Aggregation

Step 1: PROMPT THE AI

Prompt sent to ChatGPT (April, 2026):

Given the following PostgreSQL database schema:

```
CREATE TABLE classroom (
   building      VARCHAR(15),
   room_number   VARCHAR(7),
   capacity      NUMERIC(4,0),
   PRIMARY KEY (building, room_number)
);

CREATE TABLE department (
   dept_name     VARCHAR(20),
   building      VARCHAR(15),
   budget        NUMERIC(12,2) CHECK (budget > 0),
   PRIMARY KEY (dept_name)
);

CREATE TABLE course (
   course_id     VARCHAR(8),
   title         VARCHAR(50),
   dept_name     VARCHAR(20),
   credits       NUMERIC(2,0) CHECK (credits > 0),
   PRIMARY KEY (course_id),
   FOREIGN KEY (dept_name) REFERENCES department(dept_name)
      ON DELETE SET NULL
);

CREATE TABLE instructor (
   ID            VARCHAR(5),
   name          VARCHAR(20) NOT NULL,
   dept_name     VARCHAR(20),
   salary        NUMERIC(8,2) CHECK (salary > 29000),
   PRIMARY KEY (ID),
   FOREIGN KEY (dept_name) REFERENCES department(dept_name)
      ON DELETE SET NULL
```

```
);

CREATE TABLE section (
   course_id     VARCHAR(8),
   sec_id        VARCHAR(8),
   semester      VARCHAR(6) CHECK (semester IN ('Fall', 'Winter', 'Spring', 'Summer')),
   year          NUMERIC(4,0) CHECK (year > 1701 AND year < 2100),
   building      VARCHAR(15),
   room_number   VARCHAR(7),
   time_slot_id  VARCHAR(4),
   PRIMARY KEY (course_id, sec_id, semester, year),
   FOREIGN KEY (course_id) REFERENCES course(course_id)
      ON DELETE CASCADE,
   FOREIGN KEY (building, room_number) REFERENCES classroom(building,
room_number)
      ON DELETE SET NULL
);

CREATE TABLE teaches (
   ID            VARCHAR(5),
   course_id     VARCHAR(8),
   sec_id        VARCHAR(8),
   semester      VARCHAR(6),
   year          NUMERIC(4,0),
   PRIMARY KEY (ID, course_id, sec_id, semester, year),
   FOREIGN KEY (course_id, sec_id, semester, year)
      REFERENCES section(course_id, sec_id, semester, year)
      ON DELETE CASCADE,
   FOREIGN KEY (ID) REFERENCES instructor(ID)
      ON DELETE CASCADE
);

CREATE TABLE student (
   ID            VARCHAR(5),
   name          VARCHAR(20) NOT NULL,
   dept_name     VARCHAR(20),
   tot_cred      NUMERIC(3,0) CHECK (tot_cred >= 0),
   PRIMARY KEY (ID),
   FOREIGN KEY (dept_name) REFERENCES department(dept_name)
      ON DELETE SET NULL
```

```
);

CREATE TABLE takes (
   ID           VARCHAR(5),
   course_id    VARCHAR(8),
   sec_id       VARCHAR(8),
   semester     VARCHAR(6),
   year         NUMERIC(4,0),
   grade        VARCHAR(2),
   PRIMARY KEY (ID, course_id, sec_id, semester, year),
   FOREIGN KEY (course_id, sec_id, semester, year)
      REFERENCES section(course_id, sec_id, semester, year)
      ON DELETE CASCADE,
   FOREIGN KEY (ID) REFERENCES student(ID)
      ON DELETE CASCADE
);

CREATE TABLE advisor (
   s_ID         VARCHAR(5),
   i_ID         VARCHAR(5),
   PRIMARY KEY (s_ID),
   FOREIGN KEY (i_ID) REFERENCES instructor(ID)
      ON DELETE SET NULL,
   FOREIGN KEY (s_ID) REFERENCES student(ID)
      ON DELETE CASCADE
);

CREATE TABLE time_slot (
   time_slot_id  VARCHAR(4),
   day          VARCHAR(1),
   start_hr     NUMERIC(2) CHECK (start_hr >= 0 AND start_hr < 24),
   start_min    NUMERIC(2) CHECK (start_min >= 0 AND start_min < 60),
   end_hr       NUMERIC(2) CHECK (end_hr >= 0 AND end_hr < 24),
   end_min      NUMERIC(2) CHECK (end_min >= 0 AND end_min < 60),
   PRIMARY KEY (time_slot_id, day, start_hr, start_min)
);

CREATE TABLE prereq (
   course_id    VARCHAR(8),
   prereq_id    VARCHAR(8),
```

```
    PRIMARY KEY (course_id, prereq_id),
    FOREIGN KEY (course_id) REFERENCES course(course_id)
        ON DELETE CASCADE,
    FOREIGN KEY (prereq_id) REFERENCES course(course_id)
);
```

The database contains approximately:- 2,000 students- 30,000 enrollment records in "takes"- 200 courses- 100 sections- 50 instructors- 20 departments

Write a SQL query to: Find the total number of course enrollments and the average total credits (tot_cred) of students in each department. Rank departments from highest enrollment count to lowest. Please use standard PostgreSQL syntax.

Step 2: AI-Generated SQL (Unmodified)

```
SELECT
    s.dept_name,
    COUNT(t.course_id) AS total_enrollments,
    AVG(s.tot_cred) AS avg_total_credits,
    RANK() OVER (ORDER BY COUNT(t.course_id) DESC) AS dept_rank
FROM student s
LEFT JOIN takes t
    ON s.ID = t.ID
GROUP BY s.dept_name
ORDER BY total_enrollments DESC, s.dept_name;
```

This query:

- joins student to takes using student ID
- counts all enrollment records per department
- computes the average tot_cred of students in each department
- ranks departments from highest to lowest enrollment using RANK()

Step 3: Run EXPLAIN ANALYZE

```
EXPLAIN (ANALYZE, BUFFERS, FORMAT TEXT)
SELECT
    s.dept_name,
```

```
    COUNT(t.course_id) AS total_enrollments,
    AVG(s.tot_cred) AS avg_total_credits,
    RANK() OVER (ORDER BY COUNT(t.course_id) DESC) AS dept_rank
FROM student s
LEFT JOIN takes t
    ON s.ID = t.ID
GROUP BY s.dept_name
ORDER BY total_enrollments DESC, s.dept_name;
```

Execution Plan Output

```
"Incremental Sort  (cost=884.64..885.85 rows=20 width=57) (actual time=10.511..10.513
rows=20.00 loops=1)"
"  Sort Key: (count(t.course_id)) DESC, s.dept_name"
"  Presorted Key: (count(t.course_id))"
"  Full-sort Groups: 1  Sort Method: quicksort  Average Memory: 26kB  Peak Memory: 26kB"
"  Buffers: shared hit=235"
"  ->  WindowAgg  (cost=884.62..884.95 rows=20 width=57) (actual time=10.494..10.504
rows=20.00 loops=1)"
"        Window: w1 AS (ORDER BY (count(t.course_id)) ROWS UNBOUNDED
PRECEDING)"
"        Storage: Memory  Maximum Storage: 17kB"
"        Buffers: shared hit=235"
"        ->  Sort  (cost=884.60..884.65 rows=20 width=49) (actual time=10.481..10.483
rows=20.00 loops=1)"
"              Sort Key: (count(t.course_id)) DESC"
"              Sort Method: quicksort  Memory: 26kB"
"              Buffers: shared hit=235"
"              ->  HashAggregate  (cost=883.92..884.17 rows=20 width=49) (actual
time=10.459..10.467 rows=20.00 loops=1)"
"                    Group Key: s.dept_name"
"                    Batches: 1  Memory Usage: 32kB"
"                    Buffers: shared hit=235"
"                    ->  Hash Right Join  (cost=60.00..658.92 rows=30000 width=17) (actual
time=0.471..5.990 rows=30000.00 loops=1)"
"                          Hash Cond: ((t.id)::text = (s.id)::text)"
"                          Buffers: shared hit=235"
"                          ->  Seq Scan on takes t  (cost=0.00..520.00 rows=30000 width=9) (actual
time=0.009..0.996 rows=30000.00 loops=1)"
"                                Buffers: shared hit=220"
```

```
"                    ->  Hash  (cost=35.00..35.00 rows=2000 width=18) (actual time=0.443..0.443
rows=2000.00 loops=1)"
"                          Buckets: 2048  Batches: 1  Memory Usage: 119kB"
"                          Buffers: shared hit=15"
"                          ->  Seq Scan on student s  (cost=0.00..35.00 rows=2000 width=18) (actual
time=0.015..0.196 rows=2000.00 loops=1)"
"                                Buffers: shared hit=15"
"Planning:"
"  Buffers: shared hit=12"
"Planning Time: 0.367 ms"
"Execution Time: 10.620 ms"
```

Step 4: IDENTIFY ISSUES

Issue 1: Join Student and Takes Condition

In this query: AS avg_total_credits, because the student-to-takes join occurred before aggregating, each student appears once for every row in takes, meaning that AVG(s.tot_cred) is not the average credits per student in the department, but rather a weighted average where students with more course enrollments count more times.

Issue 2: Type Cast in Join Condition

In the join condition: (t.id)::text = (s.id)::text. PostgreSQL is casting both varchar columns to text for the comparison. While this doesn't prevent index use here (there is no index to use), this implicit casting pattern can prevent index usage when indexes exist.

Step 5: OPTIMIZE

Creating Indexes

```
-- Index for joining takes to student on ID
CREATE INDEX idx_takes_id ON takes(ID);

-- Index for filtering takes by course/id
CREATE INDEX idx_takes_id_course_id ON takes(ID, course_id);
```

Optimized query

```
– Original query
SELECT
   s.dept_name,
   COUNT(t.course_id) AS total_enrollments,
   AVG(s.tot_cred) AS avg_total_credits,
   RANK() OVER (ORDER BY COUNT(t.course_id) DESC) AS dept_rank
FROM student s
LEFT JOIN takes t
   ON s.ID = t.ID
GROUP BY s.dept_name
ORDER BY total_enrollments DESC, s.dept_name;

– New query
WITH enrollment_counts AS (
   SELECT
      s.dept_name,
      COUNT(t.course_id) AS total_enrollments
   FROM student s
   LEFT JOIN takes t
      ON s.ID = t.ID
   GROUP BY s.dept_name
),
credit_averages AS (
   SELECT
      dept_name,
      AVG(tot_cred) AS avg_total_credits
   FROM student
   GROUP BY dept_name
)
SELECT
   e.dept_name,
   e.total_enrollments,
   c.avg_total_credits,
   RANK() OVER (ORDER BY e.total_enrollments DESC) AS dept_rank
FROM enrollment_counts e
JOIN credit_averages c
   ON e.dept_name = c.dept_name
ORDER BY e.total_enrollments DESC, e.dept_name;
```

New Execution Plan

"Incremental Sort (cost=855.36..856.56 rows=20 width=57) (actual time=10.005..10.009 rows=20.00 loops=1)"
" Sort Key: (count(t.course_id)) DESC, s.dept_name"
" Presorted Key: (count(t.course_id))"
" Full-sort Groups: 1 Sort Method: quicksort Average Memory: 26kB Peak Memory: 26kB"
" Buffers: shared hit=250"
" -> WindowAgg (cost=855.33..855.66 rows=20 width=57) (actual time=9.988..9.998 rows=20.00 loops=1)"
"   Window: w1 AS (ORDER BY (count(t.course_id)) ROWS UNBOUNDED PRECEDING)"
"   Storage: Memory Maximum Storage: 17kB"
"   Buffers: shared hit=250"
"   -> Sort (cost=855.31..855.36 rows=20 width=49) (actual time=9.980..9.983 rows=20.00 loops=1)"
"   Sort Key: (count(t.course_id)) DESC"
"   Sort Method: quicksort Memory: 25kB"
"   Buffers: shared hit=250"
"   -> Hash Join (cost=854.37..854.88 rows=20 width=49) (actual time=9.957..9.974 rows=20.00 loops=1)"
"   Hash Cond: ((student.dept_name)::text = (s.dept_name)::text)"
"   Buffers: shared hit=250"
"   -> HashAggregate (cost=45.00..45.25 rows=20 width=41) (actual time=0.445..0.457 rows=20.00 loops=1)"
"   Group Key: student.dept_name"
"   Batches: 1 Memory Usage: 32kB"
"   Buffers: shared hit=15"
"   -> Seq Scan on student (cost=0.00..35.00 rows=2000 width=13) (actual time=0.009..0.083 rows=2000.00 loops=1)"
"   Buffers: shared hit=15"
"   -> Hash (cost=809.12..809.12 rows=20 width=17) (actual time=9.494..9.495 rows=20.00 loops=1)"
"   Buckets: 1024 Batches: 1 Memory Usage: 10kB"
"   Buffers: shared hit=235"
"   -> HashAggregate (cost=808.92..809.12 rows=20 width=17) (actual time=9.479..9.483 rows=20.00 loops=1)"
"   Group Key: s.dept_name"
"   Batches: 1 Memory Usage: 32kB"
"   Buffers: shared hit=235"
"   -> Hash Right Join (cost=60.00..658.92 rows=30000 width=13) (actual time=0.415..5.914 rows=30000.00 loops=1)"

```
"                     Hash Cond: ((t.id)::text = (s.id)::text)"
"                     Buffers: shared hit=235"
"                     ->  Seq Scan on takes t  (cost=0.00..520.00 rows=30000 width=9)
(actual time=0.012..0.983 rows=30000.00 loops=1)"
"                           Buffers: shared hit=220"
"                     ->  Hash  (cost=35.00..35.00 rows=2000 width=14) (actual
time=0.387..0.387 rows=2000.00 loops=1)"
"                           Buckets: 2048  Batches: 1  Memory Usage: 109kB"
"                           Buffers: shared hit=15"
"                           ->  Seq Scan on student s  (cost=0.00..35.00 rows=2000 width=14)
(actual time=0.010..0.151 rows=2000.00 loops=1)"
"                                 Buffers: shared hit=15"
"Planning:"
"  Buffers: shared hit=12"
"Planning Time: 0.416 ms"
"Execution Time: 10.084 ms"
```

Performance Comparison

Metric	Before (Full Scan)	After (Optimized Query)	Change
Scan on takes	Seq Scan (30,000 rows)	Seq Scan (30,000 rows)	No change
Buffer Pages	235	250	+ 6%
Execution Time	10.620 ms	10.084 ms	- 6%

(For the indexing portion, I partially created my own code and used ChatGPT for the other part. For optimizing the codes, I tried doing it by hand, but I wasn't able to do it right, so I went ahead and used AI to correct it for me, which worked).

Multi-Table Join:

Prompt sent to ChatGPT (April, 2026):

Write a SQL query to: For each instructor, list their name, department, the courses they teach, the semester and year, and the number of students enrolled in each of their sections. Include only sections with at least 10 students. Please use standard PostgreSQL syntax.

AI-Generated SQL (Unmodified)

```
SELECT
    i.name AS instructor_name,
    i.dept_name AS department,
    t.course_id,
    s.semester,
    s.year,
    COUNT(k.ID) AS student_count
FROM instructor i
JOIN teaches t
    ON i.ID = t.ID
JOIN section s
    ON t.course_id = s.course_id
   AND t.sec_id = s.sec_id
   AND t.semester = s.semester
   AND t.year = s.year
LEFT JOIN takes k
    ON s.course_id = k.course_id
   AND s.sec_id = k.sec_id
   AND s.semester = k.semester
   AND s.year = k.year
GROUP BY
    i.ID,
    i.name,
    i.dept_name,
    t.course_id,
    s.sec_id,
    s.semester,
    s.year
HAVING COUNT(k.ID) >= 10
ORDER BY
```

```
    i.name,
    s.year,
    s.semester,
    t.course_id;
```

PART 2: EXECUTION PLAN ANALYSIS

```
EXPLAIN (ANALYZE, BUFFERS, FORMAT TEXT)
SELECT
    i.name AS instructor_name,
    i.dept_name AS department,
    t.course_id,
    s.semester,
    s.year,
    COUNT(k.ID) AS student_count
FROM instructor i
JOIN teaches t
    ON i.ID = t.ID
JOIN section s
    ON t.course_id = s.course_id
   AND t.sec_id = s.sec_id
   AND t.semester = s.semester
   AND t.year = s.year
LEFT JOIN takes k
    ON s.course_id = k.course_id
   AND s.sec_id = k.sec_id
   AND s.semester = k.semester
   AND s.year = k.year
GROUP BY
    i.ID,
    i.name,
    i.dept_name,
    t.course_id,
    s.sec_id,
    s.semester,
    s.year
HAVING COUNT(k.ID) >= 10
ORDER BY
    i.name,
```

```
    s.year,
    s.semester,
    t.course_id;
```

Execution Plan Output

```
"Sort  (cost=2804.32..2829.32 rows=10000 width=164) (actual time=21.788..21.800
rows=100.00 loops=1)"
"  Sort Key: i.name, s.year, s.semester, t.course_id"
"  Sort Method: quicksort  Memory: 31kB"
"  Buffers: shared hit=225 read=1"
"  ->  HashAggregate  (cost=1764.93..2139.93 rows=10000 width=164) (actual
time=20.956..21.023 rows=100.00 loops=1)"
"        Group Key: s.year, s.semester, t.course_id, i.id, s.sec_id"
"        Filter: (count(k.id) >= 10)"
"        Batches: 1  Memory Usage: 545kB"
"        Buffers: shared hit=222 read=1"
"        ->  Hash Right Join  (cost=29.21..1314.93 rows=30000 width=161) (actual
time=0.992..12.016 rows=30000.00 loops=1)"
"              Hash Cond: (((k.course_id)::text = (s.course_id)::text) AND ((k.sec_id)::text =
(s.sec_id)::text) AND ((k.semester)::text = (s.semester)::text) AND (k.year = s.year))"
"              Buffers: shared hit=222 read=1"
"              ->  Seq Scan on takes k  (cost=0.00..520.00 rows=30000 width=21) (actual
time=0.012..1.046 rows=30000.00 loops=1)"
"                    Buffers: shared hit=220"
"              ->  Hash  (cost=27.21..27.21 rows=100 width=160) (actual time=0.957..0.964
rows=100.00 loops=1)"
"                    Buckets: 1024  Batches: 1  Memory Usage: 16kB"
"                    Buffers: shared hit=2 read=1"
"                    ->  Hash Join  (cost=23.90..27.21 rows=100 width=160) (actual time=0.874..0.935
rows=100.00 loops=1)"
"                          Hash Cond: (((t.course_id)::text = (s.course_id)::text) AND ((t.sec_id)::text =
(s.sec_id)::text) AND ((t.semester)::text = (s.semester)::text) AND (t.year = s.year))"
"                          Buffers: shared hit=2 read=1"
"                          ->  Hash Join  (cost=19.90..22.16 rows=100 width=156) (actual
time=0.805..0.828 rows=100.00 loops=1)"
"                                Hash Cond: ((t.id)::text = (i.id)::text)"
"                                Buffers: shared hit=1 read=1"
"                                ->  Seq Scan on teaches t  (cost=0.00..2.00 rows=100 width=21) (actual
time=0.007..0.010 rows=100.00 loops=1)"
```

```
"                         Buffers: shared hit=1"
"                       ->  Hash  (cost=14.40..14.40 rows=440 width=140) (actual
time=0.790..0.790 rows=50.00 loops=1)"
"                             Buckets: 1024  Batches: 1  Memory Usage: 11kB"
"                             Buffers: shared read=1"
"                             ->  Seq Scan on instructor i  (cost=0.00..14.40 rows=440 width=140)
(actual time=0.770..0.773 rows=50.00 loops=1)"
"                                   Buffers: shared read=1"
"                   ->  Hash  (cost=2.00..2.00 rows=100 width=16) (actual time=0.062..0.068
rows=100.00 loops=1)"
"                         Buckets: 1024  Batches: 1  Memory Usage: 13kB"
"                         Buffers: shared hit=1"
"                         ->  Seq Scan on section s  (cost=0.00..2.00 rows=100 width=16) (actual
time=0.033..0.040 rows=100.00 loops=1)"
"                               Buffers: shared hit=1"
"Planning:"
"  Buffers: shared hit=75 read=4 dirtied=1"
"Planning Time: 4.257 ms"
"Execution Time: 22.552 ms"
```

Step 4: IDENTIFY ISSUES

Issue 1: Left Join and Having Clause Condition

The issue with the query is that it removes rows where no matching students exist, because groups with no matches have COUNT(k,ID) equal to zero, failing the having condition. Additionally, the left join condition doesn't really preserve unmatched sections in the final result.

Issue 2: Incorrect Aggregation

If an instructor is listed multiple times in the teaches table for the same section, or if the relationship between teaches and section isn't strictly 1:1, the join will create duplicate rows for every student in takes table.

Step 5: OPTIMIZE

Creating Indexes

```
CREATE INDEX idx_takes_section_join
ON takes (course_id, sec_id, semester, year);
```

```
CREATE INDEX idx_teaches_id ON teaches(ID);
```

Optimized query

```
-- Original query

SELECT
    i.name AS instructor_name,
    i.dept_name AS department,
    t.course_id,
    s.semester,
    s.year,
    COUNT(k.ID) AS student_count
FROM instructor i
JOIN teaches t
    ON i.ID = t.ID
JOIN section s
    ON t.course_id = s.course_id
   AND t.sec_id = s.sec_id
   AND t.semester = s.semester
   AND t.year = s.year
LEFT JOIN takes k
    ON s.course_id = k.course_id
   AND s.sec_id = k.sec_id
   AND s.semester = k.semester
   AND s.year = k.year
GROUP BY
    i.ID,
    i.name,
    i.dept_name,
    t.course_id,
    s.sec_id,
    s.semester,
    s.year
HAVING COUNT(k.ID) >= 10
ORDER BY
    i.name,
    s.year,
    s.semester,
```

```
    t.course_id;

-- New query
SELECT
        i.name AS instructor_name,
        i.dept_name AS department,
        S.course_id,
        S.sec_id,
        S.semester,
        s.year,
COUNT(k.ID) AS student_count
FROM instructor i
JOIN teaches t
        ON i.ID = t.ID
JOIN section s
        ON t.course_id = s.course_id
        AND t.sec_id = s.sec_id
        AND t.semester = s.semester
        AND t.year = s.year
JOIN takes k
        ON s.course_id = k.course_id
        AND s.sec_id = k.sec_id
        AND s.semester = k.semester
        AND s.year = k.year
GROUP BY
        i.ID,
        i.name,
        I.dept_name,
        S.course_id,
        S.sec_id,
        S.semester,
        s.year
HAVING COUNT(k.ID) >= 10
ORDER BY
        i.name,
        S.year,
        S.semester,
        S.course_id,
        s.sec_id;
```

New Execution Plan

```
"Sort  (cost=2560.44..2585.44 rows=10000 width=164) (actual time=28.013..28.019
rows=100.00 loops=1)"
"  Sort Key: i.name, s.year, s.semester, s.course_id, s.sec_id"
"  Sort Method: quicksort  Memory: 31kB"
"  Buffers: shared hit=16733"
"  ->  HashAggregate  (cost=1521.06..1896.06 rows=10000 width=164) (actual
time=27.806..27.883 rows=100.00 loops=1)"
"        Group Key: s.year, s.semester, s.course_id, s.sec_id, i.id"
"        Filter: (count(k.id) >= 10)"
"        Batches: 1  Memory Usage: 545kB"
"        Buffers: shared hit=16733"
"        ->  Nested Loop  (cost=24.20..1071.06 rows=30000 width=161) (actual
time=0.108..18.415 rows=30000.00 loops=1)"
"              Buffers: shared hit=16733"
"              ->  Hash Join  (cost=23.90..27.21 rows=100 width=172) (actual time=0.070..0.176
rows=100.00 loops=1)"
"                    Hash Cond: (((t.course_id)::text = (s.course_id)::text) AND ((t.sec_id)::text =
(s.sec_id)::text) AND ((t.semester)::text = (s.semester)::text) AND (t.year = s.year))"
"                    Buffers: shared hit=3"
"                    ->  Hash Join  (cost=19.90..22.16 rows=100 width=156) (actual time=0.027..0.069
rows=100.00 loops=1)"
"                          Hash Cond: ((t.id)::text = (i.id)::text)"
"                          Buffers: shared hit=2"
"                          ->  Seq Scan on teaches t  (cost=0.00..2.00 rows=100 width=21) (actual
time=0.008..0.014 rows=100.00 loops=1)"
"                                Buffers: shared hit=1"
"                          ->  Hash  (cost=14.40..14.40 rows=440 width=140) (actual time=0.015..0.015
rows=50.00 loops=1)"
"                                Buckets: 1024  Batches: 1  Memory Usage: 11kB"
"                                Buffers: shared hit=1"
"                                ->  Seq Scan on instructor i  (cost=0.00..14.40 rows=440 width=140)
(actual time=0.006..0.009 rows=50.00 loops=1)"
"                                      Buffers: shared hit=1"
"                    ->  Hash  (cost=2.00..2.00 rows=100 width=16) (actual time=0.039..0.040
rows=100.00 loops=1)"
"                          Buckets: 1024  Batches: 1  Memory Usage: 13kB"
"                          Buffers: shared hit=1"
```

"   -> Seq Scan on section s (cost=0.00..2.00 rows=100 width=16) (actual time=0.014..0.021 rows=100.00 loops=1)"
"   Buffers: shared hit=1"
"   -> Memoize (cost=0.30..12.17 rows=6 width=21) (actual time=0.019..0.160 rows=300.00 loops=100)"
"   Cache Key: t.course_id, t.sec_id, t.semester, t.year"
"   Cache Mode: logical"
"   Hits: 0 Misses: 100 Evictions: 0 Overflows: 0 Memory Usage: 1616kB"
"   Buffers: shared hit=16730"
"   -> Index Scan using idx_takes_section_join on takes k (cost=0.29..12.16 rows=6 width=21) (actual time=0.018..0.105 rows=300.00 loops=100)"
"   Index Cond: (((course_id)::text = (t.course_id)::text) AND ((sec_id)::text = (t.sec_id)::text) AND ((semester)::text = (t.semester)::text) AND (year = t.year))"
"   Index Searches: 100"
"   Buffers: shared hit=16730"
"Planning:"
" Buffers: shared hit=8"
"Planning Time: 0.925 ms"
"Execution Time: 28.424 ms"

Performance Comparison

Metric	Before (Full Scan)	After (Indexed Filter)	Change
Scan on takes	Seq Scan (30,000 rows)	Index Scan on takes (100 index searches)	No change
Buffer Pages	225	16733	+ 7302.21%
Execution Time	22.552 ms	28.424 ms	+ 26.04%

(For the indexing portion, I partially created my own code and used ChatGPT for the other part. For optimizing the codes, I tried doing it by hand, but I wasn't able to do it right, so I went ahead and used AI to correct it for me, which worked).

Correlated Subquery:

Prompt sent to ChatGPT (April, 2026):

Write a SQL query to: Find all courses where every section ever offered had at least one student who received an ‘A’ grade.

AI-Generated SQL (Unmodified)

```
SELECT c.course_id, c.title
FROM course c
WHERE NOT EXISTS (
   SELECT 1
   FROM section s
   WHERE s.course_id = c.course_id
    AND NOT EXISTS (
       SELECT 1
       FROM takes t
       WHERE t.course_id = s.course_id
         AND t.sec_id = s.sec_id
         AND t.semester = s.semester
         AND t.year = s.year
         AND t.grade = 'A'
    )
);
```

This works by returning courses for which there does **not** exist any section without at least one student earning an `'A'`.

PART 2: EXECUTION PLAN ANALYSIS

```
EXPLAIN (ANALYZE, BUFFERS, FORMAT TEXT)
SELECT c.course_id, c.title
FROM course c
WHERE NOT EXISTS (
   SELECT 1
   FROM section s
   WHERE s.course_id = c.course_id
```

```
    AND NOT EXISTS (
        SELECT 1
        FROM takes t
        WHERE t.course_id = s.course_id
          AND t.sec_id = s.sec_id
          AND t.semester = s.semester
          AND t.year = s.year
          AND t.grade = 'A'
    )
);
```

Execution Plan Output

```
"Hash Right Anti Join  (cost=6.50..606.48 rows=115 width=21) (actual time=1.652..1.664
rows=115.00 loops=1)"
"  Hash Cond: ((s.course_id)::text = (c.course_id)::text)"
"  Buffers: shared hit=223"
"  ->  Nested Loop Anti Join  (cost=0.00..599.25 rows=100 width=4) (actual time=1.550..1.567
rows=100.00 loops=1)"
"        Join Filter: (((t.course_id)::text = (s.course_id)::text) AND ((t.sec_id)::text =
(s.sec_id)::text) AND ((t.semester)::text = (s.semester)::text) AND (t.year = s.year))"
"        Buffers: shared hit=221"
"        ->  Seq Scan on section s  (cost=0.00..2.00 rows=100 width=16) (actual time=0.006..0.009
rows=100.00 loops=1)"
"              Buffers: shared hit=1"
"        ->  Materialize  (cost=0.00..595.00 rows=1 width=16) (actual time=0.015..0.015
rows=0.00 loops=100)"
"              Storage: Memory  Maximum Storage: 17kB"
"              Buffers: shared hit=220"
"              ->  Seq Scan on takes t  (cost=0.00..595.00 rows=1 width=16) (actual
time=1.540..1.541 rows=0.00 loops=1)"
"                    Filter: ((grade)::text = 'A'::text)"
"                    Rows Removed by Filter: 30000"
"                    Buffers: shared hit=220"
"  ->  Hash  (cost=4.00..4.00 rows=200 width=21) (actual time=0.068..0.068 rows=200.00
loops=1)"
"        Buckets: 1024  Batches: 1  Memory Usage: 19kB"
"        Buffers: shared hit=2"
"        ->  Seq Scan on course c  (cost=0.00..4.00 rows=200 width=21) (actual time=0.021..0.033
rows=200.00 loops=1)"
```

" Buffers: shared hit=2"
"Planning:"
" Buffers: shared hit=40 read=3 dirtied=2"
"Planning Time: 4.602 ms"
"Execution Time: 1.688 ms"

STEP 4: IDENTIFY ISSUES

Issue 1: Double-Negative Logic

The two NOT EXISTS lines make it difficult because they return the course only if there is no section, and where there is no A student

Issue 2: Courses W/O Sections

Major logical edge case due to if no course has no matching rows in section, the outer subquery finds nothing.

Step 5: OPTIMIZE

Creating Indexes

```
CREATE INDEX idx_section_course_id
ON section (course_id);

CREATE INDEX idx_takes_section_grade
ON takes (course_id, sec_id, semester, year, grade);
```

Optimized Query

```
-- Original query retained:
SELECT c.course_id, c.title
FROM course c
WHERE NOT EXISTS (
   SELECT 1
   FROM section s
   WHERE s.course_id = c.course_id
     AND NOT EXISTS (
        SELECT 1
```

```
        FROM takes t
        WHERE t.course_id = s.course_id
          AND t.sec_id = s.sec_id
          AND t.semester = s.semester
          AND t.year = s.year
          AND t.grade = 'A'
    )
);
```

(The original query is kept because it captures the “every section has at least one A student” condition. Alternative rewrites either changed the intended logic or produced no qualifying rows in the dataset. Indexes were implemented to strengthen the correlated conditions, although no structural rewrite was adopted.)

Window Function:

Prompt sent to ChatGPT (April, 2026):

Write a SQL query to: For each department, rank instructors by salary from highest to lowest. Show the instructor's name, department, salary, their rank within the department, and what percentile they fall into.

AI-Generated SQL (Unmodified)

```
SELECT
   name AS instructor_name,
   dept_name AS department,
   salary,
   RANK() OVER (
      PARTITION BY dept_name
      ORDER BY salary DESC
   ) AS salary_rank,
   PERCENT_RANK() OVER (
      PARTITION BY dept_name
      ORDER BY salary DESC
   ) AS salary_percentile
FROM instructor
ORDER BY department, salary_rank, instructor_name;
```

PART 2: EXECUTION PLAN ANALYSIS

```
EXPLAIN (ANALYZE, BUFFERS, FORMAT TEXT)
SELECT
   name AS instructor_name,
   dept_name AS department,
   salary,
   RANK() OVER (
      PARTITION BY dept_name
      ORDER BY salary DESC
   ) AS salary_rank,
```

```
    PERCENT_RANK() OVER (
        PARTITION BY dept_name
        ORDER BY salary DESC
    ) AS salary_percentile
FROM instructor
ORDER BY department, salary_rank, instructor_name;
```

Execution Plan Output

```
"Incremental Sort  (cost=33.80..55.62 rows=440 width=146) (actual time=0.168..0.180
rows=50.00 loops=1)"
"  Sort Key: dept_name, (rank() OVER w1), name"
"  Presorted Key: dept_name"
"  Full-sort Groups: 2  Sort Method: quicksort  Average Memory: 27kB  Peak Memory: 27kB"
"  Buffers: shared hit=1"
"  ->  WindowAgg  (cost=33.74..43.62 rows=440 width=146) (actual time=0.118..0.144
rows=50.00 loops=1)"
"        Window: w1 AS (PARTITION BY dept_name ORDER BY salary ROWS UNBOUNDED
PRECEDING)"
"        Storage: Memory  Maximum Storage: 17kB"
"        Buffers: shared hit=1"
"        ->  Sort  (cost=33.72..34.82 rows=440 width=130) (actual time=0.100..0.102 rows=50.00
loops=1)"
"              Sort Key: dept_name, salary DESC"
"              Sort Method: quicksort  Memory: 27kB"
"              Buffers: shared hit=1"
"              ->  Seq Scan on instructor  (cost=0.00..14.40 rows=440 width=130) (actual
time=0.030..0.034 rows=50.00 loops=1)"
"                    Buffers: shared hit=1"
"Planning:"
"  Buffers: shared hit=9"
"Planning Time: 0.129 ms"
"Execution Time: 0.207 ms"
```

STEP 4: IDENTIFY ISSUES

Rank() Issue

Rank() skips rank numbers after ties

Percent_Rank()
Percent_Rank() assigns the top row values of 0 rather than 1

Sorting Windows Functions

STEP 5: OPTIMIZE

```
CREATE INDEX idx_instructor_dept_salary
ON instructor (dept_name, salary DESC);

CREATE INDEX idx_instructor_dept_salary_name
ON instructor (dept_name, salary DESC, name);
```

Optimized Query

```
-- Original Query
SELECT
   name AS instructor_name,
   dept_name AS department,
   salary,
   RANK() OVER (
      PARTITION BY dept_name
      ORDER BY salary DESC
   ) AS salary_rank,
   PERCENT_RANK() OVER (
      PARTITION BY dept_name
      ORDER BY salary DESC
   ) AS salary_percentile
FROM instructor
ORDER BY department, salary_rank, instructor_name;

-- New Query
SELECT
   name AS instructor_name,
   dept_name AS department,
   salary,
   DENSE_RANK() OVER (
      PARTITION BY dept_name
```

```
        ORDER BY salary DESC
    ) AS salary_rank,
    PERCENT_RANK() OVER (
        PARTITION BY dept_name
        ORDER BY salary DESC
    ) AS salary_percentile
FROM instructor
ORDER BY department, salary_rank, instructor_name;
```

New Execution Plan

```
"Incremental Sort  (cost=2.94..6.29 rows=50 width=146) (actual time=0.268..0.295 rows=50.00 loops=1)"
"  Sort Key: dept_name, (dense_rank() OVER w1), name"
"  Presorted Key: dept_name"
"  Full-sort Groups: 2  Sort Method: quicksort  Average Memory: 27kB  Peak Memory: 27kB"
"  Buffers: shared hit=1"
"  ->  WindowAgg  (cost=2.91..4.04 rows=50 width=146) (actual time=0.202..0.244 rows=50.00 loops=1)"
"        Window: w1 AS (PARTITION BY dept_name ORDER BY salary ROWS UNBOUNDED PRECEDING)"
"        Storage: Memory  Maximum Storage: 17kB"
"        Buffers: shared hit=1"
"        ->  Sort  (cost=2.91..3.04 rows=50 width=130) (actual time=0.183..0.185 rows=50.00 loops=1)"
"              Sort Key: dept_name, salary DESC"
"              Sort Method: quicksort  Memory: 27kB"
"              Buffers: shared hit=1"
"              ->  Seq Scan on instructor  (cost=0.00..1.50 rows=50 width=130) (actual time=0.045..0.051 rows=50.00 loops=1)"
"                    Buffers: shared hit=1"
"Planning Time: 0.825 ms"
"Execution Time: 0.370 ms"
```

Performance Comparison

Metric	Before (Original Query)	After (Optimized Query)	Change
Scan on takes	Seq Scan (50 rows)	Seq Scan (50 rows)	+ 50%
Buffer Pages	1	1	No change
Execution Time	0.207 ms	0.370 ms	+ 78.74%

Complex Query:

Prompt sent to ChatGPT (April, 2026):

Write a SQL query to: “Find students who have taken courses offered by at least 3 different departments but have never taken a course taught by their own advisor.

AI-Generated SQL (Unmodified)

```
SELECT
   s.ID,
   s.name,
   s.dept_name
FROM student s
JOIN takes t
   ON s.ID = t.ID
JOIN course c
   ON t.course_id = c.course_id
WHERE s.ID IN (
   SELECT t1.ID
   FROM takes t1
   JOIN course c1
      ON t1.course_id = c1.course_id
   GROUP BY t1.ID
   HAVING COUNT(DISTINCT c1.dept_name) >= 3
)
AND NOT EXISTS (
   SELECT 1
   FROM advisor a
   JOIN teaches te
      ON a.i_ID = te.ID
   WHERE a.s_ID = s.ID
     AND te.course_id = t.course_id
     AND te.sec_id = t.sec_id
     AND te.semester = t.semester
     AND te.year = t.year
)
```

```
GROUP BY s.ID, s.name, s.dept_name
ORDER BY s.ID;
```

PART 2: EXECUTION PLAN ANALYSIS

```
EXPLAIN (ANALYZE, BUFFERS, FORMAT TEXT)
SELECT
   s.ID,
   s.name,
   s.dept_name
FROM student s
JOIN takes t
   ON s.ID = t.ID
JOIN course c
   ON t.course_id = c.course_id
WHERE s.ID IN (
   SELECT t1.ID
   FROM takes t1
   JOIN course c1
      ON t1.course_id = c1.course_id
   GROUP BY t1.ID
   HAVING COUNT(DISTINCT c1.dept_name) >= 3
)
AND NOT EXISTS (
   SELECT 1
   FROM advisor a
   JOIN teaches te
      ON a.i_ID = te.ID
   WHERE a.s_ID = s.ID
     AND te.course_id = t.course_id
     AND te.sec_id = t.sec_id
     AND te.semester = t.semester
     AND te.year = t.year
)
GROUP BY s.ID, s.name, s.dept_name
ORDER BY s.ID;
```

Execution Data Output

```
"Group  (cost=4253.80..4253.81 rows=1 width=20) (actual time=165.906..168.686
rows=2000.00 loops=1)"
"  Group Key: s.id"
"  Buffers: shared hit=59373"
"  ->  Sort  (cost=4253.80..4253.81 rows=1 width=20) (actual time=165.891..166.894
rows=29387.00 loops=1)"
"        Sort Key: s.id"
"        Sort Method: quicksort  Memory: 1960kB"
"        Buffers: shared hit=59373"
"        ->  Nested Loop  (cost=3378.41..4253.79 rows=1 width=20) (actual time=40.766..120.575
rows=29387.00 loops=1)"
"              Buffers: shared hit=59373"
"              ->  Hash Anti Join  (cost=3378.27..4253.63 rows=1 width=24) (actual
time=40.732..63.940 rows=29387.00 loops=1)"
"                    Hash Cond: (((s.id)::text = (a.s_id)::text) AND ((t.course_id)::text =
(te.course_id)::text) AND ((t.sec_id)::text = (te.sec_id)::text) AND ((t.semester)::text =
(te.semester)::text) AND (t.year = te.year))"
"                    Buffers: shared hit=599"
"                    ->  Hash Join  (cost=3184.44..3809.67 rows=10005 width=36) (actual
time=38.262..52.475 rows=30000.00 loops=1)"
"                          Hash Cond: ((t.id)::text = (s.id)::text)"
"                          Buffers: shared hit=587"
"                          ->  Hash Join  (cost=3124.44..3723.36 rows=10005 width=26) (actual
time=37.867..46.224 rows=30000.00 loops=1)"
"                                Hash Cond: ((t.id)::text = (t1.id)::text)"
"                                Buffers: shared hit=572"
"                                ->  Seq Scan on takes t  (cost=0.00..520.00 rows=30000 width=21) (actual
time=0.005..1.267 rows=30000.00 loops=1)"
"                                      Buffers: shared hit=220"
"                                ->  Hash  (cost=3116.10..3116.10 rows=667 width=5) (actual
time=37.847..37.850 rows=2000.00 loops=1)"
"                                      Buckets: 2048 (originally 1024)  Batches: 1 (originally 1)  Memory
Usage: 90kB"
"                                      Buffers: shared hit=352"
"                                      ->  GroupAggregate  (cost=1.71..3116.10 rows=667 width=5) (actual
time=0.141..37.556 rows=2000.00 loops=1)"
"                                            Group Key: t1.id"
"                                            Filter: (count(DISTINCT c1.dept_name) >= 3)"
"                                            Buffers: shared hit=352"
```

```
"                                  ->  Incremental Sort  (cost=1.71..2941.10 rows=30000 width=14)
(actual time=0.135..33.718 rows=30000.00 loops=1)"
"                                        Sort Key: t1.id, c1.dept_name"
"                                        Presorted Key: t1.id"
"                                        Full-sort Groups: 757  Sort Method: quicksort  Average
Memory: 26kB  Peak Memory: 26kB"
"                                        Buffers: shared hit=352"
"                                        ->  Nested Loop  (cost=0.44..1940.07 rows=30000 width=14)
(actual time=0.021..10.373 rows=30000.00 loops=1)"
"                                              Buffers: shared hit=352"
"                                              ->  Index Only Scan using takes_pkey on takes t1
(cost=0.29..1178.29 rows=30000 width=9) (actual time=0.008..2.735 rows=30000.00 loops=1)"
"                                                    Heap Fetches: 0"
"                                                    Index Searches: 1"
"                                                    Buffers: shared hit=182"
"                                              ->  Memoize  (cost=0.15..0.17 rows=1 width=13) (actual
time=0.000..0.000 rows=1.00 loops=30000)"
"                                                    Cache Key: t1.course_id"
"                                                    Cache Mode: logical"
"                                                    Hits: 29915  Misses: 85  Evictions: 0  Overflows: 0
Memory Usage: 10kB"
"                                                    Buffers: shared hit=170"
"                                                    ->  Index Scan using course_pkey on course c1
(cost=0.14..0.16 rows=1 width=13) (actual time=0.002..0.002 rows=1.00 loops=85)"
"                                                          Index Cond: ((course_id)::text =
(t1.course_id)::text)"
"                                                          Index Searches: 85"
"                                                          Buffers: shared hit=170"
"                        ->  Hash  (cost=35.00..35.00 rows=2000 width=20) (actual time=0.386..0.387
rows=2000.00 loops=1)"
"                              Buckets: 2048  Batches: 1  Memory Usage: 123kB"
"                              Buffers: shared hit=15"
"                              ->  Seq Scan on student s  (cost=0.00..35.00 rows=2000 width=20) (actual
time=0.016..0.148 rows=2000.00 loops=1)"
"                                    Buffers: shared hit=15"
"                     ->  Hash  (cost=104.64..104.64 rows=3964 width=21) (actual time=2.443..2.445
rows=3951.00 loops=1)"
"                           Buckets: 4096  Batches: 1  Memory Usage: 244kB"
"                           Buffers: shared hit=12"
```

```
"                    ->  Hash Join  (cost=56.00..104.64 rows=3964 width=21) (actual
time=0.598..1.250 rows=3951.00 loops=1)"
"                          Hash Cond: ((te.id)::text = (a.i_id)::text)"
"                          Buffers: shared hit=12"
"                          ->  Seq Scan on teaches te  (cost=0.00..2.00 rows=100 width=21) (actual
time=0.034..0.041 rows=100.00 loops=1)"
"                                Buffers: shared hit=1"
"                          ->  Hash  (cost=31.00..31.00 rows=2000 width=10) (actual
time=0.550..0.550 rows=2000.00 loops=1)"
"                                Buckets: 2048  Batches: 1  Memory Usage: 102kB"
"                                Buffers: shared hit=11"
"                                ->  Seq Scan on advisor a  (cost=0.00..31.00 rows=2000 width=10)
(actual time=0.019..0.229 rows=2000.00 loops=1)"
"                                      Buffers: shared hit=11"
"          ->  Index Only Scan using course_pkey on course c  (cost=0.14..0.16 rows=1 width=4)
(actual time=0.002..0.002 rows=1.00 loops=29387)"
"                Index Cond: (course_id = (t.course_id)::text)"
"                Heap Fetches: 29387"
"                Index Searches: 29387"
"                Buffers: shared hit=58774"
"Planning:"
"  Buffers: shared hit=36"
"Planning Time: 1.067 ms"
"Execution Time: 169.172 ms"
```

STEP 4: IDENTIFY ISSUES

GROUP BY Issue

For this issue, it is an issue since it seems to be mainly used to remove duplicate rows created by the joins rather than the true performance aggregation.

Join Course Issue

It’s unnecessary due to not containing columns from course being used outside the subquery.

STEP 5: OPTIMIZE

```
CREATE INDEX idx_takes_id_course
ON takes (ID, course_id);
```

```
CREATE INDEX idx_course_course_id_dept
ON course (course_id, dept_name);

CREATE INDEX idx_advisor_s_id_i_id
ON advisor (s_ID, i_ID);
```

Optimizing Queries

```
-- Original query
SELECT
   s.ID,
   s.name,
   s.dept_name
FROM student s
JOIN takes t
   ON s.ID = t.ID
JOIN course c
   ON t.course_id = c.course_id
WHERE s.ID IN (
   SELECT t1.ID
   FROM takes t1
   JOIN course c1
      ON t1.course_id = c1.course_id
   GROUP BY t1.ID
   HAVING COUNT(DISTINCT c1.dept_name) >= 3
)
AND NOT EXISTS (
   SELECT 1
   FROM advisor a
   JOIN teaches te
      ON a.i_ID = te.ID
   WHERE a.s_ID = s.ID
     AND te.course_id = t.course_id
     AND te.sec_id = t.sec_id
     AND te.semester = t.semester
     AND te.year = t.year
)
GROUP BY s.ID, s.name, s.dept_name
ORDER BY s.ID;
```

```
-- New query
SELECT
   s.ID,
   s.name,
   s.dept_name
FROM student s
JOIN takes t
   ON s.ID = t.ID
WHERE s.ID IN (
   SELECT t1.ID
   FROM takes t1
   JOIN course c1
      ON t1.course_id = c1.course_id
   GROUP BY t1.ID
   HAVING COUNT(DISTINCT c1.dept_name) >= 3
)
AND NOT EXISTS (
   SELECT 1
   FROM advisor a
   JOIN teaches te
      ON a.i_ID = te.ID
   WHERE a.s_ID = s.ID
     AND te.course_id = t.course_id
     AND te.sec_id = t.sec_id
     AND te.semester = t.semester
     AND te.year = t.year
)
GROUP BY s.ID, s.name, s.dept_name
ORDER BY s.ID;
```

New Execution Plan

```
"Group  (cost=3989.64..3989.65 rows=1 width=20) (actual time=99.870..102.368 rows=2000.00
loops=1)"
"  Group Key: s.id"
"  Buffers: shared hit=533"
"  ->  Sort  (cost=3989.64..3989.64 rows=1 width=20) (actual time=99.865..100.716
rows=29387.00 loops=1)"
```

```
"       Sort Key: s.id"
"       Sort Method: quicksort  Memory: 1960kB"
"       Buffers: shared hit=533"
"       ->  Hash Anti Join  (cost=3114.27..3989.63 rows=1 width=20) (actual
time=37.607..55.641 rows=29387.00 loops=1)"
"             Hash Cond: (((s.id)::text = (a.s_id)::text) AND ((t.course_id)::text =
(te.course_id)::text) AND ((t.sec_id)::text = (te.sec_id)::text) AND ((t.semester)::text =
(te.semester)::text) AND (t.year = te.year))"
"             Buffers: shared hit=533"
"             ->  Hash Join  (cost=2920.44..3545.67 rows=10005 width=36) (actual
time=35.959..47.491 rows=30000.00 loops=1)"
"                   Hash Cond: ((t.id)::text = (s.id)::text)"
"                   Buffers: shared hit=521"
"                   ->  Hash Join  (cost=2860.44..3459.36 rows=10005 width=26) (actual
time=35.589..42.191 rows=30000.00 loops=1)"
"                         Hash Cond: ((t.id)::text = (t1.id)::text)"
"                         Buffers: shared hit=506"
"                         ->  Seq Scan on takes t  (cost=0.00..520.00 rows=30000 width=21) (actual
time=0.005..1.194 rows=30000.00 loops=1)"
"                               Buffers: shared hit=220"
"                         ->  Hash  (cost=2852.10..2852.10 rows=667 width=5) (actual
time=35.574..35.577 rows=2000.00 loops=1)"
"                               Buckets: 2048 (originally 1024)  Batches: 1 (originally 1)  Memory Usage:
90kB"
"                               Buffers: shared hit=286"
"                               ->  GroupAggregate  (cost=1.57..2852.10 rows=667 width=5) (actual
time=0.145..35.264 rows=2000.00 loops=1)"
"                                     Group Key: t1.id"
"                                     Filter: (count(DISTINCT c1.dept_name) >= 3)"
"                                     Buffers: shared hit=286"
"                                     ->  Incremental Sort  (cost=1.57..2677.10 rows=30000 width=14)
(actual time=0.140..31.640 rows=30000.00 loops=1)"
"                                           Sort Key: t1.id, c1.dept_name"
"                                           Presorted Key: t1.id"
"                                           Full-sort Groups: 757  Sort Method: quicksort  Average Memory:
26kB  Peak Memory: 26kB"
"                                           Buffers: shared hit=286"
"                                           ->  Nested Loop  (cost=0.44..1676.07 rows=30000 width=14)
(actual time=0.022..9.308 rows=30000.00 loops=1)"
"                                                 Buffers: shared hit=286"
```

```
"                                    ->  Index Only Scan using idx_takes_id_course on takes t1
(cost=0.29..914.29 rows=30000 width=9) (actual time=0.008..2.053 rows=30000.00 loops=1)"
"                                          Heap Fetches: 0"
"                                          Index Searches: 1"
"                                          Buffers: shared hit=116"
"                                    ->  Memoize  (cost=0.15..0.17 rows=1 width=13) (actual
time=0.000..0.000 rows=1.00 loops=30000)"
"                                          Cache Key: t1.course_id"
"                                          Cache Mode: logical"
"                                          Hits: 29915  Misses: 85  Evictions: 0  Overflows: 0  Memory
Usage: 10kB"
"                                          Buffers: shared hit=170"
"                                          ->  Index Only Scan using idx_course_course_id_dept on
course c1  (cost=0.14..0.16 rows=1 width=13) (actual time=0.002..0.002 rows=1.00 loops=85)"
"                                                Index Cond: (course_id = (t1.course_id)::text)"
"                                                Heap Fetches: 85"
"                                                Index Searches: 85"
"                                                Buffers: shared hit=170"
"                ->  Hash  (cost=35.00..35.00 rows=2000 width=20) (actual time=0.362..0.363
rows=2000.00 loops=1)"
"                      Buckets: 2048  Batches: 1  Memory Usage: 123kB"
"                      Buffers: shared hit=15"
"                      ->  Seq Scan on student s  (cost=0.00..35.00 rows=2000 width=20) (actual
time=0.014..0.141 rows=2000.00 loops=1)"
"                            Buffers: shared hit=15"
"             ->  Hash  (cost=104.64..104.64 rows=3964 width=21) (actual time=1.631..1.633
rows=3951.00 loops=1)"
"                   Buckets: 4096  Batches: 1  Memory Usage: 244kB"
"                   Buffers: shared hit=12"
"                   ->  Hash Join  (cost=56.00..104.64 rows=3964 width=21) (actual
time=0.394..0.783 rows=3951.00 loops=1)"
"                         Hash Cond: ((te.id)::text = (a.i_id)::text)"
"                         Buffers: shared hit=12"
"                         ->  Seq Scan on teaches te  (cost=0.00..2.00 rows=100 width=21) (actual
time=0.044..0.048 rows=100.00 loops=1)"
"                               Buffers: shared hit=1"
"                         ->  Hash  (cost=31.00..31.00 rows=2000 width=10) (actual time=0.338..0.338
rows=2000.00 loops=1)"
"                               Buckets: 2048  Batches: 1  Memory Usage: 102kB"
"                               Buffers: shared hit=11"
```

"   -> Seq Scan on advisor a (cost=0.00..31.00 rows=2000 width=10) (actual time=0.014..0.146 rows=2000.00 loops=1)"
"   Buffers: shared hit=11"
"Planning:"
" Buffers: shared hit=32"
"Planning Time: 1.665 ms"
"Execution Time: 102.977 ms"

Performance Comparison

Metric	Before (Full Scan)	After (Indexed Filter)	Change
Scan on takes	Seq Scan (30,000 rows)	Seq Scan (30,000 rows)	- 5.76%
Buffer Pages	59,373	533	- 99.10%
Execution Time	169.172 ms	102.977 ms	- 39.13%

Essay

Across all queries, the AI did consistently well for generating an output with the right queries for multi-table joins, correlated subqueries, window functions, and complex queries. However, it never created any indexes until I prompted it to do one for all five. For example: Query 4, ChatGPT's query gave me 50 rows for sequential scans, but returned 50 rows when prompted manually. I wouldn't trust AI to give me the queries without review when getting the queries for the first time. I think that it's important to test the query out, and verify it with other AI models. If it's not verified, it could lead to wrong creations of schema, tables, indexes, etc. I would integrate AI query generation when I'm trying to build a schema. For creating more attributes, I would verify the query with one or two more AI models before executing them. This exercise overall taught me how to use AI SQL optimization when unsure with the AI query outputs.

Full AI Prompts

https://chatgpt.com/c/69cdb12b-7264-8333-b476-7c54d63b0c04
https://chatgpt.com/c/69ce0e47-8844-8333-b40f-04dc1c670d81

AI Disclaimer

I used ChatGPT as a support tool while completing this lab. Specifically, I used it to review some of my SQL queries, check whether my logic and syntax were correct, and provide feedback when a query needed correction. In some cases, ChatGPT suggested corrected versions of the queries, which I reviewed and used to help improve my final answers. The final queries and explanations submitted in this lab reflect my own understanding of the assignment and the corrections I chose to apply.