



Gurukul Technicals Institute

Computer education · Robotics · Science projects · Delhi

FREE EBOOK

PROGRAMMING AND WEB

SQL Practice Workbook

Learn SQL by querying a school database, step by step

INSIDE THIS BOOK

- Databases, tables and keys
- Building the school database
- Asking questions with SELECT
- Better filters: LIKE, IN, BETWEEN and NULL
- Counting and summarising with GROUP BY
- ...and 5 more chapters, practice questions and a glossary

Beginner

Students learning databases

engineergurukul.com

+91 97116 99511

SQL Practice Workbook

Build a small school database and learn SQL by asking it real questions. You will write queries that filter, sort, count and join tables, change data safely and design tables well, then test yourself with 50 practice queries and their answers.

WHO IT IS FOR	Students learning databases
LEVEL	Beginner
SUBJECT AREA	Programming and web
WHAT IS INSIDE	10 chapters, practice questions with answers, and a glossary

CONTENTS

1	Databases, tables and keys	4
2	Building the school database	5
3	Asking questions with SELECT	8
4	Better filters: LIKE, IN, BETWEEN and NULL	11
5	Counting and summarising with GROUP BY	13
6	Joining tables	15
7	Queries inside queries	17
8	Changing data safely	19
9	Constraints, indexes and views	21
10	Normalisation: designing tables well	23
	Practice questions	24
	Answers	25

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, tables and keys

Every school keeps records. There is an admission register, an attendance register and a marks register. On paper, finding "all students from Noida who scored above 80 in Maths" can take an hour. A computer can answer it in a blink, if the records are kept in a **database**. A database is an organised store of information that a program can search and change.

To talk to a database we use **SQL**, short for Structured Query Language. People say it as "S-Q-L" or "sequel". A **query** is a question or an instruction written in SQL. You learn SQL the way you learn to cycle: by doing it. So we build one small school database and keep asking it new questions.

Tables, rows and columns

A database keeps its information in **tables**. A table looks like a page in a register.

student_id	name	city
1	Aarav Mehta	Delhi
2	Diya Patel	Noida
3	Kabir Singh	Noida

- A **row** is one record: everything about one student.
- A **column** is one kind of detail, such as the city, for every student.
- Each box where a row meets a column holds one **value**, such as "Noida".

Data types

When we make a table, we tell the database what kind of value each column holds. This is the column's **data type**.

Data type	What it holds	Example
<code>INT</code>	Whole numbers	78
<code>VARCHAR(50)</code>	Text up to 50 characters	'Diya Patel'
<code>CHAR(1)</code>	Text of exactly one character	'F'
<code>DATE</code>	A date, written year-month-day	'2012-09-02'
<code>DECIMAL(8,2)</code>	Numbers with two decimal places	1250.50

Text and dates go inside single quotes. Numbers do not. A shop table could store a price of ! 1,250.50 in a `DECIMAL(8,2)` column as `1250.50`, with no ! sign and no comma.

Keys

A **primary key** is a column whose value is different in every row and is never empty. A roll number is a good primary key. A name is not, because two students can share one.

A **foreign key** is a column that points to the primary key of another table. If the students table has a `class_id` column, and that number must match a real class in the classes table, then `class_id` is a foreign key. It links the two tables, and the database stops you from entering a class that does not exist.

REMEMBER

Primary key = "who is this row?" Foreign key = "which row in the other table does this belong to?"

Where to type SQL

Install MySQL Server 8 and a tool to type queries into: either MySQL Workbench, which has windows and buttons, or the `mysql` command-line client. End every SQL statement with a semicolon (;). SQL keywords such as `SELECT` can be typed in small or capital letters. This book writes them in capitals so you can spot them. A line that starts with two dashes (--) is a comment, which the database ignores.

DID YOU KNOW

Everything in this book runs in MySQL 8. MariaDB, a close cousin of MySQL, runs almost all of it too.

CHAPTER 2

Building the school database

Our school is small: three classes, four teachers and ten students. Here is the plan.

Table	One row means	Primary key
teachers	One teacher	teacher_id
classes	One class, such as 8A	class_id
students	One student	student_id
marks	One student's mark in one subject	student_id + subject

The marks table needs two columns together as its key. One student has many marks, and one subject has many marks, but one student has only one mark in Maths. A key made of two or more columns is called a **composite key**.

The full script

Type this carefully and run it all at once. The first three lines throw away any old copy of the practice database, so you can run the script again whenever you want clean data.

```
DROP DATABASE IF EXISTS school;
CREATE DATABASE school;
USE school;

CREATE TABLE teachers (
    teacher_id INT PRIMARY KEY,
    name        VARCHAR(50) NOT NULL,
    subject     VARCHAR(30) NOT NULL
);

CREATE TABLE classes (
    class_id    INT PRIMARY KEY,
    class_name  VARCHAR(10) NOT NULL UNIQUE,
    room        INT,
    teacher_id  INT,
    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id)
);

CREATE TABLE students (
    student_id INT PRIMARY KEY,
    name        VARCHAR(50) NOT NULL,
    gender      CHAR(1),
    city        VARCHAR(30),
    dob         DATE,
    class_id    INT,
    FOREIGN KEY (class_id) REFERENCES classes(class_id)
);

CREATE TABLE marks (
    student_id INT,
    subject     VARCHAR(30),
    marks       INT CHECK (marks BETWEEN 0 AND 100),
    PRIMARY KEY (student_id, subject),
    FOREIGN KEY (student_id) REFERENCES students(student_id)
);

INSERT INTO teachers VALUES
(1, 'Meena Sharma', 'Maths'),
(2, 'Arjun Rao', 'Science'),
(3, 'Farah Khan', 'English'),
(4, 'Harpreet Gill', 'Computers');

INSERT INTO classes VALUES
(1, '8A', 101, 1),
(2, '8B', 102, 2),
(3, '9A', 201, 3);

INSERT INTO students VALUES
(1, 'Aarav Mehta', 'M', 'Delhi', '2012-04-15', 1),
(2, 'Diya Patel', 'F', 'Noida', '2012-09-02', 1),
(3, 'Kabir Singh', 'M', 'Noida', '2012-01-20', 1),
(4, 'Ananya Iyer', 'F', 'Delhi', '2012-11-30', 2),
(5, 'Rohan Das', 'M', 'Gurugram', '2013-02-14', 2),
```

```
(6, 'Sneha Joshi', 'F', 'Faridabad', '2012-06-08', 2),
(7, 'Imran Ali', 'M', 'Delhi', '2011-08-25', 3),
(8, 'Pooja Verma', 'F', 'Noida', '2011-03-17', 3),
(9, 'Vikram Nair', 'M', 'Delhi', '2011-12-05', 3),
(10, 'Neha Gupta', 'F', 'Gurugram', NULL, 3);
```

```
INSERT INTO marks VALUES
```

```
(1, 'Maths', 78), (1, 'Science', 85), (1, 'English', 72),
(2, 'Maths', 92), (2, 'Science', 88), (2, 'English', 95),
(3, 'Maths', 45), (3, 'Science', 52), (3, 'English', 60),
(4, 'Maths', 88), (4, 'Science', 91), (4, 'English', 84),
(5, 'Maths', 35), (5, 'Science', 48), (5, 'English', 55),
(6, 'Maths', 67), (6, 'Science', 74), (6, 'English', 81),
(7, 'Maths', 81), (7, 'Science', 69), (7, 'English', 77),
(8, 'Maths', 95), (8, 'Science', 97), (8, 'English', 90),
(9, 'Maths', 58), (9, 'Science', 38), (9, 'English', 64);
```

SAFETY

`DROP DATABASE` deletes a whole database with no undo. Use it only on a practice database like this one, never on real school or office data.

Reading the script

- `CREATE TABLE` makes an empty table and lists its columns with their data types.
- `NOT NULL` means the column can never be left empty. `UNIQUE` means no two rows may share the value, so two classes cannot both be called 8A.
- `CHECK (marks BETWEEN 0 AND 100)` refuses any mark outside 0 to 100.
- `INSERT INTO ... VALUES` adds rows. Values go in the same order as the columns.
- `NULL` means "no value" or "not known yet". Neha Gupta joined late, so her date of birth has not been entered, and she has no marks yet. She will be useful later.

The tables are created in an order that matters. Classes point to teachers, so teachers must exist first. Students point to classes, and marks point to students.

TRY THIS

Run `SELECT COUNT(*) FROM marks;` after the script. You should see 27: nine students with three marks each. If you see a different number, check your `INSERT` lines.

CHAPTER 3

Asking questions with SELECT

`SELECT` is the statement you will use most. It only reads data, so you cannot break anything with it.

SQL

```
SELECT * FROM students;  
SELECT name, city FROM students;
```

The star (*) means "all columns". The second query shows only two. Asking only for the columns you need is a good habit.

Choosing rows with WHERE

WHERE keeps only the rows that pass a test.

SQL

```
SELECT name FROM students WHERE city = 'Delhi';
```

This gives Aarav Mehta, Ananya Iyer, Imran Ali and Vikram Nair. Here are the tests you can use.

Operator	Meaning	Example
=	equal to	city = 'Noida'
<>	not equal to	city <> 'Delhi'
> and <	greater than, less than	marks > 80
>= and <=	at least, at most	marks >= 40

Combine tests with AND (both must be true), OR (either may be true) and NOT (turns a test around).

SQL

```
SELECT student_id, marks FROM marks  
WHERE subject = 'Maths' AND marks >= 80;
```

TIP

When you mix AND and OR, use brackets to make your meaning clear: WHERE city = 'Delhi' AND (gender = 'F' OR class_id = 3).

Sorting with ORDER BY and LIMIT

Rows come back in no promised order. ORDER BY sorts them. ASC means smallest first (this is the default) and DESC means largest first. LIMIT keeps only the first few rows.

SQL

```
SELECT student_id, marks FROM marks  
WHERE subject = 'Maths'  
ORDER BY marks DESC  
LIMIT 3;
```

This shows the three best Maths marks: 95, 92 and 88. You can sort by more than one column: `ORDER BY city, name` sorts by city, and by name inside each city.

DISTINCT and AS

`DISTINCT` removes repeated rows. `AS` gives a column a friendlier heading, called an **alias**.

SQL

```
SELECT name AS student, city AS lives_in FROM students;
```

REMEMBER

The parts of a query always come in this order: SELECT, FROM, WHERE, ORDER BY, LIMIT. Writing LIMIT before ORDER BY gives an error.

Drills 1 to 5

Try each drill before you read the answer below it.

Drill 1. Show the names and dates of birth of all girls.

SQL

```
SELECT name, dob FROM students WHERE gender = 'F';
```

Drill 2. List the students of class 2 in alphabetical order. (Answer: Ananya Iyer, Rohan Das, Sneha Joshi.)

SQL

```
SELECT name FROM students WHERE class_id = 2 ORDER BY name;
```

Drill 3. Show the three highest Science marks with student ids. (Answer: 97, 91 and 88, from students 8, 4 and 2.)

SQL

```
SELECT student_id, marks FROM marks  
WHERE subject = 'Science' ORDER BY marks DESC LIMIT 3;
```

Drill 4. Show each city that students come from, once each.

SQL

```
SELECT DISTINCT city FROM students ORDER BY city;
```

Drill 5. Show Maths marks below 60. (Answer: students 3, 5 and 9, with 45, 35 and 58.)

SQL

```
SELECT student_id, marks FROM marks  
WHERE subject = 'Maths' AND marks < 60;
```

Better filters: LIKE, IN, BETWEEN and NULL

The `=` test needs an exact match. Real questions are often looser: "names starting with A", "students from any of these three cities", "marks from 60 to 80". SQL has a neat word for each.

LIKE for patterns

`LIKE` matches text against a pattern with two special characters, called **wildcards**:

- `%` stands for any number of characters, even none.
- `_` (underscore) stands for exactly one character.

Pattern	Matches	From our students
'A%'	starts with A	Aarav Mehta, Ananya Iyer
'%a'	ends with a	Aarav Mehta, Pooja Verma, Neha Gupta
'%an%'	has "an" anywhere	Ananya Iyer, Rohan Das, Imran Ali
'_____%'	four letters, then a space	Diya Patel, Neha Gupta

SQL

SELECT name FROM students WHERE name LIKE 'A%';

DID YOU KNOW

With MySQL's usual settings, `LIKE` ignores capital and small letters, so `'a%'` also finds "Aarav".

IN for a list

`IN` checks whether a value is in a list. It is a short way of writing many ORs.

SQL

SELECT name, city FROM students
WHERE city IN ('Noida', 'Faridabad');

`NOT IN` does the opposite: `WHERE city NOT IN ('Delhi')`.

BETWEEN for a range

`BETWEEN a AND b` includes both ends. So `marks BETWEEN 60 AND 80` keeps 60, 80 and everything in between. It works for dates too.

SQL

```
SELECT name, dob FROM students
WHERE dob BETWEEN '2012-01-01' AND '2012-12-31';
```

NULL needs special care

`NULL` is not zero and not a blank. It means "unknown". Because it is unknown, it is never equal to anything, not even another `NULL`. So `WHERE dob = NULL` finds nothing at all. You must write `IS NULL` or `IS NOT NULL`.

SQL

```
SELECT name FROM students WHERE dob IS NULL;
```

This finds Neha Gupta.

REMEMBER

Always write `IS NULL`, never `= NULL`. The wrong version gives no error, just an empty answer, which makes it hard to spot.

Drills 6 to 10

Drill 6. Find students whose name contains "an".

SQL

```
SELECT name FROM students WHERE name LIKE '%an%';
```

Drill 7. Find students from Noida or Faridabad, using `IN`. (Answer: Diya Patel, Kabir Singh, Sneha Joshi, Pooja Verma.)

SQL

```
SELECT name FROM students
WHERE city IN ('Noida', 'Faridabad');
```

Drill 8. Show English marks from 70 to 85. (Answer: 72, 84, 81 and 77.)

SQL

```
SELECT student_id, marks FROM marks
WHERE subject = 'English' AND marks BETWEEN 70 AND 85;
```

Drill 9. Find students born in 2012. (Answer: five students. Neha is not counted, because her date is unknown.)

SQL

```
SELECT name FROM students
WHERE dob BETWEEN '2012-01-01' AND '2012-12-31';
```

Drill 10. Find students whose date of birth is missing.

```
SQL
SELECT name FROM students WHERE dob IS NULL;
```

CHAPTER 5

Counting and summarising with GROUP BY

So far each answer was a list of rows. Often a teacher wants one number instead: how many students, the class average, the highest mark. **Aggregate functions** take many rows and squeeze them into one value.

Function	What it gives	Maths example
COUNT(*)	number of rows	9 marks
SUM(marks)	total	639
AVG(marks)	average	71.0
MAX(marks)	highest	95
MIN(marks)	lowest	35

```
SQL
SELECT COUNT(*), AVG(marks), MAX(marks), MIN(marks)
FROM marks WHERE subject = 'Maths';
```

COUNT(column) counts only the rows where that column is not NULL. So COUNT(*) on students gives 10 but COUNT(dob) gives 9. ROUND(value, 1) rounds to one decimal place, which makes averages tidy.

GROUP BY: one answer per group

GROUP BY splits the rows into groups and runs the aggregate on each group. Think of a teacher sorting answer sheets into three piles, one per subject, and then totalling each pile.

```
SQL
SELECT subject, ROUND(AVG(marks), 1) AS average
FROM marks
GROUP BY subject;
```

subject	average
English	75.3

subject	average
Maths	71.0
Science	71.3

REMEMBER

Every column in SELECT must either be inside an aggregate function or be listed in GROUP BY. Otherwise MySQL 8 usually stops with an error, because it cannot tell which single value to show.

HAVING: filtering groups

WHERE filters rows before they are grouped. **HAVING** filters the groups after they are made. You cannot use an aggregate inside WHERE, so "subjects with an average above 72" needs HAVING.

SQL

```
SELECT subject, ROUND(AVG(marks), 1) AS average
FROM marks
GROUP BY subject
HAVING AVG(marks) > 72;
```

Only English (75.3) passes.

EXAMPLE

"How many marks below 60 in each subject?" uses both: `SELECT subject, COUNT(*) FROM marks WHERE marks < 60 GROUP BY subject;` WHERE first throws out marks of 60 and above, then GROUP BY counts what is left. The answer is English 1, Maths 3, Science 3.

The full order of a query is now: SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY, LIMIT.

Drills 11 to 15

Drill 11. How many students come from each city? (Answer: Delhi 4, Noida 3, Gurugram 2, Faridabad 1.)

SQL

```
SELECT city, COUNT(*) AS students FROM students
GROUP BY city ORDER BY students DESC;
```

Drill 12. Show the highest and lowest mark in each subject.

SQL

```
SELECT subject, MAX(marks) AS highest, MIN(marks) AS lowest
FROM marks GROUP BY subject;
```

Drill 13. Show each student's total marks, highest total first. (Student 8 leads with 282.)

SQL

```
SELECT student_id, SUM(marks) AS total FROM marks
GROUP BY student_id ORDER BY total DESC;
```

Drill 14. Which cities have more than two students? (Answer: Delhi and Noida.)

SQL

```
SELECT city, COUNT(*) FROM students
GROUP BY city HAVING COUNT(*) > 2;
```

Drill 15. Which students have a total above 240? (Answer: students 2, 4 and 8.)

SQL

```
SELECT student_id, SUM(marks) AS total FROM marks
GROUP BY student_id HAVING SUM(marks) > 240;
```

CHAPTER 6

Joining tables

The marks table says student 8 scored 97 in Science, but who is student 8? The name lives in the students table. A **JOIN** puts matching rows from two tables side by side, so you can see both at once.

INNER JOIN

An **inner join** keeps only the rows that have a match in both tables. You say how they match after the word **ON**.

SQL

```
SELECT s.name, c.class_name
FROM students AS s
INNER JOIN classes AS c ON s.class_id = c.class_id
ORDER BY c.class_name, s.name;
```

Here **s** and **c** are short aliases for the table names, so **s.name** means "the name column of students". Aliases save typing and make it clear which table each column comes from. This matters because students and teachers both have a column called **name**, and the next query uses both. Writing just **JOIN** means the same as **INNER JOIN**.

Joining three tables

You can chain joins. This finds each Noida student's class teacher.

SQL

```
SELECT s.name, c.class_name, t.name AS class_teacher
FROM students s
JOIN classes c ON s.class_id = c.class_id
JOIN teachers t ON c.teacher_id = t.teacher_id
WHERE s.city = 'Noida';
```

name	class_name	class_teacher
Diya Patel	8A	Meena Sharma
Kabir Singh	8A	Meena Sharma
Pooja Verma	9A	Farah Khan

LEFT JOIN

An inner join quietly drops rows with no partner. Neha Gupta has no marks, so an inner join of students and marks leaves her out. A **left join** keeps every row of the left table (the one written first). Where there is no match, the columns from the right table show NULL.

SQL

```
SELECT s.name, m.marks
FROM students s
LEFT JOIN marks m
  ON s.student_id = m.student_id AND m.subject = 'Maths'
ORDER BY s.name;
```

All ten students appear, and Neha's mark shows NULL. To find only the students with no marks, keep the rows where the right side is NULL:

SQL

```
SELECT s.name
FROM students s
LEFT JOIN marks m ON s.student_id = m.student_id
WHERE m.student_id IS NULL;
```

TIP

In a left join, a test on the right-hand table belongs in ON, as in the Maths example above. If you put `m.subject = 'Maths'` in WHERE instead, Neha's NULL row fails the test and she disappears again.

Drills 16 to 19

Drill 16. Show each class with its class teacher's name.

SQL

```
SELECT c.class_name, t.name AS class_teacher
FROM classes c JOIN teachers t ON c.teacher_id = t.teacher_id;
```

Drill 17. Show the name and Maths mark of each student, highest first. (Pooja Verma tops with 95.)

SQL

```
SELECT s.name, m.marks
FROM students s JOIN marks m ON s.student_id = m.student_id
WHERE m.subject = 'Maths' ORDER BY m.marks DESC;
```

Drill 18. Show every teacher and their class, including teachers without one. (Harpreet Gill shows NULL.)

SQL

```
SELECT t.name, c.class_name
FROM teachers t
LEFT JOIN classes c ON t.teacher_id = c.teacher_id;
```

Drill 19. Show the average Science mark of each class. (Answer: 8A 75.0, 8B 71.0, 9A 68.0.)

SQL

```
SELECT c.class_name, ROUND(AVG(m.marks), 1) AS science_avg
FROM marks m
JOIN students s ON m.student_id = s.student_id
JOIN classes c ON s.class_id = c.class_id
WHERE m.subject = 'Science'
GROUP BY c.class_name;
```

CHAPTER 7

Queries inside queries

Sometimes you need one answer before you can ask the real question. "Who scored above the Maths average?" needs the average first. A **subquery** is a query written inside brackets within another query. The inner query runs first and hands its answer to the outer one.

A subquery that gives one value

SQL

```
SELECT student_id, marks FROM marks
WHERE subject = 'Maths'
AND marks > (SELECT AVG(marks) FROM marks
             WHERE subject = 'Maths');
```

The inner query works out 71. The outer query then keeps marks above 71: students 1, 2, 4, 7 and 8. You could type 71 by hand, but the subquery stays correct even after the marks change.

A subquery that gives a list

With `IN`, the inner query can return many values.

SQL

```
SELECT name FROM students
WHERE student_id IN
  (SELECT student_id FROM marks WHERE marks < 40);
```

The inner query finds students 5 and 9. The outer query turns those numbers into names: Rohan Das and Vikram Nair.

REMEMBER

Be careful with `NOT IN` when the inner list could contain a `NULL`. If even one `NULL` is in the list, `NOT IN` returns no rows at all. Add `WHERE column IS NOT NULL` inside the subquery to be safe, as in Drill 22.

A subquery in FROM

A subquery can also act as a temporary table. MySQL needs you to give it an alias.

SQL

```
SELECT ROUND(AVG(total), 1) AS average_total
FROM (SELECT student_id, SUM(marks) AS total
      FROM marks GROUP BY student_id) AS t;
```

The inner query makes one total per student. The outer query averages those totals: 217.7.

TRY THIS

Many subqueries can also be written as joins. Rewrite the "marks below 40" query with a `JOIN` and `DISTINCT`, and check that you get the same two names.

Drills 20 to 22

Drill 20. Who has the highest Science mark? Use `MAX` so that ties would also show. (Answer: Pooja Verma.)

SQL

```
SELECT name FROM students
WHERE student_id IN (
  SELECT student_id FROM marks
  WHERE subject = 'Science'
  AND marks = (SELECT MAX(marks) FROM marks
               WHERE subject = 'Science'));
```

Drill 21. List the students in the same class as Kabir Singh. (Answer: Aarav Mehta, Diya Patel, Kabir Singh.)

SQL

```
SELECT name FROM students
WHERE class_id =
  (SELECT class_id FROM students WHERE name = 'Kabir Singh');
```

Drill 22. Which teachers are not class teachers? (Answer: Harpreet Gill.)

SQL

```
SELECT name FROM teachers
WHERE teacher_id NOT IN
  (SELECT teacher_id FROM classes
   WHERE teacher_id IS NOT NULL);
```

CHAPTER 8

Changing data safely

`SELECT` only reads. `UPDATE` changes values that are already there, and `DELETE` removes whole rows. Once they run, there is no undo button, so professionals follow a careful routine.

UPDATE

SQL

```
UPDATE students SET city = 'Noida' WHERE student_id = 5;
```

This moves Rohan Das to Noida. `SET` names the column and its new value. `WHERE` decides which rows change. You can also calculate a new value from the old one. After a recheck, Vikram's Science paper earns two more marks:

SQL

```
UPDATE marks SET marks = marks + 2
WHERE student_id = 9 AND subject = 'Science';
```

SAFETY

An `UPDATE` or `DELETE` without a `WHERE` changes every row in the table. `UPDATE students SET city = 'Noida';` moves the whole school to Noida in one second.

DELETE

SQL

```
DELETE FROM marks
WHERE student_id = 3 AND subject = 'English';
```

Foreign keys protect you here. Try `DELETE FROM students WHERE student_id = 1;` and MySQL refuses, because Aarav still has marks that point to him. You would have to delete his marks first. This rule stops "orphan" rows, which are marks that belong to nobody.

The safe routine

1. Write the WHERE part first, as a SELECT: `SELECT * FROM students WHERE student_id = 5;`
2. Check that exactly the right rows come back.
3. Change `SELECT *` into `UPDATE ... SET ...` or `DELETE`, keeping the same WHERE.
4. Run a SELECT again to confirm the change.

Transactions: a trial run

A **transaction** groups changes so you can keep them all or cancel them all. MySQL's usual table type, InnoDB, supports this.

SQL

```
START TRANSACTION;
DELETE FROM marks WHERE subject = 'English';
SELECT COUNT(*) FROM marks;
ROLLBACK;
```

The count shows 18 inside the transaction.

`ROLLBACK` cancels the delete, and all 27 marks are back. If you are happy with a change, type `COMMIT` instead to make it permanent.

TIP

MySQL has a safety switch: `SET SQL_SAFE_UPDATES = 1;` makes it refuse an UPDATE or DELETE unless its WHERE uses a key column or it has a LIMIT. MySQL Workbench usually turns it on for you.

Drills 23 to 25

Drill 23. Neha's date of birth arrives: 29 May 2011. Record it.

SQL

```
UPDATE students SET dob = '2011-05-29' WHERE student_id = 10;
```

Drill 24. Class 8B (class_id 2) moves to room 105.

SQL

```
UPDATE classes SET room = 105 WHERE class_id = 2;
```

Drill 25. Kabir's English mark was entered by mistake. Remove only that row.

SQL

```
DELETE FROM marks  
WHERE student_id = 3 AND subject = 'English';
```

When you finish these drills, run the script from Chapter 2 again to get back the original data.

CHAPTER 9

Constraints, indexes and views

Constraints: rules the database enforces

A **constraint** is a rule written into a table. The database checks it on every INSERT and UPDATE, so bad data cannot get in, however tired the person typing is.

Constraint	Rule	In our school
PRIMARY KEY	unique and never empty	student_id
FOREIGN KEY	must match a row in another table	students.class_id
NOT NULL	must have a value	teachers.name
UNIQUE	no two rows the same	classes.class_name
CHECK	must pass a test	marks from 0 to 100
DEFAULT	value used when none is given	not used in our tables

Try breaking each rule and read the error:

SQL

```
INSERT INTO marks VALUES (10, 'Maths', 105);  
INSERT INTO students  
VALUES (11, 'Riya Sen', 'F', 'Delhi', '2012-07-07', 9);  
INSERT INTO classes VALUES (4, '8A', 103, 4);
```

The first breaks the CHECK rule. The second names class 9, which does not exist. The third repeats the class name 8A. All three are refused.

You can add a rule to an existing table with **ALTER TABLE**:

SQL

```
ALTER TABLE classes
ADD CONSTRAINT chk_room CHECK (room BETWEEN 100 AND 999);
```

DID YOU KNOW

MySQL began enforcing CHECK rules in version 8.0.16. Older versions accepted the rule but quietly ignored it.

Indexes: the index at the back of a book

To find "photosynthesis" in a science book, you do not read every page. You look in the index at the back. A database **index** works the same way. Without one, MySQL reads every row to find students from Noida. With one, it jumps straight to them.

SQL

```
CREATE INDEX idx_students_city ON students (city);
```

MySQL makes an index for every primary key on its own, and for foreign keys too. Indexes are not free, though. Each one takes space, and every INSERT, UPDATE or DELETE must also update it. Add indexes to columns you often search, join or sort on, not to every column.

TIP

Put the word `EXPLAIN` before a `SELECT` to see whether MySQL plans to use an index. With ten students you will not notice a difference, but on a table with lakhs of rows an index can make a search many times faster.

Views: saved queries

A **view** is a saved `SELECT` with a name. You use it like a table, but it holds no data of its own. Each time you query it, the database runs the saved `SELECT` again, so it is always up to date.

SQL

```
CREATE VIEW report_card AS
SELECT s.name, c.class_name, m.subject, m.marks
FROM students s
JOIN classes c ON s.class_id = c.class_id
JOIN marks m ON s.student_id = m.student_id;

SELECT * FROM report_card WHERE name = 'Diya Patel';
```

Now anyone can get a report card without writing a three-table join. Views also help with privacy: a view can show names and marks while hiding columns such as date of birth. Remove a view with `DROP VIEW report_card;`

Normalisation: designing tables well

Why did we use four tables instead of one big sheet? Imagine the school office keeping everything in a single table:

roll	name	class	class_teacher	subjects
1	Aarav Mehta	8A	Meena Sharma	Maths 78, Science 85
2	Diya Patel	8A	Meena Sharma	Maths 92, Science 88

This causes trouble. If Meena Sharma's name is misspelt, it must be corrected in every row of 8A. If one row is missed, the data now disagrees with itself. Searching for "all Science marks above 80" means pulling numbers out of text. **Normalisation** is a set of steps that removes these problems by splitting data into well-shaped tables. Each step is called a **normal form**.

First normal form (1NF)

Every cell holds one single value, and there are no repeating groups such as "subject1, subject2, subject3" columns. Our sheet breaks 1NF because one cell holds "Maths 78, Science 85". The fix is one row per student per subject, which is exactly our marks table.

Second normal form (2NF)

The table is in 1NF, and every non-key column depends on the whole primary key, not just part of it. This matters only when the key has two or more columns. Suppose the marks table also stored the student's name:

student_id	subject	marks	name
1	Maths	78	Aarav Mehta
1	Science	85	Aarav Mehta

The key is student_id + subject. The mark needs both parts. But the name depends only on student_id, so it repeats for every subject. That is a **partial dependency**. The fix is to keep the name in the students table only.

Third normal form (3NF)

The table is in 2NF, and no non-key column depends on another non-key column. Suppose students stored the class teacher's name. The teacher depends on the class, and the class depends on the student. That chain is called a **transitive dependency**. The fix is to move the teacher to the classes table, which is how we built it.

REMEMBER

A simple test for 3NF: every column should describe the key, the whole key, and nothing but the key.

EXAMPLE

A Lucknow coaching centre keeps one sheet: receipt number, student name, student phone, course name, course length, and amount paid in ! . The course length depends on the course, not on the receipt, so it repeats on every receipt for that course. Splitting it into students, courses and payments tables removes the repetition.

Normalisation does not mean using as many tables as possible. For everyday data that people type in and correct, 3NF is the usual goal: every fact is stored in one place only.

CHECK YOURSELF

Practice questions

Run the script from Chapter 2 again before you start, so that your results match the answers. These 25 questions and the 25 drills inside the chapters make up the book's 50 practice queries.

1. List the names and cities of all boys.
2. Show all marks of student 4, highest first, with the subject.
3. Show the names and dates of birth of the two youngest students.
4. How many students are in class 3?
5. List students whose names start with S or P.
6. Show every mark of 90 or above, with student id and subject.
7. List the students who do not live in Delhi.
8. How many marks in the whole table are below 40?
9. Find the average English mark, rounded to two decimal places.
10. Show each student whose average mark is below 50, with that average rounded to one decimal place.
11. Count the boys and the girls in one query.
12. List every student with their class name, sorted by class and then by name.
13. Show each student's name and total marks, highest total first.
14. Using a LEFT JOIN, find the students who have no marks.
15. Who is the class teacher of Imran Ali?
16. Show the number of students in each class, by class name, including any class with no students.

17. Show the names and marks of students who scored above the overall Science average.
18. Show the students whose total is higher than Aarav Mehta's total (student 1).
19. Rohan Das's Maths paper (student 5) is rechecked and his mark becomes 39. Write the statement.
20. Vikram Nair (student 9) is leaving, and his marks must go. Write the safe sequence: a check first, then the DELETE inside a transaction, and a count to confirm.
21. Create a view called `toppers` that shows the name, subject and mark for every mark of 90 or above.
22. Add a CHECK constraint so that room numbers must be from 100 to 999.
23. Create an index to speed up searches on the subject column of marks.
24. A table has the columns `roll_no`, `name` and `phones`, where `phones` holds "98xxxxxx01,97xxxxxx02". Which normal form does it break? Write a CREATE TABLE for the fix.
25. A table has the key (`student_id`, `subject`) and the columns `marks` and `subject_teacher`. Which normal form does it break, and why? Write a CREATE TABLE for the new table that fixes it.

ANSWER KEY

Answers

1. `SELECT name, city FROM students WHERE gender = 'M';` gives Aarav, Kabir, Rohan, Imran and Vikram.
2. `SELECT subject, marks FROM marks WHERE student_id = 4 ORDER BY marks DESC;` gives Science 91, Maths 88, English 84.
3. `SELECT name, dob FROM students ORDER BY dob DESC LIMIT 2;` gives Rohan Das and Ananya Iyer. The latest date of birth belongs to the youngest student. Neha's NULL date sorts last in DESC order, so she is not picked.
4. `SELECT COUNT(*) FROM students WHERE class_id = 3;` gives 4.
5. `SELECT name FROM students WHERE name LIKE 'S%' OR name LIKE 'P%';` gives Sneha Joshi and Pooja Verma.
6. `SELECT student_id, subject, marks FROM marks WHERE marks >= 90;` gives 6 rows: two for student 2, one for student 4 and three for student 8.
7. `SELECT name FROM students WHERE city <> 'Delhi';` gives 6 students. `WHERE city NOT IN ('Delhi')` also works.
8. `SELECT COUNT(*) FROM marks WHERE marks < 40;` gives 2 (Rohan's Maths and Vikram's Science).
9. `SELECT ROUND(AVG(marks), 2) FROM marks WHERE subject = 'English';` gives 75.33.
10. `SELECT student_id, ROUND(AVG(marks), 1) FROM marks GROUP BY student_id HAVING AVG(marks) < 50;` gives student 5 with 46.0. It needs HAVING, because the test is on an average.
11. `SELECT gender, COUNT(*) FROM students GROUP BY gender;` gives F 5 and M 5.
12. `SELECT s.name, c.class_name FROM students s JOIN classes c ON s.class_id = c.class_id ORDER BY c.class_name, s.name;`

13. `SELECT s.name, SUM(m.marks) AS total
FROM students s JOIN marks m ON
s.student_id = m.student_id GROUP BY
s.student_id, s.name ORDER BY total
DESC;` Pooja Verma is first with 282.
14. `SELECT s.name FROM students s LEFT JOIN
marks m ON s.student_id = m.student_id
WHERE m.student_id IS NULL;` gives Neha Gupta.
15. `SELECT t.name FROM students s JOIN
classes c ON s.class_id = c.class_id
JOIN teachers t ON c.teacher_id =
t.teacher_id WHERE s.name = 'Imran
Ali';` gives Farah Khan.
16. `SELECT c.class_name,
COUNT(s.student_id) FROM classes c LEFT
JOIN students s ON c.class_id =
s.class_id GROUP BY c.class_name;` gives
8A 3, 8B 3, 9A 4. `COUNT(s.student_id)` would
show 0 for an empty class, where `COUNT(*)`
would wrongly show 1.
17. `SELECT s.name, m.marks FROM students s
JOIN marks m ON s.student_id =
m.student_id WHERE m.subject =
'Science' AND m.marks > (SELECT
AVG(marks) FROM marks WHERE subject =
'Science');` gives Aarav, Diya, Ananya, Sneha
and Pooja.
18. `SELECT s.name, SUM(m.marks) FROM
students s JOIN marks m ON s.student_id
= m.student_id GROUP BY s.student_id,
s.name HAVING SUM(m.marks) > (SELECT
SUM(marks) FROM marks WHERE student_id
= 1);` Aarav has 235, so the answer is Diya,
Ananya and Pooja.
19. `UPDATE marks SET marks = 39 WHERE
student_id = 5 AND subject = 'Maths';`
Both parts of the key are in WHERE, so exactly
one row changes.
20. `SELECT * FROM marks WHERE student_id =
9;` shows his 3 rows. Then `START
TRANSACTION;`, `DELETE FROM marks WHERE
student_id = 9;` and `SELECT COUNT(*) FROM
marks;`, which shows 24. Type `COMMIT;` to keep
the change or `ROLLBACK;` to cancel it.
21. `CREATE VIEW toppers AS SELECT s.name,
m.subject, m.marks FROM students s JOIN
marks m ON s.student_id = m.student_id
WHERE m.marks >= 90;` It shows 6 rows.
22. `ALTER TABLE classes ADD CONSTRAINT
chk_room CHECK (room BETWEEN 100 AND
999);` After this, setting a room to 5 is refused.
23. `CREATE INDEX idx_marks_subject ON marks
(subject);` The primary key index starts with
student_id, so it is of little help to a search on
subject alone.
24. It breaks 1NF, because one cell holds two values.
Keep one row per phone number: `CREATE
TABLE student_phones (roll_no INT,
phone VARCHAR(15), PRIMARY KEY
(roll_no, phone));`
25. It breaks 2NF. The teacher depends only on the
subject, which is part of the key, not on the
whole key. Move it to its own table: `CREATE
TABLE subject_teachers (subject
VARCHAR(30) PRIMARY KEY, teacher
VARCHAR(50));`

Glossary

Aggregate function

a function such as COUNT, SUM, AVG, MAX or MIN that turns many rows into one value.

Alias

a short or friendly name given to a column or table with AS.

Column

one kind of detail stored for every row, such as city.

Composite key

a primary key made of two or more columns together.

Constraint

a rule built into a table that the database checks on every change.

Database

an organised store of information that programs can search and change.

Data type

the kind of value a column holds, such as INT, VARCHAR or DATE.

Foreign key

a column whose values must match the primary key of another table.

GROUP BY

the clause that splits rows into groups so an aggregate runs once per group.

HAVING

the clause that filters groups after GROUP BY.

Index

a lookup structure that helps the database find rows without reading them all.

Join

combining rows from two tables that match on a column.

LEFT JOIN

a join that keeps every row of the first table, with NULLs where there is no match.

Normalisation

designing tables so that each fact is stored in exactly one place.

NULL

a marker meaning a value is missing or unknown.

Primary key

a column, or set of columns, that is unique and never empty for each row.

Query

a question or instruction written in SQL.

Row

one record in a table, such as one student.

SQL

Structured Query Language, the language used to work with databases made of tables.

Subquery

a query written inside brackets within another query.

Table

a set of rows and columns that stores one kind of thing.

Transaction

a group of changes that are kept together with COMMIT or cancelled together with ROLLBACK.

View

a saved SELECT query that can be used like a table.

Wildcard

a special character in a LIKE pattern: % for any characters, _ for exactly one.



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.