```sql
-- Drop tables if already exist (to avoid errors while testing)
DROP TABLE IF EXISTS attendance;
DROP TABLE IF EXISTS marks;
DROP TABLE IF EXISTS subjects;
DROP TABLE IF EXISTS students;

-- 1. Students Table
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    name VARCHAR(100),
    class VARCHAR(20)
);

-- 2. Subjects Table
CREATE TABLE subjects (
    subject_id INT PRIMARY KEY,
    subject_name VARCHAR(100)
);

-- 3. Marks Table
CREATE TABLE marks (
    mark_id INT PRIMARY KEY AUTO_INCREMENT,
    student_id INT,
    subject_id INT,
    marks_obtained INT,
    exam_date DATE,
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (subject_id) REFERENCES subjects(subject_id)
);

-- 4. Attendance Table
```

```
CREATE TABLE attendance (
    attendance_id INT PRIMARY KEY AUTO_INCREMENT,
    student_id INT,
    date DATE,
    status ENUM('Present', 'Absent'),
    FOREIGN KEY (student_id) REFERENCES students(student_id)
);

-- -----------------------------
-- Insert Sample Data
-- -----------------------------

-- Students
INSERT INTO students VALUES
(1, 'Amit', '10A'),
(2, 'Riya', '10A'),
(3, 'Vikram', '10A');

-- Subjects
INSERT INTO subjects VALUES
(101, 'Math'),
(102, 'Science'),
(103, 'English');

-- Marks
INSERT INTO marks (student_id, subject_id, marks_obtained, exam_date) VALUES
(1, 101, 90, '2025-03-01'),
(1, 102, 78, '2025-03-02'),
(1, 103, 85, '2025-03-03'),
(2, 101, 65, '2025-03-01'),
(2, 102, 70, '2025-03-02'),
```

```sql
(2, 103, 60, '2025-03-03'),
(3, 101, 40, '2025-03-01'),
(3, 102, 55, '2025-03-02'),
(3, 103, 50, '2025-03-03');

-- Attendance
INSERT INTO attendance (student_id, date, status) VALUES
(1, '2025-03-01', 'Present'),
(1, '2025-03-02', 'Absent'),
(1, '2025-03-03', 'Present'),
(2, '2025-03-01', 'Present'),
(2, '2025-03-02', 'Present'),
(2, '2025-03-03', 'Present'),
(3, '2025-03-01', 'Absent'),
(3, '2025-03-02', 'Present'),
(3, '2025-03-03', 'Absent');

-- -----------------------------
-- Useful Queries
-- -----------------------------

-- 1. Average Marks per Student
SELECT s.student_id, s.name, AVG(m.marks_obtained) AS avg_marks
FROM students s
JOIN marks m ON s.student_id = m.student_id
GROUP BY s.student_id, s.name;

-- 2. Subject-wise Class Average
SELECT sub.subject_name, AVG(m.marks_obtained) AS avg_class_marks
FROM subjects sub
JOIN marks m ON sub.subject_id = m.subject_id
```

```sql
GROUP BY sub.subject_name;


-- 3. Attendance Percentage
SELECT
   s.student_id,
   s.name,
   COUNT(CASE WHEN a.status = 'Present' THEN 1 END) * 100.0 / COUNT(*) AS
attendance_percentage
FROM students s
JOIN attendance a ON s.student_id = a.student_id
GROUP BY s.student_id, s.name;


-- 4. Final Report with Grade
CREATE OR REPLACE VIEW student_performance AS
SELECT
   s.student_id,
   s.name,
   AVG(m.marks_obtained) AS avg_marks,
   COUNT(CASE WHEN a.status = 'Present' THEN 1 END) * 100.0 / COUNT(*) AS
attendance_percentage,
   CASE
      WHEN AVG(m.marks_obtained) >= 85 THEN 'A'
      WHEN AVG(m.marks_obtained) >= 70 THEN 'B'
      WHEN AVG(m.marks_obtained) >= 50 THEN 'C'
      ELSE 'D'
   END AS grade
FROM students s
JOIN marks m ON s.student_id = m.student_id
JOIN attendance a ON s.student_id = a.student_id
GROUP BY s.student_id, s.name;
```

Kya karta hai yeh script (overview)

Ye script:

1. 4 tables banati hai: students, subjects, marks, attendance.

2. Sample data insert karti hai (3 students, 3 subjects, kuch marks aur attendance rows).

3. Kuch useful SELECT queries chalati hai (avg marks, subject average, attendance %).

4. Ek view student_performance banati hai jo har student ka avg marks, attendance % aur grade dikhata hai.

Table-by-table explanation

1) DROP TABLE IF EXISTS ...

DROP TABLE IF EXISTS attendance;

DROP TABLE IF EXISTS marks;

DROP TABLE IF EXISTS subjects;

DROP TABLE IF EXISTS students;

Ye purane tables hata deta hai agar pehle se exist karte hain — testing ke liye useful.

Order important hai: child tables (attendance, marks) ko pehle drop kiya gaya taaki foreign-key errors na aaye.

2) students

```
CREATE TABLE students (
   student_id INT PRIMARY KEY,
   name VARCHAR(100),
   class VARCHAR(20)
);
```

student_id primary key — unique identifier.

name, class basic info.

3) subjects

```
CREATE TABLE subjects (
   subject_id INT PRIMARY KEY,
   subject_name VARCHAR(100)
);
```

subject_id unique id per subject.

4) marks

```
CREATE TABLE marks (
    mark_id INT PRIMARY KEY AUTO_INCREMENT,
    student_id INT,
    subject_id INT,
    marks_obtained INT,
    exam_date DATE,
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (subject_id) REFERENCES subjects(subject_id)
);
```

mark_id auto increment primary key.

student_id aur subject_id foreign keys — referential integrity (won't allow marks for non-existent student/subject).

marks_obtained integer; exam_date se time-series analysis possible.

Note: InnoDB automatically creates indexes for FK columns if needed, lekin explicit indexes add karna best practice hai for performance.

5) attendance

```
CREATE TABLE attendance (
    attendance_id INT PRIMARY KEY AUTO_INCREMENT,
    student_id INT,
    date DATE,
    status ENUM('Present', 'Absent'),
    FOREIGN KEY (student_id) REFERENCES students(student_id)
);
```

status ENUM with two values. Simple to read, lekin ENUM ki limitations hain (future changes). Alternative: tinyint(1) ya status table use kar sakte ho.

Sample data (kya insert hua aur kyon)

3 students: Amit(1), Riya(2), Vikram(3).

3 subjects: Math(101), Science(102), English(103).

Marks: har student ke liye 3 exam rows (ek per subject).

Attendance: 3 dates per student, Present/Absent status.

Ye seeds dashboard banane aur queries test karne ke liye hain.

Queries — detail + kya karta hai + sample calculations

Query 1: Average Marks per Student

```
SELECT s.student_id, s.name, AVG(m.marks_obtained) AS avg_marks
FROM students s
JOIN marks m ON s.student_id = m.student_id
GROUP BY s.student_id, s.name;
```

JOIN = inner join → sirf un students ko show karega jin ke marks table mein rows hain.

AVG(m.marks_obtained) per-student average marks return karega.

Example (manual calculation):

Amit (student_id=1) marks: 90, 78, 85

Step-by-step:

Sum = 90 + 78 + 85 = 253.

Count = 3.

AVG = 253 / 3 = 84.333333...

So avg_marks ≈ 84.3333.

Riya (2): 65 + 70 + 60 = 195 → 195 / 3 = 65.0000

Vikram (3): 40 + 55 + 50 = 145 → 145 / 3 = 48.333333...

> Note: AVG returns NULL if student has no marks.

Query 2: Subject-wise Class Average

SELECT sub.subject_name, AVG(m.marks_obtained) AS avg_class_marks

FROM subjects sub

```
JOIN marks m ON sub.subject_id = m.subject_id
GROUP BY sub.subject_name;
```

Per subject average across all students.

Example quick:

Math marks: 90,65,40 → Sum=195, Count=3 → Avg=195/3=65.0

Science: 78,70,55 → Sum=203 → Avg=203/3 ≈ 67.6667

English: 85,60,50 → Sum=195 → Avg=65.0

Query 3: Attendance Percentage

```
SELECT
   s.student_id,
   s.name,
   COUNT(CASE WHEN a.status = 'Present' THEN 1 END) * 100.0 / COUNT(*) AS attendance_percentage
FROM students s
JOIN attendance a ON s.student_id = a.student_id
GROUP BY s.student_id, s.name;
```

COUNT(CASE WHEN … THEN 1 END) counts present rows (because CASE returns 1 when Present, NULL otherwise; COUNT ignores NULLs).

COUNT(*) is total attendance records for that student.

* 100.0 ensures floating point division.

Example (Amit):

Attendance rows: Present, Absent, Present → Present count = 2, Total = 3

attendance % = (2 * 100.0) / 3 = 200 / 3 = 66.666666... ≈ 66.6667%

Riya: Present, Present, Present → 3/3 → 100%

Vikram: Absent, Present, Absent → 1/3 → 33.3333%

Important caveat: If a student has zero attendance rows, COUNT(*) = 0 → division by zero error. (Script assumes at least one attendance row per student.)

Query 4: Final Report VIEW (student_performance)

```
CREATE OR REPLACE VIEW student_performance AS
SELECT
   s.student_id,
   s.name,
   AVG(m.marks_obtained) AS avg_marks,
   COUNT(CASE WHEN a.status = 'Present' THEN 1 END) * 100.0 / COUNT(*) AS
attendance_percentage,
   CASE
      WHEN AVG(m.marks_obtained) >= 85 THEN 'A'
      WHEN AVG(m.marks_obtained) >= 70 THEN 'B'
```

```
        WHEN AVG(m.marks_obtained) >= 50 THEN 'C'
        ELSE 'D'
    END AS grade
FROM students s
JOIN marks m ON s.student_id = m.student_id
JOIN attendance a ON s.student_id = a.student_id
GROUP BY s.student_id, s.name;
```

Ye view student-level summary banata hai: avg_marks, attendance_percentage, grade.

JOIN marks aur JOIN attendance dono inner joins hain, to students jin ke dono records hain (marks aur attendance) unko include karta hai.

Expected output (sample numeric):

For Amit: avg_marks ≈ 84.3333 → Grade = B (kyunki >=70 aur <85). Attendance ≈ 66.6667%.

Riya: avg_marks = 65 → Grade C, Attendance = 100%

Vikram: avg ≈ 48.3333 → Grade D, Attendance ≈ 33.3333%

Quick: How to see results

Run:

```
SELECT * FROM student_performance;
```

You should get rows:

student_id, name, avg_marks, attendance_percentage, grade

(Values as calculated above.)

Doubts:_

1. students s ka matlab kya hai?

```
FROM students s
```

Yahan students table ko ek short name (alias) s diya gaya hai.

Matlab: ab aapko bar-bar students.student_id likhne ki zarurat nahi, sirf s.student_id likh sakte ho.

👉 Example:

```
SELECT s.student_id, s.name
FROM students s;
```

ye bilkul same hai jaise

```
SELECT students.student_id, students.name
FROM students;
```

---

2. marks m aur subjects sub kya hai?

```
JOIN marks m ON s.student_id = m.student_id
JOIN subjects sub ON sub.subject_id = m.subject_id
```

Yahan bhi marks table ko alias m diya gaya hai.

subjects table ko alias sub diya gaya hai.

Toh ab aap likh sakte ho:

m.marks_obtained → marks table ka marks_obtained column

sub.subject_name → subjects table ka subject_name column

---

3. Kyun alias use karte hain?

Without alias:

```
SELECT students.student_id, students.name, marks.marks_obtained
FROM students
JOIN marks ON students.student_id = marks.student_id;
```

With alias (short & clean):

```
SELECT s.student_id, s.name, m.marks_obtained
FROM students s
JOIN marks m ON s.student_id = m.student_id;
```

👉 Dono same result denge, bas dusra version chhota aur readable hai.