DATABASE

A DATABASE IS JUST A PLACE TO STORE INFORMATION IN AN ORGANIZED WAY SO WE CAN FIND AND USE IT EASILY LATER.

- ✅ THE NOTEBOOK = YOUR DATABASE
- ✅ EACH PAGE = A TABLE
- ✅ EACH ROW = ONE STUDENTS'S INFO (A RECORD)
- ✅ EACH COLUMN = A DETAIL ABOUT THE STUDENT (NAME, ROLL_NO, CLASS)

TYPES OF DATABASES

A DATABASE IS LIKE A PLACE TO KEEP YOUR STUFF ORGANIZED, RIGHT?

BUT THERE ARE DIFFERENT KINDS OF BOXES (TYPES)

FOR KEEPING THAT STUFF DEPENDING ON HOW YOU WANT TO ORGANIZE IT.

RELATIONAL DATABASE (THE "NEAT BOX")

TOOLS: MYSQL, POSTGRESQL

NOSQL DATABASE (THE "FLEXIBLE BOX")

TOOLS: MONGODB,CASSANDRA

RELATIONAL DATABASE MANAGEMENT SYSTEM

✅ **RELATIONAL** = USES TABLES (LIKE GRIDS OR SPREADSHEETS)

✅ **DATABASE** = A PLACE TO STORE INFORMATION

✅ **MANAGEMENT SYSTEM** = A TOOL TO ORGANIZE AND CONTROL THE DATA

SO, AN RDBMS IS A SYSTEM THAT HELPS US STORE, ORGANIZE, AND CONNECT DATA IN TABLES.

SQL (STRUCTURED QUERY LANGUAGE)

IT IS A STANDARD LANGUAGE USED TO MANAGE AND INTERACT WITH DATABASES.

TYPES

✅ DATA DEFINITION LANGUAGE (DDL)

✅ DATA MANIPULATION LANGUAGE (DML)

✅ DATA CONTROL LANGUAGE (DCL)

✅ TRANSACTION CONTROL LANGUAGE (TCL)

✅ DATA QUERY LANGUAGE (DQL)

INSTALLATION OF MYSQL SERVER & WORKBENCH

WINDOWS

MySQL Installer

MySQL. Installer
Adding Community

- Choosing a Setup Type
- Download
- Installation
- Installation Complete

Choosing a Setup Type

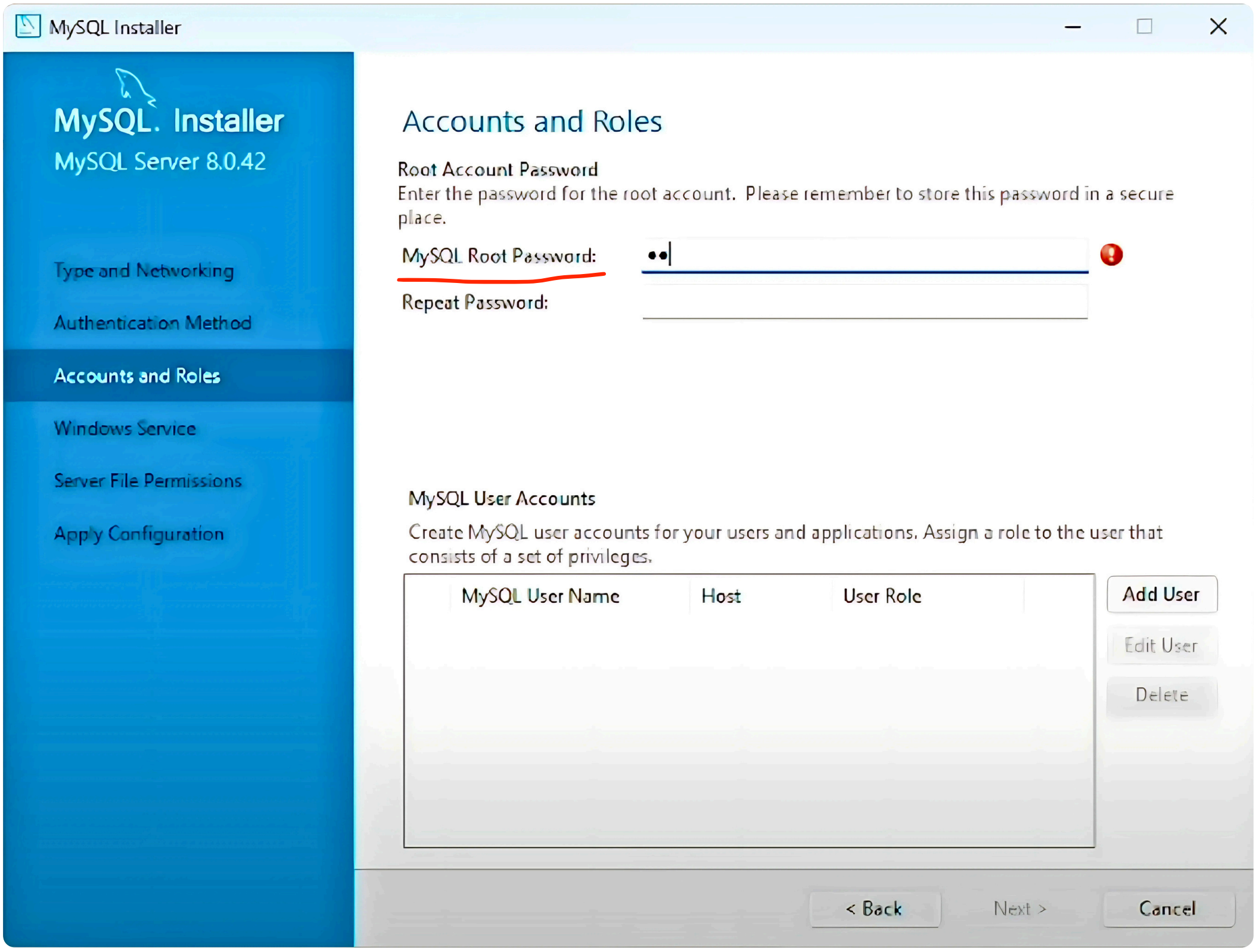
Please select the Setup Type that suits your use case.

- ○ **Server only**
  Installs only the MySQL Server product.
- ○ **Client only**
  Installs only the MySQL Client products, without a server.
- ◉ **Full**
  Installs all included MySQL products and features.
- ○ **Custom**
  Manually select the products that should be installed on the system.

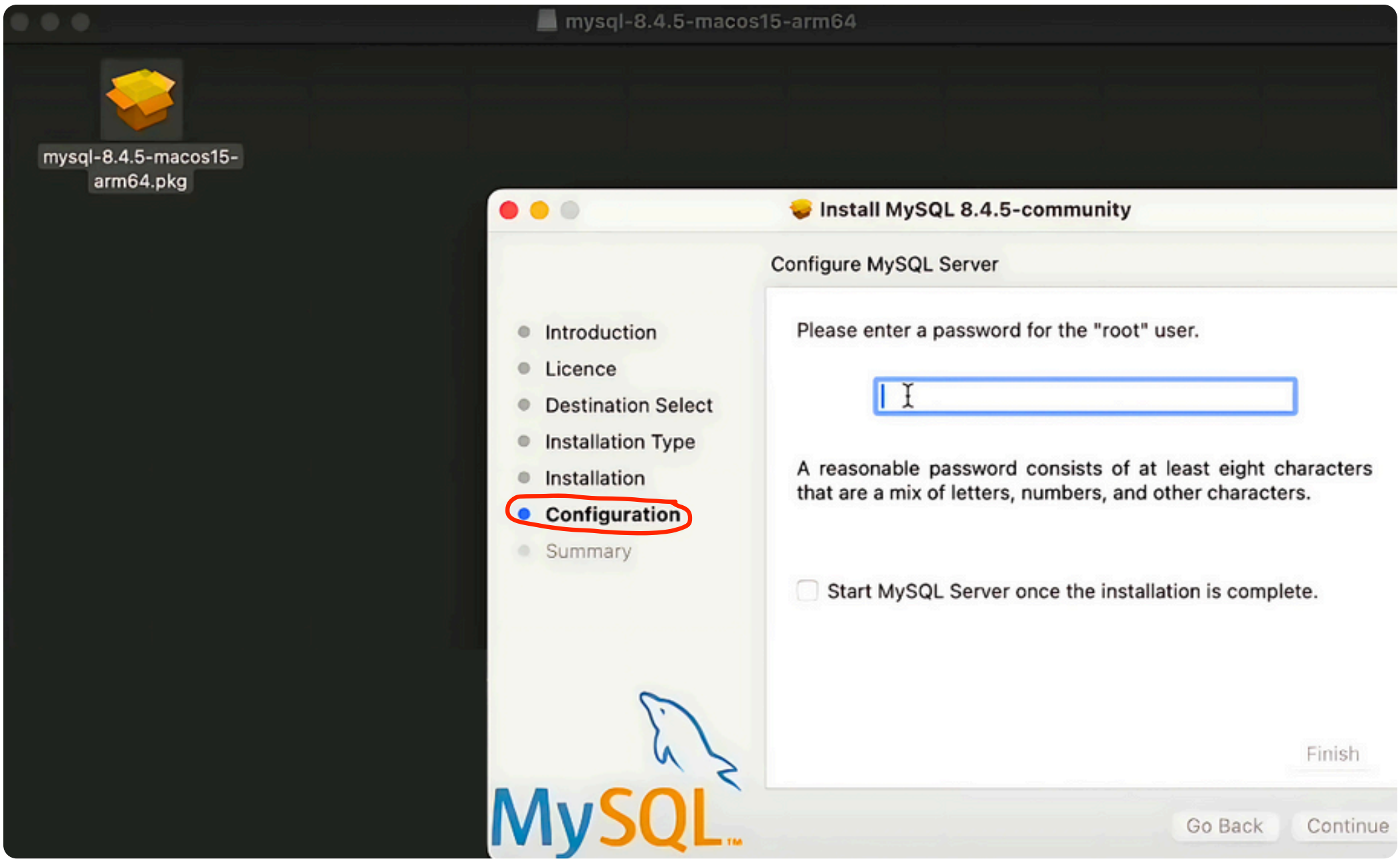
Setup Type Description

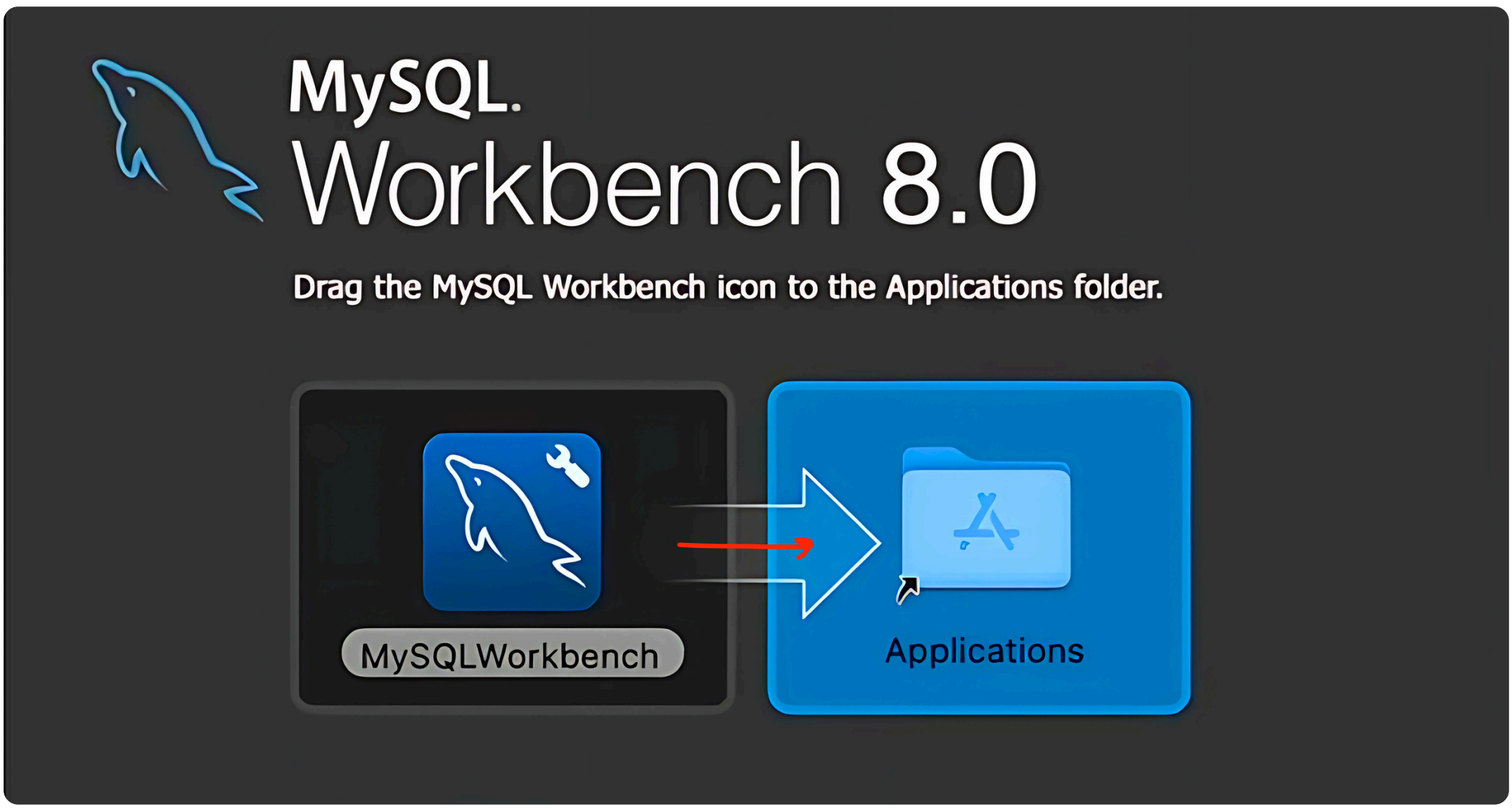
Installs all of the products available in this catalog including MySQL Server, MySQL Shell, MySQL Router, MySQL Workbench, documentation, samples and examples and more.

Next > Cancel


MySQL Installer
MySQL. Installer
MySQL Server 8.0.42
Type and Networking
Authentication Method
Accounts and Roles
Windows Service
Server File Permissions
Apply Configuration
Accounts and Roles
Root Account Password
Enter the password for the root account. Please remember to store this password in a secure place.
MySQL Root Password:
Repeat Password:
MySQL User Accounts
Create MySQL user accounts for your users and applications. Assign a role to the user that consists of a set of privileges.
MySQL User Name
Host
User Role
Add User
Edit User
Delete
< Back
Next >
Cancel

MAC





DATA TYPES IN SQL

Data Type	Description	Minimum Value	Maximum Value
TINYINT	Very small whole number	-128	127
TINYINT UNSIGNED	Only positive small numbers	0	255
SMALLINT	Small whole number	-32,768	32,767
SMALLINT UNSIGNED	Only positive small numbers	0	65,535
MEDIUMINT	Medium-size whole number	-8,388,608	8,388,607
MEDIUMINT UNSIGNED	Only positive medium numbers	0	16,777,215
INT / INTEGER	Standard whole number	-2,147,483,648	2,147,483,647
INT UNSIGNED	Only positive whole number	0	4,294,967,295
BIGINT	Very large whole number	-9,223,372,036,854,770,000	9,223,372,036,854,770,000
BIGINT UNSIGNED	Only positive large number	0	18,446,744,073,709,500,000
FLOAT	Approximate number with decimals	~ -3.4E38	~ 3.4E38
DOUBLE	Bigger, more precise than FLOAT	~ -1.7E308	~ 1.7E308

DATE	Just the **date**	YYYY-MM-DD	22 Apr 2025
TIME	Just the **time of day**	HH:MM:SS	14:30:00
DATETIME	**Date and time** together	YYYY-MM-DD HH:MM:SS	2025-04-22 14:30:00
YEAR	Only the **year**	YYYY	2025

CHAR(n)	Fixed-length text	Always exactly n chars	'Hi ' (with spaces)
VARCHAR(n)	Variable-length text	Up to n chars	'Hello'
TEXT	Large text	No strict limit (varies)	Long paragraph
TINYTEXT	Very small text	Max ~255 characters	'Short msg'
MEDIUMTEXT	Medium-sized text	Max ~16 million chars	Article or blog
LONGTEXT	Very large text	Max ~4 billion chars	Full novel

✅ DATA DEFINITION LANGUAGE (DDL)

IT'S USED TO DEFINE AND MANAGE THE STRUCTURE OF YOUR DATABASE.

1. ***CREATE***
2. ***ALTER***
3. ***DROP***
4. ***TRUNCATE***
5. ***RENAME***

COMMAND	WHAT IT DOES	REAL-WORLD EXAMPLE
CREATE	MAKES A NEW TABLE/DATABASE	BUILD A NEW TOY BOX
ALTER	CHANGES THE STRUCTURE (ADD/REMOVE COLUMN)	ADD DRAWER TO TOY BOX
DROP	DELETES THE ENTIRE TABLE	SMASH THE TOY BOX
TRUNCATE	EMPTIES ALL DATA INSIDE THE TABLE	THROW OUT ALL TOYS, KEEP THE BOX
RENAME	CHANGES THE NAME OF THE TABLE	GIVE YOUR TOY BOX A NEW NAME

1. CREATE

BUILDING A BRAND-NEW TOY BOX (A TABLE)

✅ ***SYNTAX:***

```
CREATE TABLE Students (
   ID INT,
   Name VARCHAR(50),
   Age INT
);
```

2. ALTER

YOU ALREADY BUILT A TOY BOX, AND NOW YOU WANT TO

CHANGE SOMETHINGS.

✅ ***SYNTAX:***

```
ALTER TABLE Students ADD Grade VARCHAR(2);
```

3. DROP

THIS IS LIKE SMASHING YOUR BLOCK TOWER TO THE GROUND

IT'S GONE FOREVER!

✅ ***SYNTAX:***

```
DROP TABLE Students;
```

4. TRUNCATE

IT WILL THROW ALL THE TOYS OUT BUT KEEP THE BOX.

✅ ***SYNTAX:***

```
TRUNCATE TABLE Students;
```

5. RENAME

GIVING IT A NEW NAME

✅ ***SYNTAX:***

```
RENAME TABLE Students TO Kids;
```

✅ DATA MANIPULATION LANGUAGE (DML)

DML IS ABOUT PLAYING WITH THE TOYS INSIDE

ADDING THEM, CHANGING THEM, OR TAKING THEM OUT!

1. ***INSERT – ADD NEW TOYS***
2. ***UPDATE – FIX OR CHANGE TOYS***
3. ***DELETE – REMOVE A TOY***

1. INSERT

YOU GOT A NEW TOY AND WANT TO ADD IT TO YOUR TOY BOX.

✅ SYNTAX:

```
INSERT INTO ToyBox (ToyID, ToyName, ToyType)
VALUES (1, 'Teddy Bear', 'Stuffed');
```

YOU ADDED A TOY WITH:

- ID: 1
- NAME: TEDDY BEAR
- TYPE: STUFFED

2. UPDATE

CHANGE SOMETHING ABOUT A TOY

✅ SYNTAX:

```
UPDATE ToyBox
SET ToyType = 'Plush'
WHERE ToyID = 1;
```

YOU FOUND THE TOY WITH TOYID = 1.

YOU CHANGED ITS TYPE FROM "STUFFED" TO "PLUSH".

3. DELETE

THROW A TOY AWAY

✅ SYNTAX:

```
DELETE FROM ToyBox
WHERE ToyID = 1;
```

YOU FOUND THE TOY WITH ID 1 AND TOOK IT OUT OF THE BOX.

✅ DATA CONTROL LANGUAGE (DCL)

IF DDL WAS FOR BUILDING YOUR TOY BOX

DML WAS FOR PLAYING WITH THE TOYS INSIDE

THEN DCL IS ABOUT GIVING PERMISSION TO OTHERS:

✅ WHO CAN SEE YOUR TOY BOX?
✅ WHO CAN ADD OR REMOVE TOYS?
✅ WHO CAN'T TOUCH ANYTHING AT ALL?

IT'S LIKE BEING THE TOY BOX BOSS 👑

1. GRANT – GIVE SOMEONE PERMISSION
2. REVOKE – TAKE THAT PERMISSION AWAY

1. GRANT

LET A FRIEND USE YOUR TOY BOX

✅ SYNTAX:

```
GRANT SELECT, INSERT ON ToyBox TO Sam;
```

- YOU GAVE SAM PERMISSION TO:
  - SELECT → LOOK AT THE TOYS IN YOUR BOX
  - INSERT → ADD NEW TOYS TO YOUR BOX

2. REVOKE

TAKE PERMISSION BACK

✅ SYNTAX:

```
REVOKE INSERT ON ToyBox FROM Sam;
```

YOU TOLD SAM: "YOU CAN NO LONGER ADD TOYS,

BUT HE CAN STILL LOOK AT THEM."

FULL TOY BOX SQL STORY

1. YOU BUILD THE TOY BOX → CREATE
2. YOU ADD TOYS → INSERT
3. YOU PEEK INSIDE → SELECT
4. YOU CHANGE TOYS → UPDATE
5. YOU REMOVE TOYS → DELETE
6. YOU LET FRIENDS PLAY → GRANT
7. YOU STOP THEM IF NEEDED → REVOKE

✅ TRANSACTION CONTROL LANGUAGE (TCL)

IMAGINE YOU'RE PLAYING WITH YOUR TOY BOX AND DOING MANY THINGS ONE AFTER ANOTHER:

- ADDING TOYS
- REMOVING SOME
- CHANGING A FEW

BUT YOU ONLY WANT TO SAVE THESE CHANGES IF EVERYTHING GOES WELL.
IF SOMETHING GOES WRONG – LIKE YOU DROPPED A TOY OR BROKE ONE – YOU WANT TO UNDO EVERYTHING.

THAT'S WHERE TCL HELPS!

COMMIT - SAVE ALL YOUR CHANGES

ROLLBACK - UNDO YOUR CHANGES

SAVEPOINT - SET A CHECKPOINT TO GO BACK TO

1.COMMIT

SAVE YOUR WORK

✅ EXAMPLE:

```
INSERT INTO ToyBox VALUES (1, 'Car', 'Vehicle');
INSERT INTO ToyBox VALUES (2, 'Doll', 'Stuffed');
COMMIT;
```

2.ROLLBACK

UNDO EVERYTHING

✅ EXAMPLE:

```
INSERT INTO ToyBox VALUES (3, 'Robot', 'Electronic');
INSERT INTO ToyBox VALUES (4, 'Ball', 'Outdoor');
ROLLBACK;
```

3.SAVEPOINT

SET A CHECKPOINT

✅ EXAMPLE:

```
INSERT INTO ToyBox VALUES (5, 'Puzzle', 'Brain');
SAVEPOINT Save1;

INSERT INTO ToyBox VALUES (6, 'Yo-Yo', 'Outdoor');
INSERT INTO ToyBox VALUES (7, 'RC Car', 'Electronic');

ROLLBACK TO Save1;
```

✅ DATA QUERY LANGUAGE (DQL)

SELECT - ASK FOR WHATS INSIDE THE TOY BOX.

EXAMPLES: -

```
SELECT * FROM ToyBox;
SELECT ToyName FROM ToyBox;
SELECT * FROM ToyBox WHERE ToyType = 'Vehicle';
SELECT COUNT(*) FROM ToyBox;
```

QUERY EXAMPLE	WHAT IT DOES
SELECT * FROM ToyBox;	SHOWS ALL THE TOYS
SELECT ToyName FROM ToyBox;	SHOWS ONLY TOY NAMES
WHERE ToyType = 'Vehicle';	FILTERS ONLY VEHICLE TOYS
ORDER BY ToyName;	SORTS TOYS BY NAME
COUNT(*)	COUNTS TOTAL NUMBER OF TOYS
GROUP BY ToyType	GROUPS TOYS BY THEIR TYPE

HAVING

THE HAVING CLAUSE IS USED TO FILTER GROUPS CREATED BY THE GROUP BY CLAUSE. IT'S SIMILAR TO WHERE, BUT WHERE FILTERS ROWS BEFORE GROUPING, WHILE HAVING FILTERS GROUPS AFTER GROUPING.

```
SELECT customer, SUM(amount) AS total_spent
FROM sales_data
GROUP BY customer
HAVING SUM(amount) > 30;
```

SQL FUNCTIONS

AGGREGATE FUNCTIONS

Function	Description
SUM()	Adds up all the values
AVG()	Calculates average
COUNT()	Counts the number of values
MAX()	Finds the maximum value
MIN()	Finds the minimum value

SCALAR FUNCTIONS

Function	Description
UPPER(str)	Converts to uppercase
LOWER(str)	Converts to lowercase
LENGTH(str)	Returns number of characters

DATE FUNCTIONS

Function	Description
NOW()	Current date and time
CURDATE()	Current date
YEAR(date)	Extracts year from date
MONTH(date)	Extracts month from date

SQL JOINS

Join Type	Description
INNER JOIN	Returns matching rows from both tables
LEFT JOIN	Returns all rows from the left table and matching rows from the right
RIGHT JOIN	Returns all rows from the right table and matching rows from the left
FULL OUTER JOIN	Returns all rows when there is a match in one of the tables
CROSS JOIN	Returns the **cartesian product** (all combinations)
SELF JOIN	A table is joined with itself

1. INNER JOIN

```
SELECT e.name, d.dept_name
FROM employees e
INNER JOIN departments d ON e.dept_id = d.dept_id;
```

2. LEFT JOIN

```
SELECT e.name, d.dept_name
FROM employees e
LEFT JOIN departments d ON e.dept_id = d.dept_id;
```

3. RIGHT JOIN

```
SELECT e.name, d.dept_name
FROM employees e
RIGHT JOIN departments d ON e.dept_id = d.dept_id;
```

4. FULL OUTER JOIN

```
SELECT e.name, d.dept_name
FROM employees e
FULL OUTER JOIN departments d ON e.dept_id = d.dept_id;
```

5. CROSS JOIN

```
SELECT e.name, d.dept_name
FROM employees e
CROSS JOIN departments d;
```

6. SELF JOIN

```
SELECT e.name AS employee,
m.name AS manager
FROM
  employees e
LEFT JOIN
  employees m ON e.manager_id = m.emp_id;
```

SQL KEYS

Key Type	Purpose
Primary Key	Uniquely identifies each row in a table
Foreign Key	Links one table to another
Unique Key	Ensures all values in a column are unique (except NULLs)
Composite Key	Combines two or more columns to uniquely identify a row

PRIMARY KEY

```
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    name VARCHAR(50),
    age INT);
```

FOREIGN KEY

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

UNIQUE KEY

```
CREATE TABLE teachers (
    teacher_id INT PRIMARY KEY,
    email VARCHAR(100) UNIQUE
);
```

COMPOSITE KEY

```
CREATE TABLE enrollments (
    student_id INT,
    course_id INT,
    PRIMARY KEY (student_id, course_id)
);
```

SQL CONSTRAINTS

Constraint	Description
NOT NULL	Column cannot have a NULL value
CHECK	Ensures the values meet a condition
DEFAULT	Sets a default value if none is provided

NOT NULL

```
CREATE TABLE users (
    user_id INT PRIMARY KEY,
    username VARCHAR(50) NOT NULL
);
```

DEFAULT

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    status VARCHAR(20) DEFAULT 'Pending'
);
```

CHECK

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

SQL VIEW

- ***A CUSTOM LENS TO LOOK AT DATA***
- ***A SAVED QUERY THAT ACTS LIKE A TABLE***

emp_id	name	dept	salary
1	Alice	HR	60000
2	Bob	IT	75000
3	Charlie	HR	58000

```
CREATE VIEW hr_employees AS
SELECT name, salary
FROM employees
WHERE dept = 'HR';


SELECT * FROM hr_employees;
```

SQL INDEX

AN INDEX IS LIKE THE INDEX IN A BOOK

IT HELPS YOU FIND DATA FASTER IN A TABLE.

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);


ALTER TABLE table_name
DROP INDEX index_name;
```

STORED PROCEDURE IN SQL

A STORED PROCEDURE IS A NAMED SET OF SQL STATEMENTS

THAT YOU SAVE IN THE DATABASE AND CAN CALL LATER.

OneShot Querie

```
create database xyz;

use xyz;



CREATE TABLE Students (

ID INT,

Name VARCHAR(50),

Age INT

);



ALTER TABLE Students

ADD Grade VARCHAR(2);



ALTER TABLE Students

drop Grade;



RENAME TABLE Students TO Kids;



INSERT INTO Kids (ID, Name, Age)

VALUES (1, 'Teddy Bear', 22);



TRUNCATE TABLE Kids;

drop table Kids;



CREATE TABLE ToyBox
(ToyID int, ToyName varchar(100), ToyType varchar(100));



INSERT INTO ToyBox (ToyID, ToyName, ToyType)

VALUES (1, 'Teddy Bear', 'Stuffed');



INSERT INTO ToyBox (ToyID, ToyName, ToyType)

VALUES (1, 'Teddy Bear', 'Stuffed'),

(2, 'Teddy Bearr', 'block'),

(3, 'Teddy Bearrr', 'Stuffed'),

(4, 'Teddy Bearrrr', 'block');



INSERT INTO ToyBox (ToyID, ToyName)
```

```
VALUES (1, 'Teddy Bearrrrr');




UPDATE ToyBox

SET ToyType = 'Plush'

WHERE ToyID = 1;


SET SQL_SAFE_UPDATES = 0;


DELETE FROM ToyBox

WHERE ToyID = 1;
```

```
GRANT SELECT, INSERT ON ToyBox TO Sam;


CREATE USER Sam IDENTIFIED BY 'password';

GRANT SELECT, INSERT ON ToyBox TO Sam;

SHOW GRANTS FOR Sam;

REVOKE INSERT ON ToyBox FROM Sam;

drop user Sam;


BEGIN;

delete from ToyBox;


INSERT INTO ToyBox (ToyID, ToyName)

VALUES (1, 'Teddy Bearrrrr');
```

```
SAVEPOINT before_more_toys;


SELECT * FROM xyz.ToyBox;


ROLLBACK TO SAVEPOINT before_more_toys;


INSERT INTO ToyBox (ToyID, ToyName)

VALUES (2, 'Car');


rollback;


COMMIT;
```

```
SELECT * FROM ToyBox ORDER BY ToyID desc;

SELECT ToyName FROM ToyBox;


SELECT ToyID FROM ToyBox WHERE ToyType = 'Stuffed';


SELECT ToyType, count(*) FROM ToyBox GROUP BY ToyType;


SELECT customer, AVG(amount) AS AVG_spent

FROM sales_data

GROUP BY customer

HAVING AVG_spent > 10;


SELECT customer, amount FROM sales_data WHERE amount = (SELECT min(amount) FROM sales_data);


select length('SHRIDHAR');


select day('2025-04-23');


SELECT e.name, d.dept_name

FROM employeess e

INNER JOIN departments d ON e.dept_id = d.dept_id;



SELECT e.name, d.dept_name

FROM employeess e

LEFT JOIN departments d ON e.dept_id = d.dept_id

UNION

SELECT e.name, d.dept_name

FROM employeess e

RIGHT JOIN departments d ON e.dept_id = d.dept_id;



SELECT e.name, d.dept_name

FROM employeess e

CROSS JOIN departments d;
```

```
SELECT e.name AS employee,
m.name AS manager
FROM
employeess e
INNER JOIN
employeess m ON e.manager_id = m.emp_id;

CREATE TABLE studentss (
student_id INT PRIMARY KEY default 5,
name VARCHAR(50),
age INT);

insert into students values(1,'shridhar',27);
insert into studentss(name,age) values('shree',25);

CREATE TABLE enrollments (
student_id INT,
FOREIGN KEY (student_id) REFERENCES students(student_id)
);

insert into enrollments values(1);



CREATE TABLE teachers (
teacher_id INT PRIMARY KEY,
email VARCHAR(100) UNIQUE
);

CREATE TABLE enrollmentss (
student_id INT,
course_id INT,
PRIMARY KEY (student_id, course_id)
);

CREATE TABLE userss (
user_id INT PRIMARY KEY,
```

```
username VARCHAR(50) NOT NULL default 'ABC'

);



insert into userss(user_id) values(1);




CREATE TABLE employeees (

emp_id INT PRIMARY KEY,

age INT CHECK (age >= 18)

);

insert into employeees values(1,15);



CREATE VIEW pen_value AS

SELECT customer, amount

FROM sales_data

WHERE product = 'Pen';



select * from pen_value;



drop view pen_value;







CREATE INDEX youtubee

ON youtube_channel_stats (subscribers);



select * from youtube_channel_stats where subscribers > 100000;



call youtube.empty();

call lakhsubs(600000);
```