



LECTURE 05

Introduction to Database Systems

SQL Practice and Query Building

Xiaofan Li

School of Computer Science, Nanjing University



Practice Roadmap

Build, test, and read SQL queries from the command line.

01 PostgreSQL practice

Connect with `psql`; create, inspect, populate, query, and remove objects.

02 Query building

Read constraint failures; combine aliases, predicates, ordering, sets, and aggregates.

Throughout: predict the result table before running the query.



Connect, Create, and Inspect

Use `psql`, the PostgreSQL command-line client. Backslash commands are client meta-commands.

```
psql -U postgres
Password: *****
postgres=# CREATE DATABASE company;
postgres=# \c company
company=# \l
```

Command	Purpose
<code>\l</code>	list databases
<code>\c db</code>	connect to database
<code>\d table</code>	describe table
<code>\q</code>	quit

SQL ends with a semicolon; most backslash commands do not. The prompt shows the active database.

Reconnect to another database before dropping `company`; you cannot drop the database you are using.



Define Related Tables

department(dno, dname)

```
CREATE TABLE department (  
    dno integer PRIMARY KEY,  
    dname varchar(40) NOT NULL  
);  
\d department
```

dno identifies a department; dname is required. \d shows types, nullability, and the primary-key index.

employee(eid, ename, dno)

```
CREATE TABLE employee (  
    eid integer PRIMARY KEY,  
    ename varchar(40) NOT NULL,  
    dno integer REFERENCES  
        department(dno)  
);
```

eid identifies an employee. The foreign key checks that each non-NULL dno exists on insertion or update.



Populate the Parent, Then the Child

Insert departments before employees, so every referenced department already exists.

1. department(dno, dname)

```
INSERT INTO department VALUES  
(10, 'Engineering'),  
(20, 'Finance');
```

2. employee(eid, ename, dno)

```
INSERT INTO employee VALUES  
(101, 'Ava', 10),  
(102, 'Noah', 20);
```

Atomic statement: a multi-row INSERT either inserts every row or fails.
The foreign key makes the parent-child relationship explicit.



Read Constraint Failures

A rejected statement explains which declared rule the new data would violate.

Duplicate primary key

```
INSERT INTO department VALUES  
(10, 'Design');
```

ERROR: duplicate key value

Department 10 already exists. A primary key cannot repeat.

Missing parent row

```
INSERT INTO employee VALUES  
(103, 'Mia', 99);
```

ERROR: foreign key violation

Department 99 does not exist. The reference cannot be satisfied.

The database rejects the statement; it does not invent missing parents or repair conflicting keys.



From Base Tables to a Join

SELECT * FROM department;

dno	dname
10	Engineering
20	Finance

SELECT * FROM employee;

eid	ename	dno
101	Ava	10
102	Noah	20

All pairs: SELECT * FROM employee, department; produces $2 \times 2 = 4$ rows.

```
SELECT E.ename, D.dname
FROM employee AS E
JOIN department AS D ON E.dno=D.dno;
```

ename	dname
Ava	Engineering
Noah	Finance

Equivalent filter: FROM employee AS E, department AS D WHERE E.dno=D.dno. Headers are attributes; rows are tuples.



Clean Up in Dependency Order

DELETE removes rows; DROP removes a database object.

01

Remove selected rows

```
DELETE FROM employee WHERE eid=102;
```

02

Drop the child table before its parent

```
DROP TABLE employee; DROP TABLE department;
```

03

Reconnect: `\c postgres` then `DROP DATABASE company;`



DISTINCT vs. ALL

Select buildings containing a classroom with capacity below 100.

CLASSROOM INPUT

building	room	capacity
Packard	101	500
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

ALL is the default. DISTINCT removes duplicate output rows.

```
SELECT DISTINCT building  
FROM classroom WHERE capacity < 100;
```

3 rows: Painter, Taylor, Watson.

```
SELECT ALL building  
FROM classroom WHERE capacity < 100;
```

4 rows: Painter, Taylor, Watson, Watson.



Match Students to Their Departments

```
SELECT student.name,  
       department.budget  
FROM student, department  
WHERE student.dept_name =  
       department.dept_name  
AND department.budget < 100000;
```

name	budget
Brandt	50000
Peltier	70000
Levy	70000
Sanchez	80000
Snow	70000
Aoi	85000
Bourikas	85000
Tanaka	90000

The conceptual product creates all student-department pairs; the equality predicate retains the matching department for each student.



Relation and Column Aliases

```
SELECT S.name AS student_name,  
       D.budget AS department_budget  
FROM student AS S, department AS D  
WHERE S.dept_name = D.dept_name  
AND D.budget < 100000;
```

S and **D** shorten relation names.

student_name and
department_budget label output
columns.

Qualify a shared attribute such as dept_name; an unshared attribute such as budget is unambiguous, though qualifying it can improve clarity.



Combining AND and OR

Find instructors earning below 80,000, plus every instructor whose department is in Watson or Taylor.

```
SELECT I.name  
FROM instructor AS I, department AS D  
WHERE D.dept_name = I.dept_name  
AND (I.salary < 80000 OR D.building IN ('Watson', 'Taylor'));
```

Parentheses make the intended grouping explicit: the department match applies to both alternatives.



LIKE Patterns

```
SELECT title  
FROM course  
WHERE course_id LIKE '___-%';
```

title
Investment Banking
World History

Each `_` matches exactly one character; `%` matches any sequence. The pattern selects identifiers with exactly three characters before the hyphen.

The hyphen is literal in `LIKE`; it separates the three-letter prefix from the numeric part.



ORDER BY with Two Keys

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

First sort rows by department name in ascending order. Within each department, place larger credit totals first.

name	dept.	credits
Tanaka	Biology	120
Zhang	Comp. Sci.	102
Brown	Comp. Sci.	58
Shankar	Comp. Sci.	32
Peltier	Physics	56
Levy	Physics	46
Snow	Physics	0



IN as a Value-List Test

```
SELECT course_id
FROM teaches
WHERE semester IN ('Fall','Spring')
AND year = 2018;
```

course_id
CS-315
MU-199
HIS-351
CS-101
CS-319
CS-319

IN is a compact disjunction. Add DISTINCT if repeated course identifiers should be removed.



Set Operation: UNION

```
SELECT course_id FROM teaches
WHERE semester='Fall' AND year=2018
UNION
SELECT course_id FROM teaches
WHERE semester='Spring' AND year=2018;
```

The two query results must be union-compatible.
UNION removes duplicates; UNION ALL retains them.

Fall courses $\cup$ Spring courses



INTERSECT vs. EXCEPT

Both examples start with names from Comp. Sci. or Finance, then combine sets of names.

INTERSECT: keep names in both sets

```
SELECT name FROM instructor
WHERE dept_name IN ('Comp. Sci.','Finance')
INTERSECT
SELECT name FROM instructor
WHERE salary < 80000;
```

Result: Srinivasan, Katz.

EXCEPT: remove names in the second

```
SELECT name FROM instructor
WHERE dept_name IN ('Comp. Sci.','Finance')
EXCEPT
SELECT name FROM instructor
WHERE salary > 70000 AND salary < 90000;
```

Result: Srinivasan, Brandt, Wu.

With unique names, these correspond to AND / exclusion predicates. EXCEPT uses the open salary interval (70000, 90000); endpoints are not removed.



AVG, GROUP BY, and HAVING

building	room	capacity
Packard	101	500
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

```
SELECT building, AVG(capacity)
FROM classroom
GROUP BY building
HAVING AVG(capacity) > 25;
```

building	avg
Packard	500
Taylor	70
Watson	40

WHERE filters rows; HAVING filters completed groups.



Scalar Aggregates: MIN and MAX

Without GROUP BY, the entire input is one group; each query returns one row and one value.

MIN: least instructor salary

```
SELECT MIN(salary) AS least_salary  
FROM instructor;
```

least_salary

40,000

MAX: most student credits

```
SELECT MAX(tot_cred) AS max_credits  
FROM student;
```

max_credits

120



Aggregate Function: COUNT

```
SELECT building,  
       COUNT(course_id) AS course_count  
FROM section  
GROUP BY building;
```

building	count
Packard	4
Painter	3
Watson	3

COUNT(course_id) counts non-NULL course identifiers in each building group. COUNT(*) counts rows.



Aggregate Function: SUM

course	department	credits
CS-101	Comp. Sci.	4
CS-190	Comp. Sci.	4
CS-315	Comp. Sci.	3
BIO-301	Biology	4
FIN-201	Finance	3

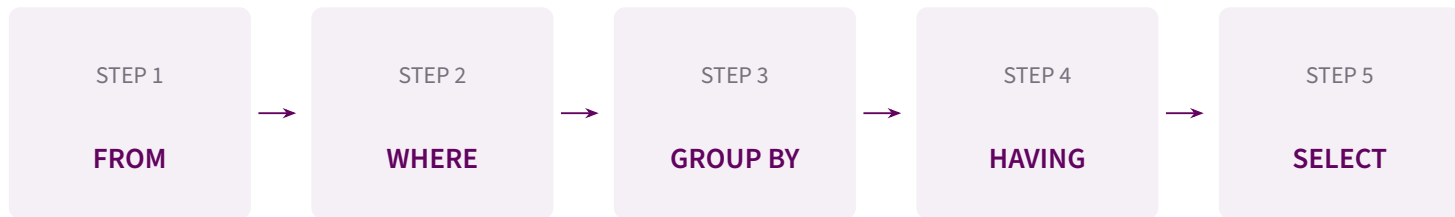
```
SELECT dept_name, SUM(credits)
FROM course
GROUP BY dept_name;
```

Comp. Sci.: $4 + 4 + 3 = 11$

For Comp. Sci., sum the three course-credit values:
 $4 + 4 + 3 = 11$.



A Reliable Query-Reading Order



Then apply DISTINCT, set operations, and ORDER BY. This logical order explains why aggregate expressions belong in HAVING, not WHERE.



Lecture Summary

01 `psql` supports a complete create, inspect, insert, query, and cleanup workflow.

02 Declared constraints reject invalid database states.

03 Predicates, set operations, and aggregates transform query results.