# SQL Practice Problems

---

## Beginner Problems

1. **Select all students** from the `Students` table.
2. Show the **first name and last name** of students who are older than 18.
3. Find all courses that start with the word "Intro".
4. List all teachers in the `Teachers` table sorted by last name.
5. Show the unique course names from the `Courses` table.
6. Find all students with no assigned `CourseID` (NULL values).

---

## Intermediate Problems

1. Count how many students are enrolled in each course.
2. Find the **youngest student** in the database.
3. Calculate the **average age** of students.
4. Show all courses that have **more than 3 students** enrolled.
5. Use an **INNER JOIN** to list students with their course names.
6. Use a **LEFT JOIN** to list all students and their courses, including students not assigned to any course.
7. Create a query that shows all courses taught by "Mr. Smith".
8. Use a `CASE` statement to label students as **"Minor"** if age < 18 and **"Adult"** otherwise.

---

## Advanced Problems

1. Create a view called `StudentCourses` that shows `StudentName`, `CourseName`, and `TeacherName`.
2. Add a new student with name "John Doe" (age 20) enrolled in course `CSE101`.
3. Update "John Doe's" age to 21.
4. Delete all students who are younger than 10.
5. Create an index on the `Courses.CourseName` column.
6. Write a query to find students who are NOT enrolled in any course.

---

## Challenge Problems

1. Find the teacher who teaches the **most students**.
2. Write a query to find courses with **no students enrolled**.
3. Show the **top 3 oldest students**.
4. Backup the database (conceptual – show SQL `BACKUP DATABASE` command).
5. Prevent duplicate entries for student emails by applying a **UNIQUE constraint**.

---

# Solutions (peek only after trying!)

1. `SELECT * FROM Students;`
2. `SELECT FirstName, LastName FROM Students WHERE Age > 18;`
3. `SELECT * FROM Courses WHERE CourseName LIKE 'Intro%';`
4. `SELECT * FROM Teachers ORDER BY LastName;`
5. `SELECT DISTINCT CourseName FROM Courses;`
6. `SELECT * FROM Students WHERE CourseID IS NULL;`
7. `SELECT CourseID, COUNT(*) FROM Students GROUP BY CourseID;`
8. `SELECT * FROM Students ORDER BY Age ASC LIMIT 1;`
9. `SELECT AVG(Age) FROM Students;`
10. `SELECT CourseID, COUNT(*) FROM Students GROUP BY CourseID HAVING COUNT(*) > 3;`
11. `SELECT s.FirstName, s.LastName, c.CourseName FROM Students s INNER JOIN Courses c ON s.CourseID = c.CourseID;`
12. `SELECT s.FirstName, s.LastName, c.CourseName FROM Students s LEFT JOIN Courses c ON s.CourseID = c.CourseID;`
13. `SELECT c.CourseName FROM Courses c INNER JOIN Teachers t ON c.TeacherID = t.TeacherID WHERE t.LastName = 'Smith';`
14. `SELECT FirstName, LastName, CASE WHEN Age < 18 THEN 'Minor' ELSE 'Adult' END AS Category FROM Students;`
15. `CREATE VIEW StudentCourses AS SELECT s.FirstName, s.LastName, c.CourseName, t.LastName AS TeacherName FROM Students s INNER JOIN Courses c ON s.CourseID = c.CourseID INNER JOIN Teachers t ON c.TeacherID = t.TeacherID;`
16. `INSERT INTO Students (FirstName, LastName, Age, CourseID) VALUES ('John', 'Doe', 20, 'CSE101');`
17. `UPDATE Students SET Age = 21 WHERE FirstName = 'John' AND LastName = 'Doe';`
18. `DELETE FROM Students WHERE Age < 10;`
19. `CREATE INDEX idx_course_name ON Courses(CourseName);`
20. `SELECT * FROM Students WHERE CourseID IS NULL;`
21. `SELECT t.LastName, COUNT(*) AS StudentCount FROM Teachers t INNER JOIN Courses c ON t.TeacherID = c.TeacherID INNER JOIN Students s ON c.CourseID = s.CourseID GROUP BY t.LastName ORDER BY StudentCount DESC LIMIT 1;`
22. `SELECT * FROM Courses c LEFT JOIN Students s ON c.CourseID = s.CourseID WHERE s.StudentID IS NULL;`
23. `SELECT * FROM Students ORDER BY Age DESC LIMIT 3;`
24. `BACKUP DATABASE SchoolDB TO DISK = 'C:\\backups\\schooldb.bak';`
25. `ALTER TABLE Students ADD CONSTRAINT UQ_StudentEmail UNIQUE (Email);`