



Gurukul Technicals Institute

Computer education · Robotics · Science projects · Delhi

FREE EBOOK

PROGRAMMING AND WEB

Databases and SQL Study Book

The complete study notes, practice questions and glossary for
Databases and SQL

INSIDE THIS BOOK

- Databases and the DBMS
- Data types, NULL and keys
- Creating tables and inserting data
- Reading data: SELECT, WHERE and ORDER BY
- Changing data: UPDATE, DELETE and transactions
- ...and 2 more chapters, practice questions and a glossary

Beginner

Students learning databases and SQL

engineergurukul.com

+91 97116 99511

Databases and SQL Study Book

Understand how a database management system stores data in related tables, and write MySQL 8 queries to create tables, add, read, change and summarise data, join tables and design them well.

WHO IT IS FOR	Students learning databases and SQL
LEVEL	Beginner
SUBJECT AREA	Programming and web
WHAT IS INSIDE	7 chapters, practice questions with answers, and a glossary

CONTENTS

1	Databases and the DBMS	3
2	Data types, NULL and keys	5
3	Creating tables and inserting data	7
4	Reading data: SELECT, WHERE and ORDER BY	10
5	Changing data: UPDATE, DELETE and transactions	12
6	Aggregate functions and GROUP BY	15
7	JOINS and normalisation	18
	Practice questions	20
	Answers	24
	Glossary	26

Free ebook: you may read, print and share it for learning; it may not be sold or changed. © 2026 Gurukul Technicals Institute. First edition, October 2026. Contact details are on the back cover.

Databases and the DBMS

What is a database?

A **database** is an organised collection of related data, stored so that it can be found, updated and shared easily. A school keeps student records, a bank keeps account balances and an online shop keeps products, customers and orders. All of these are databases.

The software that creates and manages a database is a **DBMS** (Database Management System). Users and programs do not touch the stored files directly; they send requests to the DBMS, and it does the reading and writing for them. Well-known products include MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server and SQLite.

Why not just use files?

Before databases, each program kept its data in its own files. Imagine a coaching centre where the accounts clerk and the exam clerk each keep a separate Excel file of students. When Anjali changes her mobile number, both files must be updated. If one clerk forgets, the two records disagree. A DBMS solves these problems:

Problem with separate files	How a DBMS helps
Redundancy: the same data is copied into many files	Data is stored once and shared by every program
Inconsistency: the copies stop matching	One copy means one correct value
Searching is slow and difficult	A query language finds any data quickly
Nothing stops wrong data	Constraints reject invalid data, such as a missing roll number
Two people editing at once can overwrite each other	Many users can work at the same time safely
Weak security	User accounts and permissions control who can see or change what
Data is lost if a file is damaged	Backup and recovery tools

Relational databases

Most business databases are **relational**. A relational DBMS (RDBMS), such as MySQL, stores data in **tables**, and each table holds data about one kind of thing:

roll_no	name	city
101	Anjali Verma	Delhi
102	Rohit Singh	Jaipur

- Each **column**, also called a field or attribute, holds one property, such as `name`. Every value in a column has the same data type.
- Each **row**, also called a record or tuple, holds the details of one student.
- Tables are linked to each other through shared values, as later lessons show.

SQL

SQL (Structured Query Language) is the standard language for working with relational databases. Its commands fall into groups:

- **DDL** (Data Definition Language) defines structure: `CREATE`, `ALTER`, `DROP`, `TRUNCATE`.
- **DML** (Data Manipulation Language) works with rows: `INSERT`, `UPDATE`, `DELETE`. `SELECT`, which reads data, is often placed here too, or in a group of its own called DQL (Data Query Language).
- **DCL** (Data Control Language) manages permissions: `GRANT`, `REVOKE`.
- **TCL** (Transaction Control Language) controls transactions: `COMMIT`, `ROLLBACK`.

SQL keywords are not case-sensitive, so `select` works as well as `SELECT`; writing keywords in capitals is simply a convention that makes queries easier to read. End each statement with a semicolon.

This course uses MySQL 8. Most of the SQL here works in other databases too, but small details differ between products.

KEY POINTS

- A DBMS is software that stores, protects and shares data; programs ask the DBMS instead of reading files directly.
- Compared with separate files, a DBMS reduces redundancy and inconsistency and adds constraints, safe multi-user access, security and backup.
- A relational database stores data in tables: a column is one field and a row is one record.
- SQL commands are grouped into DDL, DML, DCL and TCL.

TRY THESE EXERCISES

1. List the data a coaching centre keeps about students, fees and exams, mark every item that would be copied into more than one file if the accounts clerk and the exam clerk kept separate Excel files, and write one sentence on how a DBMS avoids each copy.
2. On paper, draw a students table with the columns roll_no, name, city and course, fill in five rows, and label one column as a field and one row as a record; then write which column could identify every student on its own.
3. Open the MySQL command-line client or MySQL Workbench, run SHOW DATABASES; and SELECT VERSION(); to check that you are using MySQL 8, then sort these commands into DDL, DML, DCL and TCL: CREATE, INSERT, GRANT, COMMIT, ALTER, DELETE, REVOKE and ROLLBACK.

CHAPTER 2

Data types, NULL and keys

Choosing data types

Every column in a table has a **data type** that decides what it can hold. The right type saves space and blocks wrong data, such as letters in a price column. The most common MySQL 8 types are:

Type	Holds	Example
INT	whole numbers	250
DECIMAL (10,2)	exact decimals: 10 digits in total, 2 after the point	14999.50
VARCHAR(50)	text of varying length, up to 50 characters	'Priya Nair'
CHAR(6)	text of a fixed length	'110001'
TEXT	long text, such as a product description	'Wireless mouse with...'
DATE	a date, written year-month-day	'2025-08-15'
DATETIME	a date and a time	'2025-08-15 10:30:00'
BOOLEAN	true or false, stored as 1 or 0	TRUE

Store money in `DECIMAL`, because `FLOAT` and `DOUBLE` hold approximate values that can cause tiny rounding errors in totals. Store phone numbers and PIN codes as text (`VARCHAR` or `CHAR`): you never do arithmetic on them, and a number column would drop the leading zero of a landline such as 011-23456789.

NULL

NULL means a value is missing or unknown. It is not the same as zero or an empty string. If a student has not given an email address, the email column can be NULL. Add `NOT NULL` to a column that must always have a value.

Primary key

A **primary key** is a column, or a set of columns, whose value identifies each row uniquely.

- No two rows can have the same primary key value.
- A primary key can never be NULL.
- A table has at most one primary key.

A roll number, an employee code or an invoice number makes a good primary key. A name does not, because two students can both be called Rahul Sharma. A primary key made of two or more columns together is called a **composite key**. In MySQL, `AUTO_INCREMENT` on an integer key makes the database generate the next number by itself.

Foreign key

A **foreign key** is a column in one table that refers to the primary key of another table. It is how tables are linked. Here is an `invoices` table:

invoice_no	customer_id	amount
5001	12	2360.00
5002	12	1180.00
5003	15	4720.00

`customer_id` here is a foreign key that refers to `customer_id`, the primary key of the `customers` table. The DBMS then enforces **referential integrity**: you cannot add an invoice for customer 99 if no customer 99 exists. Notice that a foreign key value may repeat, because one customer can have many invoices.

Other keys

- A **candidate key** is any column, or set of columns, that could serve as the primary key, such as a roll number or an email address. One is chosen as the primary key; the others become **alternate keys**.

- A **unique key** (the `UNIQUE` constraint) also blocks duplicate values, but unlike a primary key it allows NULL, and a table can have several of them.

Rule of thumb: give every table a primary key.

KEY POINTS

- Every column has a data type; use `DECIMAL` for money and text types for phone numbers and PIN codes.
- `NULL` means missing or unknown, which is different from zero or an empty string.
- A primary key uniquely identifies each row and can never be `NULL`; a table has at most one.
- A foreign key refers to the primary key of another table and keeps the link between the tables valid.
- A `UNIQUE` key blocks duplicates but allows `NULL`, and a table can have several.

TRY THESE EXERCISES

1. Choose a MySQL data type for each of these columns and give a reason for each: roll number, student name, mobile number, PIN code, course fee, date of admission, the date and time of an exam, and whether the fee is paid.
2. Plan a library database with books, members and loans tables: write each table's columns, choose a primary key for each table, and mark which columns of the loans table are foreign keys and which table each one refers to.
3. For an employees table with the columns `emp_code`, `name`, `email`, `phone`, `date_of_joining` and `dept_id`, list the candidate keys, choose the primary key, name the alternate keys, and decide which columns may be `NULL` and which should be `NOT NULL`.

CHAPTER 3

Creating tables and inserting data

Creating a database

In the MySQL command-line client or MySQL Workbench, start by creating a database and selecting it:

```
SQL
```

```
CREATE DATABASE shop;  
USE shop;
```

`SHOW DATABASES;` lists the databases your account can see on the server, and `SHOW TABLES;` lists the tables in the current one.

CREATE TABLE

`CREATE TABLE` names the table and lists each column with its data type and any **constraints**, which are rules the data must follow.

SQL

```
CREATE TABLE customers (  
  customer_id INT AUTO_INCREMENT PRIMARY KEY,  
  name        VARCHAR(60) NOT NULL,  
  phone       VARCHAR(15) UNIQUE,  
  city        VARCHAR(40) DEFAULT 'Delhi',  
  joined_on   DATE  
);  
  
CREATE TABLE invoices (  
  invoice_no   INT PRIMARY KEY,  
  customer_id INT NOT NULL,  
  invoice_date DATE NOT NULL,  
  amount       DECIMAL(10,2) NOT NULL CHECK (amount > 0),  
  FOREIGN KEY (customer_id) REFERENCES customers(customer_id)  
);
```

Constraint	Effect
PRIMARY KEY	unique and never NULL
NOT NULL	a value is required
UNIQUE	no two rows may share a value
DEFAULT	the value used when none is given
CHECK	the value must pass a test (enforced from MySQL 8.0.16)
FOREIGN KEY ... REFERENCES	the value must exist in the other table

Create the parent table (`customers`) before the table that refers to it. MySQL's default storage engine, InnoDB, enforces foreign keys and supports transactions. `DESCRIBE customers;` shows a table's columns with their types and keys.

INSERT

`INSERT INTO` adds rows. Name the columns, then give the values in the same order:

SQL

```
INSERT INTO customers (name, phone, city, joined_on)
VALUES ('Meena Iyer', '9876543210', 'Chennai', '2025-04-01');

INSERT INTO customers (name, phone)
VALUES ('Arjun Mehta', '9811122233'),
      ('Sana Khan', '9899988877');
```

- Text and dates go inside single quotes; numbers do not.
- Dates use the format 'YYYY-MM-DD'.
- One `INSERT` can add several rows, separated by commas.
- `customer_id` was left out, so `AUTO_INCREMENT` numbers the rows 1, 2 and 3. Arjun and Sana get the default city 'Delhi', and their `joined_on` is `NULL` because that column has no default.
- A row that breaks a constraint, such as a duplicate phone number, is rejected with an error.

Listing the columns is safer than relying on the table's column order, because the statement keeps working if a column is added later.

Changing and removing tables

SQL

```
ALTER TABLE customers ADD email VARCHAR(100);
ALTER TABLE customers MODIFY city VARCHAR(60);
DROP TABLE old_orders;
```

`ALTER TABLE` changes the structure of an existing table: it can add, change or remove columns. `DROP TABLE` deletes a table completely, structure and data together.

In MySQL, statements such as `CREATE`, `ALTER` and `DROP` cannot be undone with `ROLLBACK`. Take a backup before changing a table that holds real data.

KEY POINTS

- `CREATE DATABASE` makes a database and `USE` selects it.
- `CREATE TABLE` lists each column with its data type and constraints such as `PRIMARY KEY`, `NOT NULL`, `UNIQUE`, `DEFAULT` and `FOREIGN KEY`.
- `INSERT INTO table (columns) VALUES (...)` adds rows; text and dates go in single quotes, dates as 'YYYY-MM-DD'.
- `ALTER TABLE` changes a table's structure, and `DROP TABLE` deletes the table and all its data.

TRY THESE EXERCISES

1. Create a database named `institute`, select it with `USE`, and create a `courses` table with an `AUTO_INCREMENT` primary key `course_id`, a `NOT NULL` `course_name`, a `DECIMAL(8,2)` `fee` with a `CHECK` that it is greater than 0 and a `duration_months` column; check the result with `DESCRIBE courses`;
2. Create a `students` table whose `course_id` is a foreign key referencing `courses`, add four courses and eight students with multi-row `INSERT` statements and admission dates written as 'YYYY-MM-DD', then try inserting a student with a `course_id` that does not exist and read the error.
3. Use `ALTER TABLE` to add an `email` column to `students` and to change the length of the `name` column with `MODIFY` (repeat `NOT NULL` in the new definition, because `MODIFY` replaces the whole column definition and would otherwise drop it), add a `phone` column with the `UNIQUE` constraint and insert a duplicate phone number on purpose to see it rejected, and finally create a practice table and remove it with `DROP TABLE`.

CHAPTER 4

Reading data: SELECT, WHERE and ORDER BY

The examples in this lesson use this `products` table:

product_id	name	category	price	stock
1	USB keyboard	Accessories	650.00	40
2	Wireless mouse	Accessories	499.00	0
3	24-inch monitor	Displays	9800.00	12
4	Laser printer	Printers	12500.00	5

SELECT

SQL

```
SELECT * FROM products;
SELECT name, price FROM products;
SELECT name, price * 0.9 AS sale_price FROM products;
SELECT DISTINCT category FROM products;
```

- The star means every column. In real programs, name only the columns you need.
- A column can be a calculation, and `AS` gives it a readable name called an **alias**.
- `DISTINCT` removes duplicate rows from the result, so each category appears only once.

WHERE: choosing rows

WHERE keeps only the rows for which a condition is true.

SQL

```
SELECT name, price FROM products WHERE price < 1000;
SELECT name FROM products WHERE category = 'Accessories' AND stock > 0;
SELECT name FROM products WHERE price BETWEEN 500 AND 10000;
SELECT name FROM products WHERE category IN ('Displays', 'Printers');
SELECT name FROM products WHERE name LIKE '%mouse%';
```

Operator	Meaning
<code>=, <> or !=, <, >, <=, >=</code>	comparisons
<code>AND, OR, NOT</code>	combine conditions
<code>BETWEEN a AND b</code>	in a range, including both ends
<code>IN (...)</code>	equal to any value in the list
<code>LIKE</code>	pattern match: <code>%</code> is any number of characters, <code>_</code> is exactly one
<code>IS NULL, IS NOT NULL</code>	value missing or present

Put text values in single quotes. With MySQL 8's default settings, text comparisons ignore case, so `'accessories'` also matches. `AND` is worked out before `OR`, so use brackets when you mix them: `WHERE (city = 'Pune' OR city = 'Nagpur') AND amount > 5000`.

Never write `= NULL`. A comparison with `NULL` is never true, so `WHERE email = NULL` returns no rows at all. Write `WHERE email IS NULL` instead.

ORDER BY and LIMIT

SQL

```
SELECT name, price FROM products ORDER BY price DESC;
SELECT name, category, price FROM products ORDER BY category, price;
SELECT name, price FROM products ORDER BY price DESC LIMIT 3;
```

`ORDER BY` sorts the result. `ASC` (ascending: smallest first, A to Z) is the default and `DESC` sorts the other way. With two columns, rows are sorted by the first and the second breaks ties. Without `ORDER BY`, the database may return rows in any order. `LIMIT 3` returns only the first three rows, so together with `ORDER BY price DESC` it gives the three most expensive products. MySQL, PostgreSQL and SQLite use `LIMIT`; Microsoft SQL Server uses `TOP` instead.

Clause order

Write the clauses in this order: `SELECT` columns, `FROM` table, `WHERE` condition, `ORDER BY` columns, `LIMIT` count. Putting `WHERE` after `ORDER BY` is a syntax error.

KEY POINTS

- `SELECT` chooses the columns, `FROM` names the table and `WHERE` filters the rows.
- `BETWEEN` includes both ends, `IN` matches a list, and `LIKE` matches patterns with `%` and `_`.
- Test for missing values with `IS NULL`, never with `= NULL`.
- `ORDER BY` sorts in ascending order by default; `DESC` reverses it and `LIMIT` returns only the first rows.

TRY THESE EXERCISES

1. Create the lesson's products table with at least eight items, including a `supplier_email` column left `NULL` for some of them, then write queries that show every column, only name and price, a 10 percent lower price with the alias `sale_price`, and each category once using `DISTINCT`.
2. Write `WHERE` queries to find products under ! 1,000, accessories that are in stock, products priced between ! 500 and ! 10,000, products in the Displays or Printers categories using `IN`, and names containing the word mouse using `LIKE`.
3. List the products with no supplier email using `IS NULL`, then try `= NULL` and see that it returns no rows; finally show the three most expensive products with `ORDER BY price DESC LIMIT 3`, and sort all products by category and then by price.

CHAPTER 5

Changing data: UPDATE, DELETE and transactions

UPDATE

`UPDATE` changes values in rows that already exist.

SQL

```
UPDATE products
SET price = 549.00, stock = 25
WHERE product_id = 2;
```

- `SET` lists the columns to change and their new values, separated by commas.
- `WHERE` chooses which rows change. Using the primary key, as here, changes exactly one row.

- A new value can be calculated from the old one.

`UPDATE products SET price = price + 100 WHERE category = 'Printers';` raises every printer's price by ! 100.

DELETE

`DELETE` removes whole rows:

SQL

```
DELETE FROM products WHERE stock = 0 AND category = 'Accessories';
```

`DELETE` cannot remove a single value from a row. To clear one value, use `UPDATE` to set it to `NULL`.

The danger of a missing WHERE

`UPDATE` or `DELETE` without a `WHERE` clause affects every row in the table.

`UPDATE products SET price = 0;` sets the price of every product to zero, and `DELETE FROM products;` empties the whole table. Build these habits:

1. First run a `SELECT` with the same `WHERE` and check that it returns only the rows you mean to change.
2. Use the primary key in the `WHERE` whenever you can.
3. Switch on safe updates. When MySQL's `sql_safe_updates` setting is on (MySQL Workbench has a Safe Updates option for it), an `UPDATE` or `DELETE` is refused unless its `WHERE` uses a key column or it has a `LIMIT`.

Foreign keys protect related rows

If an invoice refers to customer 12 through a foreign key, MySQL by default refuses to delete customer 12, because the invoice would then point to a customer who does not exist. Delete or change the invoices first, or define the foreign key with `ON DELETE CASCADE`, which deletes the related rows automatically. Use cascading with care.

Transactions

A **transaction** groups statements so that either all of them succeed or none do. Think of transferring ! 5,000 from Ravi's account to Kavita's: taking money out of one account without adding it to the other must never happen.

SQL

```
START TRANSACTION;  
UPDATE accounts SET balance = balance - 5000 WHERE acc_no = 'SB1001';  
UPDATE accounts SET balance = balance + 5000 WHERE acc_no = 'SB1002';  
COMMIT;
```

COMMIT makes the changes permanent. **ROLLBACK**, used in place of **COMMIT**, cancels every change made since **START TRANSACTION**. Outside a transaction, MySQL works in **autocommit** mode by default: each statement is committed as soon as it runs, so it cannot be rolled back afterwards.

DELETE, TRUNCATE and DROP

Command	Removes	Notes
<code>DELETE FROM t</code> <code>WHERE ...</code>	the chosen rows	DML; can be rolled back inside a transaction
<code>TRUNCATE TABLE t</code>	every row, but keeps the table	DDL; fast, resets <code>AUTO_INCREMENT</code> , cannot be rolled back
<code>DROP TABLE t</code>	the table itself, with all its data	DDL; cannot be rolled back

KEY POINTS

- `UPDATE ... SET ... WHERE` changes values in existing rows, and `DELETE FROM ... WHERE` removes whole rows.
- Without a `WHERE` clause, `UPDATE` and `DELETE` affect every row, so test the condition with `SELECT` first.
- A transaction's changes become permanent with `COMMIT` or are cancelled with `ROLLBACK`.
- `TRUNCATE` empties a table and `DROP` removes it; in MySQL neither can be rolled back.

TRY THESE EXERCISES

1. In the products table, first run SELECT with WHERE product_id = 2, then UPDATE the same row's price and stock, raise every printer's price by ! 100 with SET price = price + 100 WHERE category = 'Printers', and check both changes with SELECT. If MySQL Workbench refuses the printer update with error 1175, its Safe Updates option is on: run SET sql_safe_updates = 0; first and SET sql_safe_updates = 1; straight after.
2. Create an accounts table with two accounts, START TRANSACTION and move ! 5,000 from one to the other with two UPDATE statements, look at the balances, run ROLLBACK and check that both are back to their old values; then repeat the transfer and finish it with COMMIT.
3. Run SET sql_safe_updates = 1; and try UPDATE products SET price = 0; to see MySQL refuse it, try deleting a course that still has students linked to it by their foreign key (such as DELETE FROM courses WHERE course_id = 1;) and read the error, then empty a practice table that has an AUTO_INCREMENT key with TRUNCATE TABLE, insert a row and check that the numbering starts again from 1.

CHAPTER 6

Aggregate functions and GROUP BY

The examples in this lesson use this `sales` table of bills from three branches:

bill_no	branch	amount	email
1	Delhi	4200	neha@example.com
2	Delhi	1800	NULL
3	Mumbai	5000	farhan@example.com
4	Mumbai	2500	NULL
5	Pune	3000	kiran@example.com

Aggregate functions

An **aggregate function** takes many rows and returns a single value.

Function	Returns	Result on this table
<code>COUNT(*)</code>	the number of rows	5
<code>COUNT(email)</code>	the number of non-NULL values in the column	3
<code>SUM(amount)</code>	the total	16500

Function	Returns	Result on this table
AVG(amount)	the average	3300
MIN(amount)	the smallest value	1800
MAX(amount)	the largest value	5000

SQL

```
SELECT COUNT(*) AS bills, SUM(amount) AS total FROM sales;
SELECT MAX(amount) FROM sales WHERE branch = 'Delhi';
SELECT COUNT(DISTINCT branch) FROM sales;
```

COUNT(*) counts rows, while COUNT(column) skips NULL values. SUM, AVG, MIN and MAX also ignore NULLs. COUNT(DISTINCT branch) counts the different values, which here is 3.

GROUP BY

GROUP BY puts rows that share a value into groups, and each aggregate is then worked out separately for every group.

SQL

```
SELECT branch, COUNT(*) AS bills, SUM(amount) AS total
FROM sales
GROUP BY branch
ORDER BY total DESC;
```

branch	bills	total
Mumbai	2	7500
Delhi	2	6000
Pune	1	3000

Every column in the SELECT list should either appear in GROUP BY or sit inside an aggregate function. MySQL 8's default SQL mode includes ONLY_FULL_GROUP_BY, which rejects most queries that break this rule. MySQL 8 does not sort grouped results for you, so add ORDER BY whenever the order matters.

HAVING versus WHERE

WHERE filters rows **before** they are grouped. HAVING filters groups **after** the aggregates are calculated, so only HAVING can test an aggregate.

SQL

```
SELECT branch, SUM(amount) AS total
FROM sales
WHERE amount >= 2000
GROUP BY branch
HAVING SUM(amount) > 4000;
```

Here **WHERE** first removes bill 2 (! 1,800). The branch totals are then Delhi 4200, Mumbai 7500 and Pune 3000, and **HAVING** keeps only Delhi and Mumbai. Writing **WHERE SUM(amount) > 4000** instead is an error.

The full clause order

SQL

```
SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ... ORDER BY ... LIMIT ...;
```

Clauses must be written in this order. Logically, the database works through them as FROM, WHERE, GROUP BY, HAVING, SELECT, ORDER BY and finally LIMIT, which is why a WHERE condition cannot use an aggregate.

KEY POINTS

- COUNT, SUM, AVG, MIN and MAX turn many rows into one value.
- COUNT(*) counts rows; COUNT(column) and the other aggregates ignore NULL values.
- GROUP BY calculates the aggregates separately for each group of rows that share a value.
- WHERE filters rows before grouping; HAVING filters groups after aggregation.

TRY THESE EXERCISES

1. Create the lesson's sales table with its five bills, entering the two missing emails as NULL without quotes (the text 'NULL' would count as a value), run COUNT(*), COUNT(email), SUM(amount), AVG(amount), MIN(amount) and MAX(amount), and check that you get 5, 3, 16500, 3300, 1800 and 5000.
2. Add five more bills spread across the three branches, then use GROUP BY branch to show each branch's number of bills, total and average amount, sorted by total from highest to lowest, and count the branches with COUNT(DISTINCT branch).
3. Write a query that leaves out bills under ! 2,000 with WHERE and keeps only branches whose total is over ! 4,000 with HAVING, then move the SUM condition into the WHERE clause and read the error MySQL gives.

JOINS and normalisation

Why join tables?

Related data lives in separate tables, linked by keys. A **JOIN** combines rows from two tables into one result, using a condition that matches them, usually a foreign key equal to a primary key. Take these two tables:

customer_id	name
1	Meena
2	Arjun
3	Sana

order_id	customer_id	amount
101	1	1500
102	1	800
103	2	2200

INNER JOIN

SQL

```
SELECT c.name, o.order_id, o.amount
FROM customers AS c
INNER JOIN orders AS o ON o.customer_id = c.customer_id;
```

An **INNER JOIN** returns only the rows that have a match in both tables: Meena twice and Arjun once. Sana has no orders, so she does not appear. **c** and **o** are table aliases that keep the query short, and **c.name** makes clear which table a column comes from. In MySQL, a plain **JOIN** means **INNER JOIN**.

LEFT and RIGHT JOIN

SQL

```
SELECT c.name, o.order_id
FROM customers AS c
LEFT JOIN orders AS o ON o.customer_id = c.customer_id;
```

A **LEFT JOIN** returns every row from the left table (the one named before JOIN) together with any matching rows from the right table. Where there is no match, the right table's columns are NULL, so Sana appears with a NULL `order_id`. Adding `WHERE o.order_id IS NULL` therefore lists the customers who have never ordered. A **RIGHT JOIN** keeps every row of the right table instead. MySQL does not support `FULL OUTER JOIN`.

Normalisation

Normalisation means organising data into tables so that each fact is stored only once. Look at this badly designed table:

order_id	customer_name	customer_phone	product
101	Meena	9876543210	Keyboard
102	Meena	9876543210	Mouse

Meena's phone number is stored twice, which causes three kinds of **anomaly**:

- **Update anomaly**: when her number changes, every one of her rows must change, and missing one leaves the data contradicting itself.
- **Insert anomaly**: a new customer cannot be recorded until they place an order.
- **Delete anomaly**: deleting a customer's only order also deletes the only record of their phone number.

The cure is to split the data into a `customers` table and an `orders` table linked by `customer_id`, as above. The rules for doing this are called **normal forms**:

1. **First normal form (1NF)**: every cell holds a single value, with no lists such as "Keyboard, Mouse" in one cell and no repeating columns such as product1 and product2.
2. **Second normal form (2NF)**: the table is in 1NF and every non-key column depends on the whole primary key, not on just part of a composite key.
3. **Third normal form (3NF)**: the table is in 2NF and no non-key column depends on another non-key column. In an employees table, `dept_name` depends on `dept_id` rather than on the employee, so departments belong in a table of their own.

Most business databases are designed to reach at least third normal form.

KEY POINTS

- A JOIN combines rows from two tables, usually by matching a foreign key to a primary key.
- INNER JOIN returns only matching rows; LEFT JOIN keeps every row of the left table and fills missing matches with NULL.
- Normalisation splits data into related tables so that each fact is stored once.
- Repeated data causes update, insert and delete anomalies.
- 1NF needs a single value in every cell, 2NF removes partial dependencies and 3NF removes dependencies between non-key columns.

TRY THESE EXERCISES

1. Create the lesson's customers and orders tables with the same rows, write an INNER JOIN with the table aliases c and o that lists each order with the customer's name, and check that Sana does not appear.
2. Change the query to a LEFT JOIN so that every customer appears, add WHERE o.order_id IS NULL to list the customers who have never ordered, then remove that WHERE and add GROUP BY c.customer_id, c.name to show each customer's number of orders with COUNT(o.order_id), so that Sana shows 0.
3. Write a flat admissions table for a computer centre in which the student name, phone, course name and course fee repeat on every row, point out one update, one insert and one delete anomaly in it, then split it into students, courses and admissions tables in third normal form and mark every primary and foreign key.

CHECK YOURSELF

Practice questions

Choose the best answer for each question. The answers, with a short explanation, are in the next chapter.

1. What does DBMS stand for?

- a) Data Backup Management Software
- b) Digital Base Memory System
- c) Database Monitoring Service
- d) Database Management System

2. In a relational table, what is a single row also called?

- a) A field
- b) A data type
- c) A constraint
- d) A record

3. A fees clerk and an exam clerk each keep their own file with the same student details. What is this duplication of data called?

- a) Data redundancy
- b) Referential integrity
- c) Normalisation
- d) Data abstraction

- 4. Which group of SQL commands contains CREATE, ALTER and DROP?**
- a) DML (Data Manipulation Language)
 - b) DDL (Data Definition Language)
 - c) DCL (Data Control Language)
 - d) TCL (Transaction Control Language)
- 5. Which SQL command gives a user permission to work with a table?**
- a) INSERT
 - b) COMMIT
 - c) GRANT
 - d) ALTER
- 6. Which MySQL data type is best for storing a price such as 14999.50 exactly?**
- a) FLOAT
 - b) VARCHAR(10)
 - c) DATE
 - d) DECIMAL(10,2)
- 7. What does NULL mean in a database column?**
- a) The number zero
 - b) The value is missing or unknown
 - c) A text value with one space in it
 - d) A value that is always false
- 8. Which statement about a primary key is true?**
- a) A table can have several primary keys
 - b) It can contain duplicate values as long as they are NULL
 - c) It must always be a text column
 - d) Its values must be unique and can never be NULL
- 9. An invoices table has a customer_id column that refers to the primary key of the customers table. What is customer_id in the invoices table?**
- a) The primary key of the invoices table
 - b) A foreign key
 - c) A composite key
 - d) A DEFAULT constraint
- 10. Which constraint stops two rows from having the same phone number, while still allowing the phone column to be NULL?**
- a) PRIMARY KEY
 - b) UNIQUE
 - c) NOT NULL
 - d) DEFAULT
- 11. Which statement creates a new table?**
- a) NEW TABLE
 - b) ADD TABLE
 - c) CREATE TABLE
 - d) MAKE TABLE
- 12. Which statement makes shop the current database, so that later statements use its tables?**
- a) OPEN shop;
 - b) USE shop;
 - c) SELECT shop;
 - d) GOTO shop;
- 13. How should the date 15 August 2025 be written in a MySQL INSERT statement?**
- a) '15-08-2025'
 - b) '2025-08-15'
 - c) 15/08/2025
 - d) #15-08-2025#

- 14. A column is defined as city VARCHAR(40) DEFAULT 'Delhi'. An INSERT statement leaves out the city column. What is stored in it?**
- a) NULL
 - b) An empty string
 - c) Delhi
 - d) Nothing, because the statement fails with an error
- 15. Which statement adds a new email column to an existing customers table?**
- a) UPDATE customers ADD email VARCHAR(100);
 - b) INSERT INTO customers COLUMN email;
 - c) CREATE COLUMN email IN customers;
 - d) ALTER TABLE customers ADD email VARCHAR(100);
- 16. Which query returns only the name and price columns of every row in products?**
- a) SELECT * FROM products WHERE name, price;
 - b) SELECT name, price FROM products;
 - c) GET name, price FROM products;
 - d) SELECT products FROM name, price;
- 17. Which condition finds products whose names begin with "Sam"?**
- a) name LIKE '%Sam'
 - b) name = 'Sam%'
 - c) name LIKE 'Sam_'
 - d) name LIKE 'Sam%'
- 18. Which prices does WHERE price BETWEEN 500 AND 1000 match?**
- a) Only prices above 500 and below 1000
 - b) Only the two values 500 and 1000
 - c) 500 to 1000, including both 500 and 1000
 - d) 500 up to 999, but not 1000
- 19. Which condition correctly finds customers with no email address recorded?**
- a) WHERE email IS NULL
 - b) WHERE email = NULL
 - c) WHERE email == NULL
 - d) WHERE NULL(email)
- 20. What does SELECT DISTINCT city FROM customers; return?**
- a) Only the city of the first customer
 - b) Each city once, with duplicate values removed
 - c) The number of customers in each city
 - d) Every city sorted in descending order
- 21. Which query returns the three cheapest products?**
- a) SELECT * FROM products LIMIT 3;
 - b) SELECT * FROM products ORDER BY price DESC LIMIT 3;
 - c) SELECT * FROM products ORDER BY price LIMIT 3;
 - d) SELECT * FROM products WHERE price LIMIT 3;
- 22. With safe updates switched off, what does UPDATE products SET price = 0; do?**
- a) Changes only the first row of the table
 - b) Changes only the rows where price is NULL
 - c) Sets the price of every product in the table to 0
 - d) Asks for confirmation before changing anything

23. Which statement removes the customer whose customer_id is 7?

- a) DROP FROM customers WHERE customer_id = 7;
- b) REMOVE customers WHERE customer_id = 7;
- c) DELETE FROM customers WHERE customer_id = 7;
- d) DELETE customer_id = 7 FROM customers;

24. Inside a transaction, which command cancels every change made since START TRANSACTION?

- a) ROLLBACK
- b) COMMIT
- c) UNDO
- d) REVOKE

25. What is the difference between TRUNCATE TABLE and DROP TABLE?

- a) TRUNCATE removes the table itself; DROP removes only the rows
- b) TRUNCATE can be rolled back in MySQL, but DROP cannot
- c) TRUNCATE removes all the rows but keeps the table; DROP removes the table itself
- d) They are two names for the same command

26. A customer still has invoices that refer to them through a foreign key that was created with no ON DELETE clause. What happens when you try to delete that customer?

- a) The customer and all their invoices are deleted
- b) MySQL refuses the delete because related invoices still refer to the customer
- c) The customer is deleted and the invoices keep pointing to the old id
- d) MySQL deletes the invoices but keeps the customer

27. What does COUNT(*) return?

- a) The total of all the values
- b) The number of columns in the table
- c) The number of non-NULL values in the first column
- d) The number of rows

28. A column named email has 10 rows, and 4 of them are NULL. What does COUNT(email) return?

- a) 6
- b) 10
- c) 4
- d) 0

29. Which query shows the total sales amount for each branch?

- a) SELECT branch, SUM(amount) FROM sales GROUP BY branch;
- b) SELECT branch, SUM(amount) FROM sales ORDER BY branch;
- c) SELECT branch, COUNT(amount) FROM sales GROUP BY branch;
- d) SELECT SUM(branch) FROM sales GROUP BY amount;

30. Which clause keeps only the branches whose total sales are above ! 50,000 after the totals are calculated?

- a) WHERE SUM(amount) > 50000
- b) ORDER BY SUM(amount) > 50000
- c) HAVING SUM(amount) > 50000
- d) LIMIT SUM(amount) > 50000

31. Which join returns only the rows that have matching values in both tables?
- a) INNER JOIN
 - b) LEFT JOIN
 - c) RIGHT JOIN
 - d) FULL OUTER JOIN
32. Customers are LEFT JOINed to orders. What does the result show for a customer who has never placed an order?
- a) The customer's name, with NULL in the order columns
 - b) Nothing, because the customer is left out
 - c) An error message
 - d) The customer's name, with 0 in the order columns
33. Meena's phone number is stored on every one of her order rows. When the number changes, missing one row leaves the data contradicting itself. What is this problem called?
- a) An update anomaly
 - b) An insert anomaly
 - c) A delete anomaly
 - d) A foreign key
34. A column holds values such as "Keyboard, Mouse, Monitor" in a single cell. Which normal form does this break?
- a) First normal form (1NF)
 - b) Second normal form (2NF)
 - c) Third normal form (3NF)
 - d) No normal form, because lists are allowed in a cell
35. Which normal form removes dependencies between non-key columns, such as dept_name depending on dept_id in an employees table?
- a) First normal form (1NF)
 - b) Second normal form (2NF)
 - c) No normal form deals with this
 - d) Third normal form (3NF)

ANSWER KEY

Answers

1. (d) **Database Management System.** A DBMS is the software that creates, stores, protects and shares the data in a database.
2. (d) **A record.** A row holds one complete record, such as the details of one student; a column is a field.
3. (a) **Data redundancy.** Redundancy means the same data is stored in several places, which wastes space and lets the copies disagree.
4. (b) **DDL (Data Definition Language).** DDL commands define and change the structure of databases and tables.
5. (c) **GRANT.** GRANT and REVOKE are DCL commands that give and take away permissions.
6. (d) **DECIMAL(10,2).** DECIMAL stores exact decimal values, while FLOAT stores approximations that can cause rounding errors in money totals.

7. (b) **The value is missing or unknown.** NULL marks a missing or unknown value; it is different from zero and from an empty or blank string.
8. (d) **Its values must be unique and can never be NULL.** A primary key identifies each row, so its values are unique and never NULL, and a table has at most one.
9. (b) **A foreign key.** A foreign key is a column that refers to the primary key of another table and links the two tables.
10. (b) **UNIQUE.** UNIQUE blocks duplicate values but allows NULL; a PRIMARY KEY would also forbid NULL.
11. (c) **CREATE TABLE.** CREATE TABLE defines a new table with its columns, data types and constraints.
12. (b) **USE shop;.** USE selects the database that following statements work in.
13. (b) **'2025-08-15'.** MySQL expects dates in year-month-day order inside single quotes.
14. (c) **Delhi.** When a column is left out of an INSERT, it takes its DEFAULT value if one is defined.
15. (d) **ALTER TABLE customers ADD email VARCHAR(100);.** ALTER TABLE changes the structure of an existing table, including adding columns.
16. (b) **SELECT name, price FROM products;.** The SELECT list names the columns to return and FROM names the table.
17. (d) **name LIKE 'Sam%'. In LIKE, % matches any number of characters, so 'Sam%' means Sam followed by anything; 'Sam_' would match exactly one more character.**
18. (c) **500 to 1000, including both 500 and 1000.** BETWEEN is inclusive: it matches both end values and everything between them.
19. (a) **WHERE email IS NULL.** A comparison with = NULL is never true; missing values are tested with IS NULL.
20. (b) **Each city once, with duplicate values removed.** DISTINCT removes duplicate rows from the result, so each city appears only once.
21. (c) **SELECT * FROM products ORDER BY price LIMIT 3;.** ORDER BY price sorts from the lowest price because ascending is the default, and LIMIT 3 keeps the first three rows.
22. (c) **Sets the price of every product in the table to 0.** Without a WHERE clause, UPDATE changes every row in the table.
23. (c) **DELETE FROM customers WHERE customer_id = 7;.** DELETE FROM removes rows, and the WHERE clause chooses which ones.
24. (a) **ROLLBACK.** ROLLBACK undoes the transaction's changes; COMMIT would make them permanent instead.
25. (c) **TRUNCATE removes all the rows but keeps the table; DROP removes the table itself.** TRUNCATE empties the table and leaves its structure in place, while DROP deletes the structure and the data together.
26. (b) **MySQL refuses the delete because related invoices still refer to the customer.** By default a foreign key blocks deleting a parent row that child rows still refer to, which protects referential integrity.
27. (d) **The number of rows.** COUNT(*) counts rows, including rows that contain NULL values.
28. (a) **6.** COUNT(column) counts only the non-NULL values, so 10 minus 4 gives 6.

- 29. (a) SELECT branch, SUM(amount) FROM sales GROUP BY branch;** GROUP BY branch makes one group per branch, and SUM(amount) adds up the amounts in each group.
- 30. (c) HAVING SUM(amount) > 50000.** HAVING filters groups after aggregation; WHERE runs before grouping and cannot use an aggregate.
- 31. (a) INNER JOIN.** An INNER JOIN keeps only matched rows; a LEFT or RIGHT JOIN also keeps unmatched rows from one side, and a FULL OUTER JOIN from both sides.
- 32. (a) The customer's name, with NULL in the order columns.** A LEFT JOIN keeps every row of the left table and fills the right table's columns with NULL where there is no match.
- 33. (a) An update anomaly.** An update anomaly happens when the same fact is stored in many rows and must be changed in all of them.
- 34. (a) First normal form (1NF).** 1NF requires every cell to hold a single, atomic value.
- 35. (d) Third normal form (3NF).** 3NF requires that no non-key column depends on another non-key column, so the department details move to their own table.

WORD LIST

Glossary

Aggregate function

A function such as COUNT, SUM, AVG, MIN or MAX that turns the values of many rows into a single result.

Anomaly

A problem caused by repeating data in a badly designed table: an update anomaly leaves some copies of a value changed and others not, an insert anomaly stops a fact being recorded on its own, and a delete anomaly loses a fact together with an unrelated row.

AUTO_INCREMENT

A MySQL column option that makes the database give each new row the next number automatically, usually for an integer primary key.

Candidate key

Any column or set of columns that could uniquely identify every row and so could be chosen as the primary key.

Column

One field of a table, such as name or city, in which every value has the same data type.

Constraint

A rule set on a column or table, such as NOT NULL, UNIQUE, CHECK or FOREIGN KEY, that the DBMS uses to reject invalid data.

Data type

The kind of value a column can hold, such as INT for whole numbers, DECIMAL for money, VARCHAR for text or DATE for dates.

Database

An organised collection of related data, stored so that it can be found, updated and shared easily.

DBMS

Database Management System, software such as MySQL that creates and manages databases and does the reading and writing of data for users and programs.

DDL

Data Definition Language, the group of SQL commands such as CREATE, ALTER, DROP and TRUNCATE that create, change and remove the structure of databases and tables.

DML

Data Manipulation Language, the group of SQL commands such as INSERT, UPDATE and DELETE that add, change and remove rows.

Foreign key

A column in one table that refers to the primary key of another table, linking the two so that every value in it, unless it is NULL, must match a row that exists there.

GROUP BY

A clause that gathers rows sharing the same value into groups, so that aggregate functions are calculated separately for each group.

HAVING

A clause that filters groups after GROUP BY has calculated the aggregates, so it can test a condition such as $SUM(amount) > 4000$.

INNER JOIN

A join that returns only the rows that have a matching row in both tables.

LEFT JOIN

A join that returns every row of the left table together with the matching rows of the right table, filling in NULL where there is no match.

Normalisation

Organising data into related tables, following normal forms such as 1NF, 2NF and 3NF, so that each fact is stored only once.

NULL

A marker meaning a value is missing or unknown, which is different from zero or an empty string and is tested with IS NULL.

Primary key

The column or set of columns whose value uniquely identifies each row of a table and can never be NULL.

Query

A request written in SQL, most often a SELECT statement, that asks the database for data.

Referential integrity

The guarantee, enforced through foreign keys, that every reference from one table points to a row that really exists in the other table.

Row

One record in a table, holding all the details of a single item such as one student or one invoice.

SQL

Structured Query Language, the standard language for creating, reading, changing and controlling access to data in relational databases.

Table

A set of rows and columns in a relational database that holds data about one kind of thing, such as customers or invoices.

Transaction

A group of SQL statements that either all take effect together with COMMIT or are all cancelled with ROLLBACK.



Gurukul Technicals Institute

Computer education institute in Delhi, with authorised member centres, robotics classes and science projects, working models and DIY kits.

COURSES

Computer & Office Skills · Accounting & Business Software · Programming & Web Development · Design & Digital Skills · Hardware & Networking · Robotics & Coding · Other Programmes

PROJECTS AND KITS

Ready-made school projects · DIY and STEM kits · Robotics and coding · Components and spares · Engineering projects

CONTACT

Head office: A-10, Chief Commissioner Colony, opposite Rana Pratap Bagh, Delhi 110007

Phone: 011 2721 1512 · **Mobile and WhatsApp:** +91 97116 99511

Email: engineergurukul@yahoo.co.in

Website: engineergurukul.com · **Instagram:** [@science_projects_club](https://www.instagram.com/science_projects_club)

More free ebooks for computer courses, robotics and science projects: engineergurukul.com/resources/ebooks/

Free to read, print and share. Not for sale.