CSL303/MAL505 — Database Management Systems

Module I: Relational Foundations & SQL
Lecture 6: Single-Table Queries & Functions

Amit Kumar Dhar

IIT Bhilai

Where we are

- Lecture 5: created tables with **DDL** and constraints.
- Today: querying **one** table (`SELECT`, `WHERE`), then extending to **joins** (spanning many tables) and **set operations**.
- This covers the most-used SQL skills in practice — we'll do lots of examples.

Agenda

- Single-table `SELECT`: projection, `WHERE`, `ORDER BY`
- Expressions and string/numeric functions
- The join mental model
- `INNER JOIN ... ON` and `USING`
- Multi-table and Self joins
- Outer joins (LEFT / RIGHT / FULL)
- Set ops: `UNION` / `INTERSECT` / `EXCEPT`

Single-Table Queries

The SELECT statement — the workhorse

```
SELECT   <columns>
FROM     <table>
WHERE    <row filter>
ORDER BY <columns>
LIMIT    <n>;
```

- Today: a **single table**. Joins next lecture.
- Maps to RA: SELECT = π, WHERE = σ.

Projection: the SELECT list

```sql
SELECT name, salary FROM instructor;   -- specific columns
SELECT * FROM instructor;              -- all columns
```

- `*` is convenient interactively, but **name your columns** in real code (stable, readable, faster).

SELECT keeps duplicates! (π does not)

```
SELECT dept_name FROM instructor;          -- duplicates kept
SELECT DISTINCT dept_name FROM instructor; -- duplicates removed (= \pi)
```

- SQL tables are **bags** (multisets), unlike pure RA relations.
- `DISTINCT` gives you the RA/set behaviour when you want it.

WHERE — filtering rows (σ)

```sql
SELECT name FROM instructor WHERE dept_name = 'Comp. Sci.';
SELECT name FROM instructor WHERE salary > 85000;
```

- Comparison operators: `= <> < <= > >=`.
- Combine with AND, OR, NOT.

```sql
WHERE dept_name = 'Comp. Sci.' AND salary > 80000
```

Quick Check: WHERE

Question: How would you find all instructors in the 'Physics' department earning more than 90,000?

Quick Check: WHERE

Question: How would you find all instructors in the 'Physics' department earning more than 90,000?

Answer:

```sql
SELECT * FROM instructor
WHERE dept_name = 'Physics' AND salary > 90000;
```

Quick Check: WHERE

Question: How would you find all instructors in the 'Physics' department earning more than 90,000?

Quick Check: WHERE

Question: How would you find all instructors in the 'Physics' department earning more than 90,000?

Answer:

```
SELECT * FROM instructor
WHERE dept_name = 'Physics' AND salary > 90000;
```

Handy WHERE predicates

- Range: `WHERE salary BETWEEN 70000 AND 90000` (inclusive).
- Membership: `WHERE dept_name IN ('Physics','Finance')`.
- NULL test: `WHERE grade IS NULL` (never `= NULL`!).
- Negation: `WHERE dept_name NOT IN (...)`, `NOT BETWEEN`.

Pattern matching with LIKE

- `%` = any sequence of characters; `_` = exactly one character.

```
SELECT name FROM instructor WHERE name LIKE 'S%';   -- starts with S
SELECT title FROM course WHERE title LIKE '%System%';
SELECT id FROM student WHERE id LIKE '_2%';         -- 2nd char is 2
```

- Case sensitivity varies by engine; Postgres has `ILIKE` for case-insensitive.

Quick Check: LIKE

Question: Write a query to find all courses whose title ends with 'Intro'.

Quick Check: LIKE

Question: Write a query to find all courses whose title ends with 'Intro'.

Answer:

```
SELECT * FROM course WHERE title LIKE '%Intro';
```

Quick Check: LIKE

Question: Write a query to find all courses whose title ends with 'Intro'.

Quick Check: LIKE

Question: Write a query to find all courses whose title ends with 'Intro'.

Answer:

```sql
SELECT * FROM course WHERE title LIKE '%Intro';
```

Expressions and computed columns

- The SELECT list can contain **expressions** (generalized projection):

```
SELECT name, salary, salary / 12 AS monthly_pay
FROM   instructor;
```

- Arithmetic: `+ - * /`. String: `||` (concatenation).
- `AS` gives the result column a readable **alias**.

```
SELECT name || ' (' || dept_name || ')' AS label FROM instructor;
```

Column and table aliases

```sql
SELECT i.name AS instructor_name, i.salary
FROM   instructor AS i
WHERE  i.salary > 90000;
```

- Table alias `i` shortens references (essential with joins).
- AS is optional in many dialects but improves clarity.

ORDER BY — sorting output

```
SELECT name, salary FROM instructor ORDER BY salary DESC;
SELECT name FROM instructor ORDER BY dept_name ASC, salary DESC;
```

- Default is **ASC**. Sort by multiple keys, each with its own direction.
- Remember: a relation has **no inherent order** — ordering exists **only** in the query output.

Quick Check: ORDER BY

Question: How do you list all students sorted by their department alphabetically, and then by their total credits descending?

Quick Check: ORDER BY

Question: How do you list all students sorted by their department alphabetically, and then by their total credits descending?

Answer:

```
SELECT * FROM student
ORDER BY dept_name ASC, tot_cred DESC;
```

Quick Check: ORDER BY

Question: How do you list all students sorted by their department alphabetically, and then by their total credits descending?

Quick Check: ORDER BY

Question: How do you list all students sorted by their department alphabetically, and then by their total credits descending?

Answer:

```
SELECT * FROM student
ORDER BY dept_name ASC, tot_cred DESC;
```

NULLs in ORDER BY

- Where do NULLs sort? It's engine-dependent.
  - PostgreSQL: **NULLS LAST** by default for **ASC**.
  - SQLite: NULLs sort **first** in **ASC**.
- Be explicit when it matters (Postgres):

```
ORDER BY grade ASC NULLS LAST;
```

LIMIT and OFFSET — top-N and paging

```
SELECT name, salary FROM instructor
ORDER BY salary DESC
LIMIT 3;                -- top 3 earners

SELECT * FROM student ORDER BY id LIMIT 10 OFFSET 20;  -- page 3
```

- `LIMIT` caps rows; `OFFSET` skips rows first.
- Always pair `LIMIT` with `ORDER BY` — otherwise "top" is undefined.

Quick Check: LIMIT

Question: How would you find the single instructor with the lowest salary?

Quick Check: LIMIT

Question: How would you find the single instructor with the lowest salary?

Answer:

```
SELECT * FROM instructor
ORDER BY salary ASC
LIMIT 1;
```

Quick Check: LIMIT

Question: How would you find the single instructor with the lowest salary?

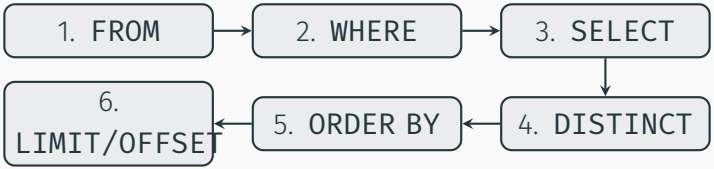
Quick Check: LIMIT

Question: How would you find the single instructor with the lowest salary?

Answer:

```
SELECT * FROM instructor
ORDER BY salary ASC
LIMIT 1;
```

Logical evaluation order of a query



- Written order $\neq$ evaluation order.
- This is why a `SELECT` alias usually can't be used in `WHERE`.

Putting it together

”Top 5 highest-paid instructors outside History, showing monthly pay.”

```
SELECT name,
       dept_name,
       salary / 12 AS monthly_pay
FROM   instructor
WHERE  dept_name <> 'History'
ORDER  BY salary DESC
LIMIT  5;
```

- Filter → project/compute → order → limit. One table, real answer.

Functions & Utilities

String functions you'll use often

```sql
SELECT upper(name), lower(dept_name), length(name),
       substr(name, 1, 3),                 -- first 3 chars
       trim('  hi  '),                      -- 'hi'
       replace(title, 'Intro.', 'Introduction')
FROM   instructor;
```

- `||` concatenates. Function names are mostly portable across SQLite/Postgres.
- Postgres also has `POSITION`, `LEFT/RIGHT`; SQLite has `instr`.

Numeric and rounding functions

```
SELECT salary,
       round(salary / 12.0, 2) AS monthly,   -- 2 decimals
       abs(-5),                               -- 5
       salary % 1000                          -- modulo
FROM   instructor;
```

- Integer vs real division: `salary / 12` may truncate if both are integers — use `salary / 12.0` to force real division.

Dates (a note before Week 3)

- SQLite has **no native date type** — store ISO strings (`'2022-03-01'`) and use `date()`, `strftime()`:

```sql
SELECT strftime('%Y', order_date) AS yr FROM orders;   -- SQLite
```

- PostgreSQL has real `DATE`/`TIMESTAMP` with `EXTRACT`, `now()`, intervals.
- We'll use dates heavily on the `store` dataset in Lab 3.

Comments and scripts

```
-- single line comment
/* multi-line
   comment */
```

- Put your work in a **`.sql` script** and run it as a batch: `sqlite3 university.db < my_queries.sql`.
- Comment each query with the task it answers — required in the labs.

CASE — conditional expressions (preview)

```
SELECT name,
       CASE WHEN salary >= 90000 THEN 'high'
            WHEN salary >= 70000 THEN 'mid'
            ELSE 'low' END AS band
FROM   instructor;
```

- CASE computes a value per row based on conditions.
- Full treatment (with NULL logic) in Lecture 9.

Quick Check: CASE

Question: Write a query that selects student names and assigns a 'Status' of 'Senior' if `tot_cred >= 90`, else 'Junior'.

Quick Check: CASE

Question: Write a query that selects student names and assigns a 'Status' of 'Senior' if `tot_cred >= 90`, else 'Junior'.

Answer:

```
SELECT name,
       CASE WHEN tot_cred >= 90 THEN 'Senior'
            ELSE 'Junior' END AS Status
FROM student;
```

Quick Check: CASE

Question: Write a query that selects student names and assigns a 'Status' of 'Senior' if `tot_cred >= 90`, else 'Junior'.

Quick Check: CASE

Question: Write a query that selects student names and assigns a 'Status' of 'Senior' if `tot_cred >= 90`, else 'Junior'.

Answer:

```
SELECT name,
       CASE WHEN tot_cred >= 90 THEN 'Senior'
            ELSE 'Junior' END AS Status
FROM student;
```

Summary — key takeaways

- `SELECT` = π, `WHERE` = σ, but SQL keeps duplicates unless `DISTINCT`.
- `ORDER BY` and `LIMIT` shape the output, but do not affect the stored relation.
- Functions (string, numeric, date) let you compute values on the fly.
- `CASE` expressions allow for conditional logic within queries.

Before next lecture

- Review single-table queries and functions.

Next time — Lecture 6a: SQL Joins & Set Operations.

Questions?