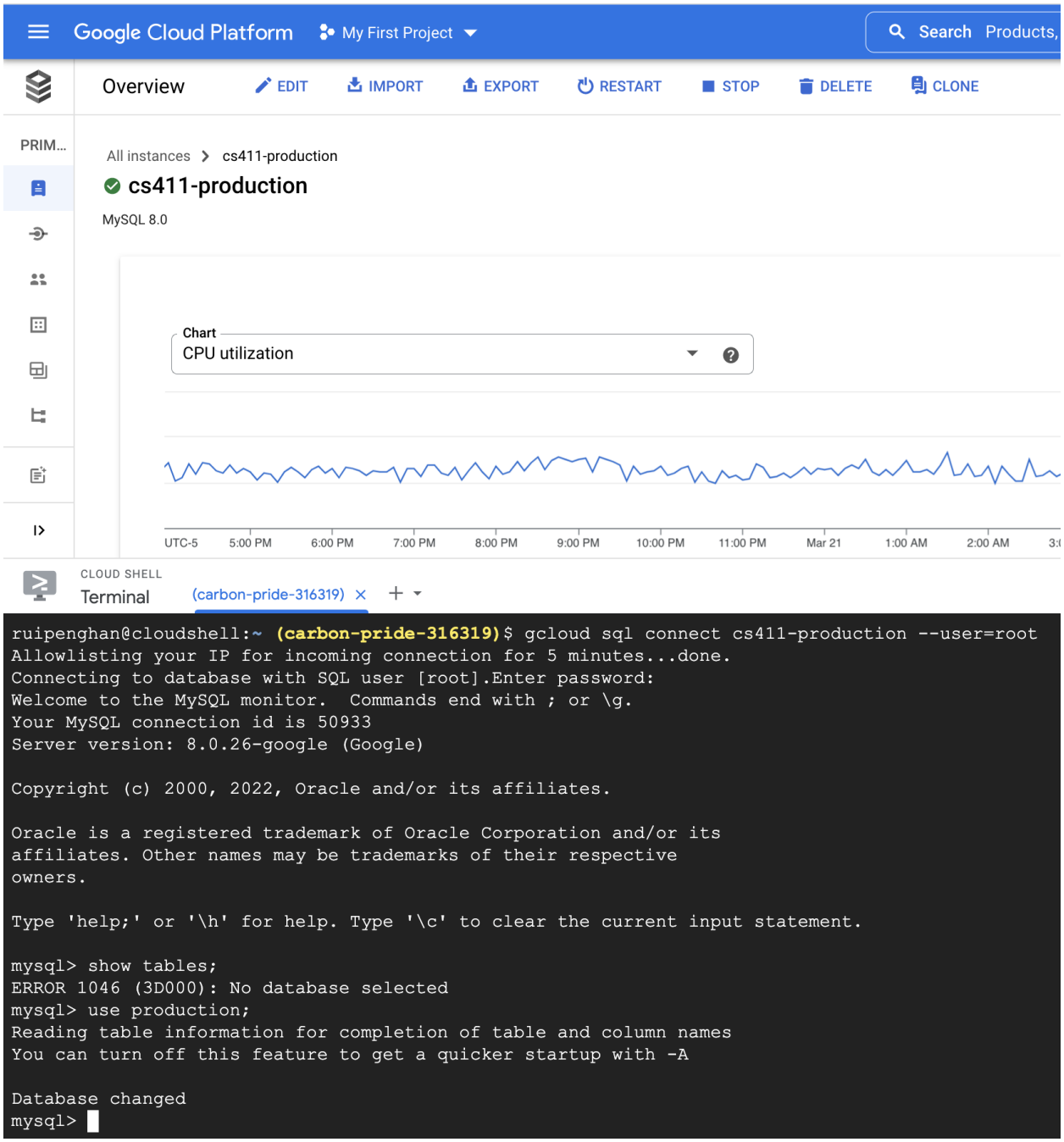
Connection Screenshot:

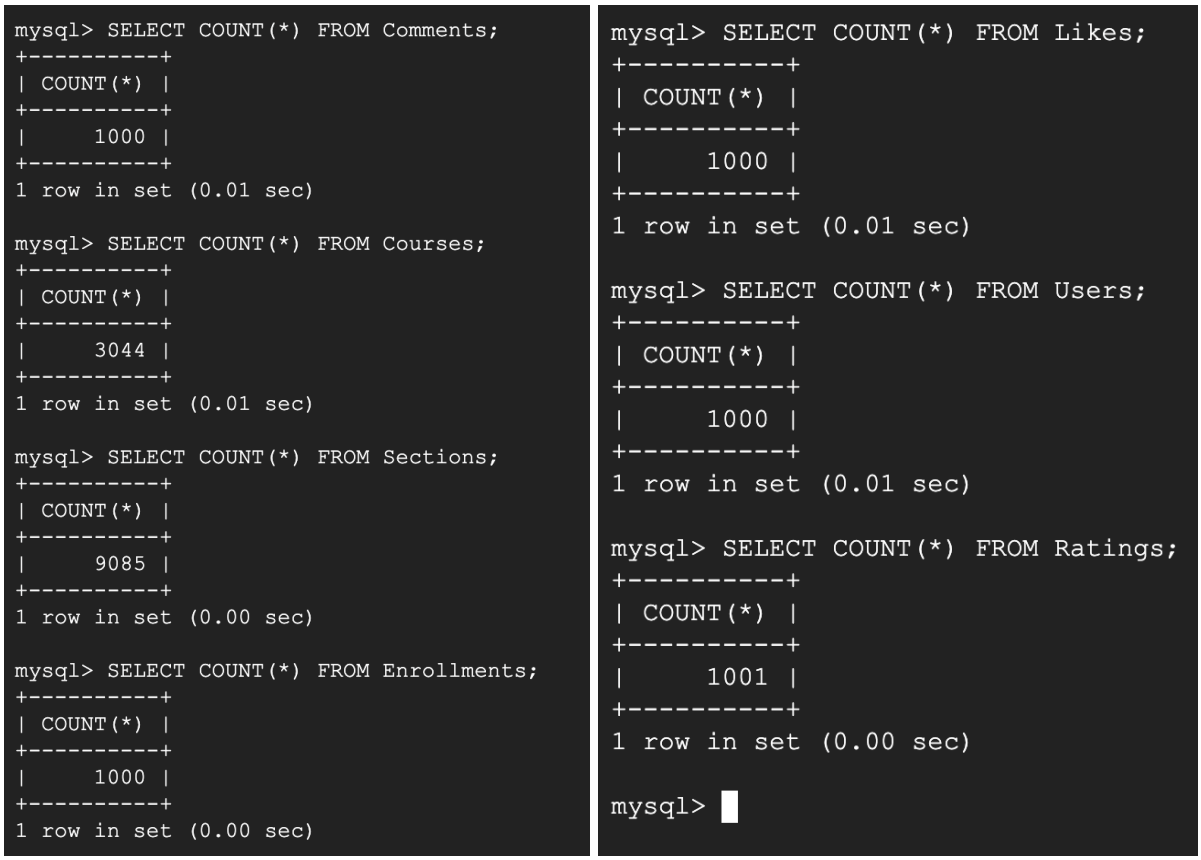


Data Definition Language:

```
CREATE TABLE Users(
   user_id INT AUTO_INCREMENT PRIMARY KEY,
   user_name VARCHAR(40) NOT NULL,
   password VARCHAR(255) NOT NULL,
   email VARCHAR(255) NOT NULL,
   first_name VARCHAR(255) NOT NULL,
   last_name VARCHAR(255) NOT NULL
);

CREATE TABLE Courses(
   id INT AUTO_INCREMENT PRIMARY KEY,
   description VARCHAR(1023),
   subject VARCHAR(40),
   coruse_number INT NOT NULL,
   title VARCHAR(40) NOT NULL
);

CREATE TABLE Sections(
   crn INT,
   course INT,
   students_count INT,
   average_gpa REAL,
   year INT,
   semester VARCHAR(10),
   professor VARCHAR(40),
   PRIMARY KEY(crn, course),
   FOREIGN KEY(course) REFERENCES Courses(id) ON DELETE CASCADE ON
UPDATE CASCADE
);

CREATE TABLE Comments(
   id INT AUTO_INCREMENT PRIMARY KEY,
   comment VARCHAR(2048) NOT NULL,
   comment_time DATETIME,
   likes INT,
   dislikes INT,
   reply INT,
   commenter INT NOT NULL,
```

```
   course INT NOT NULL,
   FOREIGN KEY (commenter) REFERENCES Users(user_id) ON DELETE
CASCADE ON UPDATE CASCADE,
   FOREIGN KEY (course) REFERENCES Courses(id) ON DELETE CASCADE ON
UPDATE CASCADE,
   FOREIGN KEY (reply) REFERENCES Comments(id) ON DELETE CASCADE ON
UPDATE CASCADE
);

CREATE TABLE Ratings(
   id INT AUTO_INCREMENT PRIMARY KEY,
   course INT NOT NULL,
   rater INT NOT NULL,
   overall_quality INT NOT NULL,
   professor INT NOT NULL,
   workload INT NOT NULL,
   rubric INT NOT NULL,
   difficulty INT NOT NULL,
   grade_Rceived CHAR NOT NULL,
   FOREIGN KEY (course) REFERENCES Courses(id) ON DELETE CASCADE ON
UPDATE CASCADE,
   FOREIGN KEY (rater) REFERENCES Users(user_id) ON DELETE CASCADE ON
UPDATE CASCADE
);

CREATE TABLE Likes(
   user INT,
   comment INT,
   isLike BOOLEAN,
   time DATETIME,
   PRIMARY KEY (user, comment),
   FOREIGN KEY (user) REFERENCES Users(user_id) ON DELETE CASCADE ON
UPDATE CASCADE,
   FOREIGN KEY (comment) REFERENCES Comments(id) ON DELETE CASCADE ON
UPDATE CASCADE
);
```

Screenshot of 1000 rows for each table:



```
mysql> SELECT COUNT(*) FROM Comments;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM Courses;
+----------+
| COUNT(*) |
+----------+
|     3044 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM Sections;
+----------+
| COUNT(*) |
+----------+
|     9085 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM Enrollments;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM Likes;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM Users;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM Ratings;
+----------+
| COUNT(*) |
+----------+
|     1001 |
+----------+
1 row in set (0.00 sec)

mysql>
```

The above seven tables each contains at least 1000 rows.

INDEXING:

First query: Union & Join (Finding the comments from CS 241 and CS 233)

```
(select c.subject, course_number, com.comment
from Courses c join Comments com on com.id = c.id
where course_number=241 and subject = 'CS')
union(select c.subject, course_number, com.comment
from Courses c join Comments com on com.id = c.id
where course_number=233 and subject = 'CS');
```

subject	course_number	comment
CS	241	QSRZB
CS	233	ENWGB

-- searching without indexing: Union Time: 2.585 Join Time: 0.417

```
explain analyze((select c.subject, course_number, com.comment
from Courses c join Comments com on com.id = c.id
where course_number=241 and subject = 'CS')
union(select c.subject, course_number, com.comment
from Courses c join Comments com on com.id = c.id
where course_number=233 and subject = 'CS'));
```

```
-> Table scan on <union temporary>  (cost=0.05..3.24 rows=60) (actual time=0.001..0.001 rows=2 loops=1)
  -> Union materialize with deduplication  (cost=672.00..675.18 rows=60) (actual time=2.585..2.586 rows=2 loops=1)
    -> Nested loop inner join  (cost=332.99 rows=30) (actual time=0.417..1.269 rows=1 loops=1)
      -> Filter: ((c.`subject` = 'CS') and (c.course_number = 241))  (cost=322.55 rows=30) (actual time=0.406..1.257 rows=1 loops=1)
```

-- only subject gets index: Union Time: 0.177 Join Time: 0.094

```
create INDEX index_subject on Courses(subject);
explain analyze((select c.subject, course_number, com.comment
from Courses c join Comments com on com.id = c.id
where course_number=241 and subject = 'CS')
union(select c.subject, course_number, com.comment
from Courses c join Comments com on com.id = c.id
where course_number=233 and subject = 'CS'));
drop index index_subject on Courses;
```

```
-> Table scan on <union temporary>  (cost=0.28..2.61 rows=9) (actual time=0.001..0.001 rows=2 loops=1)
  -> Union materialize with deduplication  (cost=28.95..31.28 rows=9) (actual time=0.177..0.177 rows=2 loops=1)
    -> Nested loop inner join  (cost=13.87 rows=5) (actual time=0.094..0.102 rows=1 loops=1)
      -> Filter: (c.course_number = 241)  (cost=12.22 rows=5) (actual time=0.086..0.093 rows=1 loops=1)
```

-- creating an searching irrelevant index Union Time: 2.642 Join Time: 0.436

```
create INDEX index_id on Courses(id);
explain analyze((select c.subject, course_number, com.comment
from Courses c join Comments com on com.id = c.id
where course_number=241 and subject = 'CS')
union(select c.subject, course_number, com.comment
from Courses c join Comments com on com.id = c.id
where course_number=233 and subject = 'CS'));
drop index index_id on Courses;
```

```
-> Table scan on <union temporary>  (cost=0.05..3.24 rows=60) (actual time=0.001..0.002 rows=2 loops=1)
  -> Union materialize with deduplication  (cost=672.00..675.18 rows=60) (actual time=2.642..2.642 rows=2 loops=1)
    -> Nested loop inner join  (cost=332.99 rows=30) (actual time=0.436..1.390 rows=1 loops=1)
      -> Filter: ((c.`subject` = 'CS') and (c.course_number = 241))  (cost=322.55 rows=30) (actual time=0.425..1.380 rows=1 loops=1)
```

-- only course number get index Union Time: 0.095 Join Time: 0.052

```
create index index_course_number on Courses(course_number);
explain analyze((select c.subject, course_number, com.comment
from Courses c join Comments com on com.id = c.id
where course_number=241 and subject = 'CS')
union(select c.subject, course_number, com.comment
from Courses c join Comments com on com.id = c.id
where course_number=233 and subject = 'CS'));
drop index index_course_number on Courses;
```

```
-> Table scan on <union temporary>  (cost=2.09..2.51 rows=1) (actual time=0.002..0.002 rows=2 loops=1)
  -> Union materialize with deduplication  (cost=5.75..6.17 rows=1) (actual time=0.095..0.095 rows=2 loops=1)
    -> Nested loop inner join  (cost=2.66 rows=1) (actual time=0.052..0.055 rows=1 loops=1)
      -> Filter: (c.`subject` = 'CS')  (cost=2.34 rows=1) (actual time=0.043..0.045 rows=1 loops=1)
```

-- both subject and course number get index
Union Time: 0.107 Join Time: 0.057

```
create INDEX index_subject on Courses(subject);
create index index_course_number on Courses(course_number);
explain analyze((select c.subject, course_number, com.comment
from Courses c join Comments com on com.id = c.id
```

```
where course_number=241 and subject = 'CS')
union(select c.subject, course_number, com.comment
from Courses c join Comments com on com.id = c.id
where course_number=233 and subject = 'CS'));
drop index index_subject on Courses;
drop index index_course_number on Courses;
```

```
-> Table scan on <union temporary>  (cost=2.39..2.51 rows=1) (actual time=0.001..0.001 rows=2 loops=1)
  -> Union materialize with deduplication  (cost=5.92..6.04 rows=1) (actual time=0.107..0.107 rows=2 loops=1)
    -> Nested loop inner join  (cost=2.65 rows=1) (actual time=0.057..0.070 rows=1 loops=1)
      -> Filter: ((c.`subject` = 'CS') and (c.course_number = 241))  (cost=2.30 rows=1) (actual time=0.047..0.060 rows=1 loops=1)
```

Second query:
Join Subquery Group-by (Finding the subjects that average overall-quality score is better than CS)

```
select c.subject,avg(r.overall_quality) as
average_quality
from Courses c natural join Ratings r
group by c.subject
having avg(r.overall_quality) >= (select
avg(r1.overall_quality)
from Courses c1 natural join Ratings r1
group by c1.subject
having subject = 'CS')
order by avg(r.overall_quality) desc;
```

subject	average_quality
CHIN	7.0000
BIOP	6.7500
BCS	6.6667
ASRM	6.6154
AFRO	6.0417
ASST	6.0000
CB	6.0000
CHLH	5.8333
CWL	5.5385
CLCV	5.4615
CS	5.4255

-- searching without indexing: Subquery Time: 5.005 Aggregation Time: 2.723

```
explain analyze(select c.subject,avg(r.overall_quality) as
average_quality
from Courses c natural join Ratings r
group by c.subject
having avg(r.overall_quality) >= (select avg(r1.overall_quality)
from Courses c1 natural join Ratings r1
group by c1.subject
having subject = 'CS')
order by avg(r.overall_quality) desc);
```

```
-> Sort: average_quality DESC  (actual time=5.060..5.061 rows=11 loops=1)
  -> Filter: (avg(r.overall_quality) >= (select #2))  (actual time=5.005..5.031 rows=11 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.014 rows=58 loops=1)
      -> Aggregate using temporary table  (actual time=2.723..2.739 rows=58 loops=1)
```

-- set indexing irrelevant column Subquery Time: 5.086 Aggregation Time: 2.728

```
create INDEX index_professor on Ratings(professor);
explain analyze(select c.subject,avg(r.overall_quality) as
average_quality
from Courses c natural join Ratings r
group by c.subject
having avg(r.overall_quality) >= (select avg(r1.overall_quality)
from Courses c1 natural join Ratings r1
group by c1.subject
having subject = 'CS')
order by avg(r.overall_quality) desc);
drop index index_professor on Ratings;
```

```
-> Sort: average_quality DESC  (actual time=5.138..5.139 rows=11 loops=1)
  -> Filter: (avg(r.overall_quality) >= (select #2))  (actual time=5.086..5.111 rows=11 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.013 rows=58 loops=1)
      -> Aggregate using temporary table  (actual time=2.728..2.743 rows=58 loops=1)
```

-- set indexing on overall_quality Subquery Time: 5.692 Aggregation Time: 3.233

```
create INDEX index_overall_quality on Ratings(overall_quality);
explain analyze(select c.subject,avg(r.overall_quality) as
average_quality
from Courses c natural join Ratings r
group by c.subject
having avg(r.overall_quality) >= (select avg(r1.overall_quality)
from Courses c1 natural join Ratings r1
group by c1.subject
having subject = 'CS')
order by avg(r.overall_quality) desc);
drop index index_overall_quality on Ratings;
```

```
-> Sort: average_quality DESC  (actual time=5.748..5.749 rows=11 loops=1)
  -> Filter: (avg(r.overall_quality) >= (select #2))  (actual time=5.692..5.717 rows=11 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..0.013 rows=58 loops=1)
      -> Aggregate using temporary table  (actual time=3.233..3.248 rows=58 loops=1)
```

-- set indexing on subjects Subquery Time: 5.235 Aggregation Time: 2.808

```
create INDEX index_subject on Courses(subject);
explain analyze(select c.subject,avg(r.overall_quality) as
```

```
average_quality
from Courses c natural join Ratings r
group by c.subject
having avg(r.overall_quality) >= (select avg(r1.overall_quality)
from Courses c1 natural join Ratings r1
group by c1.subject
having subject = 'CS')
order by avg(r.overall_quality) desc);
drop index index_subject on Courses;
```

```
-> Sort: average_quality DESC  (actual time=5.287..5.287 rows=11 loops=1)
  -> Filter: (avg(r.overall_quality) >= (select #2))  (actual time=5.235..5.261 rows=11 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.015 rows=58 loops=1)
      -> Aggregate using temporary table  (actual time=2.808..2.826 rows=58 loops=1)
```

-- set indexing on both overall_quality and subjects

Subquery Time: 5.097 Aggregation Time: 2.820

```
create INDEX index_overall_quality on Ratings(overall_quality);
create INDEX index_subject on Courses(subject);
explain analyze(select c.subject,avg(r.overall_quality) as
average_quality
from Courses c natural join Ratings r
group by c.subject
having avg(r.overall_quality) >= (select avg(r1.overall_quality)
from Courses c1 natural join Ratings r1
group by c1.subject
having subject = 'CS')
order by avg(r.overall_quality) desc);
drop index index_subject on Courses;
drop index index_overall_quality on Ratings;
```

```
-> Sort: average_quality DESC  (actual time=5.149..5.150 rows=11 loops=1)
  -> Filter: (avg(r.overall_quality) >= (select #2))  (actual time=5.097..5.123 rows=11 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.013 rows=58 loops=1)
      -> Aggregate using temporary table  (actual time=2.820..2.835 rows=58 loops=1)
```

Index Design Summary:

After we tried these two queries, we decided for simple search we may put indices on related integer columns so that the searching time will be significantly reduced. For complex queries, while we need to use group by and subquery, we are not going to use indexing since they will only increase the execution time.

In particular, for our first query, we decided to add indices on course number so that union and join time became 0.095 and 0.052 respectively, which were more efficient compared to the original query time 2.585 and 0.417. Adding indexing on the subject column will not change the time efficiency too much but increase space complexity significantly, so we choose not to add that. For the second query, the four ways of adding indices we tried did not increase the efficiency of the query significantly. In fact, adding indices reduced the efficiency of the query. A possible reason was that adding more indices make sorting and aggregation more difficult and also take more space. Under such circumstances, we did not plan to add indices.