, it will sort in ascending order.

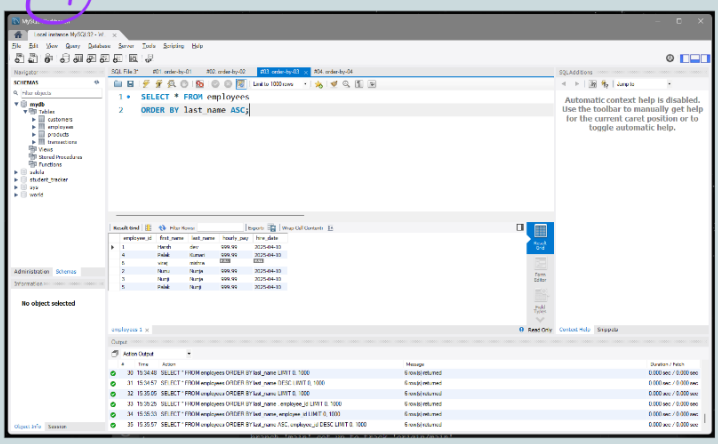




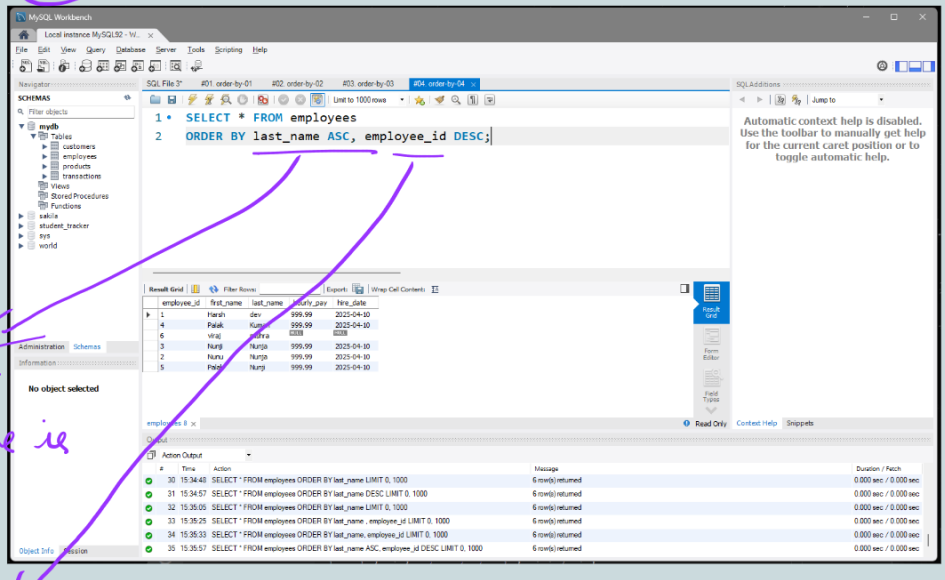
3



4



5



if two last_name is same then it will compare with employee_id.

#LIMIT Clause

The SQL LIMIT clause restricts the number of rows returned by a query, allowing you to retrieve a subset of data, which is useful for pagination or when dealing with large datasets where retrieving all rows would be inefficient.

- **Purpose:**

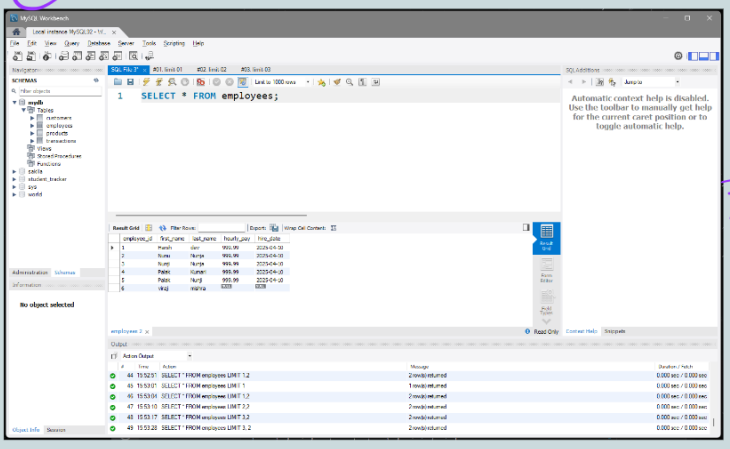
The primary function of the LIMIT clause is to control the size of the result set returned by a SELECT statement.

- **Use Cases:**
- **Data Pagination:** Displaying data in smaller chunks for user

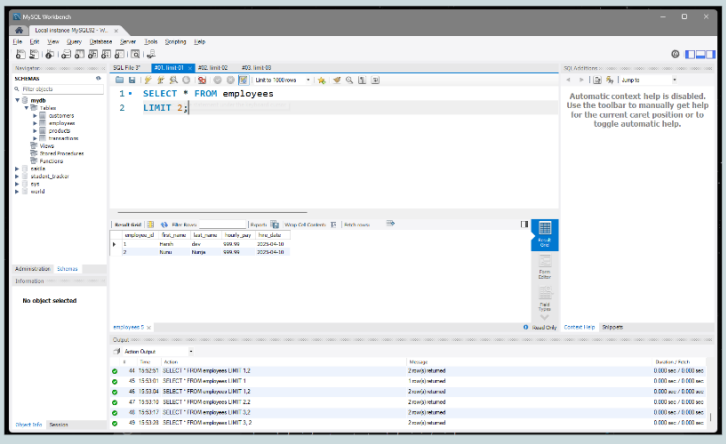
interfaces.

- **Performance Optimization:** Reducing the amount of data retrieved from large tables.
- **Retrieving Top/Bottom N Records:** Finding the highest or lowest values in a column.

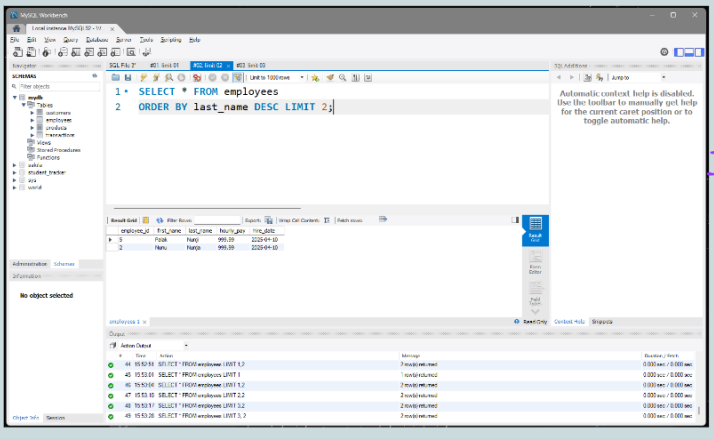
1



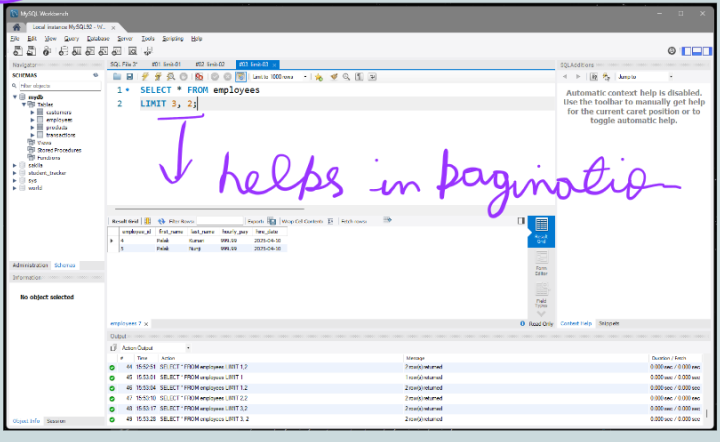
2



3



4



UNION

In SQL, the UNION operator combines the result sets of two or more SELECT statements into a single, distinct result set, removing any duplicate rows.

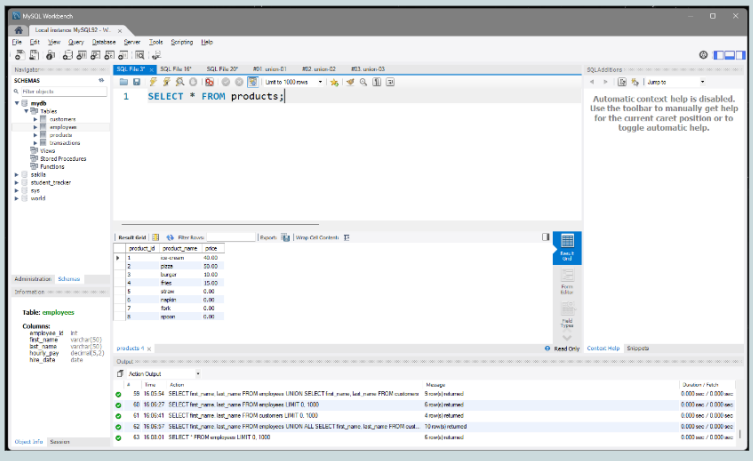
- **Purpose:**

The primary function of UNION is to merge data from different SELECT statements, effectively stacking the results of one query on top of another.

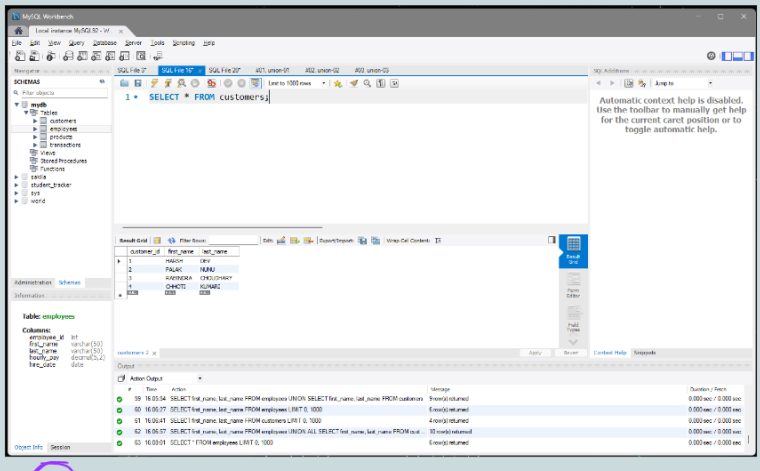
- **Requirements:**

To use UNION, the SELECT statements must have the same number of columns, and those columns must have compatible data types.

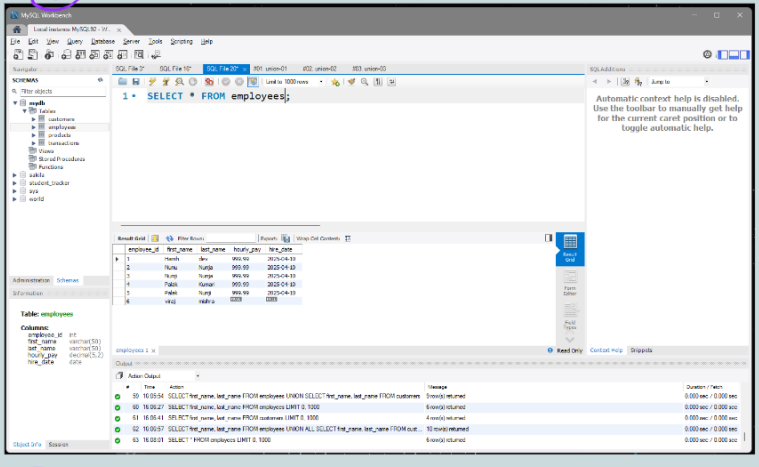
1



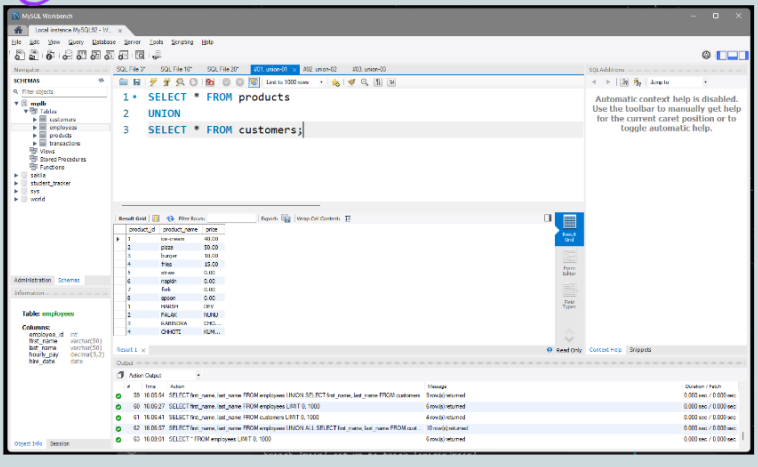
2



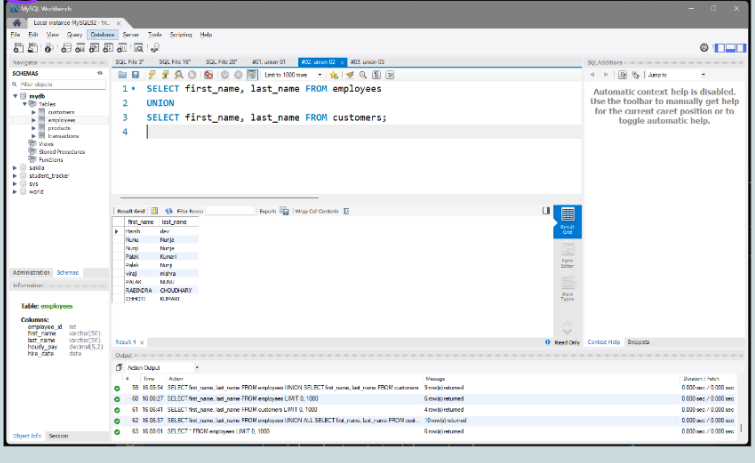
3



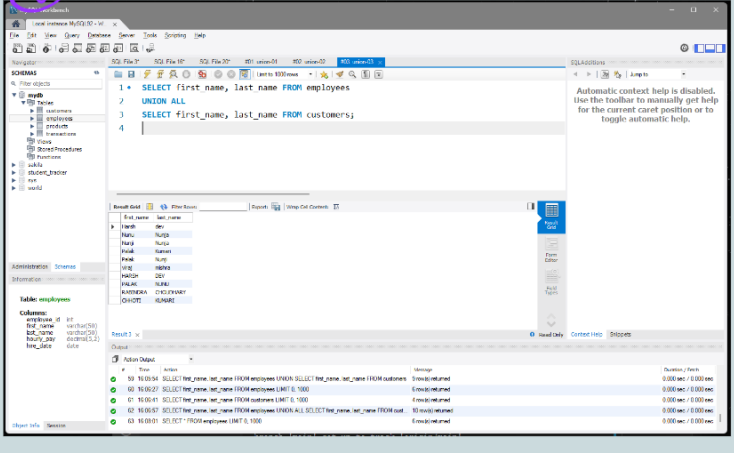
4



5



6

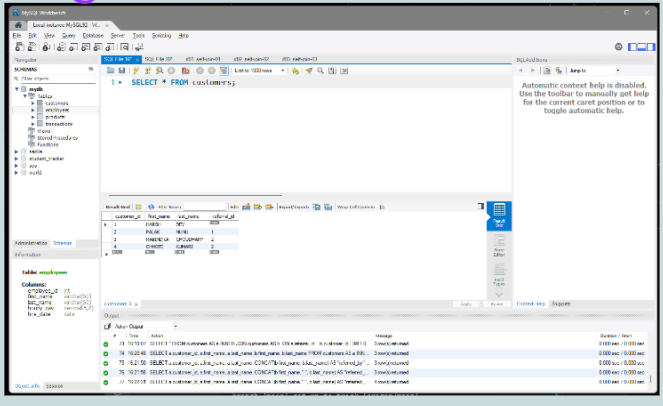


Self Join

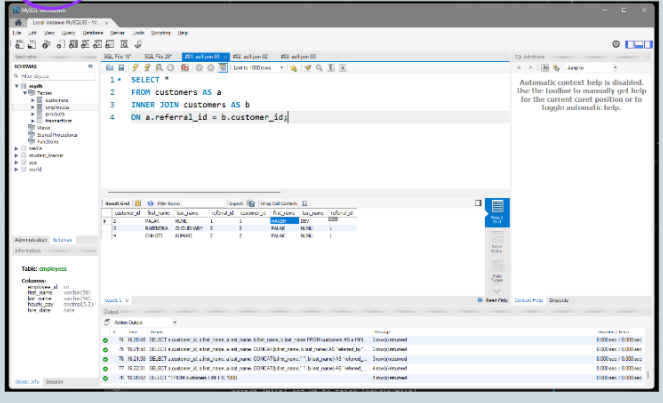
In SQL, a self join allows you to join a table with itself, enabling you to compare or relate rows within the same table by creating virtual copies of the table and establishing relationships between them.

- **Common Use Cases:**
- **Hierarchical Data:** Finding parent-child relationships in organizational charts or family trees.
- **Version Tracking:** Comparing different versions of data in the same table.
- **Recursive Queries:** Finding all descendants or ancestors of a particular record.
- **Network or Graph Data:** Working with data where nodes or entities have connections to other nodes in the same table.

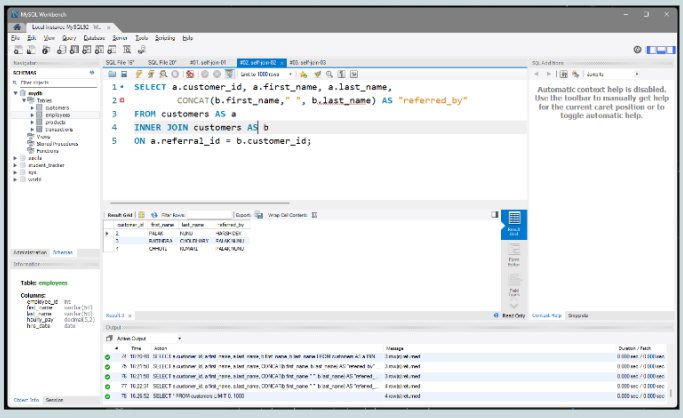
1



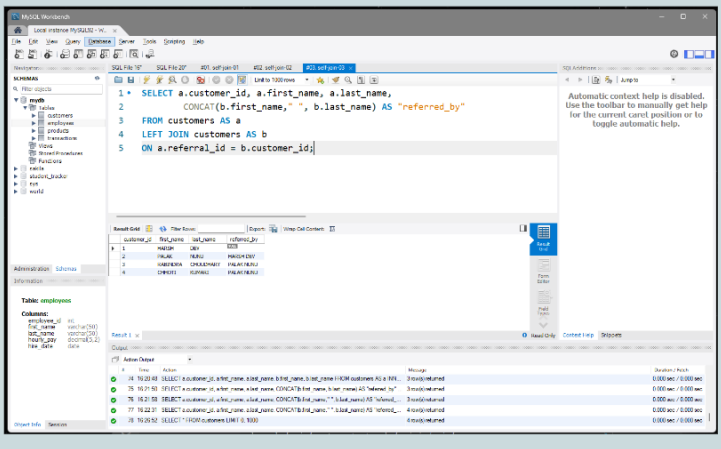
2



3



4



Views in SQL

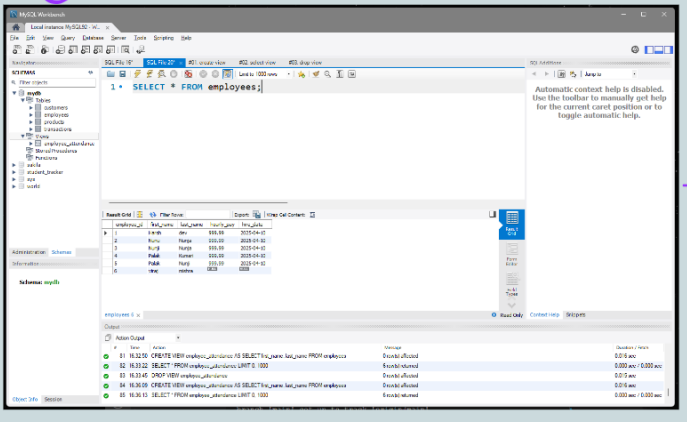
What is a View in SQL?

A view in SQL is a saved SQL query that acts as a virtual table. It can fetch data from one or more tables and present it in a customized format, allowing developers to:

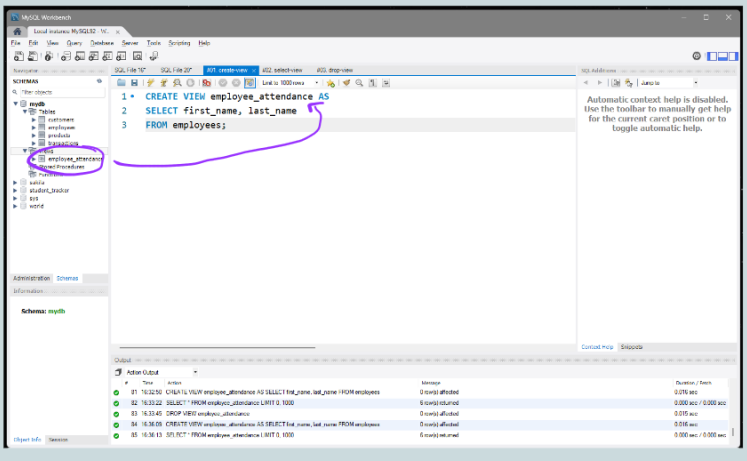
- **Simplify Complex Queries:** Encapsulate complex joins and conditions into a single object.
- **Enhance Security:** Restrict access to specific columns or rows.
- **Present Data Flexibly:** Provide tailored data views for different users.

→ Views are always up-to-date!

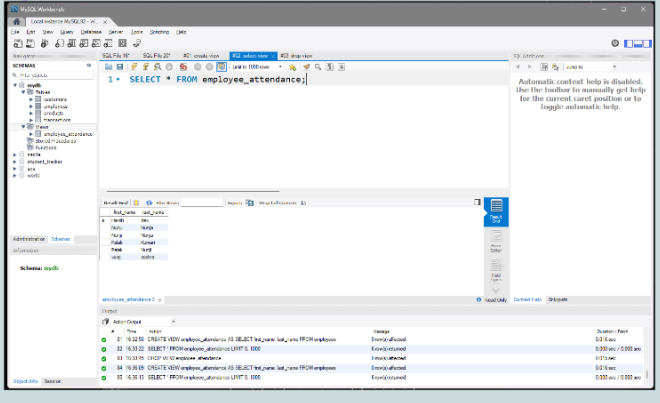
1



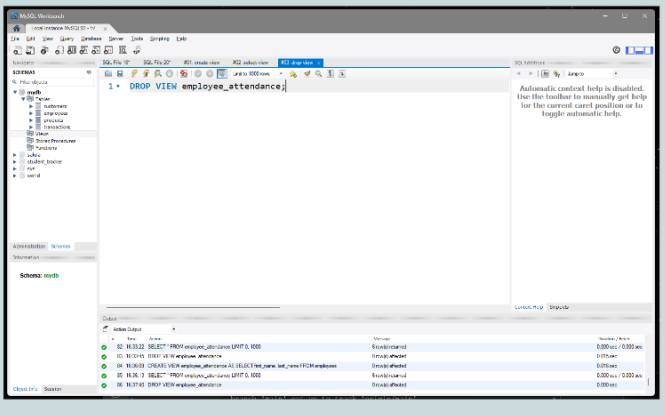
2



3



4



Index in SQL

In SQL, indexes are special lookup tables that the database search engine uses to accelerate data retrieval by quickly locating specific rows based on column values, similar to an index in a book.

- **Purpose:**

Indexes are designed to improve the speed of data retrieval operations on a table.

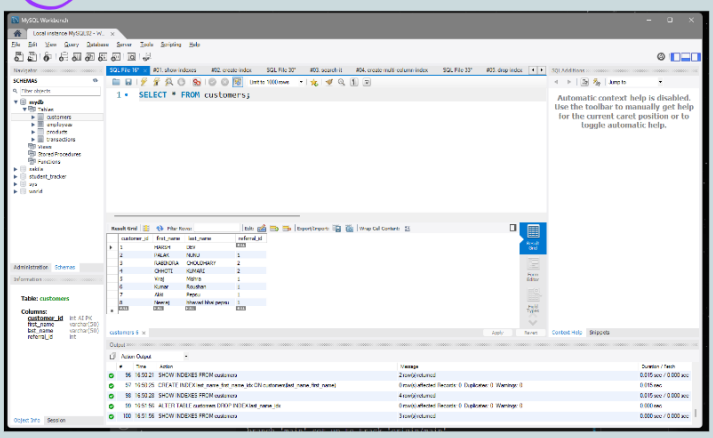
- **Types of Indexes:**
  - **Clustered Indexes:** These indexes are unique to each table and use the primary key to organize data, automatically created when the primary key is defined.
  - **Non-clustered Indexes:** These indexes are sorted references for a specific field and can be created after a table has been created and filled.

- **Benefits:**
  - **Faster Queries:** Indexes significantly reduce the time it takes to retrieve data, especially for large tables.
  - **Efficient Data Access:** They enable rapid random lookups and efficient access of ordered records.

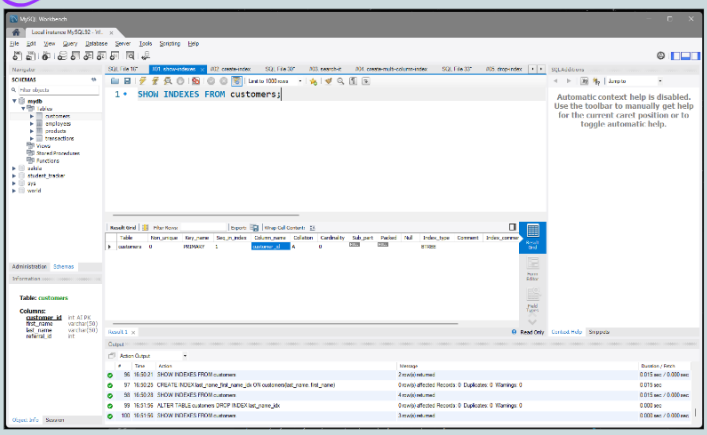
- **When to use indexes:**
  - **Frequently Searched Columns:** Create indexes on columns that are frequently used in WHERE clauses or JOIN conditions.
  - **Large Tables:** Indexes are most beneficial for large tables where full table scans would be slow.

multi-column Index

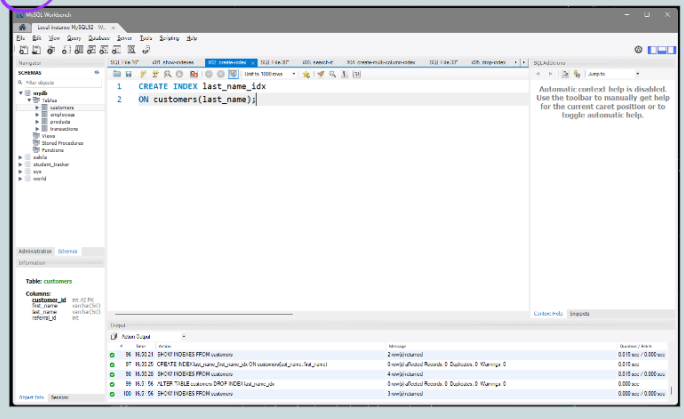
1


SELECT * FROM customers;

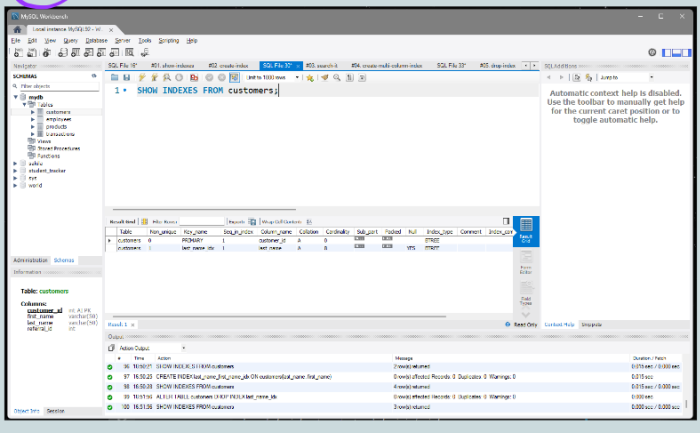
2


SHOW INDEXES FROM customers;

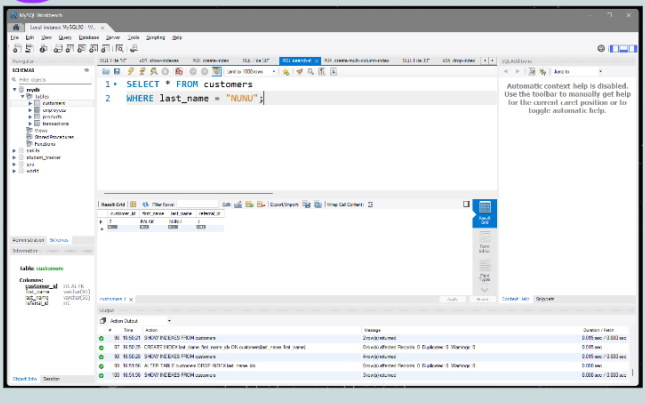
3


CREATE INDEX last_name_idx
ON customers(last_name);

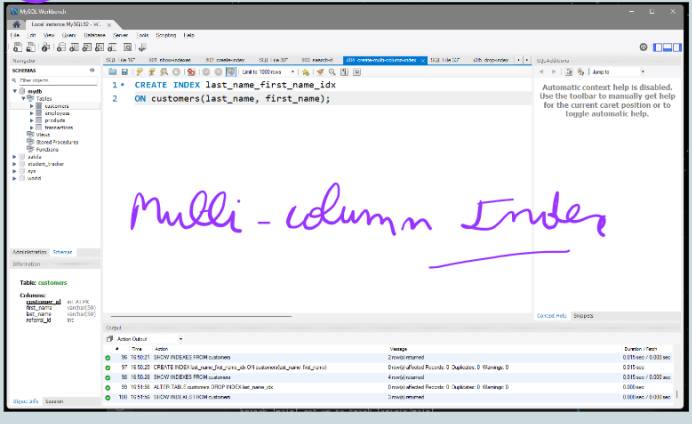
4


SHOW INDEXES FROM customers;

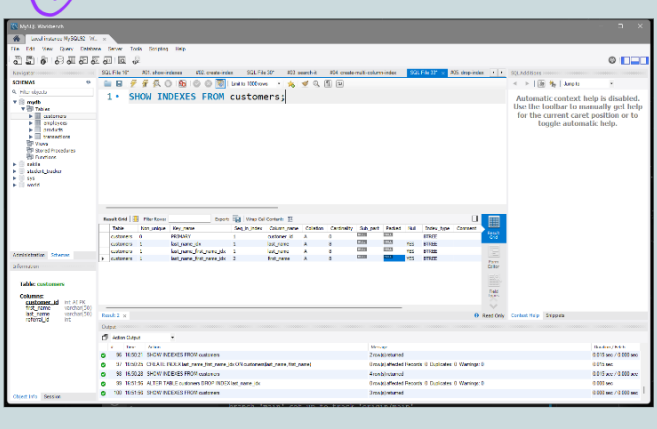
5


SELECT * FROM customers
WHERE last_name = "NUNU";

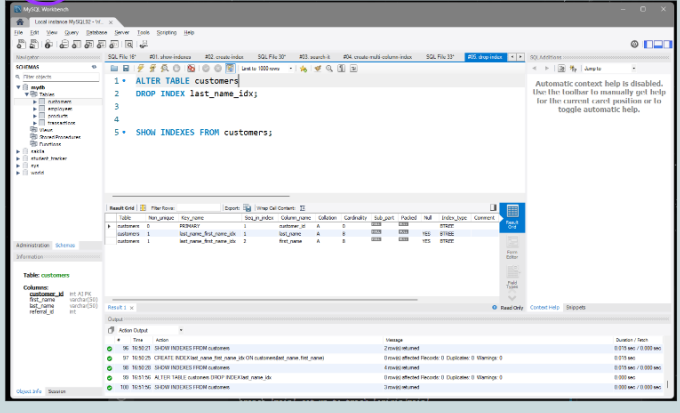
6


CREATE INDEX last_name_first_name_idx
ON customers(last_name, first_name);
Multi - column Index

7


SHOW INDEXES FROM customers;

8


ALTER TABLE customers
DROP INDEX last_name_idx;
SHOW INDEXES FROM customers;

Sub-Query

In SQL, a subquery (also called a nested query or inner query) is a query embedded within another query, allowing you to use the result of one query as input for another.

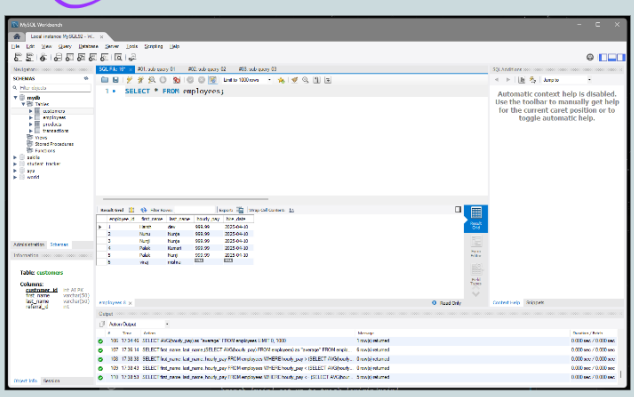
- **Purpose:**

Subqueries are used to perform complex operations that require multiple steps or logic, such as filtering data based on results from another query, aggregating data dynamically, or cross-referencing data between tables.

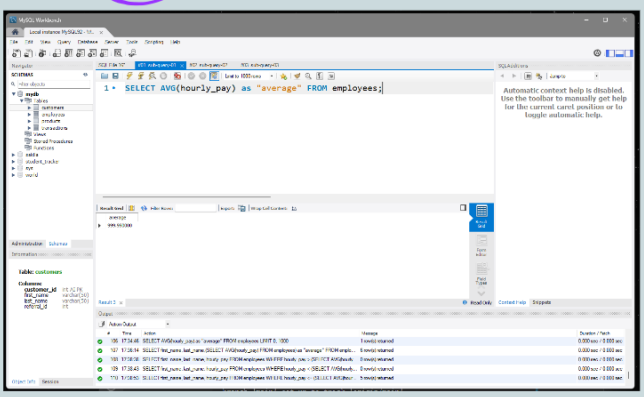
- **Types:**
- **Scalar Subqueries:** Return a single value.
- **Multi-row Subqueries:** Return multiple rows.
- **Correlated Subqueries:** Refer to values from the outer query.

- **Benefits:**
- **Simplified Complex Queries:** Subqueries can make complex queries more readable and easier to manage.
- **Improved Performance:** In some cases, subqueries can be more efficient than using multiple joins.
- **Flexibility:** Subqueries can be used to perform a wide variety of operations, making them a powerful tool for data manipulation.

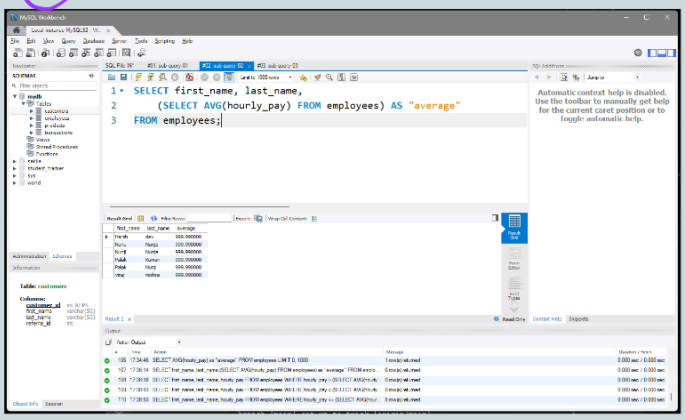
1



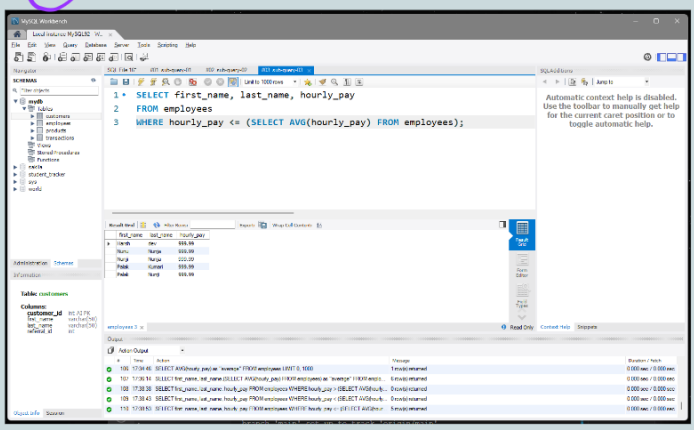
2



3



4

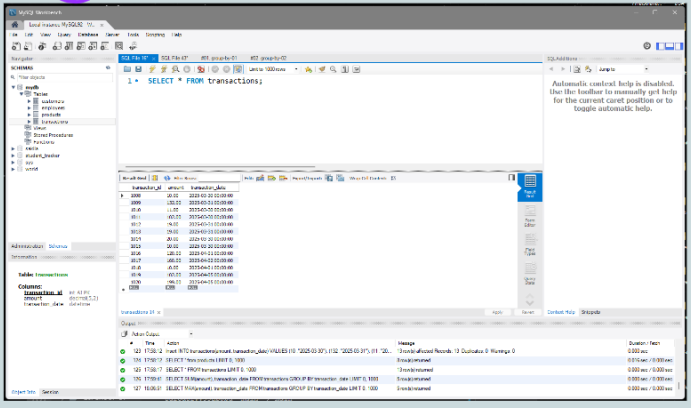


#GROUP-BY

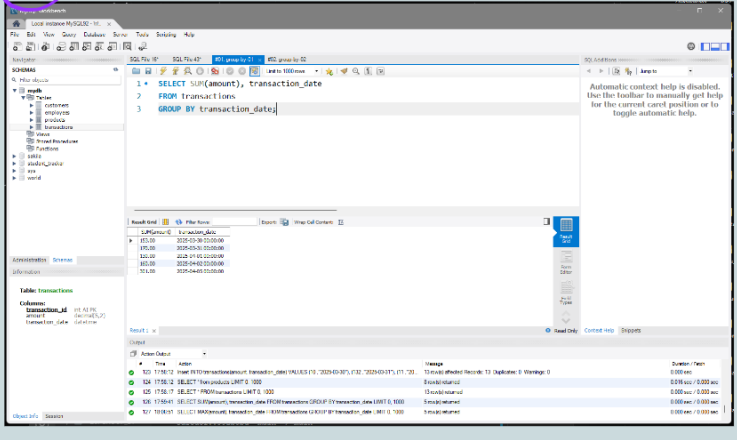
The SQL GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".
The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

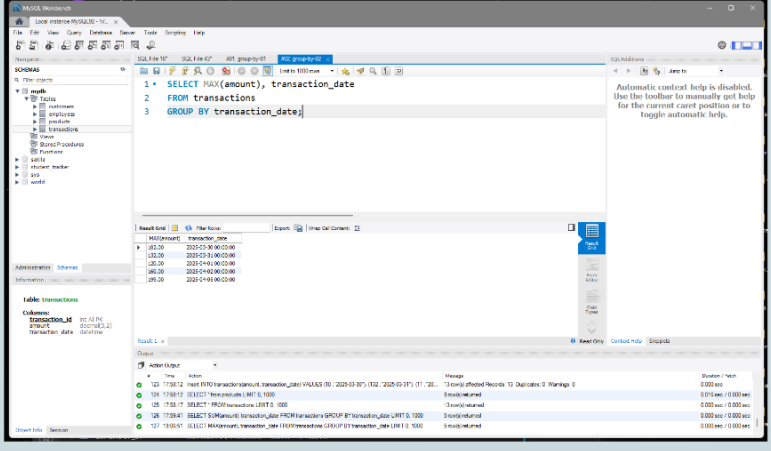
1



2



3

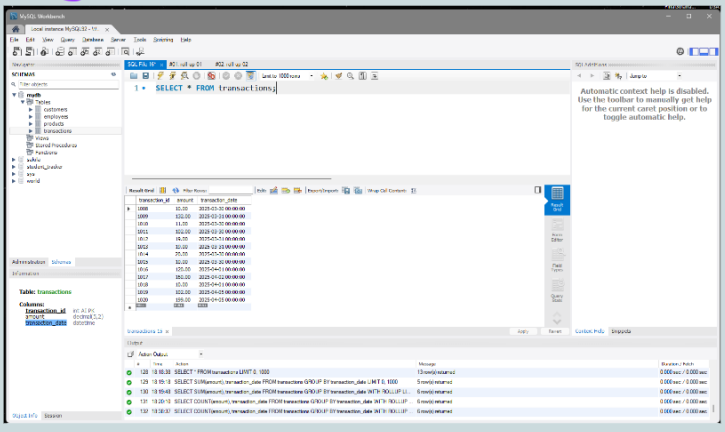


#ROLL UP

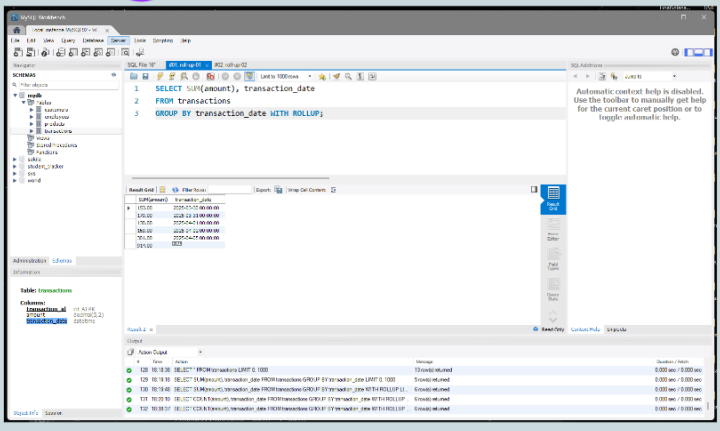
In SQL, ROLLUP is an extension of the GROUP BY clause that generates subtotals and a grand total for multiple dimensions within a dataset, allowing for analysis at different levels of granularity.

- **Purpose:** ROLLUP helps you summarize data at different levels of aggregation, providing subtotals and a final grand total in a single query.

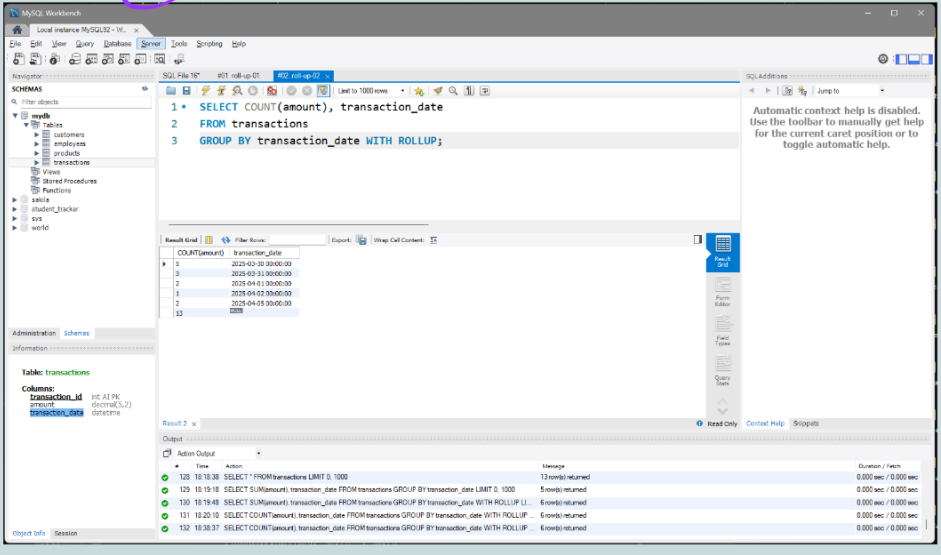
1



2



3



ON DELETE

In SQL, the ON DELETE clause, used in foreign key constraints, specifies how the database should handle records in a child table when a corresponding record in the parent table is deleted, offering options like cascading deletes, setting values to NULL, or restricting the deletion.

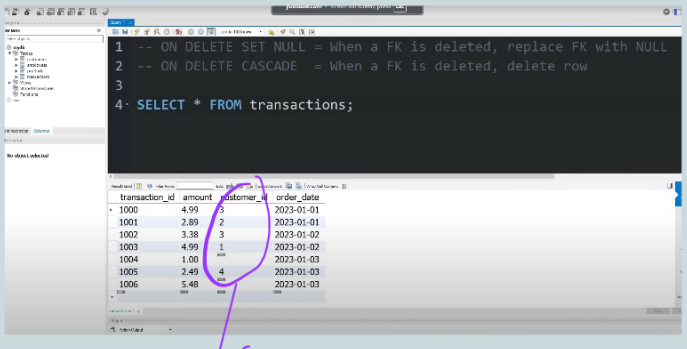
- ON DELETE CASCADE:

When a record in the parent table is deleted, all related records in the child table that reference it are automatically deleted as well.

- ON DELETE SET NULL:

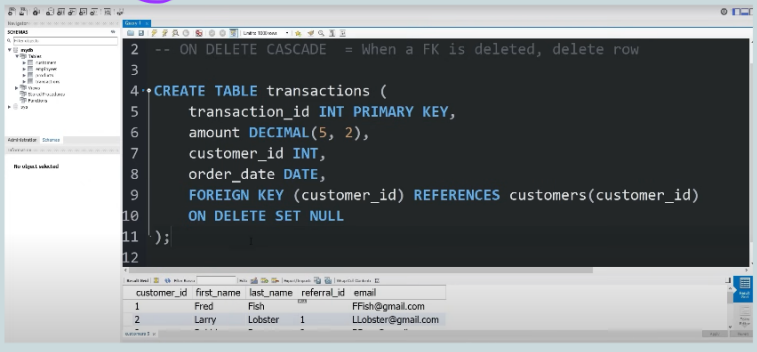
When a record in the parent table is deleted, the foreign key column in the child table is set to NULL, effectively breaking the relationship but preserving the child record.

1



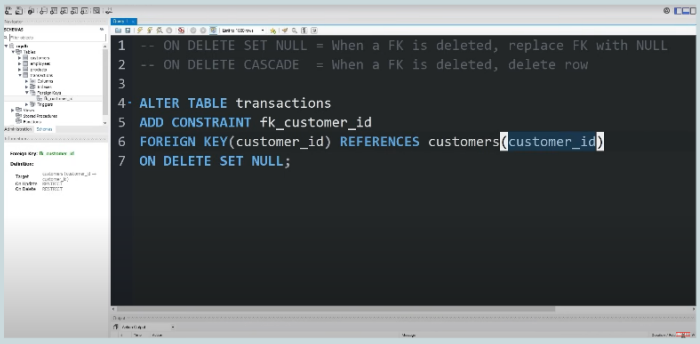
foreign key

2



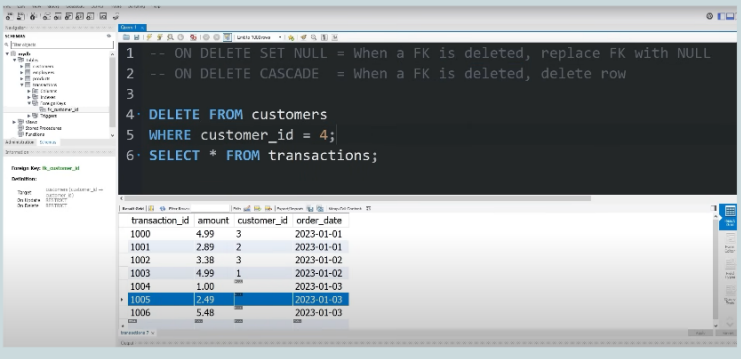
Create table with "ON DELETE SET NULL"

3

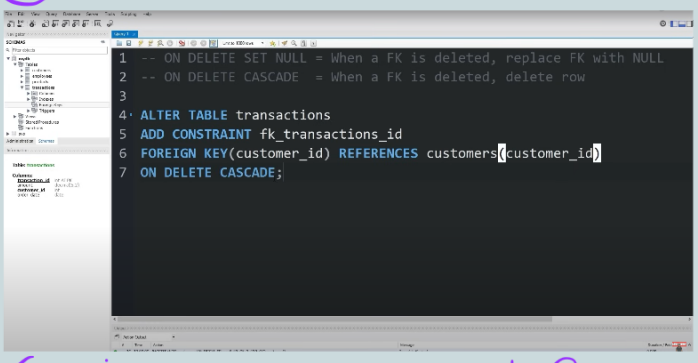


Alter table

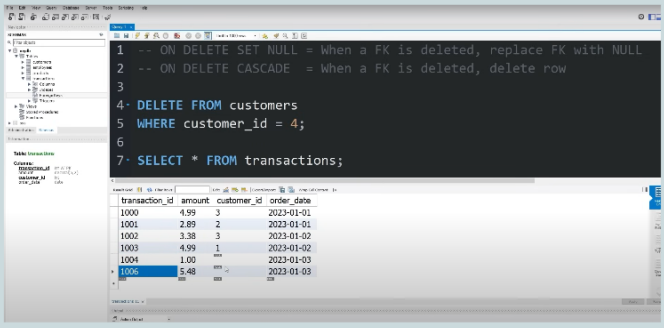
4



5



6

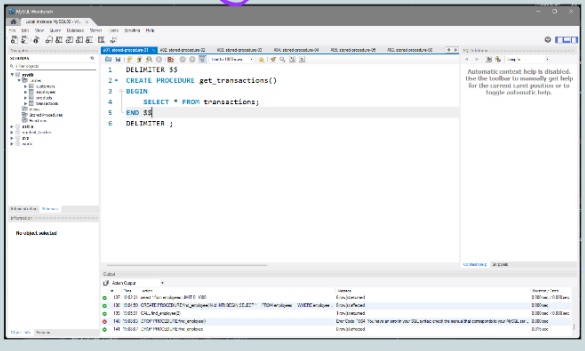


Similarly, for on delete cascade;

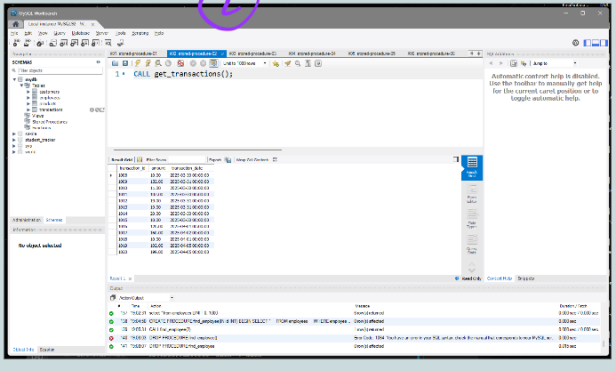
Stored Procedure

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.
So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.
You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

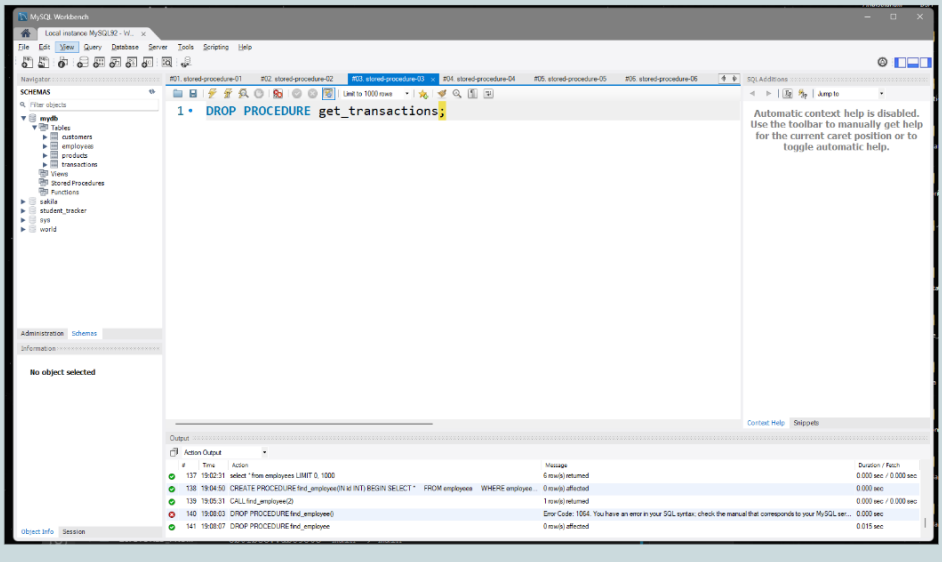
1



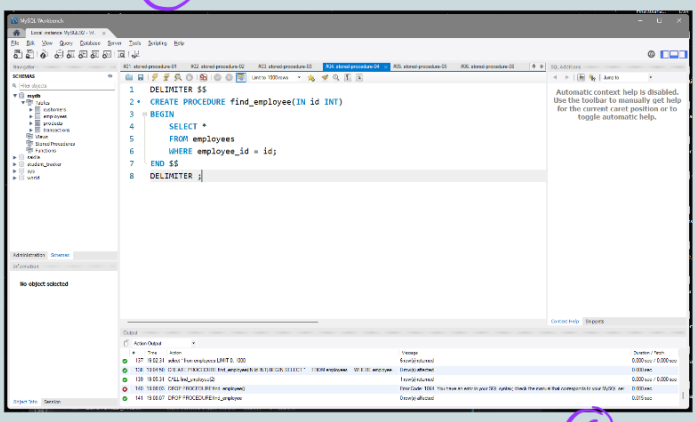
2



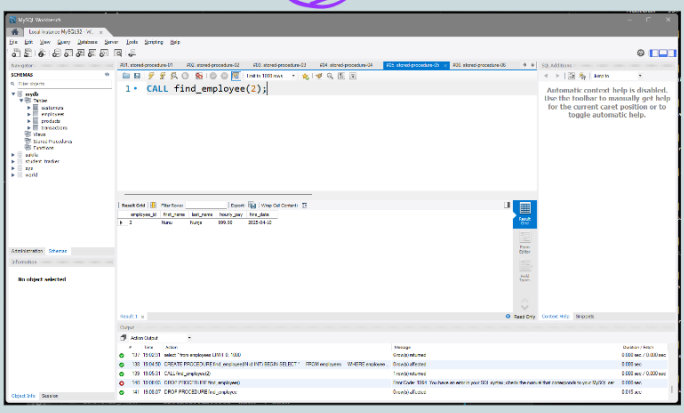
3



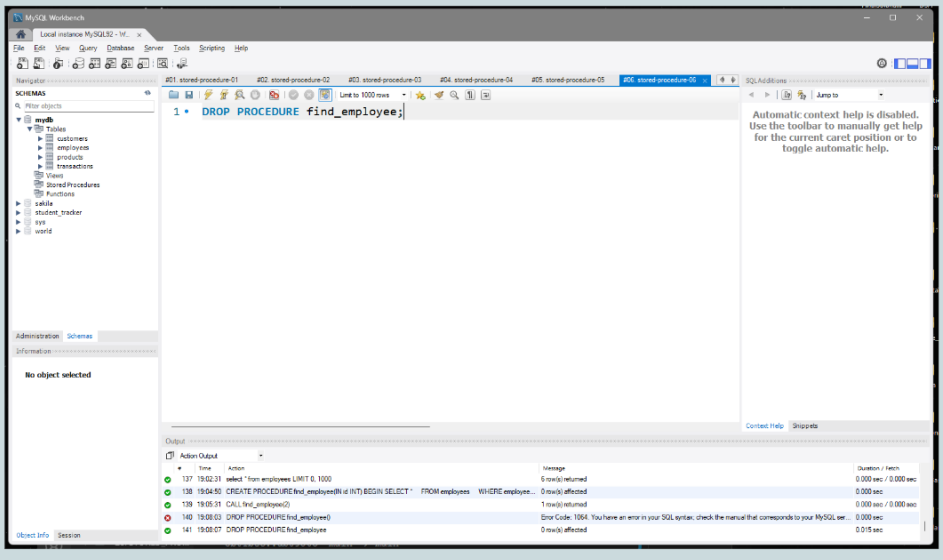
4



5



6



#TRIGGER

→In SQL, a trigger is a special type of stored procedure that automatically executes in response to specific events (like INSERT, UPDATE, or DELETE) on a table or view. They are used to maintain data integrity, enforce business rules, and automate tasks.

→What are Triggers?

- **Automatic Execution:**

Triggers are designed to run automatically when a specific event occurs on a table or view.

- **Associated with Events:**

They're triggered by events like inserting new rows, updating existing data, or deleting rows.

- **Enforcing Rules:**

Triggers can be used to enforce business rules, such as preventing certain types of data from being inserted or updated, or automatically updating related tables.

- **Maintaining Integrity:**

They play a crucial role in maintaining data integrity by ensuring that certain conditions are met before or after a data modification operation.

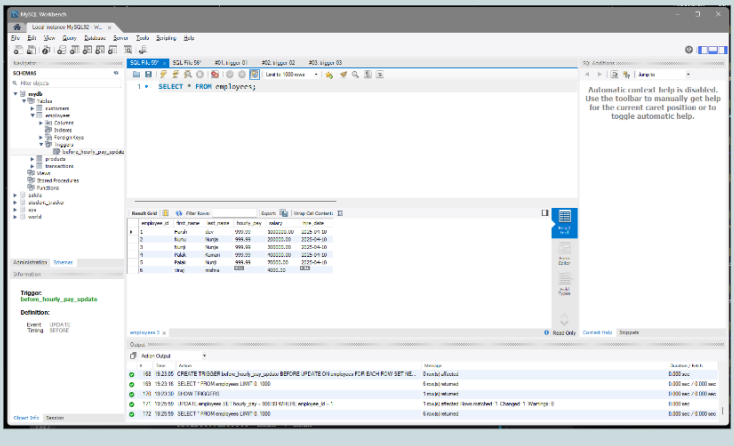
- **Automating Tasks:**

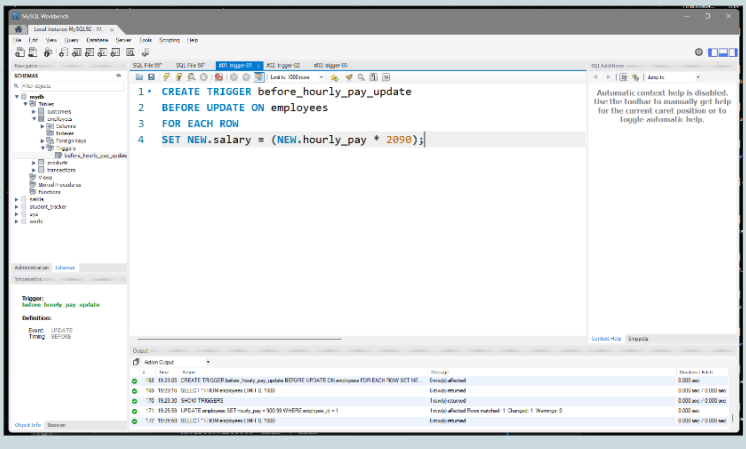
Triggers can automate tasks, such as logging changes, updating related tables, or performing calculations.

→ Types of Triggers:

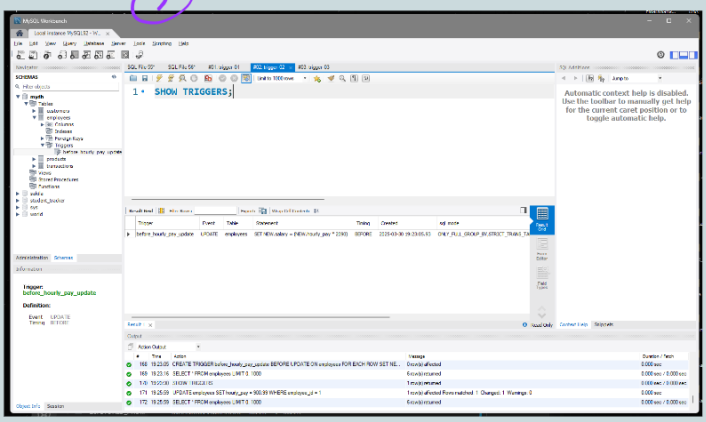
- **DML Triggers:** These are triggered by Data Manipulation Language (DML) events, such as INSERT, UPDATE, or DELETE operations on a table or view.
- **DDL Triggers:** These are triggered by Data Definition Language (DDL) events, such as CREATE, ALTER, or DROP operations on database objects.
- **Logon Triggers:** These are triggered when a user session is established.

① ②

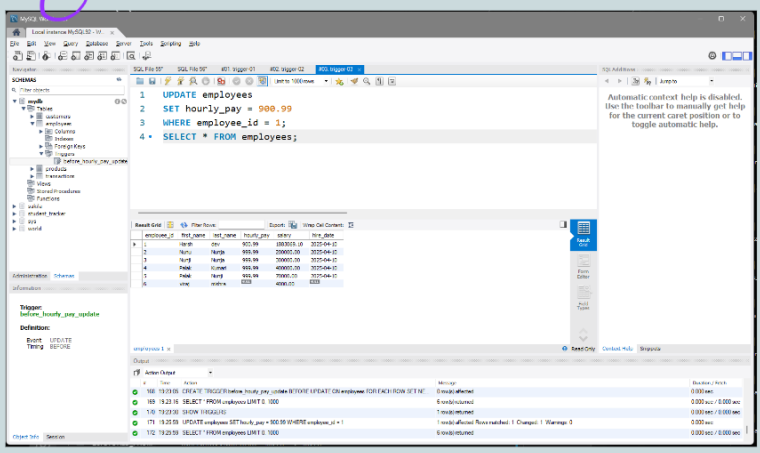




3


SHOW TRIGGERS;
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.
Trigger: before_hourly_pay_update
Definition:

4


UPDATE employees
SET hourly_pay = 900.99
WHERE employee_id = 1;
SELECT * FROM employees;
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.
Trigger: before_hourly_pay_update
Definition: