

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

MySQL Workbench

Local instance MySQL92 - W...

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

- Filter objects
- mydb
- sakila
- student_tracker
- sys
- world

Query 1 01-create-user

```
1 • CREATE DATABASE myDB;
```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Here, schema is created;

No object selected

Output

#	Time	Action	Message	Duration / Fetch
1	17:31:21	CREATE DATABASE myDB	1 row(s) affected	0.000 sec

Context Help Snippets

Object Info Session

Successful message;

- To use the currently made Database;

MySQL Workbench

Local instance MySQL92 - W...

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

- Filter objects
- mydb
- Tables
- Views
- Stored Procedures
- Functions

sakila

student_tracker

sys

world

Query 1

```
1 • USE myDB;
```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Schema: mydb

Output

#	Time	Action	Message	Duration / Fetch
1	17:31:21	CREATE DATABASE myDB	1 row(s) affected	0.000 sec
2	17:33:47	USE myDB	0 row(s) affected	0.000 sec

Context Help Snippets

Object Info Session

Successful message;

Now, we are using this database

To drop(delete) Database

The screenshot shows the MySQL Workbench interface. In the Navigator pane, under the 'Schemas' section, the 'myDB' database is listed. A yellow circle highlights this entry. A handwritten note below the Navigator says: "Here myDB is no longer visible;". In the Query Editor, the command `DROP DATABASE myDB;` is typed. In the Output pane, the log shows three actions: 1. CREATE DATABASE myDB, 2. USE myDB, and 3. DROP DATABASE myDB. The third action is circled in yellow with a handwritten note: "Successfully dropped".

```

Query 1
1 • DROP DATABASE myDB;

Output
Action Output
# Time Action
1 17:31:21 CREATE DATABASE myDB
2 17:39:47 USE myDB
3 17:35:10 DROP DATABASE myDB

```

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

The screenshot shows the MySQL Workbench interface. In the Navigator pane, under the 'Schemas' section, the 'myDB' database is expanded, showing its tables, views, stored procedures, and functions. A yellow circle highlights the 'myDB' entry. In the Query Editor, the command `ALTER DATABASE myDB READ ONLY = 1;` is typed. In the Output pane, the log shows four actions: 1. CREATE DATABASE myDB, 2. USE myDB, 3. ALTER DATABASE myDB READ ONLY = 1, and 4. ALTER DATABASE myDB READ ONLY = 1. The third and fourth actions are highlighted with a blue selection bar.

```

Query 1
1 • ALTER DATABASE myDB READ ONLY = 1;

Output
Action Output
# Time Action
1 17:39:04 CREATE DATABASE myDB
2 17:39:15 USE myDB
3 17:40:13 ALTER DATABASE myDB READ ONLY = 1
4 17:40:14 ALTER DATABASE myDB READ ONLY = 1

```

• To disable Read Only Mode

The screenshot shows the MySQL Workbench interface. In the top-left pane, the 'Schemas' tree shows the 'mydb' schema selected. In the central 'Query' editor, the following SQL command is entered:

```
ALTER DATABASE myDB READ ONLY = 0;
```

In the bottom-right pane, there is a note: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

The 'Output' pane at the bottom displays the execution log:

#	Time	Action	Message	Duration / Fetch
1	17:39:04	CREATE DATABASE myDB	1 row(s) affected	0.000 sec
2	17:39:15	USE myDB	0 row(s) affected	0.000 sec
3	17:40:13	ALTER DATABASE myDB READ ONLY = 1	1 row(s) affected	0.016 sec
4	17:40:14	ALTER DATABASE myDB READ ONLY = 1	1 row(s) affected	0.000 sec
5	17:43:11	DROP DATABASE myDB	Error Code: 3989 Schema 'mydb' is in read only mode.	0.000 sec
6	17:43:54	ALTER DATABASE myDB READ ONLY = 0	1 row(s) affected	0.000 sec

TABLE

• Creating Table

The screenshot shows the MySQL Workbench interface. In the top-left pane, the 'Schemas' tree shows the 'mydb' schema selected. In the central 'Query' editor, the following SQL command is entered:

```
CREATE TABLE employees (
    employee_id INT,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    hourly_pay DECIMAL(5,2),
    hire_date DATE
);
```

Handwritten annotations explain the data types and lengths:

- Annotations point to the 'VARCHAR(50)' and 'DECIMAL(5,2)' definitions with the text "max-length".
- An annotation points to the 'DECIMAL(5,2)' definition with the text "max-length after decimal".
- An annotation points to the 'DATE' definition with the text "max-length".
- A large handwritten note states: "(It can store up to 5 digit in total, including 2 decimal places.)"

The 'Output' pane at the bottom displays the execution log:

#	Time	Action	Message	Duration / Fetch
1	17:39:15	USE myDB	0 row(s) affected	0.000 sec
2	17:40:13	ALTER DATABASE myDB READ ONLY = 1	1 row(s) affected	0.016 sec
3	17:40:14	ALTER DATABASE myDB READ ONLY = 1	1 row(s) affected	0.000 sec
4	17:43:11	DROP DATABASE myDB	Error Code: 3989 Schema 'mydb' is in read only mode.	0.000 sec
5	17:43:54	ALTER DATABASE myDB READ ONLY = 0	1 row(s) affected	0.000 sec
6	17:51:03	CREATE TABLE employees(employee_id INT, first_name VARCHAR(50), ...)	0 row(s) affected	0.015 sec

Select Table

MySQL Workbench - Local instance MySQL52 - W...

File Edit View Query Database Server Tools Scripting Help

Navigator: mydb

Query 1: creating-table-01 selecting-table-01

1 • `SELECT * FROM employees;`

Result Grid | Filter Rows | Export | Wrap Cell Content:

employee_id	first_name	last_name	hourly_pay	hire_date
-------------	------------	-----------	------------	-----------

Output: Action Output

#	Time	Action	Message	Duration / Fetch
3	17:40:13	ALTER DATABASE myDB READ ONLY + 1	1 row(s) affected	0.015 sec
4	17:40:14	ALTER DATABASE myDB READ ONLY - 1	1 row(s) affected	0.000 sec
5	17:43:11	DROP DATABASE myDB	Error Code: 3999 Schema 'mydb' is in read only mode.	0.000 sec
6	17:43:54	ALTER DATABASE myDB READ ONLY + 0	1 row(s) affected	0.000 sec
7	17:51:00	CREATE TABLE employee(employee_id INT, first_name VARCHAR(50), last_name VARCHAR(50), hourly_pay DECIMAL(5,2), hire_date DATE)	0 row(s) affected	0.015 sec
8	17:53:39	SELECT * FROM employees LIMIT 0, 1000	0 row(s) returned	0.015 sec / 0.000 sec

Rename Table

MySQL Workbench - Local instance MySQL52 - W...

File Edit View Query Database Server Tools Scripting Help

Navigator: mydb

Query 1: creating-table-01 selecting-table-01 rename-table-01

1 • `RENAME TABLE employees TO workers;`

Output: Action Output

#	Time	Action	Message	Duration / Fetch
11	17:55:42	RENAME TABLE employees TO workers	0 row(s) affected	0.016 sec

Drop Table

MySQL Workbench - Local instance MySQL52 - W...

File Edit View Query Database Server Tools Scripting Help

Navigator: mydb

Query 1: creating-table-01 selecting-table-01 rename-table-01 drop-table-01

1 • `DROP TABLE employees;`

Output: Action Output

#	Time	Action	Message	Duration / Fetch
13	17:56:48	SELECT * FROM mydb.employees LIMIT 0, 1000	Error Code: 1146 Table 'mydb.employees' doesn't exist	0.000 sec
14	17:56:59	SELECT * FROM mydb.workers LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
15	17:56:01	SELECT * FROM mydb.workers LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
16	17:56:05	RENAME TABLE employees TO workers	Error Code: 1055 Table 'workers' already exists	0.000 sec
17	17:57:11	RENAME TABLE workers TO employees	0 row(s) affected	0.016 sec
18	17:57:50	DROP TABLE employees	0 row(s) affected	0.000 sec

• ALTER Table (Add Column)

MySQL Workbench Screenshot:

- Schema:** mydb
- Table:** employees
- SQL:**

```
1 • ALTER TABLE employees
2 ADD phone_number VARCHAR(15);
```
- Output:**

#	Time	Action	Message	Duration / Fetch
16	17:56:05	RENAME TABLE employees TO workers	Error Code: 1050 Table 'workers' already exists	0.000 sec
17	17:57:11	RENAME TABLE workers TO employees	0 rows(a) affected	0.01 sec
18	17:57:50	DROP TABLE employees	0 rows(a) affected	0.000 sec
19	17:59:22	CREATE TABLE employee(employee_id INT, first_name VARCHAR(50), last_name VARCHAR(50), hourly_pay DECIMAL(5,2), hire_date DATE, phone_number VARCHAR(15))	0 rows(a) affected	0.000 sec
20	17:59:32	SELECT * FROM employees LIMIT 0, 1000	0 rows(a) returned	0.016 sec / 0.000 sec
21	18:01:05	ALTER TABLE employees ADD phone_number VARCHAR(15)	0 rows(a) affected Records: 0 Duplicates: 0 Warnings: 0	0.031 sec

MySQL Workbench Screenshot:

- Schema:** mydb
- Table:** employees
- SQL:**

```
1 • SELECT * FROM mydb.employees;
```
- Output:**

employee_id	first_name	last_name	hourly_pay	hire_date	phone_number
1	John	Doe	12.50	2010-01-01	123-4567890
2	Susan	Smith	14.50	2010-01-01	987-6543210
3	Mark	Williams	16.50	2010-01-01	543-2109870
4	Linda	Johnson	18.50	2010-01-01	432-1098760
5	David	Miller	20.50	2010-01-01	321-0987650
6	Karen	Allen	22.50	2010-01-01	210-9876540
7	James	Wilson	24.50	2010-01-01	123-4567890
8	Sarah	Clark	26.50	2010-01-01	987-6543210
9	Robert	Anderson	28.50	2010-01-01	543-2109870
10	Mary	Howard	30.50	2010-01-01	432-1098760
11	Jeffrey	Miller	32.50	2010-01-01	321-0987650
12	Elizabeth	Allen	34.50	2010-01-01	210-9876540
13	David	Howard	36.50	2010-01-01	123-4567890
14	Sarah	Miller	38.50	2010-01-01	987-6543210
15	Robert	Allen	40.50	2010-01-01	543-2109870
16	Jeffrey	Howard	42.50	2010-01-01	432-1098760
17	Elizabeth	Miller	44.50	2010-01-01	321-0987650
18	David	Allen	46.50	2010-01-01	210-9876540
19	Sarah	Howard	48.50	2010-01-01	123-4567890
20	Robert	Miller	50.50	2010-01-01	987-6543210
21	Jeffrey	Allen	52.50	2010-01-01	543-2109870
22	Elizabeth	Howard	54.50	2010-01-01	432-1098760
23	David	Miller	56.50	2010-01-01	321-0987650
24	Sarah	Allen	58.50	2010-01-01	210-9876540
25	Robert	Howard	60.50	2010-01-01	123-4567890
26	Jeffrey	Miller	62.50	2010-01-01	987-6543210
27	Elizabeth	Allen	64.50	2010-01-01	543-2109870
28	David	Howard	66.50	2010-01-01	432-1098760
29	Sarah	Miller	68.50	2010-01-01	321-0987650
30	Robert	Allen	70.50	2010-01-01	210-9876540
31	Jeffrey	Howard	72.50	2010-01-01	123-4567890
32	Elizabeth	Miller	74.50	2010-01-01	987-6543210
33	David	Allen	76.50	2010-01-01	543-2109870
34	Sarah	Howard	78.50	2010-01-01	432-1098760
35	Robert	Miller	80.50	2010-01-01	321-0987650
36	Jeffrey	Allen	82.50	2010-01-01	210-9876540
37	Elizabeth	Howard	84.50	2010-01-01	123-4567890
38	David	Miller	86.50	2010-01-01	987-6543210
39	Sarah	Allen	88.50	2010-01-01	543-2109870
40	Robert	Howard	90.50	2010-01-01	432-1098760
41	Jeffrey	Miller	92.50	2010-01-01	321-0987650
42	Elizabeth	Allen	94.50	2010-01-01	210-9876540
43	David	Howard	96.50	2010-01-01	123-4567890
44	Sarah	Miller	98.50	2010-01-01	987-6543210
45	Robert	Allen	100.50	2010-01-01	543-2109870
46	Jeffrey	Howard	102.50	2010-01-01	432-1098760
47	Elizabeth	Miller	104.50	2010-01-01	321-0987650
48	David	Allen	106.50	2010-01-01	210-9876540
49	Sarah	Howard	108.50	2010-01-01	123-4567890
50	Robert	Miller	110.50	2010-01-01	987-6543210
51	Jeffrey	Allen	112.50	2010-01-01	543-2109870
52	Elizabeth	Howard	114.50	2010-01-01	432-1098760
53	David	Miller	116.50	2010-01-01	321-0987650
54	Sarah	Allen	118.50	2010-01-01	210-9876540
55	Robert	Howard	120.50	2010-01-01	123-4567890
56	Jeffrey	Miller	122.50	2010-01-01	987-6543210
57	Elizabeth	Allen	124.50	2010-01-01	543-2109870
58	David	Howard	126.50	2010-01-01	432-1098760
59	Sarah	Miller	128.50	2010-01-01	321-0987650
60	Robert	Allen	130.50	2010-01-01	210-9876540
61	Jeffrey	Howard	132.50	2010-01-01	123-4567890
62	Elizabeth	Miller	134.50	2010-01-01	987-6543210
63	David	Allen	136.50	2010-01-01	543-2109870
64	Sarah	Howard	138.50	2010-01-01	432-1098760
65	Robert	Miller	140.50	2010-01-01	321-0987650
66	Jeffrey	Allen	142.50	2010-01-01	210-9876540
67	Elizabeth	Howard	144.50	2010-01-01	123-4567890
68	David	Miller	146.50	2010-01-01	987-6543210
69	Sarah	Allen	148.50	2010-01-01	543-2109870
70	Robert	Howard	150.50	2010-01-01	432-1098760
71	Jeffrey	Miller	152.50	2010-01-01	321-0987650
72	Elizabeth	Allen	154.50	2010-01-01	210-9876540
73	David	Howard	156.50	2010-01-01	123-4567890
74	Sarah	Miller	158.50	2010-01-01	987-6543210
75	Robert	Allen	160.50	2010-01-01	543-2109870
76	Jeffrey	Howard	162.50	2010-01-01	432-1098760
77	Elizabeth	Miller	164.50	2010-01-01	321-0987650
78	David	Allen	166.50	2010-01-01	210-9876540
79	Sarah	Howard	168.50	2010-01-01	123-4567890
80	Robert	Miller	170.50	2010-01-01	987-6543210
81	Jeffrey	Allen	172.50	2010-01-01	543-2109870
82	Elizabeth	Howard	174.50	2010-01-01	432-1098760
83	David	Miller	176.50	2010-01-01	321-0987650
84	Sarah	Allen	178.50	2010-01-01	210-9876540
85	Robert	Howard	180.50	2010-01-01	123-4567890
86	Jeffrey	Miller	182.50	2010-01-01	987-6543210
87	Elizabeth	Allen	184.50	2010-01-01	543-2109870
88	David	Howard	186.50	2010-01-01	432-1098760
89	Sarah	Miller	188.50	2010-01-01	321-0987650
90	Robert	Allen	190.50	2010-01-01	210-9876540
91	Jeffrey	Howard	192.50	2010-01-01	123-4567890
92	Elizabeth	Miller	194.50	2010-01-01	987-6543210
93	David	Allen	196.50	2010-01-01	543-2109870
94	Sarah	Howard	198.50	2010-01-01	432-1098760
95	Robert	Miller	200.50	2010-01-01	321-0987650
96	Jeffrey	Allen	202.50	2010-01-01	210-9876540
97	Elizabeth	Howard	204.50	2010-01-01	123-4567890
98	David	Miller	206.50	2010-01-01	987-6543210
99	Sarah	Allen	208.50	2010-01-01	543-2109870
100	Robert	Howard	210.50	2010-01-01	432-1098760
101	Jeffrey	Miller	212.50	2010-01-01	321-0987650
102	Elizabeth	Allen	214.50	2010-01-01	210-9876540
103	David	Howard	216.50	2010-01-01	123-4567890
104	Sarah	Miller	218.50	2010-01-01	987-6543210
105	Robert	Allen	220.50	2010-01-01	543-2109870
106	Jeffrey	Howard	222.50	2010-01-01	432-1098760
107	Elizabeth	Miller	224.50	2010-01-01	321-0987650
108	David	Allen	226.50	2010-01-01	210-9876540
109	Sarah	Howard	228.50	2010-01-01	123-4567890
110	Robert	Miller	230.50	2010-01-01	987-6543210
111	Jeffrey	Allen	232.50	2010-01-01	543-2109870
112	Elizabeth	Howard	234.50	2010-01-01	432-1098760
113	David	Miller	236.50	2010-01-01	321-0987650
114	Sarah	Allen	238.50	2010-01-01	210-9876540
115	Robert	Howard	240.50	2010-01-01	123-4567890
116	Jeffrey	Miller	242.50	2010-01-01	987-6543210
117	Elizabeth	Allen	244.50	2010-01-01	543-2109870
118	David	Howard	246.50	2010-01-01	432-1098760
119	Sarah	Miller	248.50	2010-01-01	321-0987650
120	Robert	Allen	250.50	2010-01-01	210-9876540
121	Jeffrey	Howard	252.50	2010-01-01	123-4567890
122	Elizabeth	Miller	254.50	2010-01-01	987-6543210
123	David	Allen	256.50	2010-01-01	543-2109870
124	Sarah	Howard	258.50	2010-01-01	432-1098760
125	Robert	Miller	260.50	2010-01-01	321-0987650
126	Jeffrey	Allen	262.50	2010-01-01	210-9876540
127	Elizabeth	Howard	264.50	2010-01-01	123-4567890
128	David	Miller	266.50	2010-01-01	987-6543210
129	Sarah	Allen	268.50	2010-01-01	543-2109870
130	Robert	Howard	270.50	2010-01-01	432-1098760
131	Jeffrey	Miller	272.50	2010-01-01	321-0987650
132	Elizabeth	Allen	274.50	2010-01-01	210-9876540
133	David	Howard	276.50	2010-01-01	123-4567890
134	Sarah	Miller	278.50	2010-01-01	987-6543210
135	Robert	Allen	280.50	2010-01-01	543-2109870
136	Jeffrey	Howard	282.50	2010-01-01	432-1098760
137	Elizabeth	Miller	284.50	2010-01-01	321-0987650
138	David	Allen	286.50	2010-01-01	210-9876540
139	Sarah	Howard	288.50	2010-01-01	123-4567890
140	Robert	Miller	290.50	2010-01-01	987-6543210
141	Jeffrey	Allen	292.50	2010-01-01	543-2109870
142	Elizabeth	Howard	294.50	2010-01-01	432-1098760
143	David	Miller	296.50	2010-01-01	321-0987650
144	Sarah	Allen	298.50	2010-01-01	210-9876540
145	Robert	Howard	300.50	2010-01-01	123-4567890
146	Jeffrey	Miller	302.50	2010-01-01	987-6543210
147	Elizabeth	Allen	304.50	2010-01-01	543-2109870
148	David	Howard	306.50	2010-01-01	432-1098760
149	Sarah	Miller	308.50	2010-01-01	321-0987650
150	Robert	Allen	310.50	2010-01-01	210-9876540
151	Jeffrey	Howard	312.50	2010-01-01	123-4567890
152	Elizabeth	Miller	314.50	2010-01-01	987-6543210
153	David	Allen	316.50	2010-01-01	543-2109870
154	Sarah	Howard	318.50	2010-01-01	432-1098760
155	Robert	Miller	320.50	2010-01-01	321-0987650
156	Jeffrey	Allen			

• ALTER Table (modify column properties)

The screenshot shows the MySQL Workbench interface with the SQL tab active. The query window contains the following code:

```
1 • ALTER TABLE employees
2 MODIFY COLUMN email VARCHAR(100);
```

The results pane shows the execution of the query with several log entries. A purple arrow points from the handwritten note "email max-size is changed." to the "email" column definition in the table schema.

email max-size is changed.

• ALTER Table (Move Column)

The screenshot shows the MySQL Workbench interface with the SQL tab active. The query window contains the following code:

```
1 • SELECT * FROM mydb.employees;
```

The results pane shows the output of the query. A purple arrow points from the handwritten note "email is here; let's move this" to the "email" column in the result grid.

The screenshot shows the MySQL Workbench interface with the SQL tab active. The query window contains the following code:

```
1 • ALTER TABLE employees
2 MODIFY email VARCHAR(100)
3 AFTER last_name;
4
5 • SELECT * FROM employees;
```

The results pane shows the execution of the query. A purple arrow points from the handwritten note "This will show table;" to the result grid, which displays the modified table structure.

• ALTER Table (move column to first)

The screenshot shows the MySQL Workbench interface with the SQL tab active. The query window contains the following code:

```
1 • ALTER TABLE employees
2 MODIFY email VARCHAR(100)
3 FIRST;
4
5 • SELECT * FROM employees;
```

The results pane shows the execution of the query. A purple arrow points from the handwritten note "Email is now at first position" to the result grid, which displays the modified table structure with the email column moved to the first position.

• ALTER Table (Drop Column)

MySQL Workbench - Local instance MySQL82 - W...

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS mydb

Tables employees

Views

Stored Procedures

Functions

sakila

student_tracker

sys

world

SQL File 13*

```
1 • ALTER TABLE employees
2 DROP COLUMN email;
3
4 • SELECT * FROM employees;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

employee_id	first_name	last_name	hourly_pay	hire_date

Administration Schemas

Information

Table: employees

Columns:

- employee_id int
- first_name varchar(50)
- last_name varchar(50)
- hourly_pay decimal(5,2)
- hire_date date

employees 1 x

Action Output

#	Time	Action	Message	Duration / Fetch
32	18:13:01	ALTER TABLE employees MODIFY email VARCHAR(100) AFTER last_name	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.032 sec
33	18:13:01	SELECT * FROM employees LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
34	18:14:48	ALTER TABLE employees MODIFY email VARCHAR(100) FIRST	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.016 sec
35	18:14:48	SELECT * FROM employees LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
36	18:16:51	ALTER TABLE employees DROP COLUMN email	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.031 sec
37	18:16:51	SELECT * FROM employees LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

Object Info Session

Rows

• Insert Rows

MySQL Workbench - Local instance MySQL82 - W...

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS mydb

Tables employees

Views

Stored Procedures

Functions

sakila

student_tracker

sys

world

SQL File 15*

```
1 • Insert INTO employees
2 VALUE(1,"Harsh","dev",999.99,"2025-04-10");
3
4 • SELECT * from employees;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

employee_id	first_name	last_name	hourly_pay	hire_date
1	Harsh	dev	999.99	2025-04-10

Administration Schemas

Information

Table: employees

Columns:

- employee_id int
- first_name varchar(50)
- last_name varchar(50)
- hourly_pay decimal(5,2)
- hire_date date

employees 1 x

Action Output

#	Time	Action	Message	Duration / Fetch
36	18:16:51	ALTER TABLE employee DROP COLUMN email	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.031 sec
37	18:16:51	SELECT * FROM employees LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
38	18:29:54	99999	Error Code: 1054. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '99999' at line 1	0.000 sec
39	18:29:53	Insert INTO employees VALUE(1,"Harsh","dev",999.99,"2025-04-10")	Error Code: 1254. Out of range value for column 'hourly_pay' at row 1	0.000 sec
40	18:29:54	Insert INTO employees VALUE(1,"Harsh","dev",999.99,"2025-04-10")	1 row(s) affected	0.016 sec
41	18:29:54	SELECT * FROM employees LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

Object Info Session

• Insert multiple Rows

MySQL Workbench - Local instance MySQL82 - W...

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS mydb

Tables employees

Views

Stored Procedures

Functions

sakila

student_tracker

sys

world

SQL File 16*

```
1 • Insert INTO employees
2 VALUE (2,"Nunu","Nunja",999.99,"2025-04-10"),
3 (3,"Nunji","Nunja",999.99,"2025-04-10"),
4 (4,"Palak","Kumari",999.99,"2025-04-10"),
5 (5,"Palak","Nunji",999.99,"2025-04-10");
6
7 • SELECT * from employees;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

employee_id	first_name	last_name	hourly_pay	hire_date
1	Harsh	dev	999.99	2025-04-10
2	Nunu	Nunja	999.99	2025-04-10
3	Nunji	Nunja	999.99	2025-04-10
4	Palak	Kumari	999.99	2025-04-10
5	Palak	Nunji	999.99	2025-04-10

Administration Schemas

Information

Table: employees

Columns:

- employee_id int
- first_name varchar(50)
- last_name varchar(50)
- hourly_pay decimal(5,2)
- hire_date date

employees 1 x

Action Output

#	Time	Action	Message	Duration / Fetch
36	18:29:54	99999	Error Code: 1054. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '99999' at line 1	0.000 sec
37	18:29:54	Insert INTO employees VALUE(1,"Harsh","dev",999.99,"2025-04-10")	Error Code: 1254. Out of range value for column 'hourly_pay' at row 1	0.000 sec
38	18:29:54	Insert INTO employees VALUE(1,"Nunu","Nunja",999.99,"2025-04-10")	1 row(s) affected	0.076 sec
39	18:29:54	Insert INTO employees VALUE(1,"Nunji","Nunja",999.99,"2025-04-10")	1 row(s) affected	0.000 sec / 0.000 sec
40	18:29:54	Insert INTO employees VALUE(1,"Palak","Kumari",999.99,"2025-04-10")	1 row(s) affected	0.000 sec
41	18:29:54	Insert INTO employees VALUE(1,"Palak","Nunji",999.99,"2025-04-10")	1 row(s) affected	0.000 sec
42	18:29:55	SELECT * FROM employees LIMIT 0, 1000	5 rows(s) affected	0.000 sec / 0.000 sec
43	18:29:55	SELECT * FROM employees LIMIT 0, 1000	5 rows(s) returned	0.000 sec / 0.000 sec

Object Info Session

Insert at desired column in Table

The screenshot shows the MySQL Workbench interface with a query editor containing the following SQL code:

```

1 • INSERT INTO employees(employee_id,first_name, last_name)
2 VALUES (6, "viraj", "mishra");
3
4 • SELECT * FROM employees;
    
```

The results grid displays the following data:

employee_id	first_name	last_name	hourly_rate	hire_date
1	Henry	dev	999.99	2025-04-10
2	Nuru	Yousaf	999.99	2025-04-10
3	Nuru	Marti	999.99	2025-04-10
4	Pakal	Kunsel	999.99	2025-04-10
5	Pakal	Hung	999.99	2025-04-10
6	viraj	mishra	data	data

The status bar indicates an error: "Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'data' at line 6".

Select

Select full table

The screenshot shows the MySQL Workbench interface with a query editor containing the following SQL code:

```

1 • SELECT * FROM employees;
    
```

The results grid displays the following data:

employee_id	first_name	last_name	hourly_rate	hire_date
1	Henry	dev	999.99	2025-04-10
2	Nuru	Yousaf	999.99	2025-04-10
3	Nuru	Marti	999.99	2025-04-10
4	Pakal	Kunsel	999.99	2025-04-10
5	Pakal	Hung	999.99	2025-04-10
6	viraj	mishra	data	data

The status bar indicates a success message: "Query OK, 6 rows affected, 6 rows returned, 0.000 sec".

Select specific column

The screenshot shows the MySQL Workbench interface with a query editor containing the following SQL code:

```

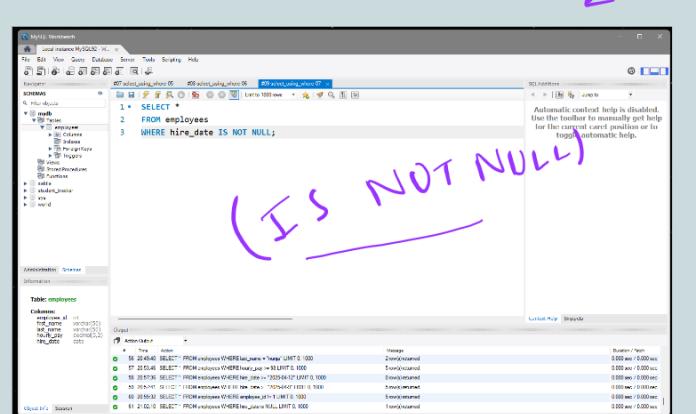
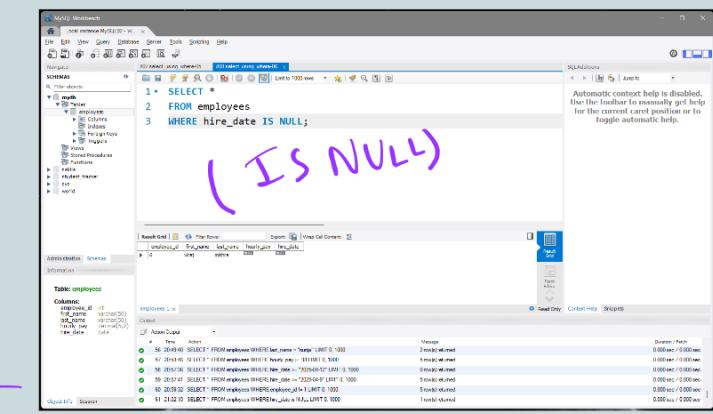
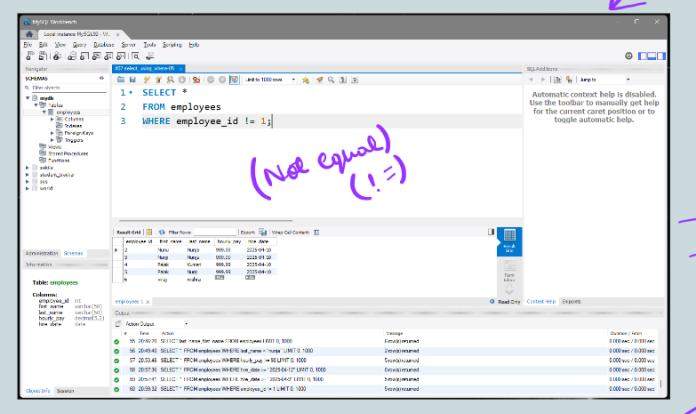
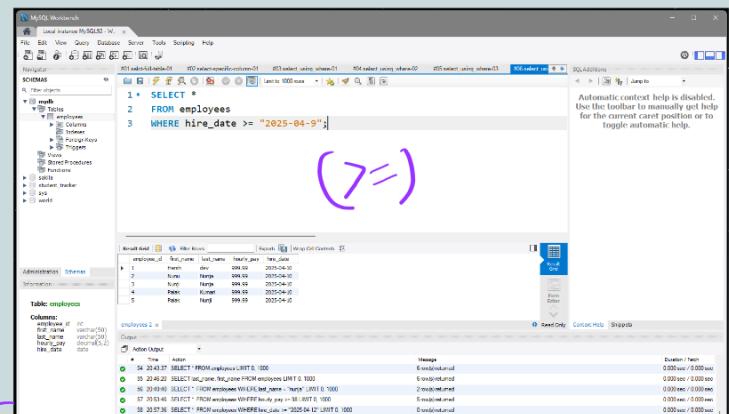
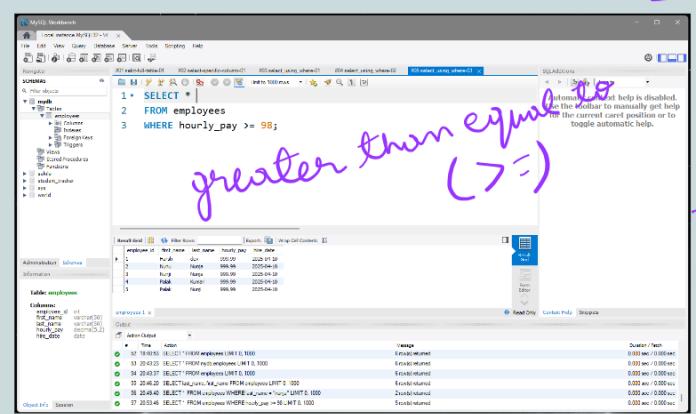
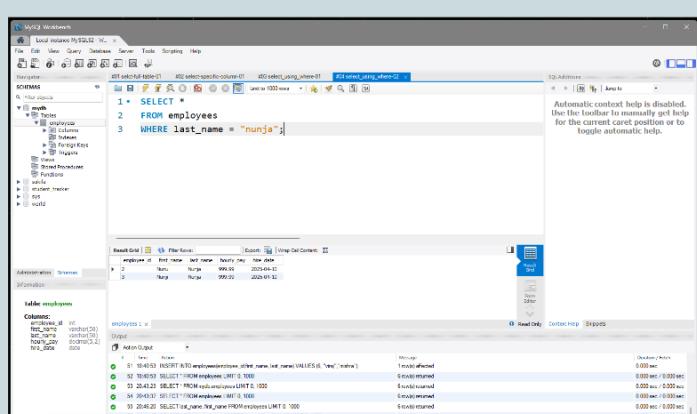
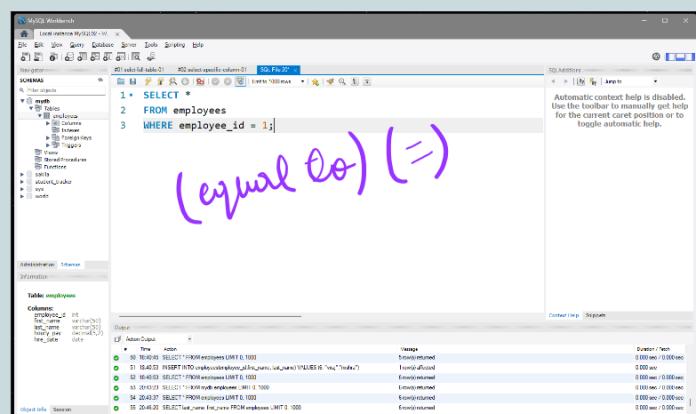
1 • SELECT last_name, first_name
2 FROM employees;
    
```

The results grid displays the following data:

last_name	first_name
Nuru	Nuru
Nuru	Marti
Kunsel	Pakal
Hung	Pakal
mishra	viraj

The status bar indicates a success message: "Query OK, 5 rows affected, 5 rows returned, 0.000 sec".

Select with where keyword



Update and Delete

• Update a single column

The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the schema `mydb` with tables `employees`, `salaries`, and `dept_emp`.
- SQL Editor:** Contains the following SQL code:

```
1 • UPDATE employees
2   SET hourly_pay = 10.25
3   WHERE employee_id = 6;
4
5 • SELECT * FROM employees;
```
- Result Grid:** Displays the `employees` table with 6 rows. The `hourly_pay` column is updated to 10.25 for employee ID 6.

employee_id	first_name	last_name	hourly_pay	hire_date
1	Kimi	Naruto	999.99	2025-04-10
2	Nuru	Nurje	999.99	2025-04-10
3	Nuru	Nurje	999.99	2025-04-10
4	Pak	Kusari	999.99	2025-04-10
5	Pak	Nurje	999.99	2025-04-10
6	Vraj	mishra	10.25	2024-03-31
- Output:** Shows the execution log with 8 entries, all completed successfully.

• Update multiple column

The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the schema `mydb` with tables `employees`, `salaries`, and `dept_emp`.
- SQL Editor:** Contains the following SQL code:

```
1 • UPDATE employees
2   SET hourly_pay = 10.25,
3       hire_date = "2024-03-31"
4   WHERE employee_id = 6;
5
6 • SELECT * FROM employees;
```
- Result Grid:** Displays the `employees` table with 6 rows. The `hourly_pay` column is updated to 10.25 and the `hire_date` column is updated to '2024-03-31' for employee ID 6.

employee_id	first_name	last_name	hourly_pay	hire_date
1	Kimi	Naruto	999.99	2025-04-10
2	Nuru	Nurje	999.99	2025-04-10
3	Nuru	Nurje	999.99	2025-04-10
4	Pak	Kusari	999.99	2025-04-10
5	Pak	Nurje	999.99	2025-04-10
6	Vraj	mishra	10.25	2024-03-31
- Output:** Shows the execution log with 8 entries, all completed successfully.

• Update to null value

The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the schema `mydb` with tables `employees`, `salaries`, and `dept_emp`.
- SQL Editor:** Contains the following SQL code:

```
1 • UPDATE employees
2   SET hourly_pay = NULL
3   WHERE employee_id = 6;
4
5 • SELECT * FROM employees;
```
- Result Grid:** Displays the `employees` table with 6 rows. The `hourly_pay` column is set to NULL for employee ID 6.

employee_id	first_name	last_name	hourly_pay	hire_date
1	Kimi	Naruto	999.99	2025-04-10
2	Nuru	Nurje	999.99	2025-04-10
3	Nuru	Nurje	999.99	2025-04-10
4	Pak	Kusari	999.99	2025-04-10
5	Pak	Nurje	999.99	2025-04-10
6	Vraj	mishra	NULL	2024-03-31
- Output:** Shows the execution log with 12 entries, all completed successfully.

• Update whole column

The screenshot shows the MySQL Workbench interface with the 'mydb' schema selected. In the SQL Editor tab, the following code is run:

```

1 • UPDATE employees
2   SET hourly_pay = 900;
3
4 • SELECT * FROM employees;
    
```

The Result Grid shows the initial state of the employees table:

employee_id	first_name	last_name	hourly_pay	hire_date
1	Harsh	dev	900.00	2015-04-10
2	Muni	Nursa	900.00	2015-04-10
3	Nuri	Nursa	900.00	2015-04-10
4	Palek	Kumari	900.00	2015-04-10
5	Palek	Nursa	900.00	2015-04-10
6	Vinej	midra	900.00	2024-03-31

The Output tab shows the execution log:

- 11 21:45:00 UPDATE employees SET hourly_pay = NULL WHERE employee_id = 6
- 12 21:45:00 SELECT * FROM employees LIMIT 0, 1000
- 13 21:47:09 UPDATE employees SET hourly_pay = 900
- 14 21:47:09 SELECT * FROM employees LIMIT 0, 1000
- 15 21:47:32 UPDATE employees SET hourly_pay = 900
- 16 21:47:32 SELECT * FROM employees LIMIT 0, 1000

• Delete whole Table

The screenshot shows the MySQL Workbench interface with the 'mydb' schema selected. In the SQL Editor tab, the following code is run:

```

1 • DELETE FROM employees;
2
3 • SELECT * FROM employees;
    
```

The Result Grid shows the initial state of the employees table:

employee_id	first_name	last_name	hourly_pay	hire_date
1	Harsh	dev	900.00	2015-04-10
2	Muni	Nursa	900.00	2015-04-10
3	Nuri	Nursa	900.00	2015-04-10
4	Palek	Kumari	900.00	2015-04-10
5	Palek	Nursa	900.00	2015-04-10
6	Vinej	midra	900.00	2024-03-31

The Output tab shows the execution log:

- 11 21:45:00 UPDATE employees SET hourly_pay = NULL WHERE employee_id = 6
- 12 21:45:00 SELECT * FROM employees LIMIT 0, 1000
- 13 21:47:09 UPDATE employees SET hourly_pay = 900
- 14 21:47:09 SELECT * FROM employees LIMIT 0, 1000
- 15 21:47:32 UPDATE employees SET hourly_pay = 900
- 16 21:47:32 SELECT * FROM employees LIMIT 0, 1000

• Delete By selecting

The screenshot shows the MySQL Workbench interface with the 'mydb' schema selected. In the SQL Editor tab, the following code is run:

```

1 • DELETE FROM employees
2   WHERE employee_id = 6;
3
4 • SELECT * FROM employees;
    
```

The Result Grid shows the initial state of the employees table:

employee_id	first_name	last_name	hourly_pay	hire_date
1	Harsh	dev	900.00	2015-04-10
2	Muni	Nursa	900.00	2015-04-10
3	Nuri	Nursa	900.00	2015-04-10
4	Palek	Kumari	900.00	2015-04-10
5	Palek	Nursa	900.00	2015-04-10

The Output tab shows the execution log:

- 13 21:47:09 UPDATE employees SET hourly_pay = 900
- 14 21:47:09 SELECT * FROM employees LIMIT 0, 1000
- 15 21:47:32 UPDATE employees SET hourly_pay = 900
- 16 21:47:32 SELECT * FROM employees LIMIT 0, 1000
- 17 21:52:16 DELETE FROM employees WHERE employee_id = 6
- 18 21:52:16 SELECT * FROM employees LIMIT 0, 1000

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:**

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.

The screenshot shows the MySQL Workbench interface. The SQL editor window has the following content:

```
1 SET AUTOCOMMIT = OFF;
```

A purple arrow points from the top of the page down towards the cursor in the SQL editor.

On the right side of the interface, there is a message box with the following text:

Annotate current help is disabled.
Use the toolbar to manually get help
for the current caret position or to
toggle automatic help.

With this setting turned off,
Our transaction will not
Save automatically.

Step - By - Step

To delete whole table : ->

1

②) But if you want to restore previous table.

The screenshot shows the MySQL Workbench interface. The top bar displays the title 'Local instance M/OS32' and the status 'Automatic context help is disabled. Double click on a column to get help for the current context position or to toggle automatic help.' A red box highlights the status message.

The main area contains a query editor with the following SQL code:

```
1. DELETE FROM employees;
2. SELECT * FROM employees;
```

Below the query editor is a results grid titled 'employees 1. N'. The grid has columns: employee_id, first_name, last_name, hire_date, and duration. It lists 33 rows of data. A red box highlights the 'duration' column header.

3

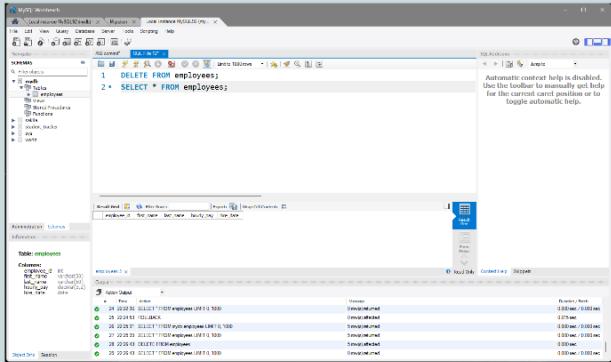
Our table is expert

The screenshot shows the MySQL Workbench application window. At the top, there's a toolbar with various icons for database management. Below the toolbar is a navigation pane on the left containing a tree view of databases ('employees', 'information_schema', 'mysql', 'performance_schema', 'sys', 'world') and other objects like tables and stored procedures. The main workspace has a query editor with the SQL tab selected, displaying the command 'ROLLBACK;'. To the right of the editor, there's a status bar with the schema name 'mydb' and a list of recent log entries. The bottom of the screen features a footer with links for 'Contact help' and 'Shortcuts'.

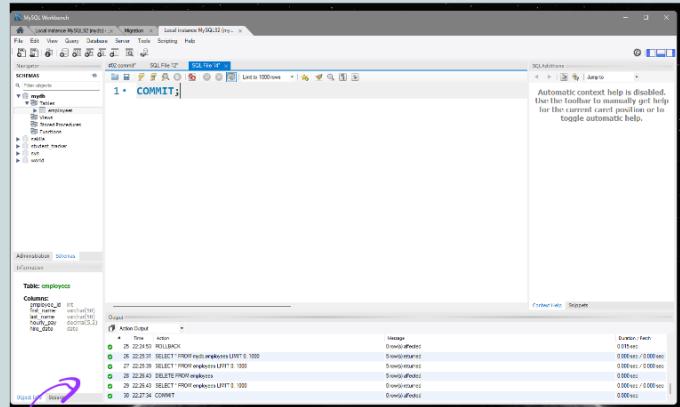
Automatic context help is disabled.
Use the toolbar to manually get help
for the current caret position or to
toggle automatic help.

but if want to delete whole table & save it.

① =

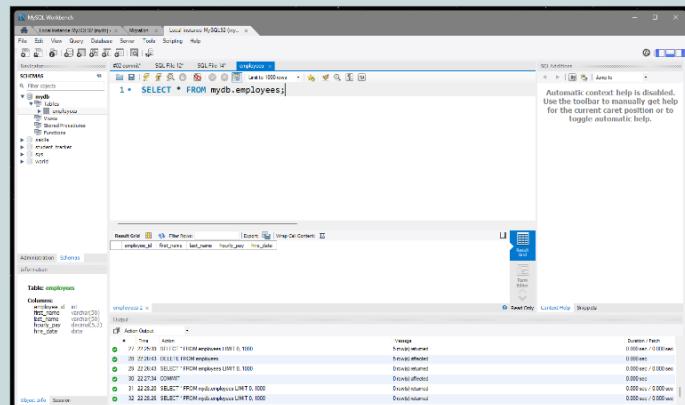


②



It will finally save in database

③



Current Date and Time

Create a test table

①

```
CREATE TABLE test (
    my_date DATE,
    my_time TIME,
    my_datetime DATETIME
);
```

②

```
SELECT * FROM test;
```

③

```
INSERT INTO test
VALUES (CURRENT_DATE(), CURRENT_TIME(), NOW());
```

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 :-)

The screenshot shows the MySQL Workbench interface. In the central SQL editor window, the following SQL code is displayed:

```

CREATE TABLE products (
    product_id INT,
    product_name VARCHAR(25) UNIQUE,
    price DECIMAL(4,2)
);

```

The 'Schemas' pane on the left shows the 'employees' schema containing the 'products' table. The 'Output' pane at the bottom displays the execution log with several entries, including the creation of the table and the addition of the unique constraint.

Alter Table and then add Unique feature after making table .

① Make table without any Unique column;

The screenshot shows the MySQL Workbench interface. In the central SQL editor window, the following SQL code is displayed:

```

CREATE TABLE products (
    product_id INT,
    product_name VARCHAR(25),
    price DECIMAL(4,2)
);

```

The 'Schemas' pane on the left shows the 'employees' schema containing the 'products' table. The 'Output' pane at the bottom displays the execution log with several entries, including the creation of the table.

② make product-name UNIQUE

The screenshot shows the MySQL Workbench interface. In the central SQL editor window, the following SQL code is displayed:

```

ALTER TABLE products
ADD CONSTRAINT
UNIQUE(product_name);

```

An arrow points from the first screenshot to this one. The 'Schemas' pane on the left shows the 'employees' schema containing the 'products' table. The 'Output' pane at the bottom displays the execution log with several entries, including the alteration of the table to add the unique constraint.

NOT NULL

- Create table with NOT NULL keyword

so, price can not be null.

The screenshot shows the MySQL Workbench interface. In the central query editor, the following SQL code is written:

```
CREATE TABLE products (
    product_id INT,
    product_name VARCHAR(25) UNIQUE,
    price DECIMAL(4,2) NOT NULL
);
```

A handwritten note "so, price can not be null." is written over the "NOT NULL" part of the code. A blue arrow points from this note to the "NOT NULL" part of the code.

- Alter table and add NOTNULL

The screenshot shows the MySQL Workbench interface. In the central query editor, the following SQL code is written:

```
ALTER TABLE products
MODIFY product_name VARCHAR(45) UNIQUE NOT NULL;
```

The MySQL Workbench interface includes a left sidebar for navigating databases and tables, and a right sidebar for viewing table structure and data.

Check Constraints

Create table with check

The screenshot shows the MySQL Workbench interface. In the central query editor, the following SQL code is written:

```
1 CREATE TABLE products (
2     product_id INT,
3     product_name VARCHAR(25) UNIQUE,
4     price DECIMAL(4,2) NOT NULL
5     CONSTRAINT check_pay CHECK (price >= 10.00)
6 );
```

The line `CONSTRAINT check_pay CHECK (price >= 10.00)` is highlighted with a blue underline. Below the code, the 'Output' pane displays the execution results:

#	Time	Action	Message	Duration / Fetch
59	23:08:16	Insert INTO employees VALUE (1, "Ice-cream", 40, 2, "pizza", 500, 3, "burger", 100, 4, "Yess", 150)	Error Code: 1364. Column count doesn't match value count at row 1	0.000 sec
60	23:08:16	Insert INTO products VALUE (1, "Ice-cream", 40, 2, "pizza", 500, 3, "burger", 100, 4, "Yess", 150)	Error Code: 1364. Out of range value for column 'price' at row 2	0.000 sec
61	23:08:16	Insert INTO products VALUE (1, "Ice-cream", 40, 2, "pizza", 500, 3, "burger", 100, 4, "Yess", 150)	4 rows affected. Records: 4 Duplicates: 0 Warnings: 0	0.019 sec
62	23:08:16	SELECT * from products LIMIT 5, 1000	4 rows returned	0.000 sec / 0.000 sec
63	23:23:19	DROP TABLE products	0 rows affected	0.018 sec
64	23:24:59	CREATE TABLE products (product_id INT, product_name VARCHAR(25) UNIQUE, price DECIM...	0 rows affected	0.021 sec

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

The screenshot shows the MySQL Workbench interface. In the central query editor, the following SQL code is written:

```
1 ALTER TABLE products
2 ADD CONSTRAINT chk_product_id CHECK(product_id >= 0);
```

The line `ADD CONSTRAINT chk_product_id CHECK(product_id >= 0);` is highlighted with a blue underline. Below the code, the 'Output' pane displays the execution results:

#	Time	Action	Message	Duration / Fetch
60	23:08:16	Insert INTO products VALUE (1, "Ice-cream", 40, 2, "pizza", 500, 3, "burger", 100, 4, "Yess", 150)	Error Code: 1364. Out of range value for column 'price' at row 2	0.000 sec
61	23:08:16	Insert INTO products VALUE (1, "Ice-cream", 40, 2, "pizza", 500, 3, "burger", 100, 4, "Yess", 150)	4 rows affected. Records: 4 Duplicates: 0 Warnings: 0	0.019 sec
62	23:08:16	SELECT * from products LIMIT 5, 1000	4 rows returned	0.000 sec / 0.000 sec
63	23:23:19	DROP TABLE products	0 rows affected	0.018 sec
64	23:24:59	CREATE TABLE products (product_id INT, product_name VARCHAR(25) UNIQUE, price DECIM...	0 rows affected	0.021 sec
65	23:48:37	ALTER TABLE products ADD CONSTRAINT chk_product_id CHECK(product_id >= 0)	0 rows affected. Records: 0 Duplicates: 0 Warnings: 0	0.002 sec

• Drop Checks

The screenshot shows the MySQL Workbench interface. In the SQL editor tab, the following code is displayed:

```
1  ALTER TABLE products
2  DROP CHECK chk_product_id;
```

The code is being run against a local MySQL instance. The results pane shows the execution of several statements, including the creation of the 'products' table and the addition of a check constraint, followed by the execution of the provided code to drop the constraint.

#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

The screenshot shows the MySQL Workbench interface. In the SQL editor tab, the following code is displayed:

```
1  CREATE TABLE products (
2      product_id INT,
3      product_name VARCHAR(25) UNIQUE,
4      price DECIMAL(4,2) DEFAULT 0
5  );
```

The code is being run against a local MySQL instance. The results pane shows the execution of several statements, including the creation of the 'products' table with a default value constraint for the 'price' column.

• Alter Table to have Default value

```

1 • ALTER TABLE products
2 • ALTER price SET DEFAULT 0;

Table: products
Columns:
product_id int
product_name varchar(25)
price decimal(4,2)

Object Info Session

```

None, odd errors

```

1 Insert INTO products (product_id , product_name)
2 VALUES (5, "straw"),
3 (6, "napkin"),
4 (7, "fork"),
5 (8, "spoon");
6
7 • SELECT * from products;

Table: products
Columns:
product_id int
product_name varchar(25)
price decimal(4,2)

Object Info Session

```

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

①

```

1 • CREATE TABLE transactions(
2   transaction_id INT,
3   amount DECIMAL(5,2),
4   transaction_date DATETIME DEFAULT NOW()
5 );

```

⇒

②

```

1 • Insert INTO transactions(transaction_id, amount)
2 VALUES (1,40),
3 (2,50),
4 (3,10),
5 (4,15);
6
7 • SELECT * from transactions;

Table: transactions
Columns:
transaction_id int
amount decimal(5,2)
transaction_date datetime

Object Info Session

```

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.

• Create Table with Primary key

The screenshot shows the MySQL Workbench interface. In the left sidebar, under the 'Schemas' section, the 'mydb' schema is selected. The 'Tables' section shows a single table named 'employees'. In the main workspace, a SQL editor window is open with the following code:

```
CREATE TABLE transactions(
    transaction_id INT PRIMARY KEY,
    amount DECIMAL(5,2),
    transaction_date DATETIME DEFAULT NOW()
);

SELECT * FROM transactions;
```

Below the SQL editor, the 'Result Grid' shows a single row of data:

transaction_id	amount	transaction_date
1	4.15	2023-06-21 10:18:18

The status bar at the bottom indicates 'Query Completed'.

Alter Table with Primary Key

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** mydb
- Tables:** employees, products, transactions
- SQL Editor:** SQL File 59*
- SQL:**

```
1 • ALTER TABLE transactions
2   ADD CONSTRAINT
3     PRIMARY KEY(transaction_id);
4
5 • SELECT * FROM transactions;
```
- Output:** Action Output
- Result Grid:** Shows the structure of the transactions table.
- Action Output:** Log of database operations.

Insering rows

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** mydb
- Tables:** employees, products, transactions
- SQL Editor:** SQL File 60*
- SQL:**

```
1 • Insert INTO transactions(transaction_id, amount)
2   VALUES (1,40),
3          (2,50),
4          (3,10),
5          (4,15);
6
7 • SELECT * from transactions;
```
- Output:** Action Output
- Result Grid:** Shows the inserted data in the transactions table.
- Action Output:** Log of database operations.

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

The screenshot shows the MySQL Workbench interface. In the SQL editor, the following code is written:

```
1 • CREATE TABLE transactions(
2     transaction_id INT PRIMARY KEY AUTO_INCREMENT,
3     amount DECIMAL(5,2),
4     transaction_date DATETIME DEFAULT NOW()
5 );
6
7 • SELECT * FROM transactions;
```

A handwritten note next to the code says "By default it is 1."

The results grid shows the table structure:

transaction_id	amount	transaction_date
1	40.00	2025-03-30 00:42:19
2	50.00	2025-03-30 00:42:19
3	10.00	2025-03-30 00:42:19
4	15.00	2025-03-30 00:42:19

The output pane shows the execution log:

Time	Action	Message	Duration / Fetch
93 00:11:18	SELECT * FROM transactions LIMIT 0, 1000	0 rows returned	0.000 sec / 0.000 sec
94 00:32:22	Insert INTO transactions(transaction_id, amount) VALUES (40), (25), (10), (15)	4 rows affected Records: 4 Duplicates: 0 Warnings: 0	0.000 sec
95 00:32:22	SELECT * FROM transactions LIMIT 0, 1000	4 rows returned	0.000 sec / 0.000 sec
96 00:39:03	DROP TABLE transactions	0 rows affected	0.016 sec
97 00:40:56	CREATE TABLE transactions(transaction_id INT PRIMARY KEY AUTO_INCREMENT, amount DECIM	0 rows affected	0.000 sec
98 00:40:56	SELECT * FROM transactions LIMIT 0, 1000	0 rows returned	0.000 sec / 0.000 sec

Inserting rows in transactions .

The screenshot shows the MySQL Workbench interface. In the SQL editor, the following code is written:

```
1 • Insert INTO transactions( amount)
2     VALUES (40),
3             (50),
4             (10),
5             (15);
6
7 • SELECT * from transactions;
```

The results grid shows the table structure:

transaction_id	amount	transaction_date
1	40.00	2025-03-30 00:42:19
2	50.00	2025-03-30 00:42:19
3	10.00	2025-03-30 00:42:19
4	15.00	2025-03-30 00:42:19

The output pane shows the execution log:

Time	Action	Message	Duration / Fetch
95 00:32:22	SELECT * FROM transactions LIMIT 0, 1000	4 rows returned	0.000 sec / 0.000 sec
96 00:39:03	DROP TABLE transactions	0 rows affected	0.016 sec
97 00:40:56	CREATE TABLE transactions(transaction_id INT PRIMARY KEY AUTO_INCREMENT, amount DECIM	0 rows affected	0.000 sec
98 00:40:56	SELECT * FROM transactions LIMIT 0, 1000	4 rows affected Records: 4 Duplicates: 0 Warnings: 0	0.000 sec / 0.000 sec
99 00:42:19	Insert INTO transactions(amount) VALUES (40), (50), (10), (15)	4 rows affected	0.000 sec
100 00:42:19	SELECT * FROM transactions LIMIT 0, 1000	4 rows returned	0.000 sec / 0.000 sec

* Alter table to start auto increment with 1000

The screenshot shows the MySQL Workbench interface. The top menu bar includes File, Edit, View, Query, Database, Server, Tools, Scripting, and Help. The left sidebar displays the Navigator and Schemas. Under the Schemas section, there is a tree view for the 'mydb' database, which contains tables like 'transaction', 'employee', 'products', 'view', 'temp', 'Stored Procedures', 'Functions', 'Materialized Views', 'student_tracker', 'sys', and 'world'. The main workspace shows a query editor with the following SQL code:

```
1 • ALTER TABLE transactions
2 AUTO_INCREMENT = 1000;
3
4
5 • SELECT * FROM transactions;
```

Below the query editor is a results grid titled 'transactions 1 x'. The grid has columns: transaction_id, amount, and transaction_date. The data is as follows:

transaction_id	amount	transaction_date
1	40.00	2025-03-30 00:40:19
2	10.00	2025-03-30 00:40:19
3	10.00	2025-03-30 00:40:19
4	15.00	2025-03-30 00:40:19

The bottom part of the interface shows the 'Object info' and 'Session' panes, and a status bar indicating the script was saved to 'C:\Users\Harsh\Desktop\SQL\#13.AUTOINCREMENT\#02_alter-table-to-start-auto-increment-with-1000.sql'.

Interestin' Rows

The screenshot shows the MySQL Workbench interface. The top navigation bar includes 'File', 'Edit', 'View', 'Query', 'Database', 'Server', 'Tools', 'Scripting', and 'Help'. The left sidebar shows the 'Schemas' tree, which includes the 'mysql' schema with tables like 'columns_priv', 'employees', 'products', 'views', 'Stored Procedures', 'Functions', 'sealevel', 'student_tracker', and 'sys', along with the 'world' schema. The main area has two tabs open: '#01_create-table-with-auto-incr...' and '#02_alter-table-to-start-auto-incr...'. The '#02' tab contains the following SQL code:

```
1 • Insert INTO transactions( amount)
2   VALUES (40),
3   (50),
4   (10),
5   (15);
6
7 • SELECT * from transactions;
```

Below the tabs is a toolbar with icons for 'Run', 'Stop', 'Reset', 'Jump to', and 'SQL Editor'. To the right of the tabs is a message: 'Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.' The results grid below the tabs displays the following data:

transaction_id	amount	transaction_date
1	40.00	2023-03-30 00:40:19
2	50.00	2023-03-30 00:40:19
3	10.00	2023-03-30 00:40:19
4	15.00	2023-03-30 00:40:19
1000	40.00	2023-03-30 00:40:41
1001	50.00	2023-03-30 00:40:41
1002	10.00	2023-03-30 00:40:41
1003	15.00	2023-03-30 00:40:41
1004	00.00	0000-00-00 00:00:00

The bottom section shows the 'Output' pane with the following log entries:

Action	Time	Action	Message	Duration / Fetch
101	00:43:59	Action	0 rows) affected Records: 0 Duplicates: 0 Warnings: 0	0.016 sec
102	00:44:14	ALTER TABLE transactions AUTO_INCREMENT = 1000	0 rows) affected Records: 0 Duplicates: 0 Warnings: 0	0.016 sec
103	00:44:18	ALTER TABLE transactions AUTO_INCREMENT = 1000	0 rows) affected Records: 0 Duplicates: 0 Warnings: 0	0.016 sec
104	00:44:18	SELECT * FROM transactions LIMIT 0, 1000	4 rows) returned	0.000 sec / 0.000 sec
105	00:45:41	Insert INTO transactions(amount) VALUES (40),(50),(10),(15)	4 rows) affected Records: 4 Duplicates: 0 Warnings: 0	0.000 sec
106	00:45:41	SELECT * from transactions LIMIT 0, 1000	9 rows) returned	0.000 sec / 0.000 sec

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

Screenshot of MySQL Workbench showing the creation of the 'customers' table. The SQL tab contains the following code:

```
1 * CREATE TABLE customers(
2 *   customer_id INT PRIMARY KEY AUTO_INCREMENT,
3 *   first_name VARCHAR(50),
4 *   last_name VARCHAR(50),
5 * );
6
7 * SELECT * FROM customers;
```

The results tab shows the table structure with columns: customer_id, first_name, and last_name. The data tab shows no data has been inserted yet.

Transactions Table

② Inserting rows

Screenshot of MySQL Workbench showing the creation of the 'transactions' table. The SQL tab contains the following code:

```
1 * CREATE TABLE transactions(
2 *   transaction_id INT PRIMARY KEY AUTO_INCREMENT,
3 *   amount DECIMAL(5,2),
4 *   transaction_date DATETIME DEFAULT NOW(),
5 *   customer_id INT,
6 *   FOREIGN KEY(customer_id) REFERENCES customers(customer_id);
7
8 * SELECT * FROM transactions;
```

The results tab shows the table structure with columns: transaction_id, amount, transaction_date, and customer_id. The data tab shows no data has been inserted yet.

③ Inserting rows

Screenshot of MySQL Workbench showing the insertion of data into the 'customers' table. The SQL tab contains the following code:

```
1 * INSERT INTO customers(first_name, last_name)
2 * VALUES ("HARSH", "DEV"),
3 *         ("PALAK", "JUNU"),
4 *         ("RABINDRA", "CHOUHARY");
5
6 * SELECT * FROM customers;
```

The results tab shows the table structure with columns: customer_id, first_name, and last_name. The data tab shows three rows have been inserted with customer IDs 1, 2, and 3, and names HARSH DEV, PALAK JUNU, and RABINDRA CHOUHARY respectively.

• Drop foreign key.

The screenshot shows the MySQL Workbench interface. In the left sidebar, under the 'mydb' schema, the 'Foreign Keys' section is selected. A blue arrow points from the text 'Drop foreign key' to the 'transactions_ibfk_1' entry in the list. The main query editor contains the following SQL code:

```

1 ALTER TABLE transactions
2 DROP FOREIGN KEY transactions_ibfk_1;

```

The output window shows the execution of these statements. The first statement succeeds, while the second fails with an error message:

```

116 00:58:25 SELECT * from customers LIMIT 0,1000
116 00:58:25 DROP TABLE transactions
117 01:01:33 CREATE TABLE `transaction`(`transaction_id` INT PRIMARY KEY AUTO_INCREMENT, `amount` DECIMAL(5,2), `customer_id` INT)
118 01:01:56 CREATE TABLE `transactions`(`transaction_id` INT PRIMARY KEY AUTO_INCREMENT, `amount` DECIMAL(5,2), `customer_id` INT)
119 01:02:06 CREATE TABLE `transactions`(`transaction_id` INT PRIMARY KEY AUTO_INCREMENT, `amount` DECIMAL(5,2), `customer_id` INT)
120 01:02:06 SELECT * FROM transactions LIMIT 0,1000

```

An error message is present in the output:

```

Err Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'DROP TABLE transactions' at line 2

```

• Alter table to make foreign key

The screenshot shows the MySQL Workbench interface. In the left sidebar, under the 'mydb' schema, the 'Tables' section is selected. A blue arrow points from the text 'Alter table to make foreign key' to the 'ADD CONSTRAINT fk_customer_id' part of the following SQL code. The main query editor contains the following SQL code:

```

1 • ALTER TABLE transactions
2 ADD CONSTRAINT fk_customer_id
3 FOREIGN KEY(customer_id) REFERENCES customers(customer_id);

```

The output window shows the execution of these statements. The first two statements succeed, while the third fails with an error message:

```

120 01:02:06 SELECT * FROM transactions LIMIT 0, 1000
121 01:09:02 ALTER TABLE transactions DROP FOREIGN KEY transactions_ibfk_1
122 01:10:44 ALTER TABLE transactions FOREIGN KEY(customer_id) REFERENCES customers(customer_id)
123 01:10:58 ALTER TABLE transactions ADD CONSTRAINT FOREIGN KEY(customer_id) REFERENCES customers(customer_id)
124 01:11:14 ALTER TABLE transactions DROP FOREIGN KEY transactions_ibfk_1
125 01:11:34 ALTER TABLE transactions ADD CONSTRAINT fk_customer_id FOREIGN KEY(customer_id) RE...

```

An error message is present in the output:

```

Err Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'ALTER TABLE transactions FOREIGN KEY(customer_id) REFERENCES customers(customer_id)' at line 2

```

A handwritten note 'name, alter your choice' is written over the code area.

→ Now, step by step insertion →

①

The screenshot shows the MySQL Workbench interface. The main query editor contains the following SQL code for creating the 'customers' table:

```

1 • CREATE TABLE `customers`(
2     `customer_id` INT PRIMARY KEY AUTO_INCREMENT,
3     `first_name` VARCHAR(30),
4     `last_name` VARCHAR(30)
5 );
6
7 • SELECT * FROM customers;

```

The results pane shows the inserted data:

customer_id	first_name	last_name
1	PAUL	DEV
2	MALIA	MUNU
3	RABIMORA	CHODHARY

②

The screenshot shows the MySQL Workbench interface. The main query editor contains the following SQL code for inserting data into the 'customers' table:

```

1 • INSERT INTO `customers`(`first_name`, `last_name`)
2     VALUES ('PAUL', 'DEV'),
3            ('MALIA', 'MUNU'),
4            ('RABIMORA', 'CHODHARY');
5
6 • SELECT * FROM customers;

```

The results pane shows the inserted data:

customer_id	first_name	last_name
1	PAUL	DEV
2	MALIA	MUNU
3	RABIMORA	CHODHARY

(3)

```

CREATE TABLE transactions (
    transaction_id INT PRIMARY KEY AUTO_INCREMENT,
    amount DECIMAL(5,2),
    transaction_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    customer_id INT,
    FOREIGN KEY(customer_id) REFERENCES customers(customer_id)
);

SELECT * FROM transactions;

```

(4)

```

ALTER TABLE transactions
AUTO_INCREMENT = 1000;

```

(5)

```

INSERT INTO transactions(amount, customer_id)
VALUES (6.99, 1),
       (2.89, 2),
       (3.39, 3),
       (4.99, 4);

SELECT * FROM transactions;

```

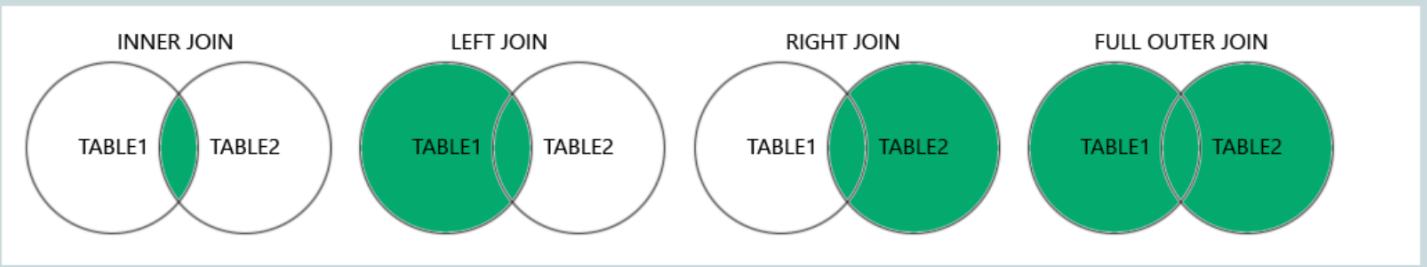
~~#JOINS~~

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

```

CREATE TABLE customers(
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50),
    last_name VARCHAR(50)
);

SELECT * FROM customers;
  
```

②

```

INSERT INTO customers(first_name, last_name)
VALUES ('KUSH', 'DEV'),
       ('VALAK', 'NUNAVU'),
       ('HABIBA', 'CHODHARY'),
       ('CHRISTI', 'NUMAR');

SELECT * FROM customers;
  
```

③

```

CREATE TABLE transactions(
    transaction_id INT PRIMARY KEY AUTO_INCREMENT,
    amount DECIMAL(10,2),
    time DATETIME DEFAULT NOW(),
    customer_id INT,
    FOREIGN KEY(customer_id) REFERENCES customers(customer_id)
);

SELECT * FROM transactions;
  
```

④

```

ALTER TABLE transactions
AUTO_INCREMENT = 1000;

SELECT * FROM transactions;
  
```

⑤

```

INSERT INTO transactions(amount, customer_id)
VALUES (4.99,1),
       (1.99,2),
       (3.99,3),
       (2.99,4);

SELECT * FROM transactions;
  
```

⑥

```

SELECT *
FROM transactions
INNER JOIN customers
ON transactions.customer_id = customers.customer_id;

SELECT * FROM transactions;
  
```

This will show all common column.

7

```

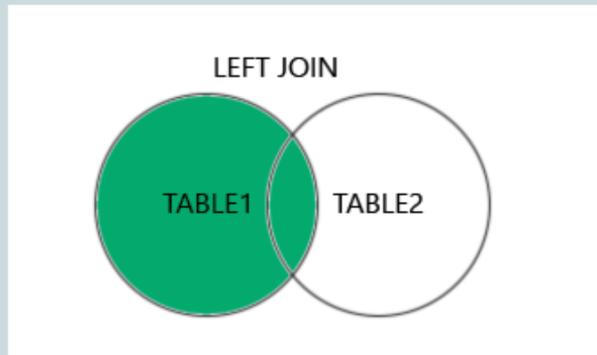
1. SELECT transaction_id, amount, first_name, last_name
2. FROM transactions INNER JOIN customers
3. ON transactions.customer_id = customers.customer_id

```

→ This will show first_name, last_name, amount f 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.



```

1. SELECT *
2. FROM transactions LEFT JOIN customers
3. ON transactions.customer_id = customers.customer_id;

```

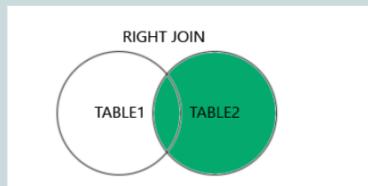
transaction_id	amount	transaction_date	customer_id	customer_id	first_name	last_name
1004	4.99	2025-03-30 01:40:29	3	3	RAENDRA	CHOURHARY
1005	2.00	2025-03-30 01:40:29	3	3	PAUL	DEV
1006	2.83	2025-03-30 01:40:29	3	3	RAENDRA	CHOURHARY
1007	4.99	2025-03-30 01:40:29	1	1	HARSH	DEV

Action Output

- 146 01:41:50 SELECT * FROM transactions INNER JOIN customers ON transactions.customer_id = customers.. Error Code: 1054. Unknown column 'transactions.customer_id' in 'on clause'
- 147 01:42:35 SELECT * FROM transactions INNER JOIN customers ON transactions.customer_id = customers.. 4 rows(s) returned
- 148 01:50:32 SELECT transaction_id, amount, first_name, last_name FROM transactions INNER JOIN customer.. 4 rows(s) returned
- 149 01:50:45 SELECT * FROM transactions INNER JOIN customers ON transactions.customer_id = customers.. 4 rows(s) returned
- 150 01:50:59 SELECT * FROM transactions LEFT JOIN customers ON transactions.customer_id = customers.. 4 rows(s) returned
- 151 01:58:59 SELECT * FROM transactions LEFT JOIN customers ON transactions.customer_id = customers.. 4 rows(s) returned

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.

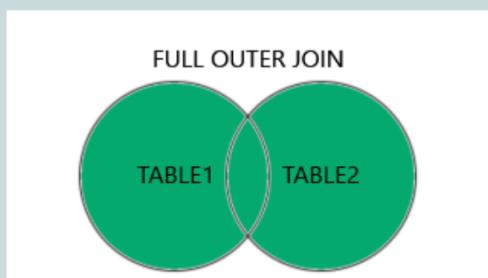


transaction_id	amount	transaction_date	customer_id	first_name	last_name
1007	1.99	2023-03-30 01:40:29	1	HARSH	KUMAR
1008	2.89	2023-03-30 01:40:29	2	RISHABH	MISHRA
1004	4.99	2023-03-30 01:40:29	3	RABINDRA	CHOWDHARY
1006	3.83	2023-03-30 01:40:29	3	RABINDRA	CHOWDHARY
1005	0.98	2023-03-30 01:40:29	4	CHINTU	KUMARE

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 Functions

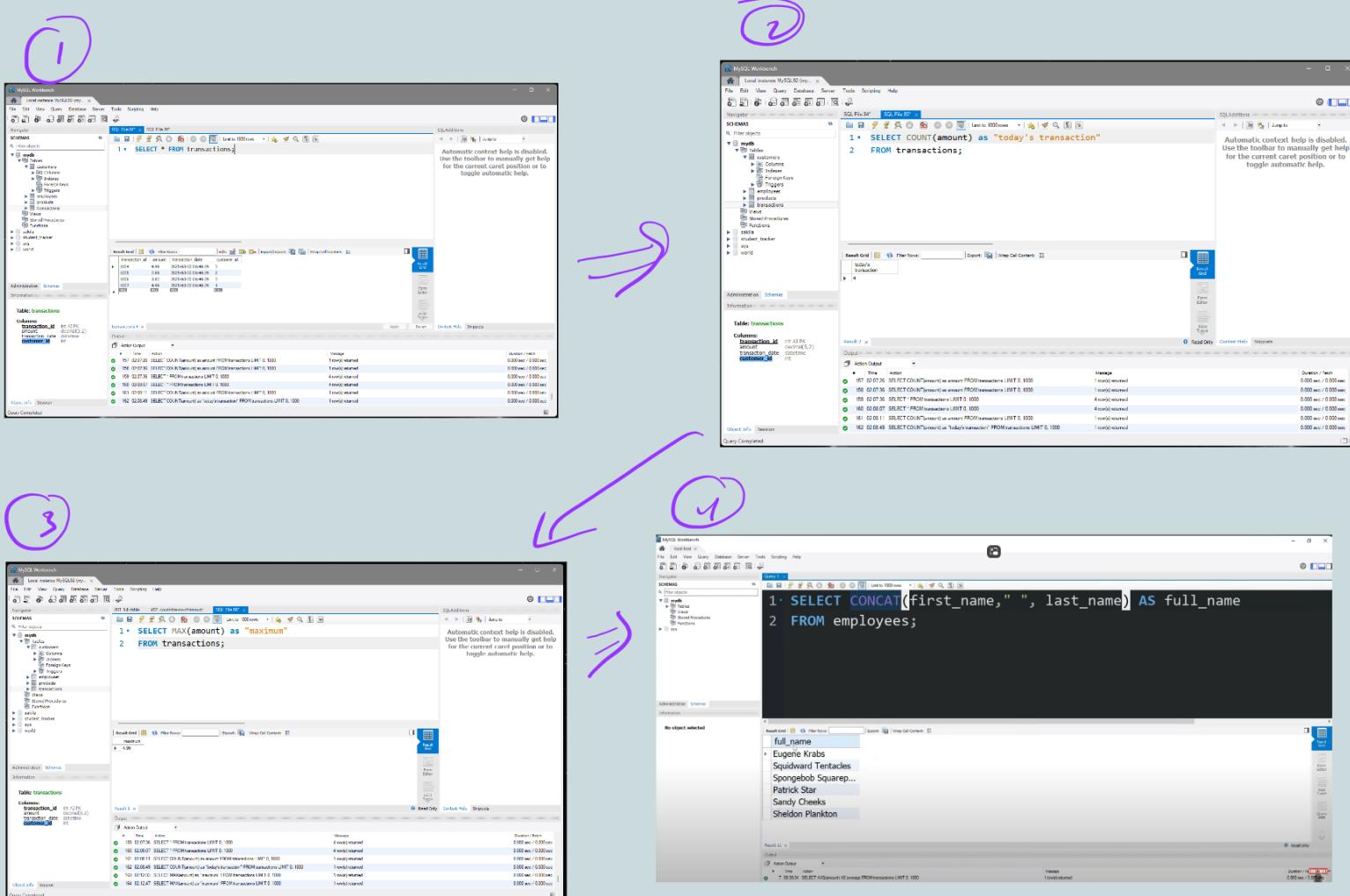
In SQL, functions are pre-written code blocks that perform specific actions on data, enabling users to manipulate, extract, and analyze information efficiently. They can be built-in (like SUM, COUNT, UPPER) or user-defined, and can be used on individual cells, records, or entire datasets.

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. String Functions (Text Manipulation)

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

Function	Description	Example
UPPER()	Converts text to uppercase.	<code>SELECT UPPER('hello');</code> → HELLO
LOWER()	Converts text to lowercase.	<code>SELECT LOWER('HELLO');</code> → hello
LENGTH()	Returns length of a string.	<code>SELECT LENGTH('SQL');</code> → 3
CONCAT()	Joins two or more strings.	<code>SELECT CONCAT('Hello', ' ', 'World');</code> → Hello World
SUBSTRING()	Extracts part of a string.	<code>SELECT SUBSTRING('database', 1, 4);</code> → data
TRIM()	Removes spaces from both ends.	<code>SELECT TRIM(' SQL ');</code> → SQL
LTRIM()	Removes spaces from the left.	<code>SELECT LTRIM(' SQL');</code> → SQL
RTRIM()	Removes spaces from the right.	<code>SELECT RTRIM('SQL ');</code> → SQL
REPLACE()	Replaces part of a string.	<code>SELECT REPLACE('SQL is great', 'great', 'awesome');</code> → SQL is awesome
LEFT()	Extracts left portion of a string.	<code>SELECT LEFT('SQL', 2);</code> → SQ
RIGHT()	Extracts right portion of a string.	<code>SELECT RIGHT('SQL', 2);</code> → QL
LOCATE()	Finds the position of a substring.	<code>SELECT LOCATE('l', 'Hello');</code> → 3
ASCII()	Returns ASCII code of a character.	<code>SELECT ASCII('A');</code> → 65
CHAR()	Converts ASCII code to a character.	<code>SELECT CHAR(65);</code> → 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.	<code>SELECT IFNULL(NULL, 'No Value');</code> → No Value
COALESCE()	Returns first non-null value.	<code>SELECT COALESCE(NULL, NULL, 'SQL', NULL);</code> → SQL
NULLIF()	Returns NULL if values are equal.	<code>SELECT NULLIF(10, 10);</code> → NULL
CAST()	Converts data type.	<code>SELECT CAST(123 AS CHAR);</code> → '123'
CONVERT()	Converts data type.	<code>SELECT CONVERT(123, CHAR);</code> → '123'

3. Mathematical Functions (Numeric Operations)

Used for performing calculations on numeric values.

Function	Description	Example
ABS()	Returns absolute value.	<code>SELECT ABS(-10);</code> → 10
CEIL()	Rounds up to next integer.	<code>SELECT CEIL(4.2);</code> → 5
FLOOR()	Rounds down to previous integer.	<code>SELECT FLOOR(4.9);</code> → 4
ROUND()	Rounds to nearest integer.	<code>SELECT ROUND(4.6);</code> → 5
MOD()	Returns remainder of division.	<code>SELECT MOD(18, 3);</code> → 1
POWER()	Returns power of a number.	<code>SELECT POWER(2, 3);</code> → 8
SQRT()	Returns square root.	<code>SELECT SQRT(16);</code> → 4
RAND()	Generates a random number between 0 and 1.	<code>SELECT RAND();</code> → 0.7463
EXP()	Returns e^x .	<code>SELECT EXP(1);</code> → 2.718
LOG()	Returns natural logarithm.	<code>SELECT LOG(10);</code> → 2.302
SIN()	Returns sine of angle (radians).	<code>SELECT SIN(PI()/2);</code> → 1
COS()	Returns cosine of angle.	<code>SELECT COS(0);</code> → 1
TAN()	Returns tangent of angle.	<code>SELECT TAN(PI()/4);</code> → 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.

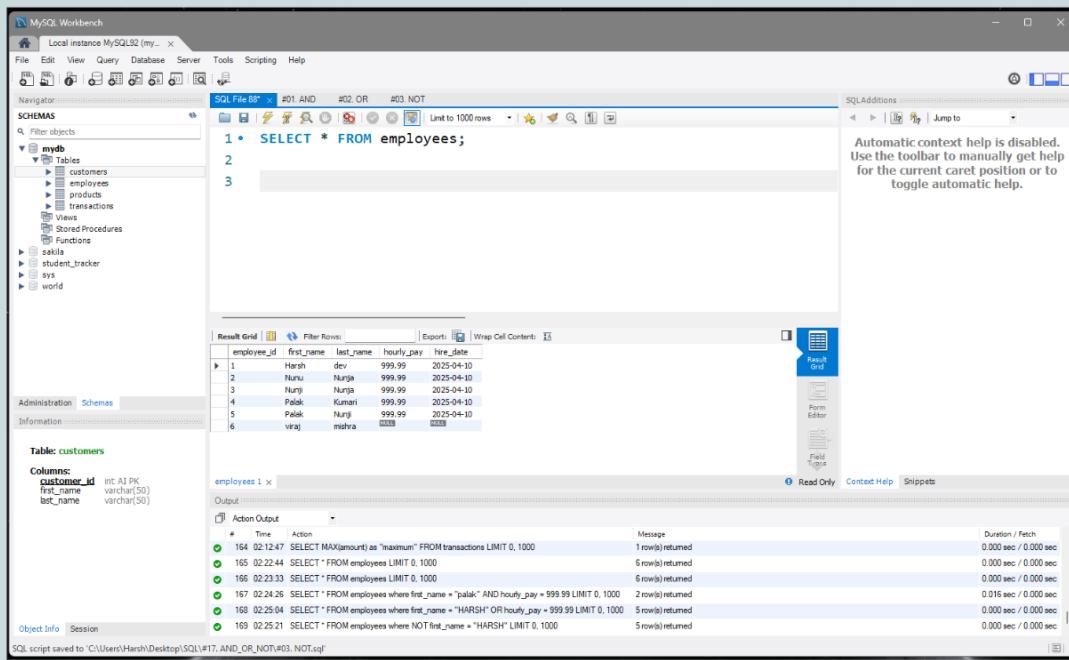
- **Example:** WHERE age > 18 AND city = 'New York' (This selects records where both conditions are met)
- OR:

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

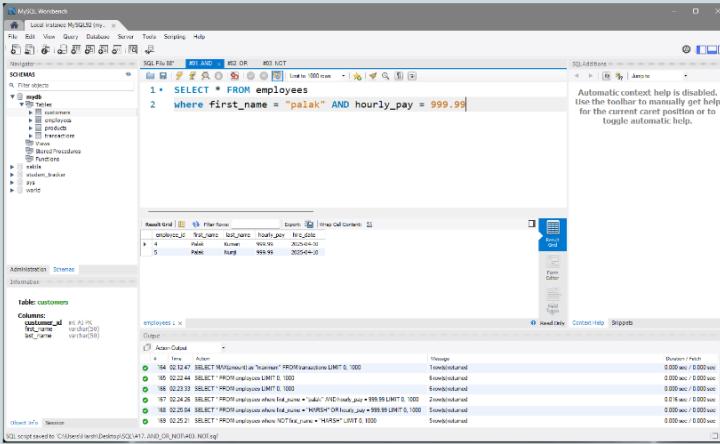
- **Example:** WHERE age > 65 OR disability = 'Yes' (This selects records where either condition is met)
- NOT:

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

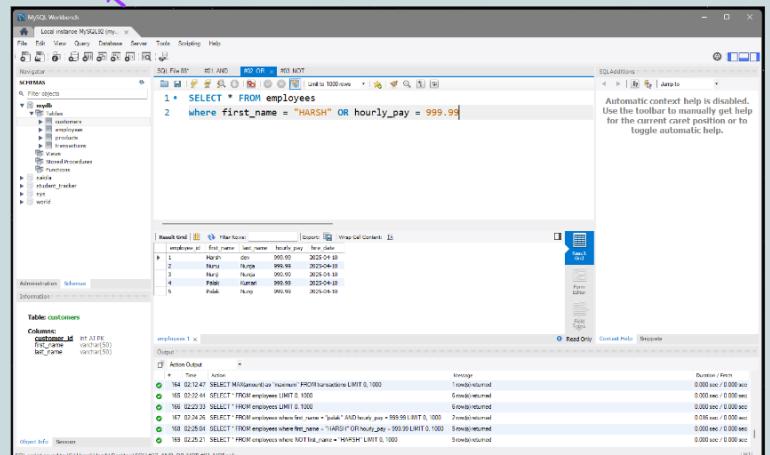
- **Example:** WHERE NOT (age < 18) (This selects records where the age is not less than 18)



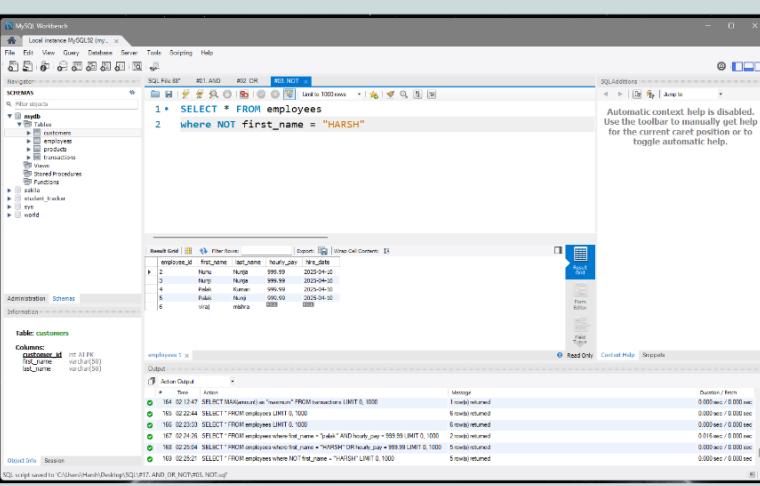
AND



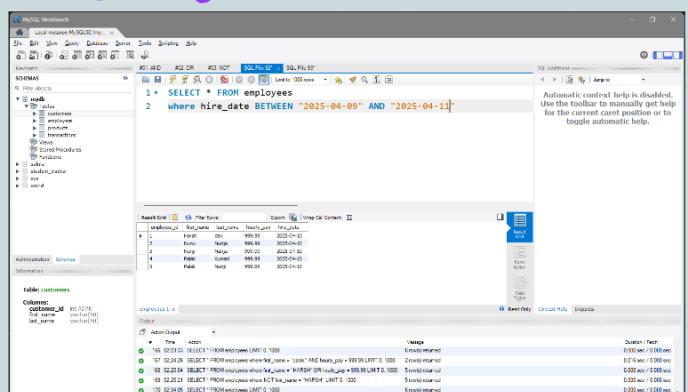
o R



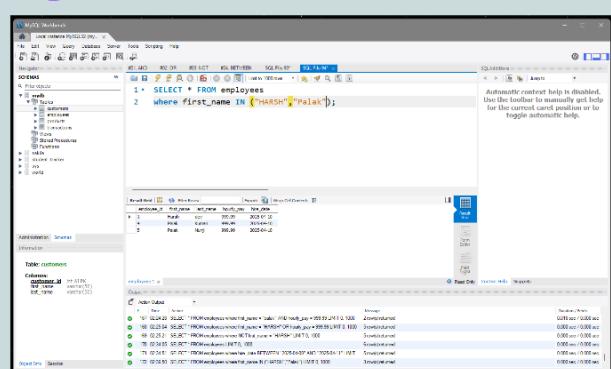
NOT



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.

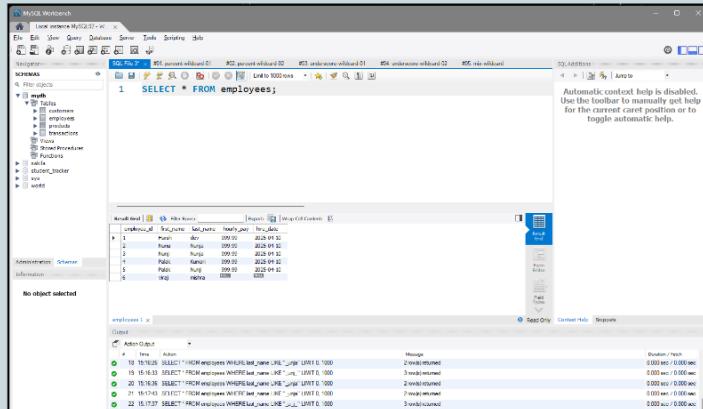
Wildcard Characters

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 **

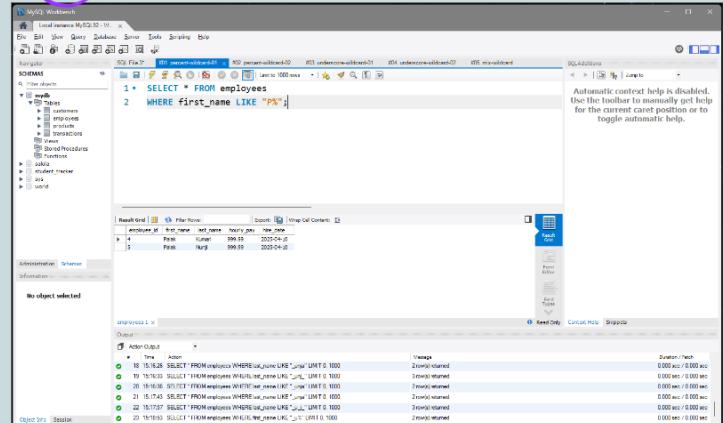
* Not supported in PostgreSQL and MySQL databases.

** Supported only in Oracle databases.

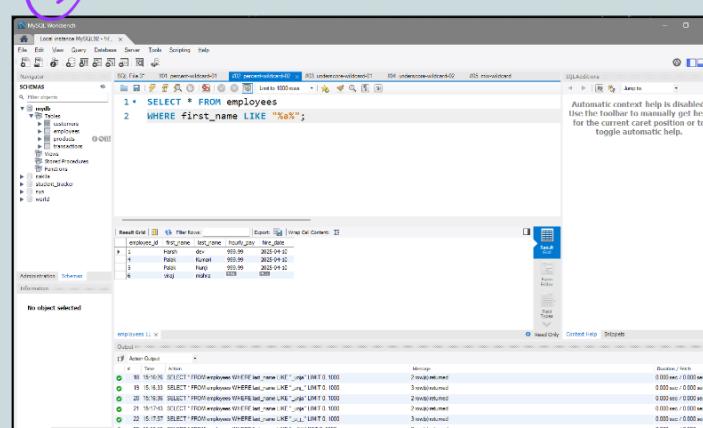
①



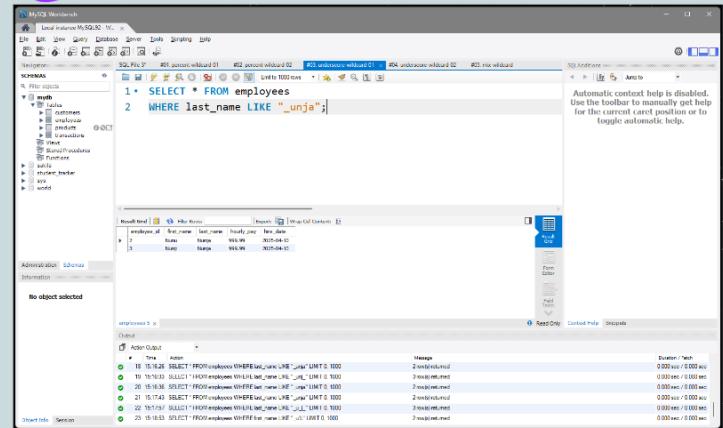
②



③



④



(5)

```

1. SELECT * FROM employees
2. WHERE last_name LIKE '_u_';

```

employee_id	first_name	last_name	hire_date
2	Herm	Urgon	2025-01-30
5	Heidi	Ungar	2025-01-30

(6)

```

1. SELECT * FROM employees
2. WHERE first_name LIKE "u%"; 

```

employee_id	first_name	last_name	hire_date
2	Herm	Urgon	2025-01-30
5	Heidi	Ungar	2025-01-30

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.

```

1. SELECT * FROM employees;

```

employee_id	first_name	last_name	hire_date
1	Pat	Adams	2025-01-30
2	Herm	Urgon	2025-01-30
3	Heidi	Ungar	2025-01-30
4	Paul	Kumar	2025-01-30
5	Heidi	Ungar	2025-01-30
6	Vern	Werner	2025-01-30

```

1. SELECT * FROM employees
2. ORDER BY last_name;

```

employee_id	first_name	last_name	hire_date
1	Pat	Adams	2025-01-30
2	Herm	Urgon	2025-01-30
3	Heidi	Ungar	2025-01-30
4	Paul	Kumar	2025-01-30
5	Heidi	Ungar	2025-01-30
6	Vern	Werner	2025-01-30

(3)

```

1. SELECT * FROM employees
2. ORDER BY last_name DESC;

1. SELECT * FROM employees
2. ORDER BY last_name ASC;

```

(4)

```

1. SELECT * FROM employees
2. ORDER BY last_name DESC;

1. SELECT * FROM employees
2. ORDER BY last_name ASC;

```

(5)

```

1. SELECT * FROM employees
2. ORDER BY last_name ASC, employee_id DESC;

```

if two
last_name is
some
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

Screenshot of MySQL Workbench showing a SELECT query: `1 * SELECT * FROM employees;`. The results show all 147 rows of the employees table. The output pane shows the execution plan with 49 separate SELECT statements for each row, indicating a full table scan for every row.

②

Screenshot of MySQL Workbench showing the same SELECT query with a `LIMIT 2` clause added: `1 * SELECT * FROM employees;` and `2 * LIMIT 2;`. The results now show only the first two rows of the employees table. The output pane shows the execution plan with only 2 SELECT statements, indicating a more efficient query execution.

③

Screenshot of MySQL Workbench showing a modified query: `1 * SELECT * FROM employees` and `2 * ORDER BY last_name DESC LIMIT 2;`. The results show the top two rows ordered by last name in descending order. The output pane shows the execution plan with 2 SELECT statements, similar to the previous screenshot.

④

Screenshot of MySQL Workbench showing a query with a larger offset: `1 * SELECT * FROM employees` and `2 * LIMIT 3,3;`. The results show the 4th through 6th rows of the employees table. The output pane shows the execution plan with 3 SELECT statements, illustrating how LIMIT can be used for pagination.

UNION -

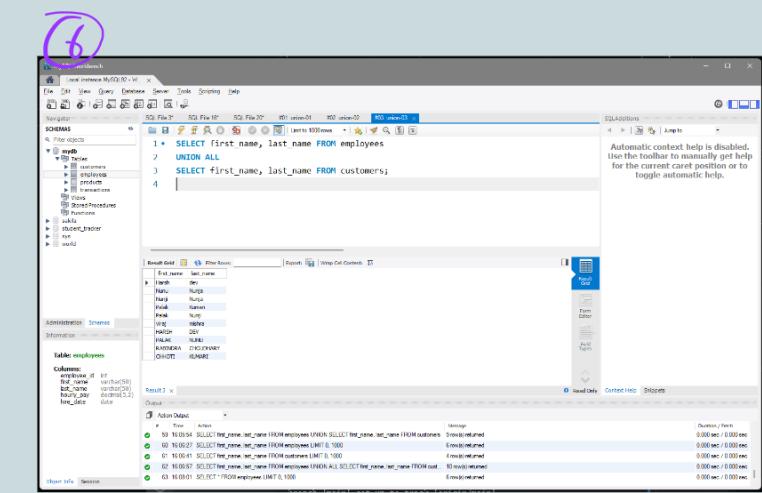
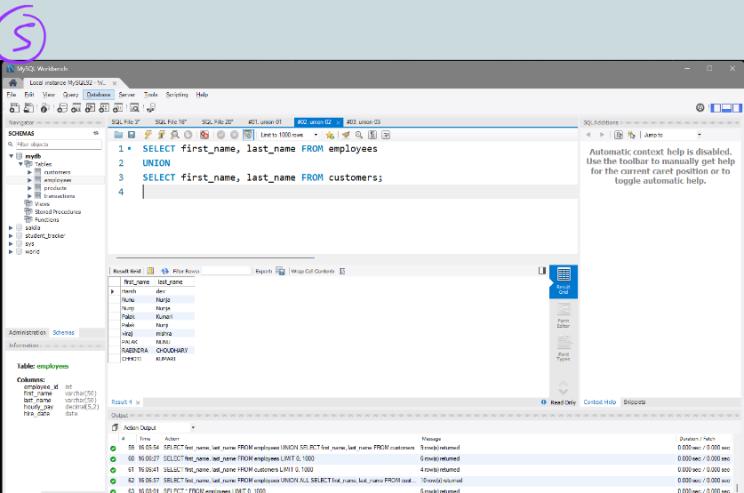
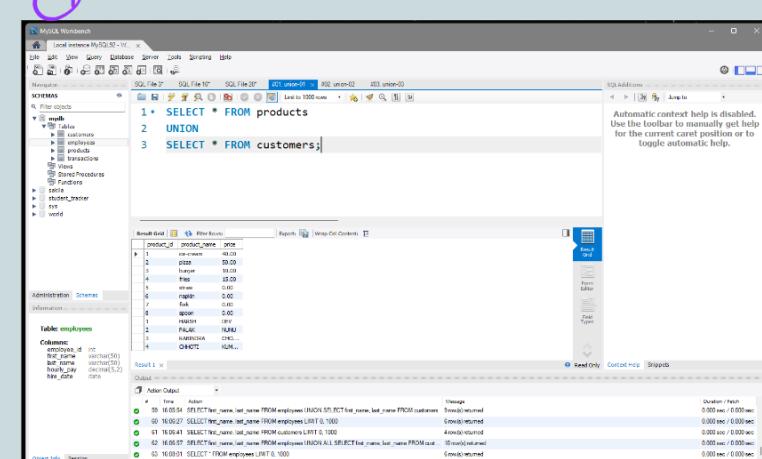
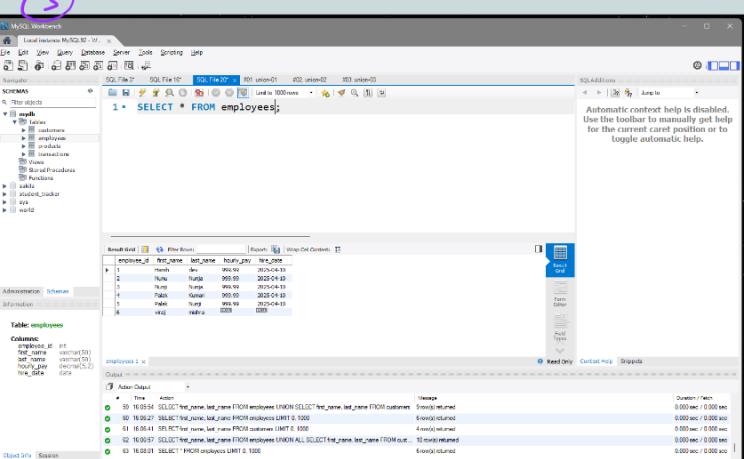
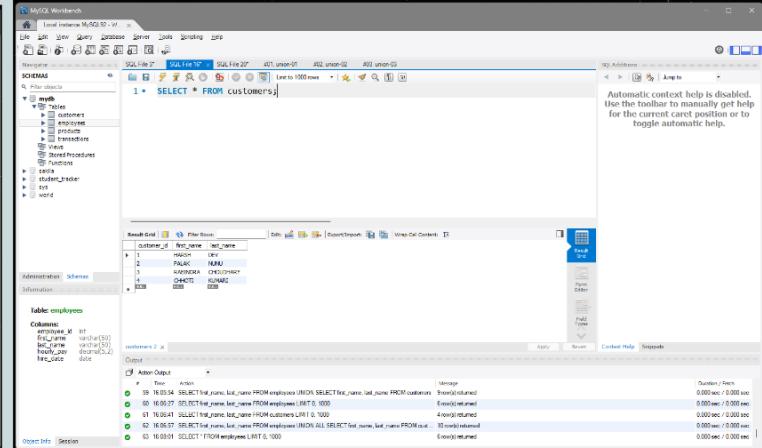
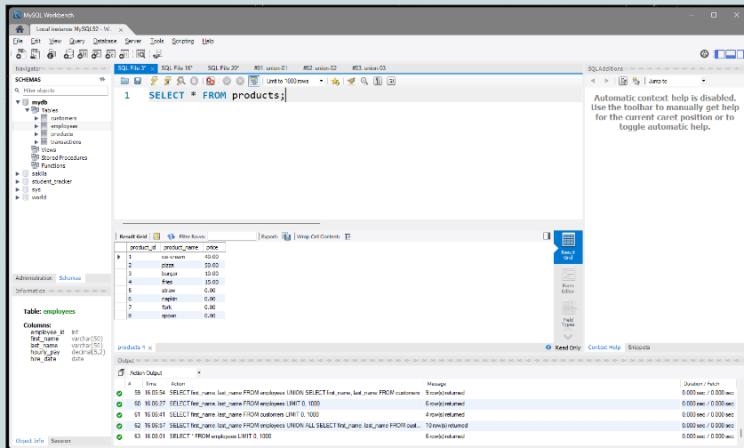
In SQL, the UNION operator combines the result sets of two or more SELECT statements into a single, distinct result set, removing any duplicate rows.

- **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:

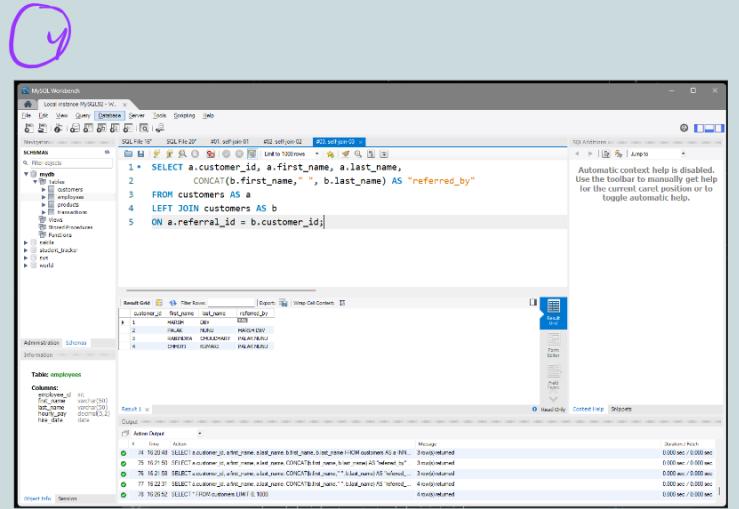
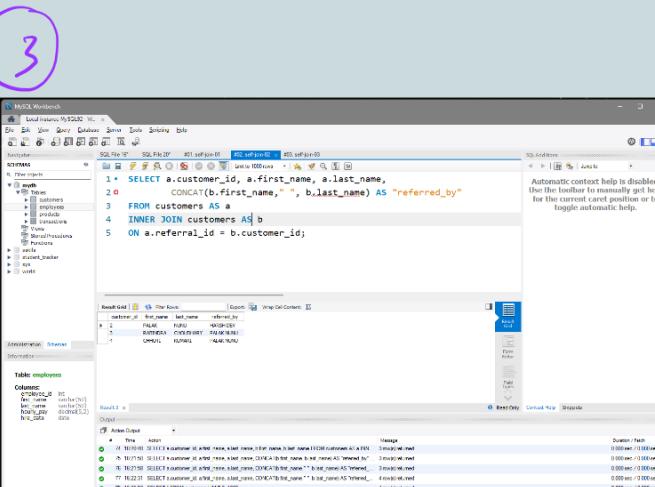
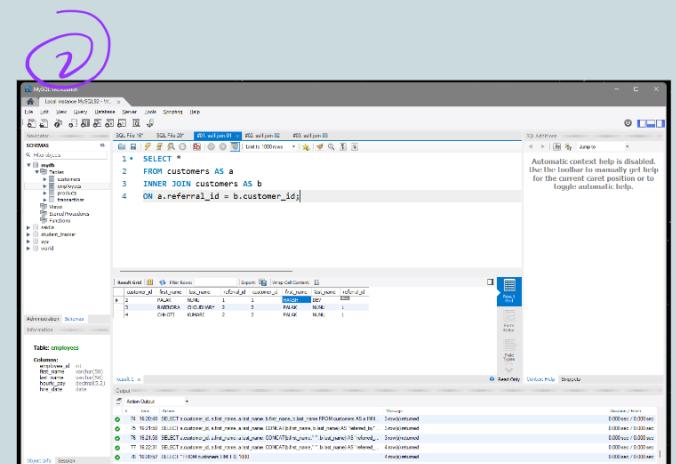
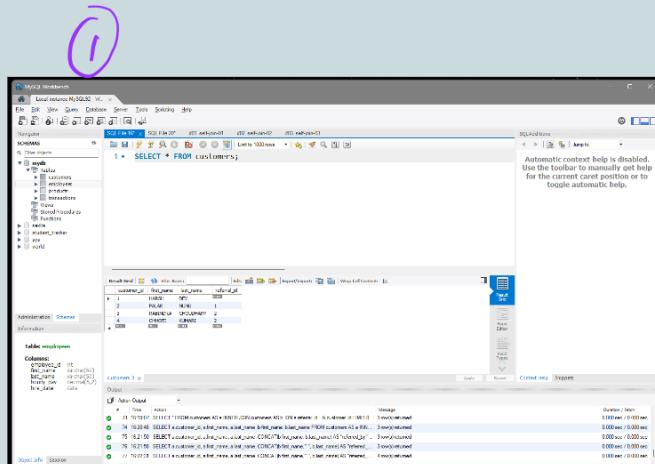
To use UNION, the SELECT statements must have the same number of columns, and those columns must have compatible data types.



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.



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

```
1 • SELECT * FROM employees;
```

emp_no	last_name	first_name	middle_name	hire_date
1	Koch	Ruth		2005-01-30
2	Koch	Norbert		2005-01-30
3	Padil	Ramona		2005-04-01
4	Padil	Harmen		2005-04-01
5	Abrahams			

②

```
1 • CREATE VIEW employee_attendance AS
2   SELECT first_name, last_name
3   FROM employees;
```



③

```
1 • SELECT * FROM employee_attendance;
```

first_name	last_name
Ruth	Koch
Norbert	Koch
Ramona	Padil
Harmen	Padil
	Abrahams



④

```
1 • DROP VIEW employee_attendance;
```

Indexes 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

MySQL Workbench - Local instance MySQL 5.7 - v18.03.0

1 • SELECT * FROM customers;

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Result Grid | Filter Rows | Import | Export | Open Cell Content

Table: customers

Columns:

- customer_id int(11) NOT NULL auto_increment
- first_name varchar(45)
- last_name varchar(45)
- email varchar(100)
- phone varchar(20)
- customer_since date
- last_update timestamp
- referral_id int(11)

Action Output

```
16:50:21 SHOW INDEXES FROM customers
16:50:23 CREATE INDEX last_name_idx ON customers(last_name)
16:50:28 SHOW INDEXES FROM customers
16:50:30 ALTER TABLE customers DROP INDEX last_name_idx
16:50:58 SHOW INDEXES FROM customers
```

2

MySQL Workbench - Local instance MySQL 5.7 - v18.03.0

1 • SHOW INDEXES FROM customers;

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Result Grid | Filter Rows | Import | Export | Open Cell Content

Table: customers

Columns:

- customer_id int(11) NOT NULL auto_increment
- first_name varchar(45)
- last_name varchar(45)
- email varchar(100)
- phone varchar(20)
- customer_since date
- last_update timestamp
- referral_id int(11)

Action Output

```
96 16:50:21 SHOW INDEXES FROM customers
97 16:50:23 CREATE INDEX first_name_idx ON customers(first_name)
98 16:50:28 SHOW INDEXES FROM customers
99 16:50:30 ALTER TABLE customers DROP INDEX first_name_idx
100 16:50:58 SHOW INDEXES FROM customers
```

3

MySQL Workbench - Local instance MySQL 5.7 - v18.03.0

1 • CREATE INDEX last_name_idx
ON customers(last_name);

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Result Grid | Filter Rows | Import | Export | Open Cell Content

Table: customers

Columns:

- customer_id int(11) NOT NULL auto_increment
- first_name varchar(45)
- last_name varchar(45)
- email varchar(100)
- phone varchar(20)
- customer_since date
- last_update timestamp
- referral_id int(11)

Action Output

```
16:50:21 SHOW INDEXES FROM customers
16:50:23 CREATE INDEX last_name_idx ON customers(last_name)
16:50:28 SHOW INDEXES FROM customers
16:50:30 ALTER TABLE customers DROP INDEX last_name_idx
16:50:58 SHOW INDEXES FROM customers
```

4

MySQL Workbench - Local instance MySQL 5.7 - v18.03.0

1 • SHOW INDEXES FROM customers;

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Result Grid | Filter Rows | Import | Export | Open Cell Content

Table: customers

Columns:

- customer_id int(11) NOT NULL auto_increment
- first_name varchar(45)
- last_name varchar(45)
- email varchar(100)
- phone varchar(20)
- customer_since date
- last_update timestamp
- referral_id int(11)

Action Output

```
96 16:50:21 SHOW INDEXES FROM customers
97 16:50:23 CREATE INDEX last_name_first_name_idx ON customers(last_name,first_name)
98 16:50:28 SHOW INDEXES FROM customers
99 16:50:30 ALTER TABLE customers DROP INDEX last_name_idx
100 16:50:58 SHOW INDEXES FROM customers
```

5

MySQL Workbench - Local instance MySQL 5.7 - v18.03.0

1 • SELECT * FROM customers
2 WHERE last_name = "NUNO";

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Result Grid | Filter Rows | Import | Export | Open Cell Content

Table: customers

Columns:

- customer_id int(11) NOT NULL auto_increment
- first_name varchar(45)
- last_name varchar(45)
- email varchar(100)
- phone varchar(20)
- customer_since date
- last_update timestamp
- referral_id int(11)

Action Output

```
16:50:21 SHOW INDEXES FROM customers
16:50:23 CREATE INDEX last_name_idx ON customers(last_name)
16:50:28 SHOW INDEXES FROM customers
16:50:30 ALTER TABLE customers DROP INDEX last_name_idx
16:50:58 SHOW INDEXES FROM customers
```

6

MySQL Workbench - Local instance MySQL 5.7 - v18.03.0

1 • CREATE INDEX last_name_first_name_idx
2 ON customers(last_name, first_name);

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Result Grid | Filter Rows | Import | Export | Open Cell Content

Table: customers

Columns:

- customer_id int(11) NOT NULL auto_increment
- first_name varchar(45)
- last_name varchar(45)
- email varchar(100)
- phone varchar(20)
- customer_since date
- last_update timestamp
- referral_id int(11)

Action Output

```
96 16:50:21 SHOW INDEXES FROM customers
97 16:50:23 CREATE INDEX last_name_idx ON customers(last_name)
98 16:50:28 SHOW INDEXES FROM customers
99 16:50:30 ALTER TABLE customers DROP INDEX last_name_idx
100 16:50:58 SHOW INDEXES FROM customers
```

Multi-column Index

7

MySQL Workbench - Local instance MySQL 5.7 - v18.03.0

1 • SHOW INDEXES FROM customers;

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Result Grid | Filter Rows | Import | Export | Open Cell Content

Table: customers

Columns:

- customer_id int(11) NOT NULL auto_increment
- first_name varchar(45)
- last_name varchar(45)
- email varchar(100)
- phone varchar(20)
- customer_since date
- last_update timestamp
- referral_id int(11)

Action Output

```
96 16:50:21 SHOW INDEXES FROM customers
97 16:50:23 CREATE INDEX last_name_idx ON customers(last_name)
98 16:50:28 SHOW INDEXES FROM customers
99 16:50:30 ALTER TABLE customers DROP INDEX last_name_idx
100 16:50:58 SHOW INDEXES FROM customers
```

8

MySQL Workbench - Local instance MySQL 5.7 - v18.03.0

1 • ALTER TABLE customers
2 DROP INDEX last_name_idx;
3
4
5 • SHOW INDEXES FROM customers;

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Result Grid | Filter Rows | Import | Export | Open Cell Content

Table: customers

Columns:

- customer_id int(11) NOT NULL auto_increment
- first_name varchar(45)
- last_name varchar(45)
- email varchar(100)
- phone varchar(20)
- customer_since date
- last_update timestamp
- referral_id int(11)

Action Output

```
96 16:50:21 SHOW INDEXES FROM customers
97 16:50:23 CREATE INDEX last_name_idx ON customers(last_name)
98 16:50:28 SHOW INDEXES FROM customers
99 16:50:30 ALTER TABLE customers DROP INDEX last_name_idx
100 16:50:58 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.

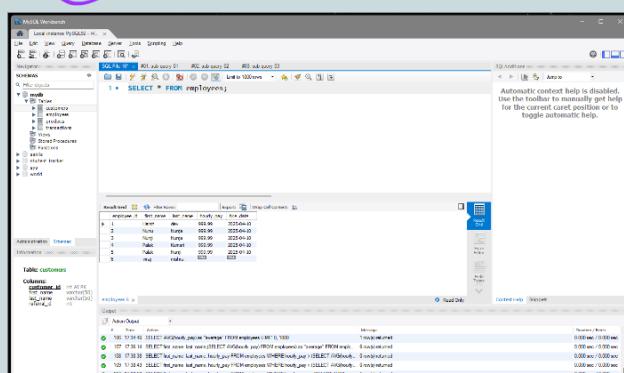
→ • Types:

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

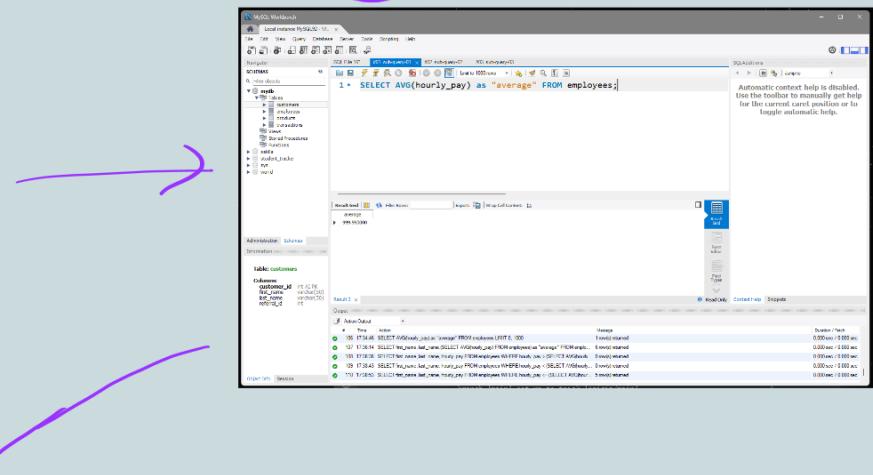
→ • Benefits:

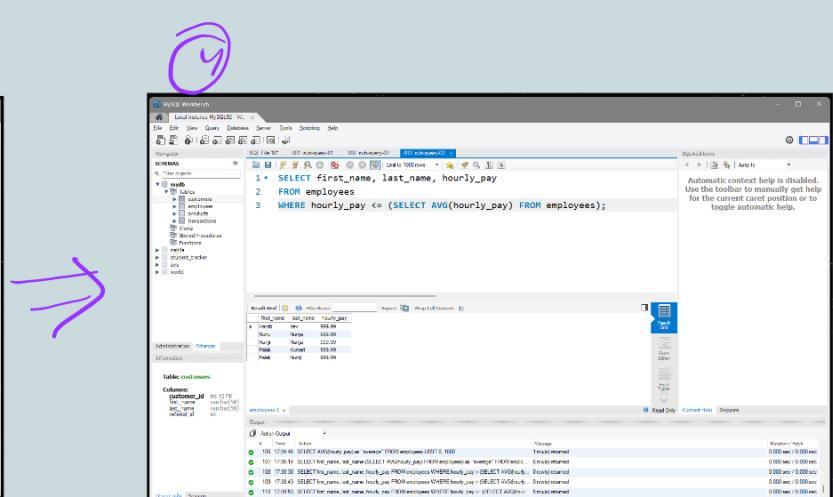
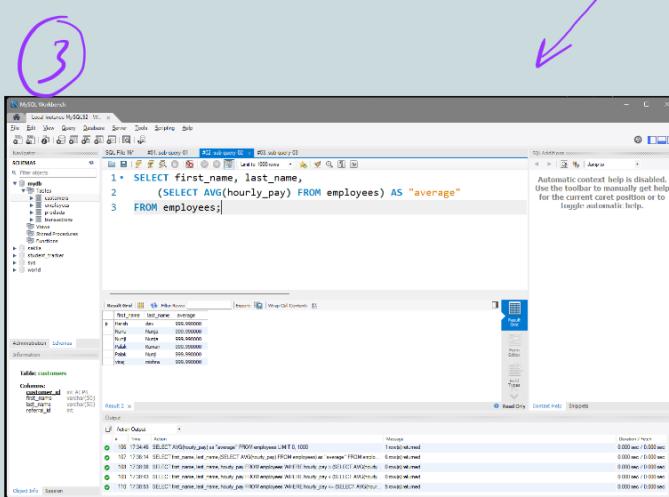
- **Simplified Complex Queries:** Subqueries can make complex queries more readable and easier to manage.
- **Improved Performance:** In some cases, subqueries can be more efficient than using multiple joins.
- **Flexibility:** Subqueries can be used to perform a wide variety of operations, making them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

①



②



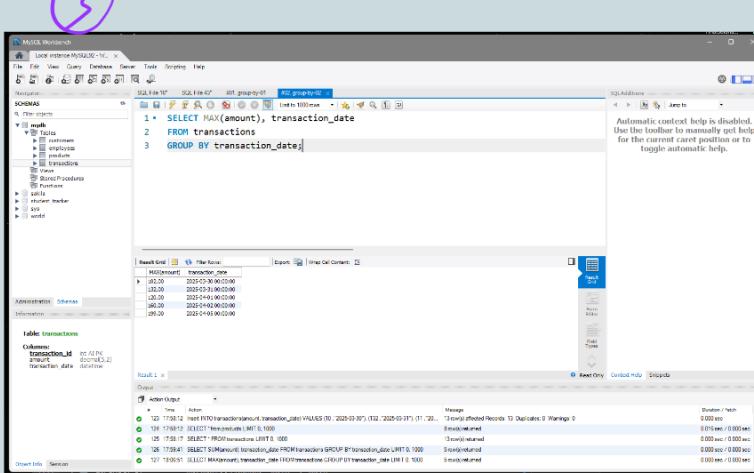
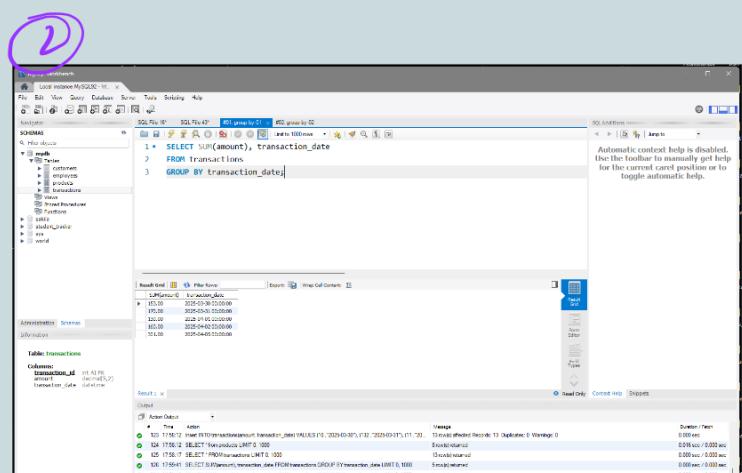
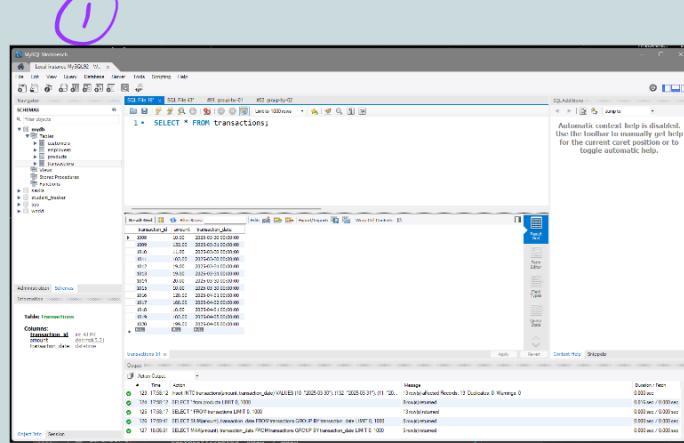


GROUP-BY

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.



#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

This screenshot shows the MySQL Workbench interface. A query editor window displays the following SQL code:

```
1. SELECT * FROM transactions;
```

The results grid shows a list of transactions with columns: transaction_id, amount, and transaction_date. The data includes various transaction IDs, amounts ranging from 10.00 to 350.50, and dates from 2023-03-20 to 2023-04-05.

②

This screenshot shows the MySQL Workbench interface. A query editor window displays the following SQL code:

```
1. SELECT sum(amount), transaction_date
2. FROM transactions
3. GROUP BY transaction_date WITH ROLLUP;
```

The results grid shows the same transaction data as the first screenshot, but with additional rows for subtotals and a grand total. The subtotal rows have a COUNT column value of 5, and the grand total row has a COUNT value of 13.

③

This screenshot shows the MySQL Workbench interface. A query editor window displays the following SQL code:

```
1. SELECT COUNT(amount), transaction_date
2. FROM transactions
3. GROUP BY transaction_date WITH ROLLUP;
```

The results grid shows the transaction data with subtotals and a grand total. The COUNT column values are 3, 3, 2, 1, 2, and 13 respectively. The output also includes a message indicating 13 rows returned.

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

```
1 -- ON DELETE SET NULL = When a FK is deleted, replace FK with NULL  
2 -- ON DELETE CASCADE = When a FK is deleted, delete row  
3  
4· SELECT * FROM transactions;
```

②

```
2 -- ON DELETE CASCADE = When a FK is deleted, delete row  
3  
4· CREATE TABLE transactions (  
5     transaction_id INT PRIMARY KEY,  
6     amount DECIMAL(5, 2),  
7     customer_id INT,  
8     order_date DATE,  
9     FOREIGN KEY (customer_id) REFERENCES customers(customer_id)  
10    ON DELETE SET NULL  
11 );  
12
```

③

```
1 -- ON DELETE SET NULL = When a FK is deleted, replace FK with NULL  
2 -- ON DELETE CASCADE = When a FK is deleted, delete row  
3  
4· ALTER TABLE transactions  
5 ADD CONSTRAINT fk_customer_id  
6 FOREIGN KEY(customer_id) REFERENCES customers(customer_id)  
7 ON DELETE SET NULL;
```

Create table with
"ON DELETE SET NULL"

④

```
1 -- ON DELETE SET NULL = When a FK is deleted, replace FK with NULL  
2 -- ON DELETE CASCADE = When a FK is deleted, delete row  
3  
4· DELETE FROM customers  
5 WHERE customer_id = 4;  
6· SELECT * FROM transactions;
```

⑤

transaction_id	amount	customer_id	order_date
1000	4.99	3	2023-01-01
1001	2.89	2	2023-01-01
1002	3.38	3	2023-01-02
1003	4.99	1	2023-01-02
1004	1.00	1	2023-01-03
1005	2.49	1	2023-01-03
1006	5.48	1	2023-01-03

Alter table

5

```

1 -- ON DELETE SET NULL = When a FK is deleted, replace FK with NULL
2 -- ON DELETE CASCADE = When a FK is deleted, delete row
3
4 ALTER TABLE transactions
5 ADD CONSTRAINT fk_transactions_id
6 FOREIGN KEY(customer_id) REFERENCES customers(customer_id)
7 ON DELETE CASCADE;

```

6

```

1 -- ON DELETE SET NULL = When a FK is deleted, replace FK with NULL
2 -- ON DELETE CASCADE = When a FK is deleted, delete row
3
4 DELETE FROM customers
5 WHERE customer_id = 4;
6
7 SELECT * FROM transactions;

```

transaction_id	amount	customer_id	order_date
1000	4.99	3	2023-01-01
1001	2.99	2	2023-01-01
1002	3.99	3	2023-01-02
1003	4.99	1	2023-01-02
1004	1.00	1	2023-01-03
1006	5.48	1	2023-01-03

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

```

1 DELIMITER //
2 CREATE PROCEDURE get_transactions()
3 BEGIN
4     SELECT * FROM transactions;
5 END //
6 DELIMITER ;

```

```

1 CALL get_transactions();

```

transaction_id	amount	customer_id	order_date
1000	4.99	3	2023-01-01
1001	2.99	2	2023-01-01
1002	3.99	3	2023-01-02
1003	4.99	1	2023-01-02
1004	1.00	1	2023-01-03
1006	5.48	1	2023-01-03

③

```

1 DROP PROCEDURE get_transactions;

```

④

```

DELIMITER $$

CREATE PROCEDURE find_employee(IN id INT)
BEGIN
    SELECT *
    FROM employees
    WHERE employee_id = id;
END $$

DELIMITER ;

```

The screenshot shows the MySQL Workbench interface with the SQL editor tab open. The code above creates a stored procedure named 'find_employee' that takes an integer parameter 'id'. It selects all columns from the 'employees' table where the 'employee_id' matches the input 'id'. The 'Output' pane at the bottom shows the execution of the CREATE PROCEDURE statement.

```

CALL find_employee(2);

```

The screenshot shows the MySQL Workbench interface with the SQL editor tab open. The code above calls the 'find_employee' stored procedure with the parameter value '2'. The results are displayed in the Result Grid pane, showing one row with employee_id 2, first_name 'Rversible', last_name 'Pineapple', hire_date '2008-05-14', and rate 120000.

⑤

```

DROP PROCEDURE find_employee;

```

The screenshot shows the MySQL Workbench interface with the SQL editor tab open. The code above drops the 'find_employee' stored procedure. The 'Output' pane at the bottom shows the execution of the DROP PROCEDURE statement.

#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.

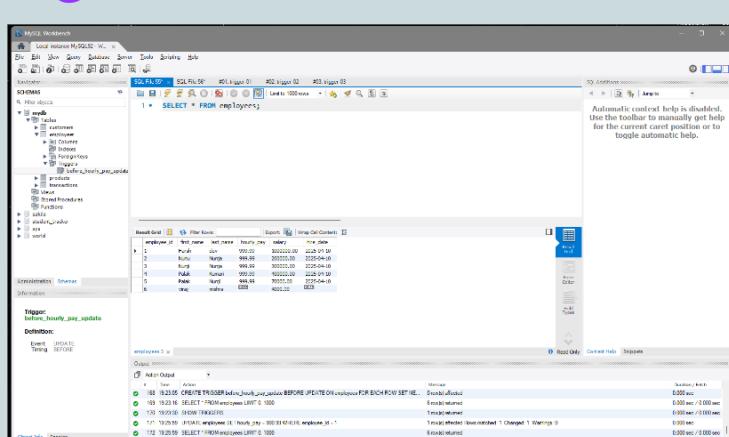
- **Automating Tasks:**

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

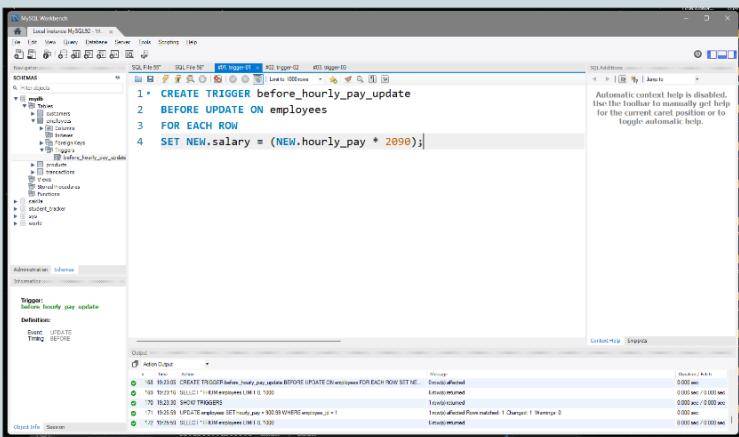
→ Types of Triggers:

- **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

Automatic context help is disabled, but the toolbar normally gets help for the current caret position or to toggle automatic help.

1. SHOW TRIGGERS;

```

1 UPDATE employees
2 SET hourly_pay = 900.00
3 WHERE employee_id = 1;
4 SELECT * FROM employees;

```

Trigger: before_hourly_pay_update

Definition:

```

Event: UPDATE
Time: BEFORE

```

Script

```

1 USE [master]
2 GO
3 CREATE TRIGGER [before_hourly_pay_update] BEFORE UPDATE ON [employees] FOR EACH ROW SET NEW.[hourly_pay] = 900.00 WHERE employee_id = 1
4 GO
5 USE [master]
6 GO
7 EXEC sp_helptext [before_hourly_pay_update]
8 GO
9 USE [master]
10 GO
11 EXEC sp_helptrigger [before_hourly_pay_update]
12 GO
13 EXEC sp_depends [before_hourly_pay_update]
14 GO
15 EXEC sp_depends [before_hourly_pay_update]
16 GO
17 EXEC sp_depends [before_hourly_pay_update]
18 GO

```

④

Automatic context help is disabled, but the toolbar normally gets help for the current caret position or to toggle automatic help.

1 UPDATE employees

2 SET hourly_pay = 900.00

3 WHERE employee_id = 1;

4 SELECT * FROM employees;

Result Grid

employee_id	first_name	last_name	hourly_pay	is_disco
1	Nana	Nancy	900.00	0
2	Mark	Marky	20000.00	0
3	John	Johny	30000.00	0
4	David	Davidy	40000.00	0
5	Adam	Adamy	50000.00	0

Information: Triggers

Trigger: before_hourly_pay_update

Definition:

```

Event: UPDATE
Time: BEFORE

```

Output: Action Output

Action	Message	Duration
1. 10:19:05 CREATE_TRIGGER[before_hourly_pay_update] BEFORE UPDATE ON [employees] FOR EACH ROW SET NEW.[hourly_pay] = 900.00 WHERE employee_id = 1	Created trigger	0.000 sec / 0.000 ms
2. 10:19:05 EXECUTE[before_hourly_pay_update]	Triggered	0.000 sec / 0.000 ms
3. 10:19:05 UPDATE[before_hourly_pay_update]	Triggered	0.000 sec / 0.000 ms
4. 10:19:05 SELECT[before_hourly_pay_update]	Triggered	0.000 sec / 0.000 ms
5. 10:19:05 EXECUTE[before_hourly_pay_update]	Triggered	0.000 sec / 0.000 ms