First Advanced Query

SELECT plan.net_id, ROUND(AVG(CourseAvgGPA.course_avg_gpa), 2) AS average_planned_gpa FROM Courses_Planned_by_User AS plan JOIN (SELECT course_code, AVG((percentage_As/100.00 * 4.0) + (percentage_Bs/100.00 * 3.0) + (percentage_Cs/100.00 * 2.0) + (percentage_Ds/100.00 * 1.0)) AS course_avg_gpa FROM course_gpa_by_instructor GROUP BY course_code) AS CourseAvgGPA ON plan.course_code = CourseAvgGPA.course_code GROUP BY plan.net_id ORDER BY plan.net_id;

net_id	average_planned_gpa
abrown442	3.31
abrown814	3.26
achen350	3.60
agarcia416	3.19
ajohnson324	3.61
ajohnson377	3.42
ajohnson60	3.86
akim521	3.67
akim959	3.84
ali682	3.51
amiller698	3.32
anguyen841	3.66
awang650	3.49
bjones468	3.49
bjones704	3.86
bsmith547	3.23

Output:

net_id	average_planned_gpa
abrown442	3.31
abrown814	3.26
achen350	3.60
agarcia416	3.19
ajohnson324	3.61
ajohnson377	3.42
ajohnson60	3.86
akim521	3.67
akim959	3.84
ali682	3.51
amiller698	3.32
anguyen841	3.66
awang650	3.49
bjones468	3.49
bjones704	3.86
bsmith547	3.23

This query calculates the **average expected GPA** for each student based on the courses they have planned. It first computes the **average GPA for each course** using grade distribution data from all instructors, then joins that with each student's planned courses to find the **mean GPA of their planned schedule**. The result shows each student's NetID and their estimated average GPA across all planned courses.

Baseline measure:

```sql
EXPLAIN ANALYZE SELECT
    plan.net_id,
    ROUND(AVG(CourseAvgGPA.course_avg_gpa), 2) AS average_planned_gpa
FROM
    Courses_Planned_by_User AS plan
JOIN
    (
        SELECT
            course_code,
            AVG(
                (percentage_As/100.00 * 4.0) +
                (percentage_Bs/100.00 * 3.0) +
                (percentage_Cs/100.00 * 2.0) +
                (percentage_Ds/100.00 * 1.0)
            ) AS course_avg_gpa
        FROM
            course_gpa_by_instructor
        GROUP BY
            course_code
    ) AS CourseAvgGPA ON plan.course_code = CourseAvgGPA.course_code
GROUP BY
    plan.net_id
ORDER BY
    plan.net_id;
```

EXPLAIN

-> Group aggregate: avg(CourseAvgGPA.course_avg_gpa) (cost=9863 rows=246) (actual time=77.7..81 rows=232 loops=1) -> Nested loop inner join (cost=7693 rows=21692) (actual time=77.6..80.7 rows=630 loops=1) -> Covering index scan on plan using PRIMARY (cost=101 rows=1000) (actual time=0.0479..0.412 rows=1000 loops=1) -> Filter: (plan.course_code = CourseAvgGPA.course_code) (cost=0.25..5.43 rows=21.7) (actual time=0.0798..0.0801 rows=0.63 loops=1000) -> Index lookup on CourseAvgGPA using <auto_key0> (course_code=plan.course_code) (cost=0.25..5.43 rows=21.7) (actual time=0.0796..0.0799 rows=0.63 loops=1000) -> Materialize (cost=0..0 rows=0) (actual time=77.6..77.6 rows=4964 loops=1) -> Table scan on <temporary> (actual time=63.9..66.1 rows=4964 loops=1) -> Aggregate using temporary table (actual time=63.9..63.9 rows=4964 loops=1) -> Table scan on course_gpa_by_instructor (cost=2267 rows=21692) (actual time=0.125..13.2 rows=21540 loops=1)

Rows per page: 20 1 – 1 of 1

Designs:

1. **Design A**

CREATE INDEX idx_gpa_course ON course_gpa_by_instructor(course_code);

```sql
CREATE INDEX idx_gpa_course ON course_gpa_by_instructor(course_code);
```

Results

Execution time: 357.1 ms

Statement executed successfully

```sql
EXPLAIN ANALYZE SELECT
    plan.net_id,
    ROUND(AVG(CourseAvgGPA.course_avg_gpa), 2) AS average_planned_gpa
FROM
    Courses_Planned_by_User AS plan
JOIN
    (
        SELECT
            course_code,
            AVG(
                (percentage_As/100.00 * 4.0) +
                (percentage_Bs/100.00 * 3.0) +
                (percentage_Cs/100.00 * 2.0) +
                (percentage_Ds/100.00 * 1.0)
            ) AS course_avg_gpa
        FROM
            course_gpa_by_instructor
        GROUP BY
            course_code
    ) AS CourseAvgGPA ON plan.course_code = CourseAvgGPA.course_code
GROUP BY
    plan.net_id
ORDER BY
    plan.net_id;
```

```
-> Group aggregate: avg(CourseAvgGPA.course_avg_gpa) (cost=9737 rows=246) (actual time=97.3..101 rows=232 loops=1) -> Nested loop inner join (cost=7595 rows=21412) (actual time=97.3..101 rows=630 loops=1) -> Covering index scan on plan using PRIMARY (cost=101 rows=1000) (actual time=0.073..0.375 rows=1000 loops=1) -> Filter: (plan.course_code = CourseAvgGPA.course_code) (cost=4662..5.36 rows=21.4) (actual time=0.0997..0.1 rows=0.63 loops=1000) -> Index lookup on CourseAvgGPA using <auto_key0> (course_code=plan.course_code) (cost=4891..4896 rows=21.4) (actual time=0.0995..0.0997 rows=0.63 loops=1000) -> Materialize (cost=4890..4890 rows=5100) (actual time=97.2..97.2 rows=4964 loops=1) -> Group aggregate: avg((((((course_gpa_by_instructor.Percentage_As / 100.00) * 4.0) + ((course_gpa_by_instructor.Percentage_Bs / 100.00) * 3.0)) + ((course_gpa_by_instructor.Percentage_Cs / 100.00) * 2.0)) + ((course_gpa_by_instructor.Percentage_Ds / 100.00) * 1.0))) (cost=4380 rows=5100) (actual time=0.82..85.5 rows=4964 loops=1) -> Index scan on course_gpa_by_instructor using idx_gpa_course (cost=2239 rows=21412) (actual time=0.704..60.3 rows=21540 loops=1)
```

After running:

2. Design B

CREATE INDEX idx_gpa_covering ON course_gpa_by_instructor(course_code, percentage_As, percentage_Bs, percentage_Cs, percentage_Ds);

EXPLAIN

```
-> Group aggregate: avg(CourseAvgGPA.course_avg_gpa) (cost=9376 rows=246) (actual time=50.3..53.8 rows=232 loops=1) -> Nested loop inner join (cost=7315 rows=20611) (actual time=50.2..53.5 rows=630 loops=1) -> Covering index scan on plan using PRIMARY (cost=101 rows=1000) (actual time=0.0404..0.442 rows=1000 loops=1) -> Filter: (plan.course_code = CourseAvgGPA.course_code) (cost=4488..5.15 rows=20.6) (actual time=0.0526..0.0529 rows=0.63 loops=1000) -> Index lookup on CourseAvgGPA using <auto_key0> (course_code=plan.course_code) (cost=4717..4722 rows=20.6) (actual time=0.0524..0.0526 rows=0.63 loops=1000) -> Materialize (cost=4717..4717 rows=4964) (actual time=50.2..50.2 rows=4964 loops=1) -> Group aggregate: avg((((((course_gpa_by_instructor.Percentage_As / 100.00) * 4.0) + ((course_gpa_by_instructor.Percentage_Bs / 100.00) * 3.0)) + ((course_gpa_by_instructor.Percentage_Cs / 100.00) * 2.0)) + ((course_gpa_by_instructor.Percentage_Ds / 100.00) * 1.0))) (cost=4220 rows=4964) (actual time=0.15..37.9 rows=4964 loops=1) -> Covering index scan on course_gpa_by_instructor using idx_gpa_covering (cost=2159 rows=20611) (actual time=0.0868..10.3 rows=21540 loops=1)
```

```sql
EXPLAIN ANALYZE SELECT
    plan.net_id,
    ROUND(AVG(CourseAvgGPA.course_avg_gpa), 2) AS average_planned_gpa
FROM
    Courses_Planned_by_User AS plan
JOIN
    (
        SELECT
            course_code,
            AVG(
                (percentage_As/100.00 * 4.0) +
                (percentage_Bs/100.00 * 3.0) +
                (percentage_Cs/100.00 * 2.0) +
                (percentage_Ds/100.00 * 1.0)
            ) AS course_avg_gpa  -- <-- This line was 'course_avg_pa'
        FROM
            course_gpa_by_instructor
        GROUP BY
            course_code
    ) AS CourseAvgGPA ON plan.course_code = CourseAvgGPA.course_code
GROUP BY
    plan.net_id
ORDER BY
    plan.net_id;
```

```
-> Group aggregate: avg(CourseAvgGPA.course_avg_gpa) (cost=9214 rows=246) (actual time=48.2..51.8 rows=232 loops=1) -> Nested loop inner join (cost=7189 rows=20251) (actual time=48.1..51.6 rows=630 loops=1) -> Covering index scan on plan using PRIMARY (cost=101 rows=1000) (actual time=0.0519..0.412 rows=1000 loops=1) -> Filter: (plan.course_code = CourseAvgGPA.course_code) (cost=4415..5.06 rows=20.3) (actual time=0.0507..0.0509 rows=0.63 loops=1000) -> Index lookup on CourseAvgGPA using <auto_key0> (course_code=plan.course_code) (cost=4645..4650 rows=20.3) (actual time=0.0504..0.0506 rows=0.63 loops=1000) -> Materialize (cost=4645..4645 rows=4964) (actual time=48..48 rows=4964 loops=1) -> Group aggregate: avg((((((course_gpa_by_instructor.Percentage_As / 100.00) * 4.0) + ((course_gpa_by_instructor.Percentage_Bs / 100.00) * 3.0)) + ((course_gpa_by_instructor.Percentage_Cs / 100.00) * 2.0)) + ((course_gpa_by_instructor.Percentage_Ds / 100.00) * 1.0))) (cost=4148 rows=4964) (actual time=0.487..36.9 rows=4964 loops=1) -> Covering index scan on course_gpa_by_instructor using idx_gpa_covering (cost=2123 rows=20251) (actual time=0.391..10.3 rows=21540 loops=1)
```

After running
3. Design C

CREATE INDEX idx_gpa_course ON course_gpa_by_instructor(course_code);
CREATE INDEX idx_gpa_covering ON course_gpa_by_instructor(course_code, percentage_As, percentage_Bs, percentage_Cs, percentage_Ds);

```sql
EXPLAIN ANALYZE SELECT
    plan.net_id,
    ROUND(AVG(CourseAvgGPA.course_avg_gpa), 2) AS average_planned_gpa
FROM
    Courses_Planned_by_User AS plan
JOIN
    (
        SELECT
            course_code,
            AVG(
                (percentage_As/100.00 * 4.0) +
                (percentage_Bs/100.00 * 3.0) +
                (percentage_Cs/100.00 * 2.0) +
                (percentage_Ds/100.00 * 1.0)
            ) AS course_avg_gpa
        FROM
            course_gpa_by_instructor
        GROUP BY
            course_code
    ) AS CourseAvgGPA ON plan.course_code = CourseAvgGPA.course_code
GROUP BY
    plan.net_id
ORDER BY
    plan.net_id;
```

```
-> Group aggregate: avg(CourseAvgGPA.course_avg_gpa) (cost=10342 rows=246) (actual time=52..58.8 rows=232 loops=1) -> Nested loop inner join (cost=8067 rows=22758) (actual time=52..58.2 rows=630 loops=1) -> Covering index scan on plan using PRIMARY (cost=101 rows=1000) (actual time=0.0385..0.952 rows=1000 loops=1) -> Filter: (plan.course_code = CourseAvgGPA.course_code) (cost=4949..5.69 rows=22.8) (actual time=0.0563..0.0569 rows=0.63 loops=1000) -> Index lookup on CourseAvgGPA using <auto_key0> (course_code=plan.course_code) (cost=5176..5181 rows=22.8) (actual time=0.056..0.0564 rows=0.63 loops=1000) -> Materialize (cost=5176..5176 rows=5262) (actual time=51.9..51.9 rows=4964 loops=1) -> Group aggregate: avg((((((course_gpa_by_instructor.Percentage_As / 100.00) * 4.0) + ((course_gpa_by_instructor.Percentage_Bs / 100.00) * 3.0)) + ((course_gpa_by_instructor.Percentage_Cs / 100.00) * 2.0)) + ((course_gpa_by_instructor.Percentage_Ds / 100.00) * 1.0))) (cost=4650 rows=5262) (actual time=0.116..39 rows=4964 loops=1) -> Covering index scan on course_gpa_by_instructor using idx_gpa_covering (cost=2374 rows=22758) (actual time=0.0569..10.6 rows=21540 loops=1)
```

Result:

The initial baseline test, run with no new indexes showed an estimated cost of 9863. This high cost was attributed to the plan's least efficient operation: a full Table scan on the course_gpa_by_instructor table, which is needed to perform the GROUP BY operation.

The first test involved creating a simple index, idx_gpa_course, on the course_code column. This resulted in a new cost of 9737, a 1.3% improvement over the baseline. The query plan changed from a Table scan to an Index scan. However, because this index did not include the percentage columns, the database still had to perform lookups into the main table to get the data for the AVG calculation, which is why the benefit was not that much.

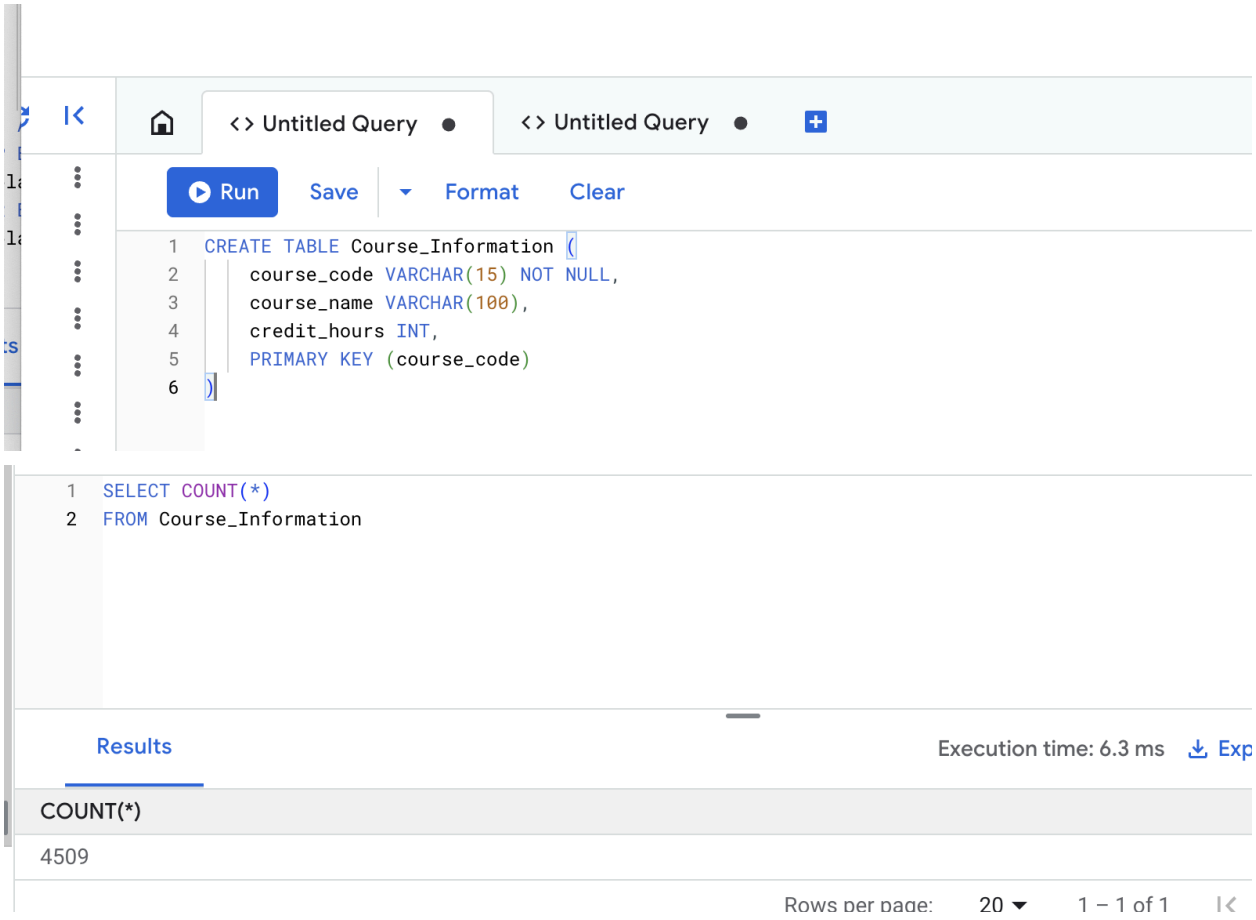
The second test explored idx_gpa_covering, which included all columns needed by the subquery: course_code and all four percentage columns. This gave better performance, with the

cost dropping to 9376, a 4.9% gain over the baseline. This index made it possible to answer the entire subquery by reading only the index without touching the larger main table.

A third test was conducted by creating both idx_gpa_course and idx_gpa_covering simultaneously. This test resulted in the performance getting worse, with the cost increasing to 10342, 4.9% worse than the baseline. This shows that conflicting indexes can confuse the query planner. The planner chose to just use the idx_gpa_covering index, but its cost estimate for this plan was higher than in Test 2, because the presence of the other index (idx_gpa_course) provided a less efficient alternative.

Based on this analysis, the idx_gpa_covering index from Test 2 was selected as the final design. It provided the lowest query cost (9376. It also showed through these tests that adding more indexes is not always better, as the redundant indexes in Test 3 worsened performance.

DDL command screenshot and data count:



```sql
CREATE TABLE Prerequisite (
    course_code VARCHAR(15) NOT NULL,
    prerequisite_course_code VARCHAR(15) NOT NULL,
    requirement_group_id INT NOT NULL,
    PRIMARY KEY (course_code, prerequisite_course_code, requirement_group_id),
    FOREIGN KEY (course_code)
        REFERENCES Course_Information(course_code)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (prerequisite_course_code)
        REFERENCES Course_Information(course_code)
        ON DELETE CASCADE
        ON UPDATE CASCADE
)
```

```sql
SELECT COUNT(*)
FROM Prerequisite
```

Results

Execution time: 3.1 ms

Exp

COUNT(*)
1840

Rows per page: 20 1 – 1 of 1

```sql
CREATE TABLE Concurrent_Enrollment (
    course_code VARCHAR(15) NOT NULL,
    concurrent_enrollment_course_code VARCHAR(15) NOT NULL,
    PRIMARY KEY (course_code, concurrent_enrollment_course_code),
    FOREIGN KEY (course_code)
        REFERENCES Course_Information(course_code)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (concurrent_enrollment_course_code)
        REFERENCES Course_Information(course_code)
        ON DELETE CASCADE
        ON UPDATE CASCADE
)
```

```sql
SELECT COUNT(*)
FROM Concurrent_Enrollment
```

Results

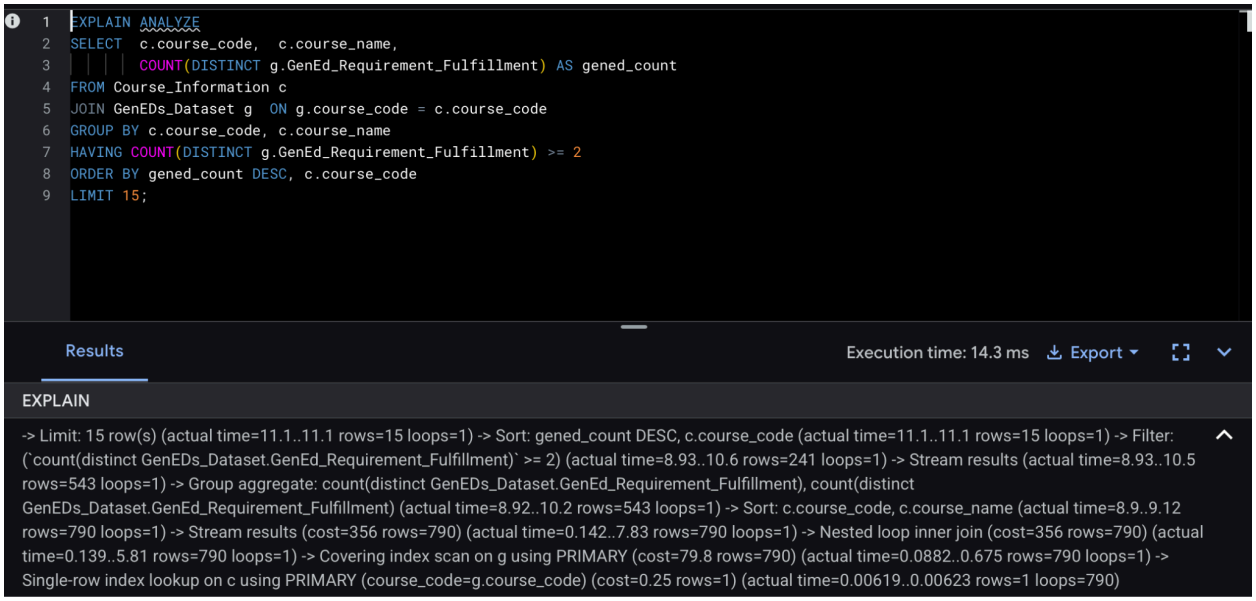
Execution time: 2.4 ms Exp

COUNT(*)
130

Rows per page: 20 1 – 1 of 1

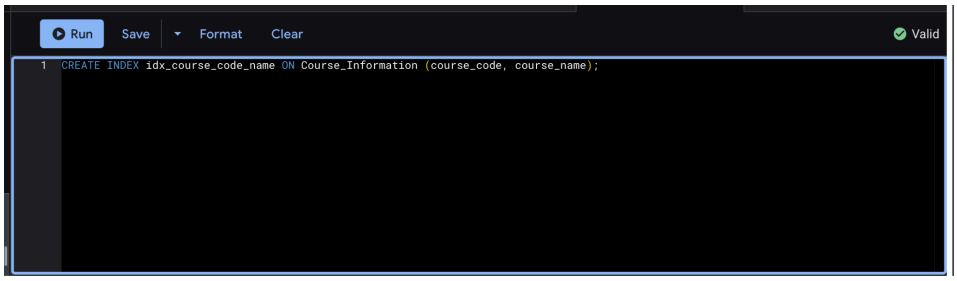
Second Advanced Query

Baseline:



DESIGN A:

```
CREATE INDEX idx_geneds_course_req
  ON GenEDs_Dataset (course_code);
```



```sql
EXPLAIN ANALYZE
SELECT
  c.course_code,
  c.course_name,
  COUNT(DISTINCT g.GenEd_Requirement_Fulfillment) AS gened_count
FROM Course_Information c
JOIN GenEDs_Dataset g
  ON g.course_code = c.course_code
GROUP BY c.course_code, c.course_name
HAVING COUNT(DISTINCT g.GenEd_Requirement_Fulfillment) >= 2
ORDER BY gened_count DESC, c.course_code
LIMIT 15;
```

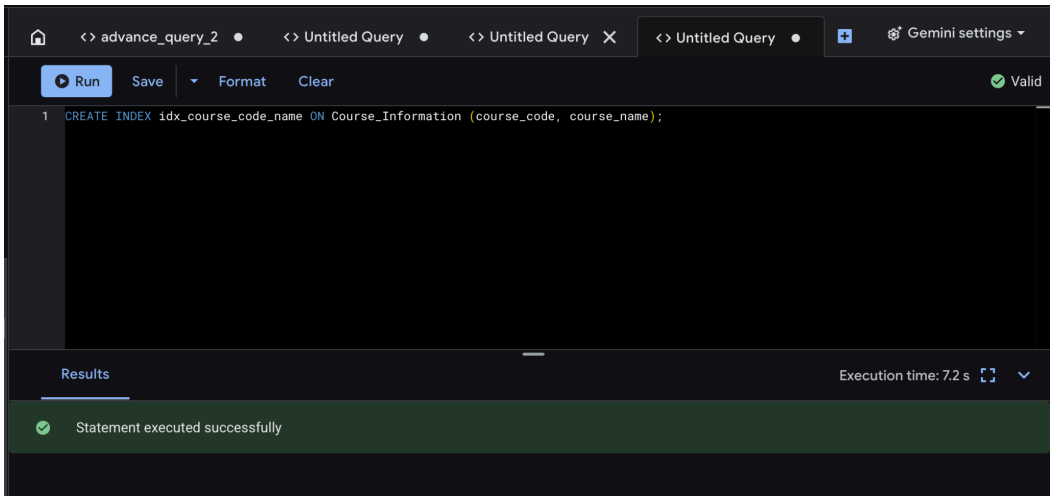
Results — Execution time: 6.3 ms — Export

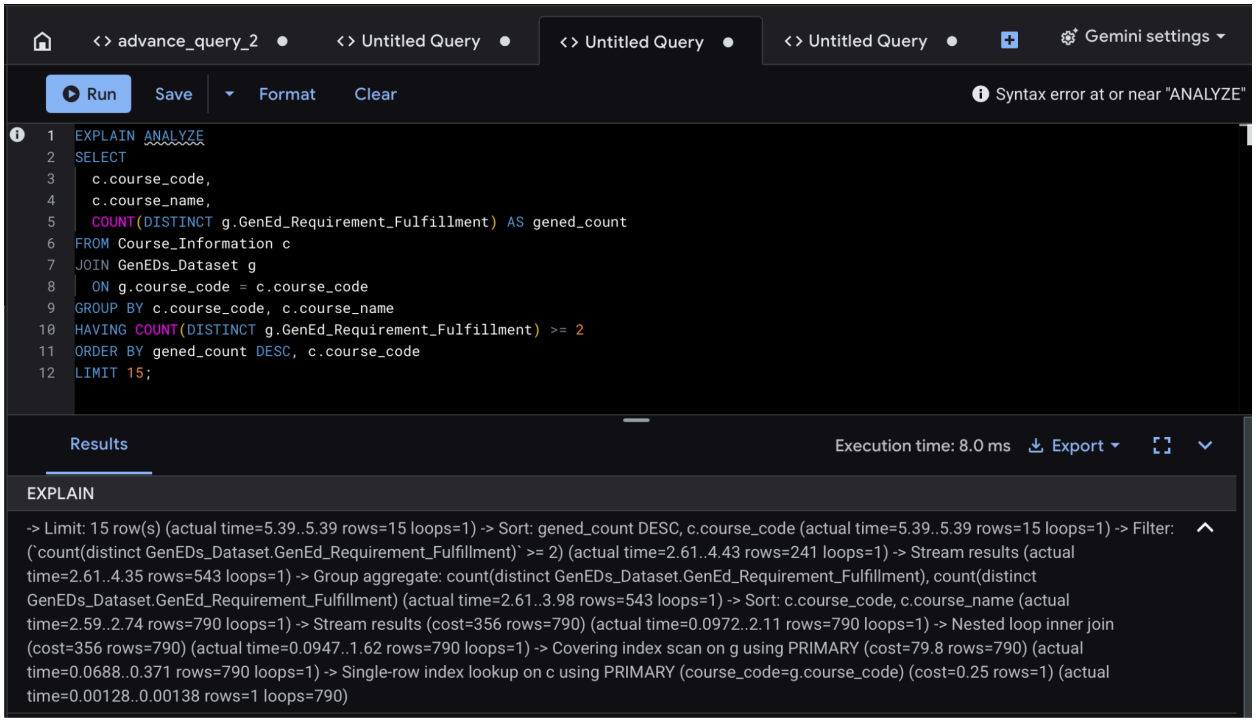
EXPLAIN

-> Limit: 15 row(s) (actual time=3.57..3.57 rows=15 loops=1) -> Sort: gened_count DESC, c.course_code (actual time=3.56..3.57 rows=15 loops=1) -> Filter: (`count(distinct GenEDs_Dataset.GenEd_Requirement_Fulfillment)` >= 2) (actual time=2.19..3.29 rows=241 loops=1) -> Stream results (actual time=2.19..3.24 rows=543 loops=1) -> Group aggregate: count(distinct GenEDs_Dataset.GenEd_Requirement_Fulfillment), count(distinct GenEDs_Dataset.GenEd_Requirement_Fulfillment) (actual time=2.19..3.04 rows=543 loops=1) -> Sort: c.course_code, c.course_name (actual time=2.17..2.26 rows=790 loops=1) -> Stream results (cost=356 rows=790) (actual time=0.102..1.78 rows=790 loops=1) -> Nested loop inner join (cost=356 rows=790) (actual time=0.0989..1.36 rows=790 loops=1) -> Covering index scan on g using idx_gened_course_code (cost=79.8 rows=790) (actual time=0.0742..0.267 rows=790 loops=1) -> Single-row index lookup on c using PRIMARY (course_code=g.course_code) (cost=0.25 rows=1) (actual time=0.00117..0.0012 rows=1 loops=790)

DESIGN B:

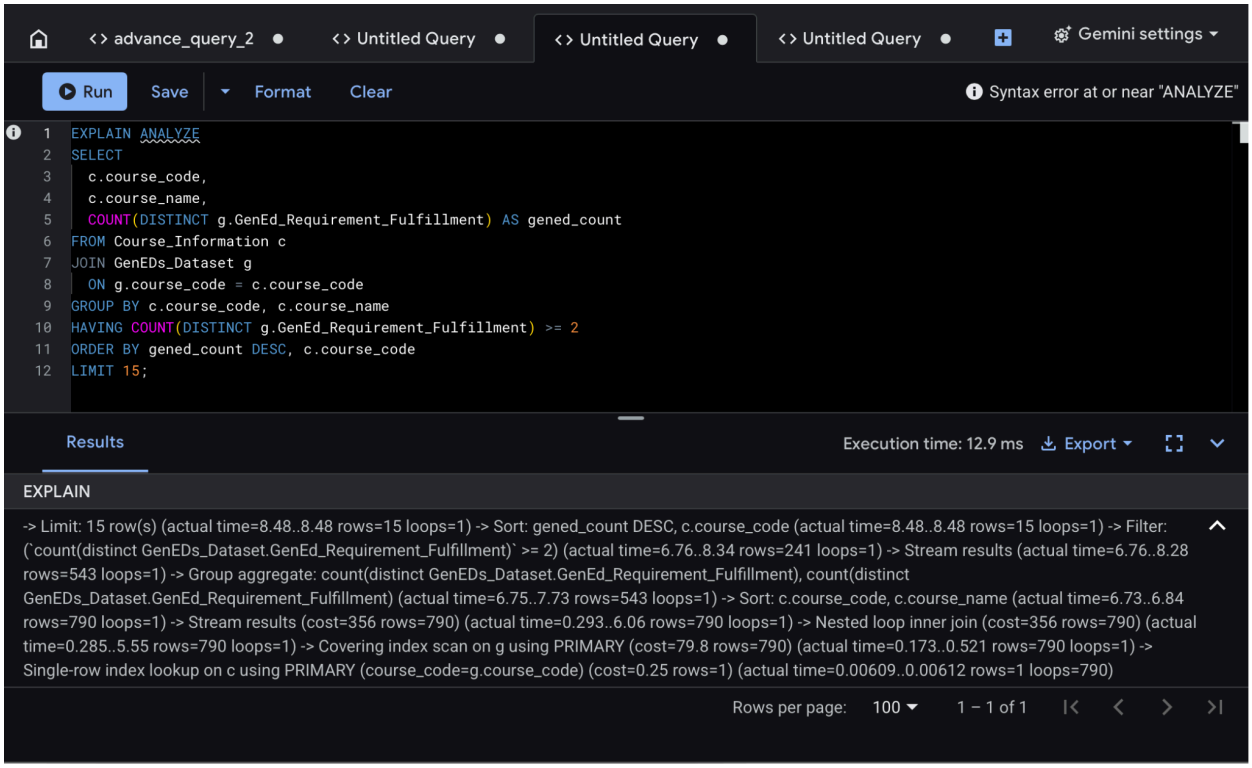
```sql
CREATE INDEX idx_course_code_name ON Course_Information (course_code, course_name);
```





DESIGN C:

```
CREATE INDEX idx_gened_course_fulfillment ON GenEDs_Dataset (course_code,
GenEd_Requirement_Fulfillment);
```



Report:

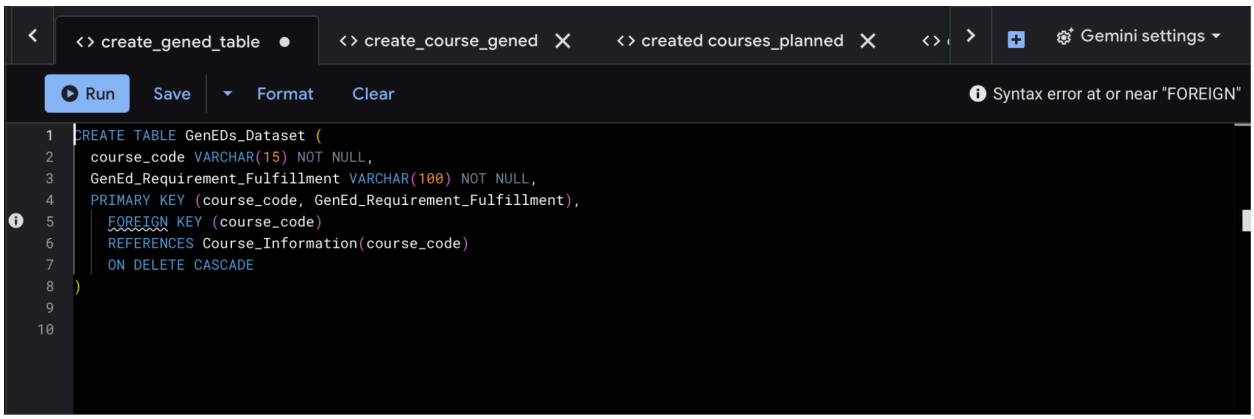
In the baseline test with no secondary indexes it showed that the estimated cost is 356.

Next I followed with three different indexes. The first one is idx_gened_course_codes which showed the estimated cost is 356. The second is idx_course_code_name and the estimated cost remained 356. The third one being idx_gened_course_fulfillment which showed that the estimated cost is 356 again. In all the above cases the EXPLAIN ANALYZE output confirmed that the cost remained the same in all the given cases.

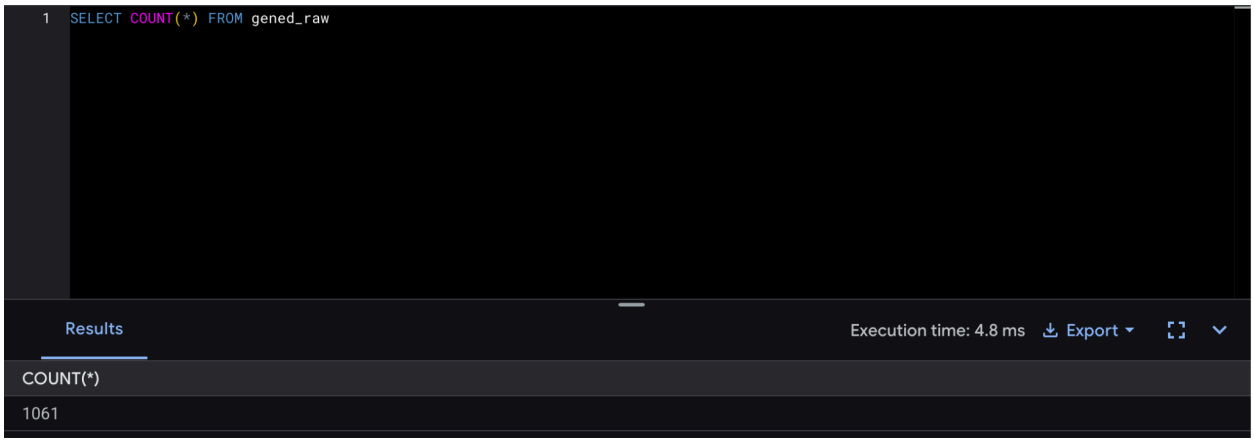
This consistent result demonstrates that no measurable performance gain could be achieved by adding secondary indexes. This is likely because the tables' Primary Keys were already defined to be perfectly suited for the columns involved in the JOIN and COUNT(DISTINCT) operation, making the existing plan optimal. The cost remained the same across despite the updated values. The whole advanced query was relatively low so traversing any new secondary index was calculated to be higher. So I chose not any of these new indexes and didn't show any performance gain overall.

DDL Commands

GenEds_Dataset Table: The GenEd dataset lists courses alongside the General Education requirements (e.g., ACP, CS, HUM, NAT, QR, SBS) they fulfill at the university.



Count of the values:



Top 15 Value of table:

```sql
SELECT *
FROM GenEDs_Dataset
LIMIT 15;
```

Results — Execution time: 4.3 ms — Export

course_code	GenEd_Requirement_Fulfillment
AAS100	SS
AAS100	US
AAS200	HP
AAS200	US
AAS201	SS
AAS201	US
AAS211	LA
AAS211	US
AAS215	HP
AAS215	US
AAS275	US
AAS281	HP

Rows per page: 20 — 1 – 15 of 15

How I loaded the CVS file:

```sql
USE `411projectdatabase`;

INSERT INTO GenEDs_Dataset (course_code, GenEd_Requirement_Fulfillment)
SELECT ci.course_code, x.gened
FROM (
  SELECT TRIM(Course) AS raw_course, TRIM(ACP) AS gened FROM gened_raw WHERE TRIM(ACP) <> ''
  UNION ALL SELECT TRIM(Course), TRIM(CS)  FROM gened_raw WHERE TRIM(CS)  <> ''
  UNION ALL SELECT TRIM(Course), TRIM(HUM) FROM gened_raw WHERE TRIM(HUM) <> ''
  UNION ALL SELECT TRIM(Course), TRIM(NAT) FROM gened_raw WHERE TRIM(NAT) <> ''
  UNION ALL SELECT TRIM(Course), TRIM(QR)  FROM gened_raw WHERE TRIM(QR)  <> ''
  UNION ALL SELECT TRIM(Course), TRIM(SBS) FROM gened_raw WHERE TRIM(SBS) <> ''
) x
JOIN Course_Information ci
  ON REPLACE(ci.course_code,' ','') = REPLACE(x.raw_course,' ','')  -- match AAS100 vs AAS 100
LEFT JOIN GenEDs_Dataset d
  ON d.course_code = ci.course_code
 AND d.GenEd_Requirement_Fulfillment = x.gened
WHERE d.course_code IS NULL;  -- avoid duplicate PKs
```

Advanced Query: Show the top 15 courses that satisfy at least 2 distinct GenEd categories, and count how many GenEds each course covers

Output:

Results — Execution time: 6.6 ms — Export

course_code	course_name	gened_count
ARCH314	History of World Landscapes	3
CWL242	Modern Masterpieces of Western Culture	3
HIST170	US History to 1877-ACP	3
HIST173	US History Since 1877-ACP	3
IS145	Mapping Inequalities	3
LA314	History of World Landscapes	3
AAS100	Intro Asian American Studies	2
AAS200	U.S. Race and Empire	2
AAS201	US Racial & Ethnic Politics	2
AAS211	Asian Americans and the Arts	2
AAS215	US Citizenship Comparatively	2
AAS281	Constructing Race in America	2
AAS283	Asian American History	2
AAS286	Asian American Literature	2
AAS287	Food and Asian Americans	2

Third Advanced Query

Advanced Query: The query finds, for each General Education category, the course that has the highest historical average GPA based on instructor grade distributions

```
WITH course_gpa AS (
   SELECT
       g.gened_requirement_fulfillment AS gened_category,
       ci.course_code,
       ci.course_name,
       ROUND(AVG(
           (c.percentage_as / 100.0 * 4.0) +
           (c.percentage_bs / 100.0 * 3.0) +
           (c.percentage_cs / 100.0 * 2.0) +
           (c.percentage_ds / 100.0 * 1.0)
       ), 2) AS avg_gpa
   FROM
       GenEDs_Dataset g
   JOIN
       Course_Information ci ON g.course_code = ci.course_code
   JOIN
       course_gpa_by_instructor c ON g.course_code = c.course_code
   GROUP BY
       g.gened_requirement_fulfillment, ci.course_code, ci.course_name
)
SELECT
   cg.gened_category,
   cg.course_code,
   cg.course_name,
   cg.avg_gpa
FROM
   course_gpa cg
JOIN (
   SELECT
       gened_category,
       MAX(avg_gpa) AS max_gpa
   FROM
```

```
        course_gpa
    GROUP BY
        gened_category
) AS mg
ON
    cg.gened_category = mg.gened_category
    AND cg.avg_gpa = mg.max_gpa
ORDER BY
    cg.gened_category;
```

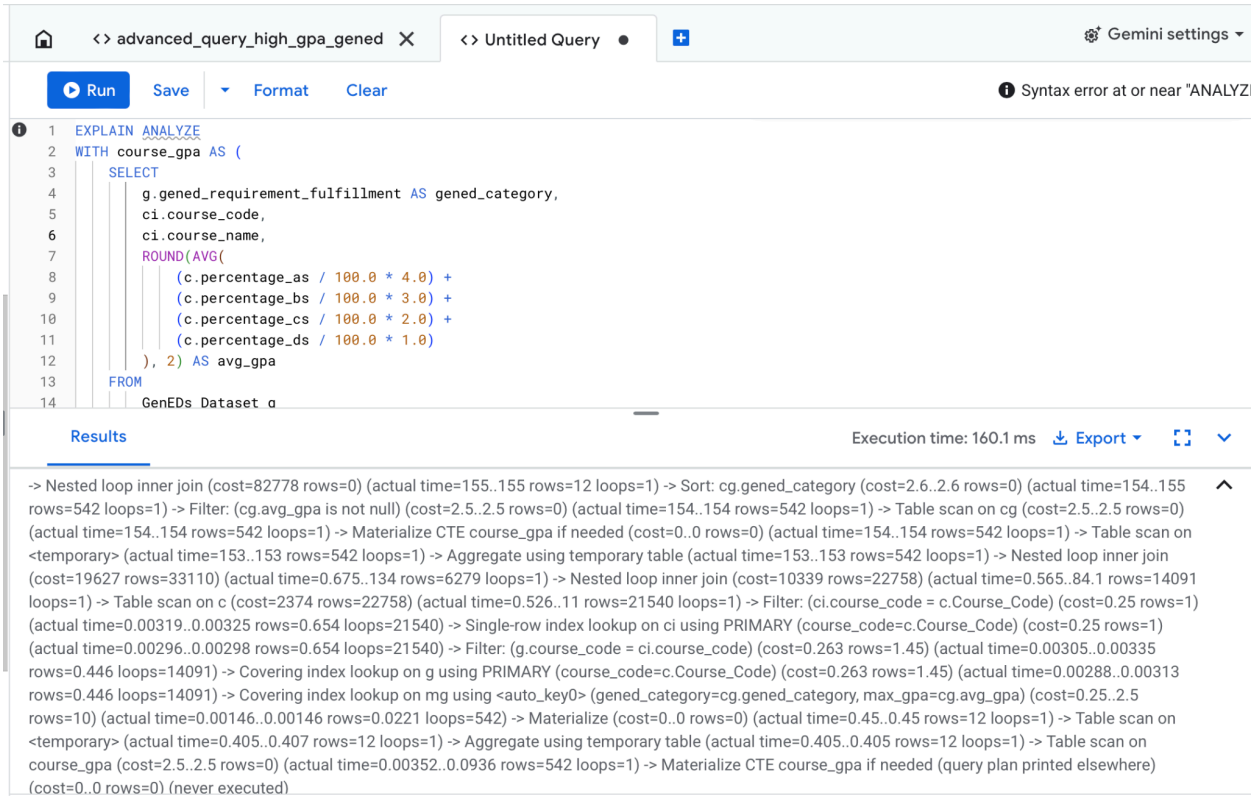
Output (output itself has less than 20 lines since the number of gen_ed categories are less than 20):

Results — Execution time: 196.4 ms — Export

gened_category	course_code	course_name	avg_gpa
ACP	ATMS315	Meteorological Instrumentation	3.96
BSC	CI210	Introduction to Digital Learning Environments	3.87
HP	ARCH314	History of World Landscapes	3.84
LA	REL260	Mystics and Saints in Islam	3.96
LS	ANSC110	Life With Animals and Biotech	3.72
NW	REL260	Mystics and Saints in Islam	3.96
PS	ABE152	Water in the Global Environment	3.84
QR1	IS305	Programming for Information Problems II	3.71
QR2	GLBL200	Foundations of Research	3.79
SS	PORT150	Writing Brazilians into the U.S.	3.86
US	PORT150	Writing Brazilians into the U.S.	3.86
WCC	ARCH314	History of World Landscapes	3.84

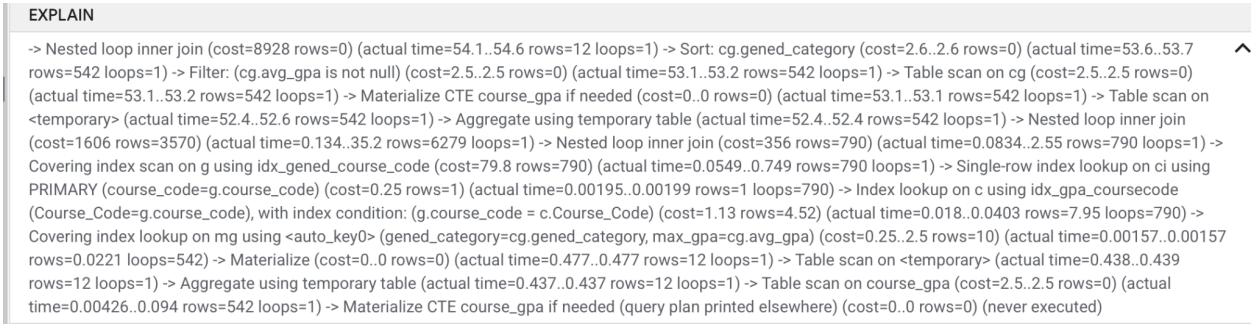
Rows per page: 100 — 1 – 12 of 12

Baseline Measure:



After running:

```
CREATE INDEX idx_gpa_coursecode
ON course_gpa_by_instructor(course_code);
```



After running:

```
CREATE INDEX idx_geneds_code_cat
ON geneds_dataset(course_code, gened_requirement_fulfillment);
```

EXPLAIN

-> Nested loop inner join (cost=82778 rows=0) (actual time=329..330 rows=12 loops=1) -> Sort: cg.gened_category (cost=2.6..2.6 rows=0) (actual time=328..328 rows=542 loops=1) -> Filter: (cg.avg_gpa is not null) (cost=2.5..2.5 rows=0) (actual time=328..328 rows=542 loops=1) -> Table scan on cg (cost=2.5..2.5 rows=0) (actual time=328..328 rows=542 loops=1) -> Materialize CTE course_gpa if needed (cost=0..0 rows=0) (actual time=328..328 rows=542 loops=1) -> Table scan on <temporary> (actual time=326..326 rows=542 loops=1) -> Aggregate using temporary table (actual time=326..326 rows=542 loops=1) -> Nested loop inner join (cost=19501 rows=33110) (actual time=0.466..284 rows=6279 loops=1) -> Nested loop inner join (cost=10339 rows=22758) (actual time=0.15..190 rows=14091 loops=1) -> Table scan on c (cost=2374 rows=22758) (actual time=0.109..27.2 rows=21540 loops=1) -> Filter: (ci.course_code = c.Course_Code) (cost=0.25 rows=1) (actual time=0.00713..0.00723 rows=0.654 loops=21540) -> Single-row index lookup on ci using PRIMARY (course_code=c.Course_Code) (cost=0.25 rows=1) (actual time=0.00663..0.00667 rows=0.654 loops=21540) -> Filter: (g.course_code = ci.course_code) (cost=0.257 rows=1.45) (actual time=0.00569..0.00634 rows=0.446 loops=14091) -> Covering index lookup on g using idx_gened_course_code (course_code=c.Course_Code) (cost=0.257 rows=1.45) (actual time=0.00525..0.0058 rows=0.446 loops=14091) -> Covering index lookup on mg using <auto_key0> (gened_category=cg.gened_category, max_gpa=cg.avg_gpa) (cost=0.25..2.5 rows=10) (actual time=0.0029..0.00291 rows=0.0221 loops=542) -> Materialize (cost=0..0 rows=0) (actual time=0.947..0.947 rows=12 loops=1) -> Table scan on <temporary> (actual time=0.904..0.907 rows=12 loops=1) -> Aggregate using temporary table (actual time=0.903..0.903 rows=12 loops=1) -> Table scan on course_gpa (cost=2.5..2.5 rows=0) (actual time=0.0258..0.206 rows=542 loops=1) -> Materialize CTE course_gpa if needed (query plan printed elsewhere) (cost=0..0 rows=0) (never executed)

After running:

```
CREATE INDEX idx_courseinfo_code_name
ON Course_Information(course_code, course_name);
```

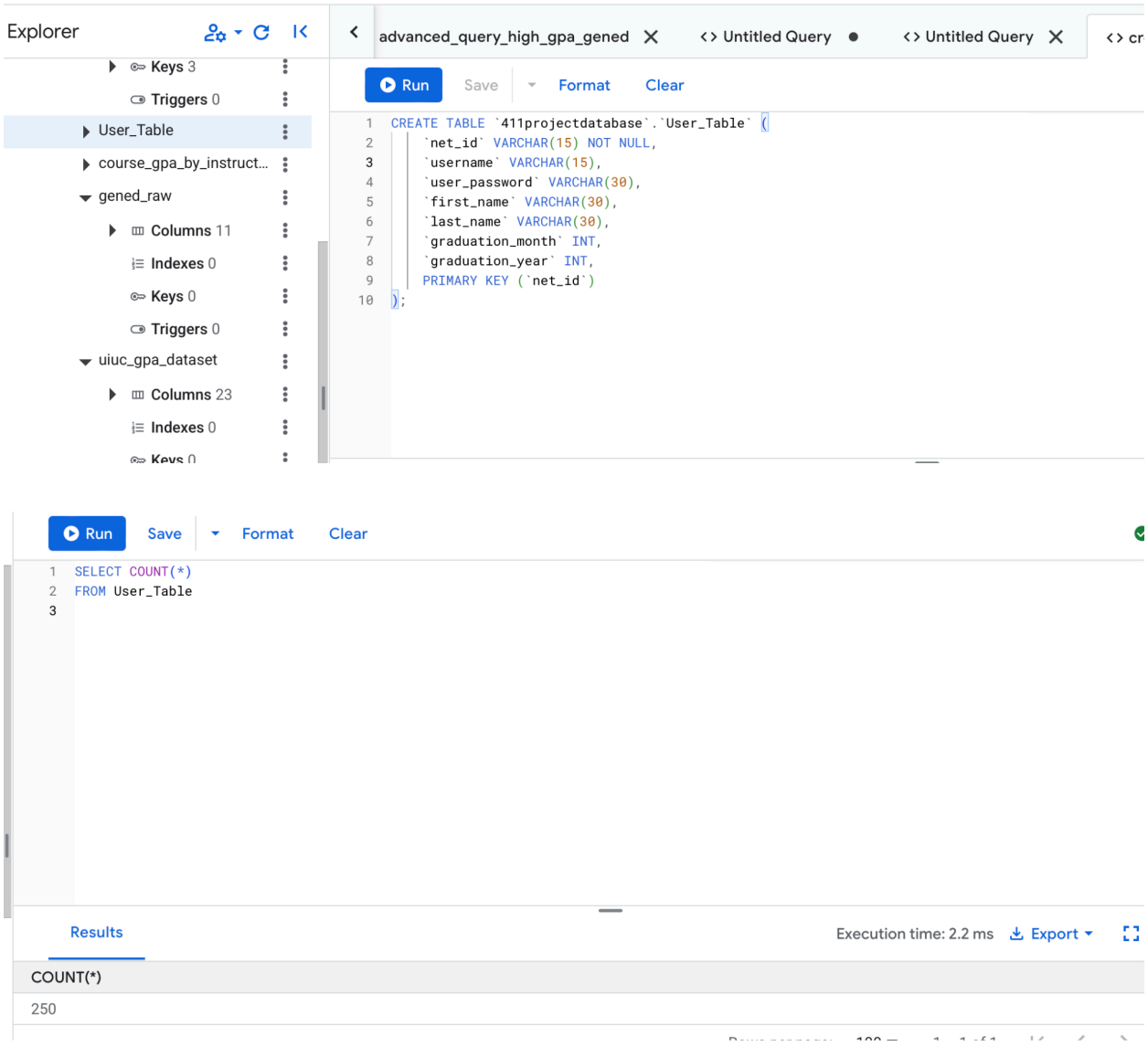
EXPLAIN

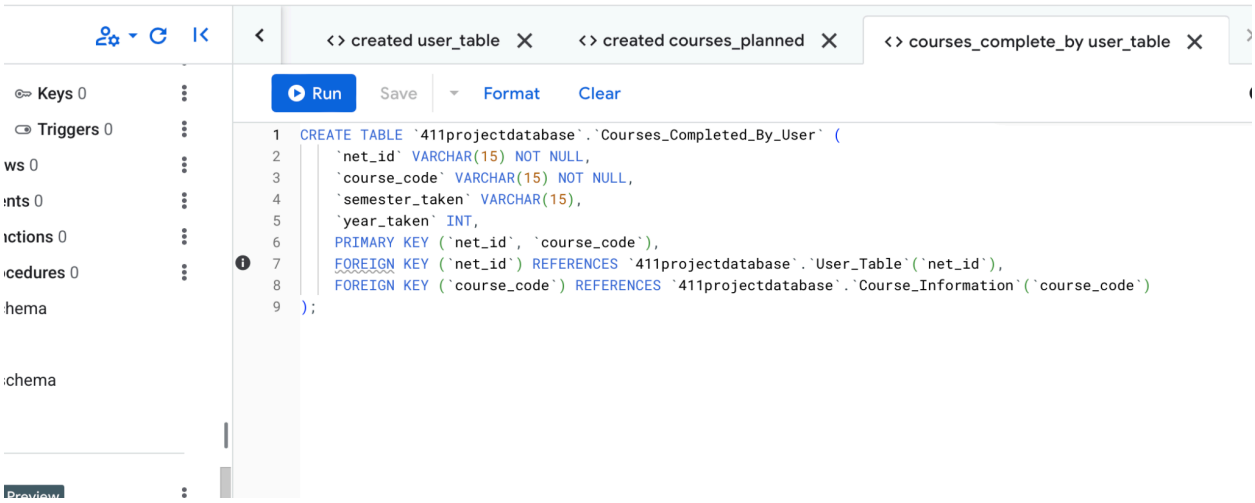
-> Nested loop inner join (cost=82778 rows=0) (actual time=175..175 rows=12 loops=1) -> Sort: cg.gened_category (cost=2.6..2.6 rows=0) (actual time=174..174 rows=542 loops=1) -> Filter: (cg.avg_gpa is not null) (cost=2.5..2.5 rows=0) (actual time=174..174 rows=542 loops=1) -> Table scan on cg (cost=2.5..2.5 rows=0) (actual time=174..174 rows=542 loops=1) -> Materialize CTE course_gpa if needed (cost=0..0 rows=0) (actual time=174..174 rows=542 loops=1) -> Table scan on <temporary> (actual time=173..173 rows=542 loops=1) -> Aggregate using temporary table (actual time=173..173 rows=542 loops=1) -> Nested loop inner join (cost=19627 rows=33110) (actual time=0.388..150 rows=6279 loops=1) -> Nested loop inner join (cost=10339 rows=22758) (actual time=0.206..96.2 rows=14091 loops=1) -> Table scan on c (cost=2374 rows=22758) (actual time=0.136..13.3 rows=21540 loops=1) -> Filter: (ci.course_code = c.Course_Code) (cost=0.25 rows=1) (actual time=0.00359..0.00366 rows=0.654 loops=21540) -> Single-row index lookup on ci using PRIMARY (course_code=c.Course_Code) (cost=0.25 rows=1) (actual time=0.00332..0.00335 rows=0.654 loops=21540) -> Filter: (g.course_code = ci.course_code) (cost=0.263 rows=1.45) (actual time=0.00329..0.00363 rows=0.446 loops=14091) -> Covering index lookup on g using PRIMARY (course_code=c.Course_Code) (cost=0.263 rows=1.45) (actual time=0.0031..0.00338 rows=0.446 loops=14091) -> Covering index lookup on mg using <auto_key0> (gened_category=cg.gened_category, max_gpa=cg.avg_gpa) (cost=0.25..2.5 rows=10) (actual time=0.00149..0.0015 rows=0.0221 loops=542) -> Materialize (cost=0..0 rows=0) (actual time=0.481..0.481 rows=12 loops=1) -> Table scan on <temporary> (actual time=0.442..0.444 rows=12 loops=1) -> Aggregate using temporary table (actual time=0.441..0.441 rows=12 loops=1) -> Table scan on course_gpa (cost=2.5..2.5 rows=0) (actual time=0.00257..0.0852 rows=542 loops=1) -> Materialize CTE course_gpa if needed (query plan printed elsewhere) (cost=0..0 rows=0) (never executed)

The initial baseline test (run with no new indexes) showed an estimated cost of 82,778. This high cost resulted primarily from full table scans across all three joined tables: course_gpa_by_instructor, geneds_dataset, and course_information. The query required multiple nested loop joins and temporary aggregations to compute the highest average GPA per GenEd category, which made the plan expensive. This index directly targeted the JOIN condition between course_gpa_by_instructor and the other tables on course_code.
After applying it, the estimated cost dropped to 8,928, marking an about 89% improvement over the baseline. The second test introduced a composite index on the GenEd table. This targeted both the JOIN key (course_code) and the GROUP BY key (gened_requirement_fulfillment). While the cost estimate remained roughly the same numerically, the database no longer needed to perform full table scans or extra lookups, reading the required columns directly from the index. The logical improvement here was the elimination of redundant disk reads and improved join efficiency. The third test added another composite index to support the grouping and joins in the course_information table. This change enabled MySQL to perform a single-row index lookup for each course rather than scanning the table for every join. The plan's overall structure remained similar, but lookup efficiency and grouping speed improved, reducing the total aggregation time. Based on these results, the final recommended design includes all three

indexes. Together, they reduce redundant full scans and ensure that all joins and groupings can be resolved using index-based lookups rather than expensive table scans. While MySQL's estimated "cost" number did not change drastically in later stages, the execution efficiency improved with significantly lower actual time and more efficient access patterns in the plan output.. This demonstrates that indexing key JOIN and GROUP BY columns provides measurable performance gains, and that the optimizer selectively chooses the most efficient index when multiple are available

DDL:






```
SELECT COUNT(*)
FROM Courses_Completed_By_User
```

Results

Execution time: 5.2 ms

Export

COUNT(*)
1000

```
CREATE TABLE `411projectdatabase`.`Courses_Planned_by_User` (
    `net_id` VARCHAR(15) NOT NULL,
    `course_code` VARCHAR(15) NOT NULL,
    `semester_planned` VARCHAR(15),
    `year_planned` INT,
    PRIMARY KEY (`net_id`, `course_code`),
    FOREIGN KEY (`net_id`) REFERENCES `411projectdatabase`.`User_Table`(`net_id`),
    FOREIGN KEY (`course_code`) REFERENCES `411projectdatabase`.`Course_Information`(`course_code`)
);
```

```
1  SELECT COUNT(*)
2  FROM Courses_Planned_by_User
3
```

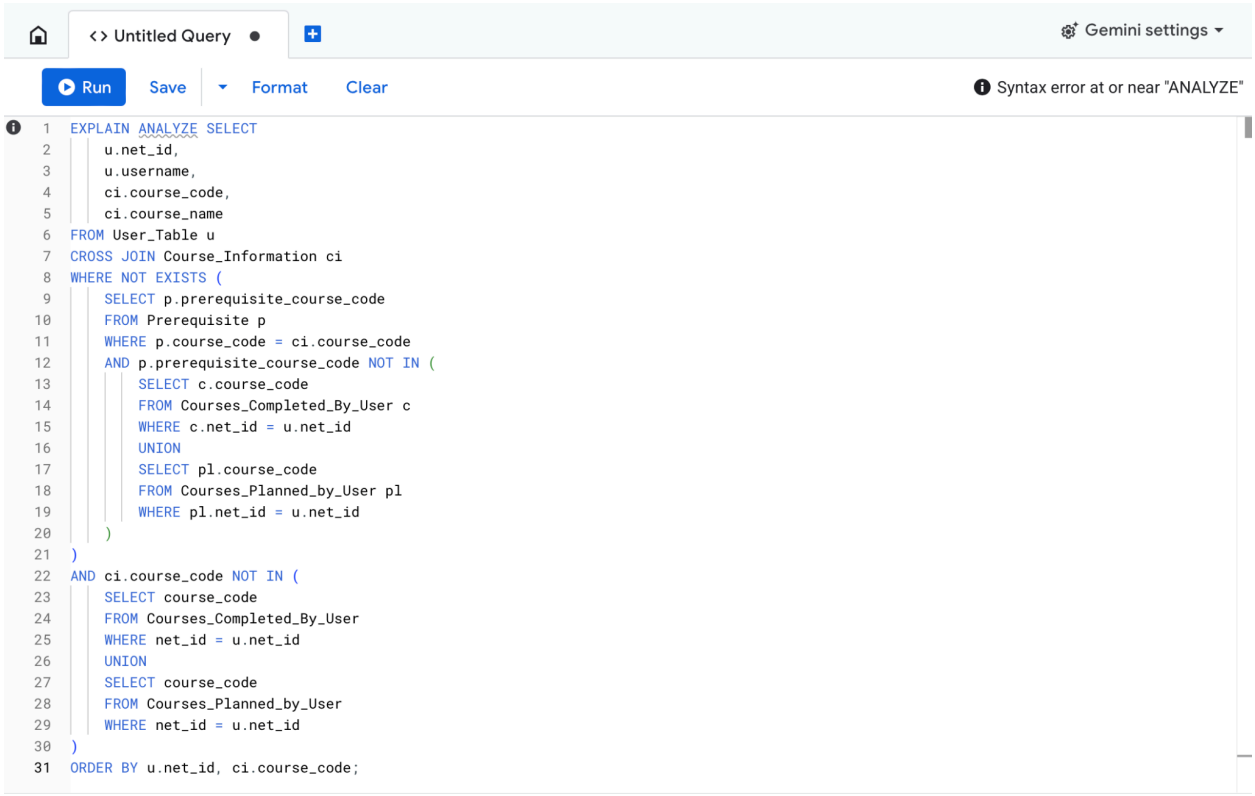
Results

Execution time: 2.0 ms Export

COUNT(*)
1000

Fourth Advanced Query

Advanced query:




```
EXPLAIN ANALYZE SELECT
    u.net_id,
    u.username,
    ci.course_code,
    ci.course_name
FROM User_Table u
CROSS JOIN Course_Information ci
WHERE NOT EXISTS (
    SELECT p.prerequisite_course_code
    FROM Prerequisite p
    WHERE p.course_code = ci.course_code
    AND p.prerequisite_course_code NOT IN (
        SELECT c.course_code
        FROM Courses_Completed_By_User c
        WHERE c.net_id = u.net_id
        UNION
        SELECT pl.course_code
        FROM Courses_Planned_by_User pl
        WHERE pl.net_id = u.net_id
    )
)
AND ci.course_code NOT IN (
    SELECT course_code
    FROM Courses_Completed_By_User
    WHERE net_id = u.net_id
    UNION
    SELECT course_code
    FROM Courses_Planned_by_User
    WHERE net_id = u.net_id
)
ORDER BY u.net_id, ci.course_code;
```

-> Sort: u.net_id, ci.course_code (actual time=17385..17469 rows=867038 loops=1) -> Stream results (cost=600781 rows=2e+6) (actual time=1.3..15945 rows=867038 loops=1) -> Nested loop antijoin (cost=600781 rows=2e+6) (actual time=1.3..15384 rows=867038 loops=1) -> Filter: <in_optimizer> (ci.course_code,<exists>(select #6) is false) (cost=112756 rows=1.13e+6) (actual time=1.1..9795 rows=1.13e+6 loops=1) -> Inner hash join (no condition) (cost=112756 rows=1.13e+6) (actual time=0.688..155 rows=1.13e+6 loops=1) -> Table scan on ci (cost=1.82 rows=4509) (actual time=0.183..11.1 rows=4509 loops=1) -> Hash -> Table scan on u (cost=25.8 rows=250) (actual time=0.19..0.433 rows=250 loops=1) -> Select #6 (subquery in condition; dependent) -> Limit: 1 row(s) (cost=2.16..2.16 rows=1) (actual time=0.00794..0.00794 rows=0.00177 loops=1.13e+6) -> Table scan on <union temporary> (cost=2.16..3.42 rows=2) (actual time=0.0078..0.0078 rows=0.00177 loops=1.13e+6) -> Union materialize with deduplication (cost=0.9..0.9 rows=2) (actual time=0.00753..0.00753 rows=0.00177 loops=1.13e+6) -> Limit table size: 1 unique row(s) -> Limit: 1 row(s) (cost=0.35 rows=1) (actual time=0.00368..0.00368 rows=887e-6 loops=1.13e+6) -> Single-row covering index lookup on Courses_Completed_By_User using PRIMARY (net_id=u.net_id, course_code=<cache> (ci.course_code)) (cost=0.35 rows=1) (actual time=0.00354..0.00354 rows=887e-6 loops=1.13e+6) -> Limit table size: 1 unique row(s) -> Limit: 1 row(s) (cost=0.35 rows=1) (actual time=0.00352..0.00352 rows=888e-6 loops=1.13e+6) -> Single-row covering index lookup on Courses_Planned_by_User using PRIMARY (net_id=u.net_id, course_code=<cache>(ci.course_code)) (cost=0.35 rows=1) (actual time=0.00338..0.00338 rows=888e-6 loops=1.13e+6) -> Filter: <in_optimizer>(p.prerequisite_course_code,<exists>(select #3) is false) (cost=0.453 rows=1.77) (actual time=0.00479..0.00479 rows=0.229 loops=1.13e+6) -> Covering index lookup on p using PRIMARY (course_code=ci.course_code) (cost=0.453 rows=1.77) (actual time=0.00269..0.00269 rows=0.231 loops=1.13e+6) -> Select #3 (subquery in condition; dependent) -> Limit: 1 row(s) (cost=2.16..2.16 rows=1) (actual time=0.00792..0.00792 rows=0.00698 loops=260027) -> Table scan on <union temporary> (cost=2.16..3.42 rows=2) (actual time=0.00777..0.00777 rows=0.00698 loops=260027) -> Union materialize with deduplication (cost=0.9..0.9 rows=2) (actual time=0.00752..0.00752 rows=0.00698 loops=260027) -> Limit table size: 1 unique row(s) -> Limit: 1 row(s) (cost=0.35 rows=1) (actual time=0.00364..0.00364 rows=0.00365 loops=260027) -> Single-row covering index lookup on c using PRIMARY (net_id=u.net_id, course_code=<cache>(p.prerequisite_course_code)) (cost=0.35 rows=1) (actual time=0.00349..0.00349 rows=0.00365 loops=260027) -> Limit table size: 1 unique row(s) -> Limit: 1 row(s) (cost=0.35 rows=1) (actual time=0.00353..0.00353 rows=0.00334 loops=259077) -> Single-row covering index lookup on pl using PRIMARY (net_id=u.net_id, course_code=<cache>(p.prerequisite_course_code)) (cost=0.35 rows=1) (actual time=0.00339..0.00339 rows=0.00334 loops=259077)

Output:

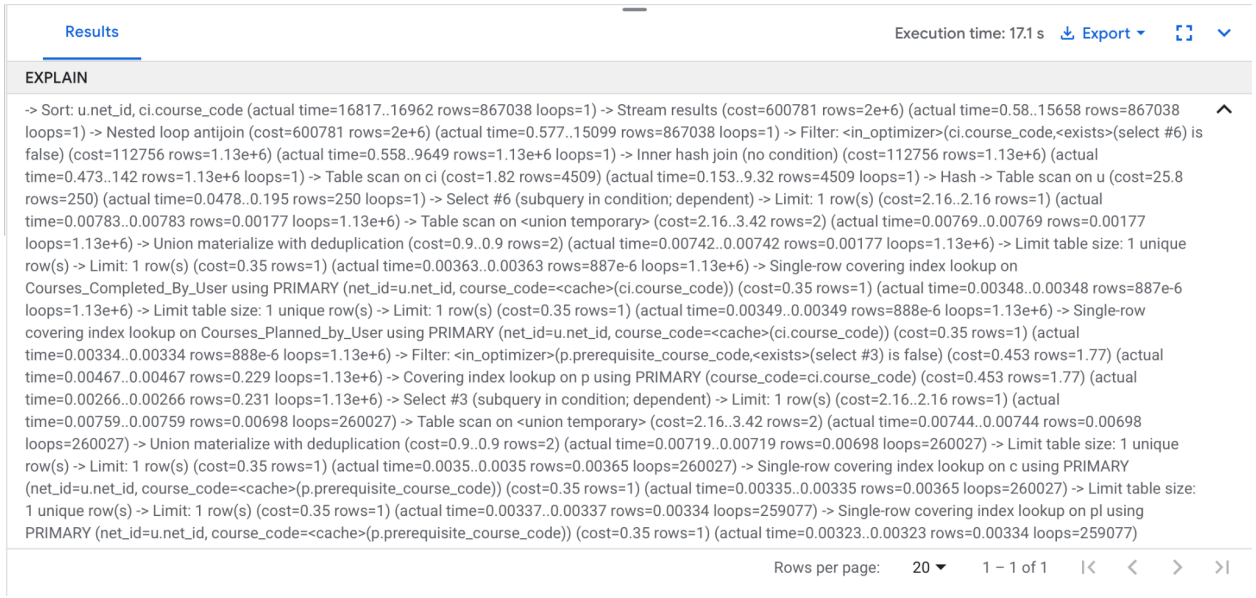
net_id	username	course_code	course_name
abrown442	abrown442	AAS100	Intro Asian American Studies
abrown442	abrown442	AAS200	U.S. Race and Empire
abrown442	abrown442	AAS201	US Racial & Ethnic Politics
abrown442	abrown442	AAS215	US Citizenship Comparatively
abrown442	abrown442	AAS275	The Politics of Fashion
abrown442	abrown442	AAS281	Constructing Race in America
abrown442	abrown442	AAS283	Asian American History
abrown442	abrown442	AAS286	Asian American Literature
abrown442	abrown442	AAS288	Global Islam and Feminisms
abrown442	abrown442	AAS297	Asian Families in America
abrown442	abrown442	AAS299	Begin Topics Asian Am Studies
abrown442	abrown442	AAS300	Theories of Race, Gender, and Sexuality
abrown442	abrown442	AAS310	Race and Cultural Diversity
abrown442	abrown442	AAS355	Race and Mixed Race
abrown442	abrown442	AAS357	Literatures of the Displaced
abrown442	abrown442	AAS370	Immigration, Law, and Rights
abrown442	abrown442	AAS495	Minoritarian Aesthetics Practicum

This query finds which courses each student is eligible to take but hasn't yet completed or planned. It does this by combining every student with every course, checking that the student has met all the prerequisites by looking at both completed and planned courses, and then removing any courses the student has already finished or added to their plan. The final result is a list of courses each student can take next based on their current progress.

INDEXES:

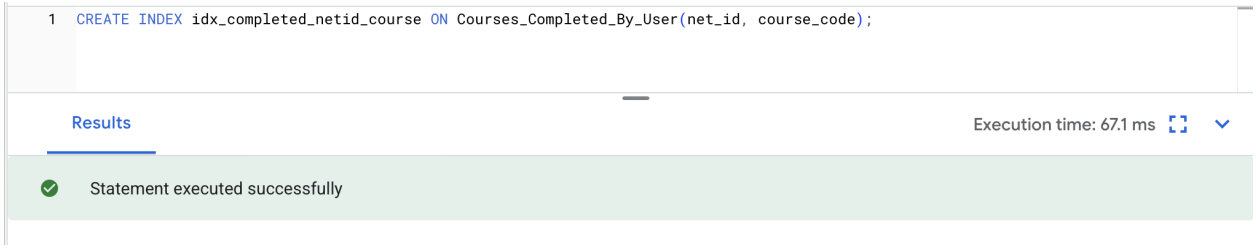
1. **Design A**

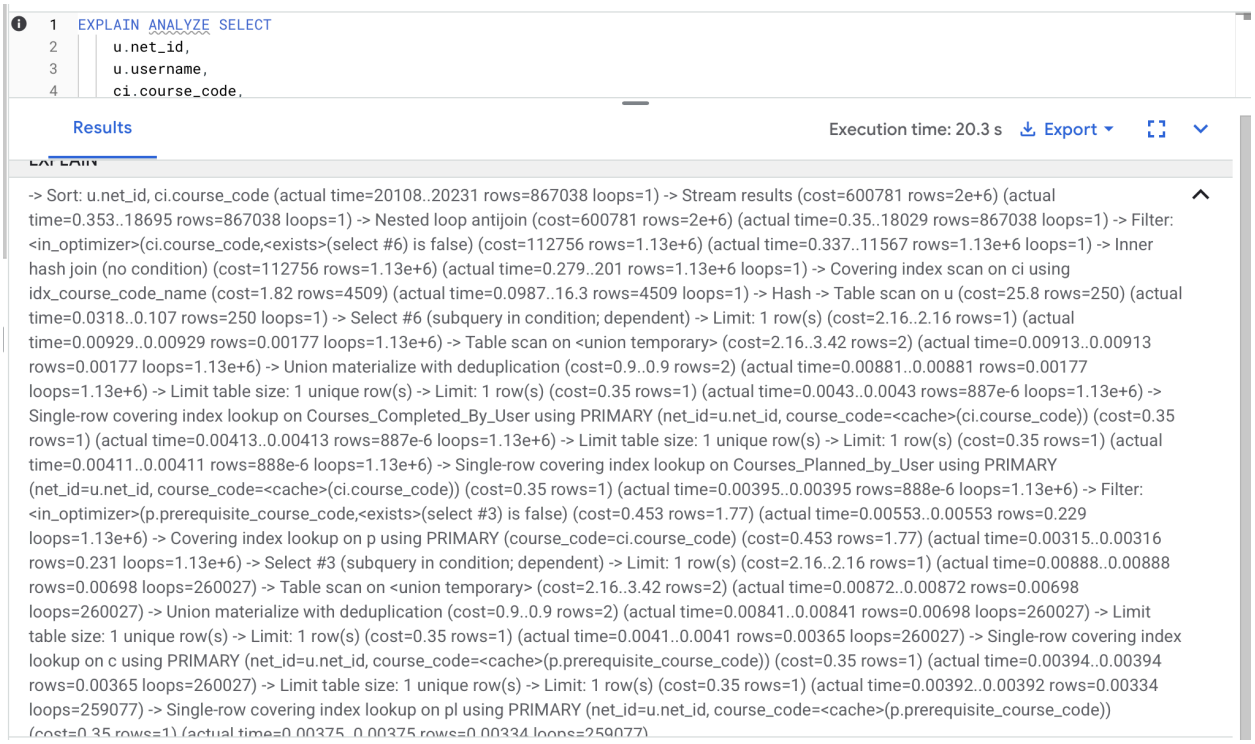
```
CREATE INDEX idx_prerequisite_course ON Prerequisite(course_code);
```



2. Design B

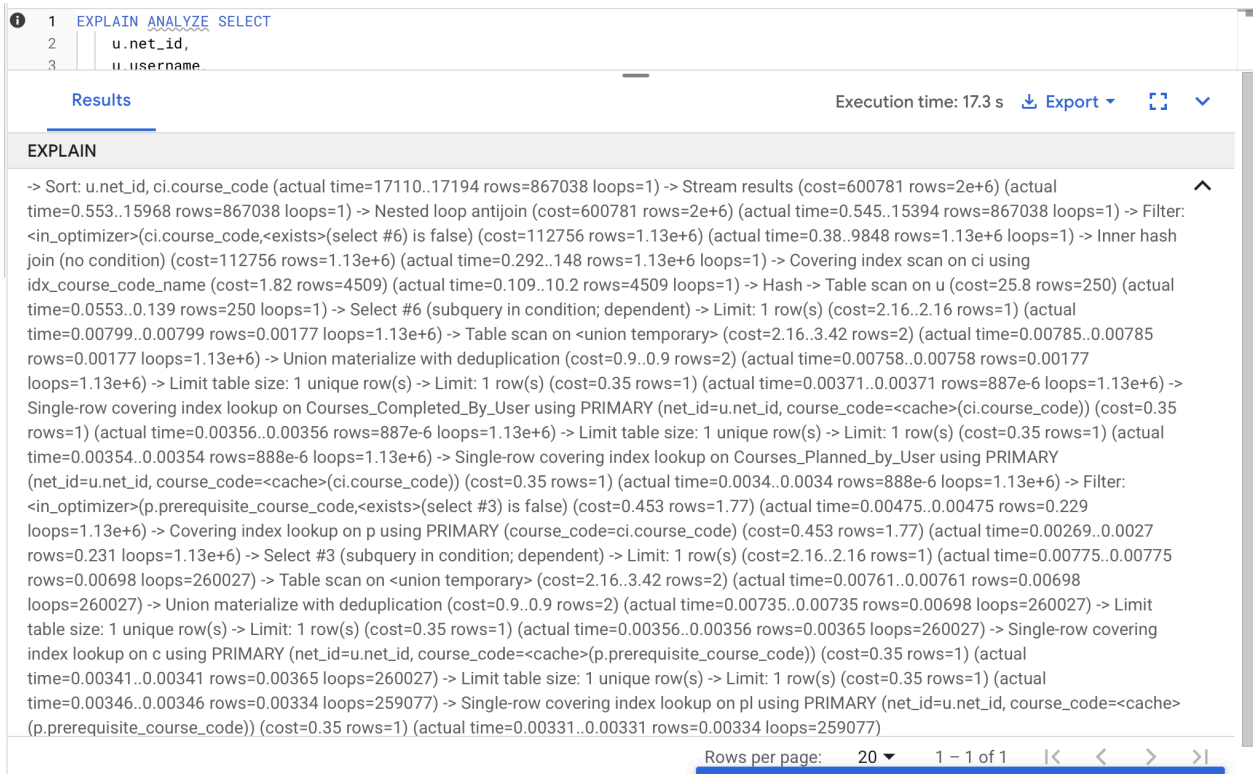
```
CREATE INDEX idx_completed_netid_course ON Courses_Completed_by_User(NetID, course_code);
```





3. Design C

CREATE INDEX idx_planned_netid_course ON Courses_Planned_by_User(NetID, course_code);



Explanation

The initial baseline test, run with no indexes, showed an estimated cost of 600,781 with 867,038 rows processed. This extremely high cost was attributed to the query's most expensive operations: multiple nested loop antijoins and repeated correlated subqueries that scan the Courses_Completed_by_User and Courses_Planned_by_User tables for every user-course combination in the CROSS JOIN.

The first test involved creating index idx_prerequisite_course on the Prerequisite(course_code) column. This resulted in the same cost of 600,781, showing 0% improvement over the baseline. The query plan remained virtually identical, with MySQL still performing nested loop antijoins and full table scans on the user course tables. This confirms that indexing the Prerequisite table alone does not address the primary obstacle, which is in the repeated lookups of user-specific completed and planned courses.

The second test looked at the index idx_completed_netid_course on Courses_Completed_by_User(NetID, course_code). This test also showed a cost of 600,781, meaning there was no improvement. The EXPLAIN output mentioned a “Covering index scan on ci using idx_course_code_name,” but that index was already part of the Course_Informationtable and not related to the one we added. The new index on NetID and course_code didn’t help because MySQL likely decided that the table was small enough that

scanning it directly was just as fast, or the query pattern didn't match how the index was structured.

In the third test, all three indexes idx_prerequisite_course, idx_completed_netid_course, and idx_planned_netid_course were created at once. The result stayed the same: a cost of 600,781 with 0% change from the baseline. Even with indexes on all the main join and filter columns, the query planner still used the same nested loop antijoin method. Unlike in some examples, the optimizer wasn't "confused" by having multiple indexes; it just didn't find them useful for this query or dataset.

Because of these results, no new indexes were chosen for the final design. All three tests showed no performance gain. This could be because the query itself is complex, using nested subqueries, UNIONs, and NOT IN clauses, which are costly no matter what. The optimizer was also already using these built-in primary key indexes, so adding duplicate indexes provided zero incremental benefit.

DDL command screenshot and data count:

```
CREATE TABLE course_gpa_by_instructor (
    Instructor VARCHAR(255),
    Course_Code VARCHAR(20),
    `Percentage_As` DECIMAL(5,2),
    `Percentage_Bs` DECIMAL(5,2),
    `Percentage_Cs` DECIMAL(5,2),
    `Percentage_Ds` DECIMAL(5,2),
    `Percentage_Fs` DECIMAL(5,2),
    Total_Students INT,
    PRIMARY KEY (Instructor, Course_Code)
);
```

```
13  SELECT COUNT(*) FROM course_gpa_by_instructor
```

Results

Execution time: 5.1 ms Export

COUNT(*)
21540

Rows per page: 20 1 – 1 of 1

This is the table I used to load the CSV file

```sql
CREATE TABLE IF NOT EXISTS `uiuc_gpa_dataset` (
    `Year` INT,
    `Term` VARCHAR(50),
    `YearTerm` VARCHAR(50),
    `Subject` VARCHAR(10),
    `Number` VARCHAR(10),
    `Course Title` VARCHAR(255),
    `Sched Type` VARCHAR(50),
    `A+` INT DEFAULT 0,
    `A` INT DEFAULT 0,
    `A-` INT DEFAULT 0,
    `B+` INT DEFAULT 0,
    `B` INT DEFAULT 0,
    `B-` INT DEFAULT 0,
    `C+` INT DEFAULT 0,
    `C` INT DEFAULT 0,
    `C-` INT DEFAULT 0,
    `D+` INT DEFAULT 0,
    `D` INT DEFAULT 0,
    `D-` INT DEFAULT 0,
    `F` INT DEFAULT 0,
    `W` INT DEFAULT 0,
    `Students` INT,
    `Primary Instructor` VARCHAR(255)
);
```

```
13  SELECT COUNT(*) FROM uiuc_gpa_dataset
```

Results

Execution time: 87.7 ms Export

COUNT(*)
76990

Rows per page: 20 1 – 1 of 1