Lab 2 — SQL Querying I: Filtering & Joins

Module I · Week 2 · Duration: 2 hours Dataset: `data/university.sql` · **Deliverable:** `lab02_solution.sql`

Objectives

Practice, on a realistic multi-table schema, the single-table querying skills from Lecture 5 and the multi-table querying skills from Lecture 6:

1. Single-table filtering, sorting, `DISTINCT`, `LIMIT`, and string functions.
2. Using conditional logic (`CASE`) and pattern matching (`LIKE`).
3. **Inner joins** across two, three, and four tables.
4. **Outer joins** and the "find the unmatched rows" (anti-join) pattern.
5. **Self joins**.
6. Recognizing and avoiding **row-explosion** and **cross-join** pitfalls.
7. Running SQL from a **Python** script.

> **Time budget (≈2 h):** setup ~10 min · Part 1 ~30 min · Part 2 ~30 min · Part 3 ~25 min · Part 4 ~15 min · Part 5 (Python) ~10 min.

Setup (≈10 min)

```
sqlite3 lab2.db < data/university.sql
sqlite3 lab2.db ".tables"
```

Enable foreign keys in every session (`PRAGMA foreign_keys = ON;`). Skim the schema — the relationships you'll join on:

```
student.dept_name   -> department       instructor.dept_name -> department
course.dept_name    -> department       takes.id  -> student
teaches.id -> instructor                takes/teaches (course_id, sec_id, semester, year) -> section
prereq.course_id / prereq.prereq_id -> course      advisor.s_id/i_id -> student/instructor
```

Write each answer into `lab02_solution.sql`, **commented with its number**.

Part 1 — Single-table queries (≈30 min)

1. Titles of all courses in the `'Comp. Sci.'` department.
2. Instructors earning **between** 70,000 and 90,000, highest salary first, showing their `name` and a computed column `monthly_pay` (salary / 12.0 rounded to 2 decimal places).
3. Distinct semesters that appear in `section`.
4. The **3** students with the highest `tot_cred` (id, name, tot_cred). If there is a tie, sort by `name` alphabetically.
5. Select `course_id` and `title` for courses whose title contains the word "System" (or "system"), but does **not** contain "Intro" anywhere in the title.
6. Instructors **not** in the History or Music departments — i.e. `dept_name NOT IN ('History','Music')`, ordered by department then name.
7. Number of instructors (single number) using `COUNT(*)`.

8. Categorize instructors' salaries using a `CASE` statement: "High" (≥ 90000), "Medium" ($70000 \ldots 89999$), and "Low" (< 70000). Show `name`, `salary`, and the new column `salary_category`.
9. **(Hard)** Find all students who have a `tot_cred` that is a non-zero multiple of 10. (Hint: use the modulo operator `%` and explicitly exclude `tot_cred = 0`).

Checkpoint: Q4's top student is Tanaka (120). Q7 returns 12.

Part 2 — Inner joins (≈30 min)

10. Each **student's name** with their **department's building**.
11. Each **instructor's name** and the **titles of courses** they teach (join `instructor → teaches → section → course`).
12. Every **student**, the **course title** they took, and their **grade** (join `student → takes → section → course`).
13. Names of students advised by instructor **'Katz'** (join `advisor → student`, filter on the instructor).
14. Course title and the **capacity** of the room its section uses (join `section → classroom → course`).
15. Distinct **department buildings** that host at least one **section** (careful: which tables connect a section to a building?).
16. Names of instructors who teach in the **'Taylor'** building.

Watch row counts. Q12 returns *one row per enrollment*, so a student who took 3 courses appears 3 times. That's correct here — but note it.

Checkpoint: Q13 returns Zhang and Brown (Katz advises 00128 and 76543).

Part 3 — Outer joins & anti-joins (≈25 min)

17. **All** sections and the **id of the instructor** teaching them — including sections with **no** instructor (`LEFT JOIN section → teaches`).
18. Using the result of Q17, list only the sections that have **no assigned instructor** (the `WHERE ... IS NULL` anti-join pattern).
19. **All** departments and the **number of students** in each — showing **0** for departments with none (`LEFT JOIN` + `COUNT`).
20. Instructors who teach **nothing** (in `instructor` but not referenced by `teaches`) — via `LEFT JOIN ... IS NULL` **or** `NOT EXISTS`.
21. Every course title, and the name of an instructor who taught it — including courses **taught by no one** (chain of `LEFT JOIN`s).

The outer-join trap: in Q17, if you filter on a `teaches` column in the `WHERE` clause, you silently turn the `LEFT JOIN` back into an inner join. Put such conditions in the `ON` clause. Explain in a comment why.

Checkpoint: Q18 returns two rows — `(CS-319, 2)` and `(MATH-101, 1)` — the sections with no row in `teaches`.

Part 4 — Self joins & gotchas (≈15 min)

22. All **pairs of instructors in the same department**, with no self-pairs and no mirror duplicates (`a.id < b.id`).
23. Each **course title with its prerequisite's title** (self-join on `course` via `prereq`).
24. **Row-explosion demo:** run `SELECT COUNT(*) FROM student, department;` and, in a comment, explain why the number equals `#students × #departments` and what single clause turns it into a meaningful join.
25. **INNER vs LEFT counts:** report the row count of `section INNER JOIN teaches` vs `section LEFT JOIN teaches` and explain the difference in one sentence.

Checkpoint: Q24 returns 105 (15 × 7).

Part 5 — SQL from Python (≈10 min)

Create `explore.py` (you will submit this):

```
import sqlite3

conn = sqlite3.connect("lab2.db")
conn.row_factory = sqlite3.Row          # access columns by name
cur = conn.cursor()

# a parameterized query - the ? is filled safely by the driver
dept = "Comp. Sci."
cur.execute("SELECT name, salary FROM instructor WHERE dept_name = ?", (dept,))
for row in cur.fetchall():
    print(row["name"], row["salary"])

# an aggregate
cur.execute("SELECT COUNT(*) AS n FROM student")
print("total students:", cur.fetchone()["n"])

conn.close()
```

Run it:

```
python3 explore.py
```

Task: modify the script to accept a department name from the command line (`import sys; dept = sys.argv[1]`) and print all instructors in that department — still using a **parameterized** query (never an f-string).

Checkpoint: `python3 explore.py Physics` lists Einstein and Gold.

Stretch goals (optional)

- Rewrite Q20 both ways (`LEFT JOIN ... IS NULL` and `NOT EXISTS`) and confirm identical results.
- List students who took a course **taught by their own advisor**.
- Find departments whose building also hosts a section of a **different** department's course.

Submission

Submit `lab02_solution.sql` with all 25 answers, each preceded by a comment of the form `-- Q10: each student's name with department building`. For questions 24 and 25 (and the Q17/Q18 trap), include a short comment explaining the result. At the bottom of your SQL file, paste the output of your Part 5 `explore.py` run in a block comment.

Also submit: - `explore.py` (your modified version).

Self-check: `sqlite3 fresh.db < data/university.sql` then pasting your queries runs with no errors, and your join queries return the checkpoint values given above.