Database-Design

Contents

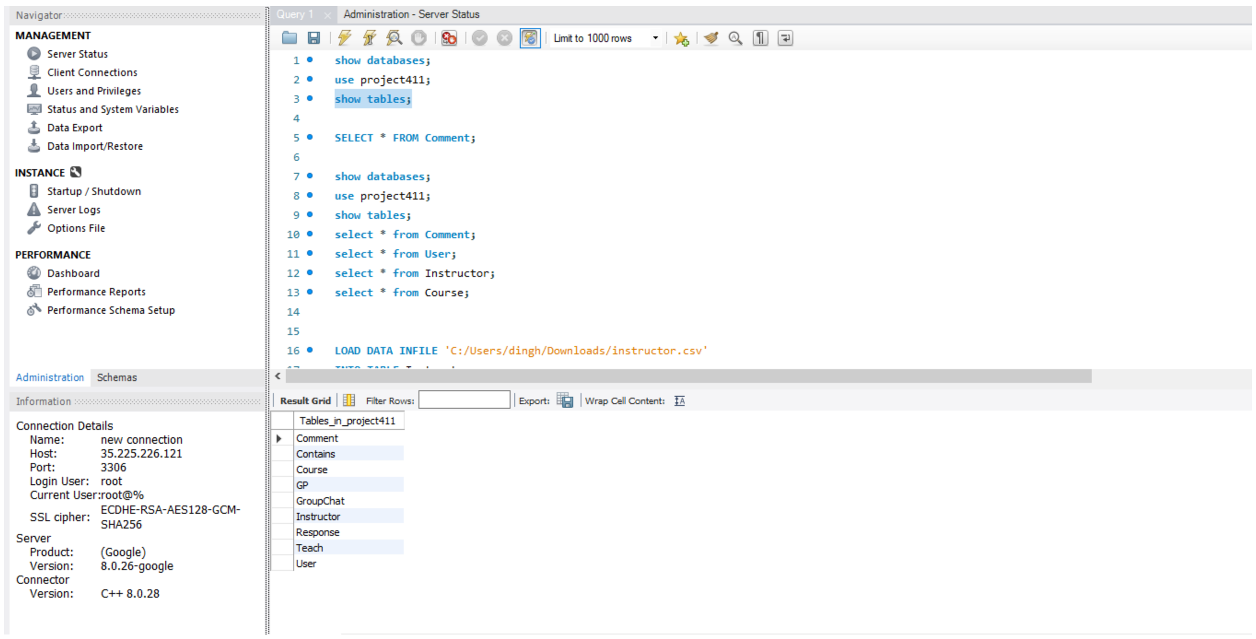
1.Data Definition Language

So far, we have created all tables which would be used in our final project. The DDL are showed as listed.

```
Create Table User(
User_ID INT,
    User_Name Varchar(255),
    User_Gender Varchar(30),
    Primary Key (User_ID)
);
Create Table Comment(
Comment_ID Int,
    Comment_Text Varchar(1024),
    Comment_Score Int,
    Comment_Course_ID Int,
    Comment_User_ID Int,
    Primary Key (Comment_ID),
    Foreign Key (Comment_Course_ID) References Course(Course_ID),
    Foreign Key (Comment_User_ID) References User(User_ID)
```

```
);
Create Table Course(
Course_ID Int,
Course_Num Int,
    Course_Department Varchar(255),
    Course_Instructor_ID Int,
    Primary Key (Course_ID),
    Foreign Key (Course_Instructor_ID) References Instructor(Instructor_ID)
);
Create Table Instructor(
Instructor_ID int,
    Instructor_Name Varchar(255),
    Instructor_Department Varchar(255),
Primary Key (Instructor_ID)
);
Create Table Response(
Response_ID Int,
    Response_Text Varchar(1024),
    Response_Comment_ID Int,
    Primary Key (Response_ID),
    Foreign Key (Response_Comment_ID) References Comment(Comment_ID)
);
Create Table GP(
   User_ID Int,
    Group_ID Int,
    Primary Key (User_ID),
    Primary Key (Group_ID),
    Foreign Key (User_ID) References User(Course_ID)
);
Create Table Teach(
Course_ID Int,
    Instructor_ID Int,
    Primary Key (Course_ID),
    Primary Key (Instructor_ID),
    Foreign Key (Course_ID) References Course(Course_ID),
    Foreign Key (Instructor_ID) References Instructor(Instructor_ID)
);
Create Table GroupChat(
Group_Chat_ID Int,
    Message_Info Varchar(1024),
    Receive_From_ID Int,
    Send_To_ID Int,
    Primary Key (Group_Chat_ID)
);
```

The databases screenshot is as follows:



Connection Details
Name: new connection
Host: 35.225.226.121
Port: 3306
Login User: root
Current User:root@%
SSL cipher: ECDHE-RSA-AES128-GCM-SHA256
Server
Product: (Google)
Version: 8.0.26-google
Connector
Version: C++ 8.0.28

```
show databases;
use project411;
show tables;

SELECT * FROM Comment;

show databases;
use project411;
show tables;
select * from Comment;
select * from User;
select * from Instructor;
select * from Course;


LOAD DATA INFILE 'C:/Users/dingh/Downloads/instructor.csv'
```

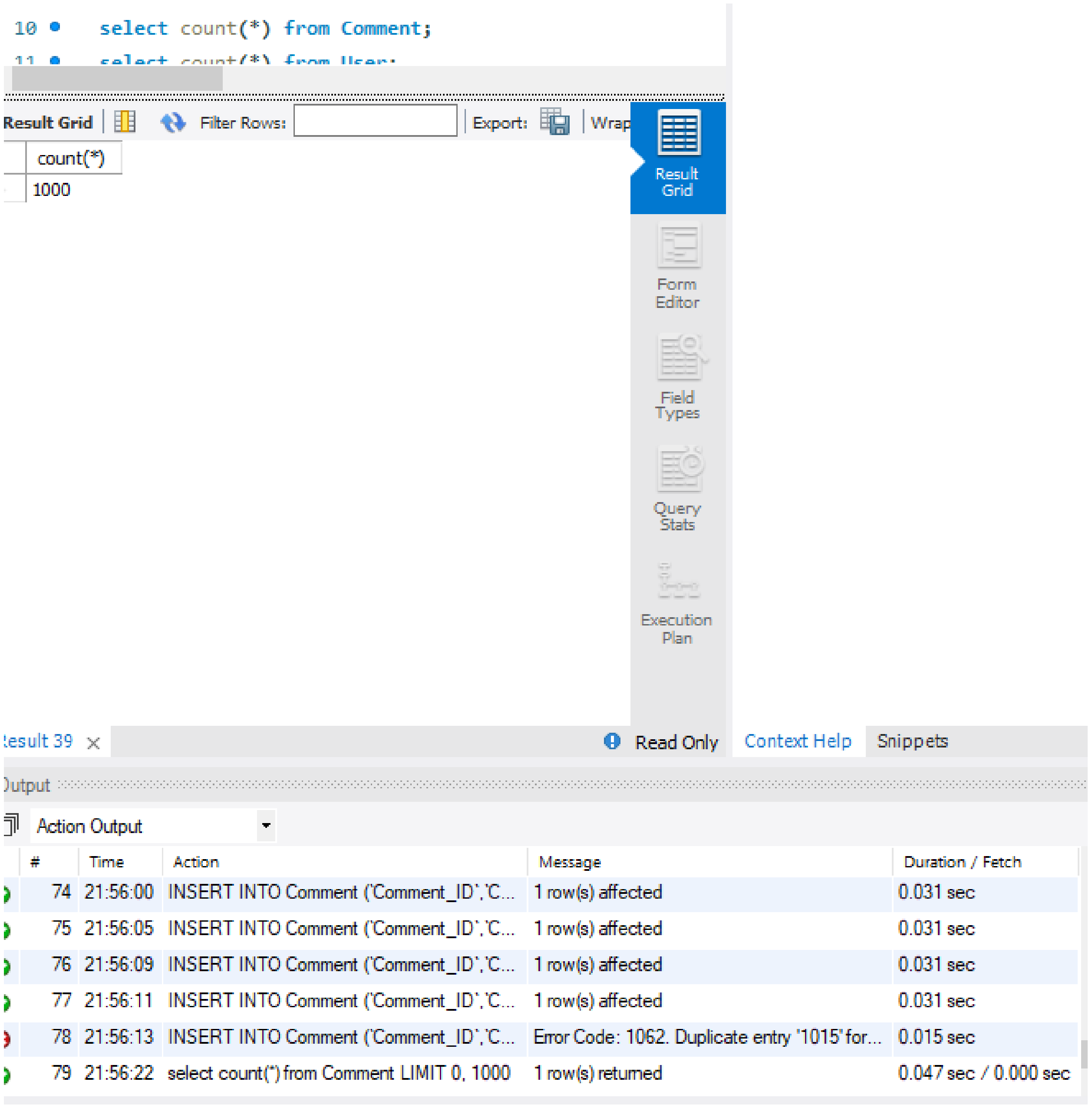
Tables_in_project411
Comment
Contains
Course
GP
GroupChat
Instructor
Response
Teach
User

2. Insert data to tables

We insert data into User, Comment, Course, and Instructor tables. Most of the imported are real. All real data comes from rateMyProfessor website. We crawl all U of I instructor information that stored at the website and imported them into Instructor. Data stored at Comment table are rate comments for CS and EE major. However, we had to create User information because all rates posted on the website are anonymous. The rows for comment, instructor and user tables are over 1000.

Screenshot for count:

Comment

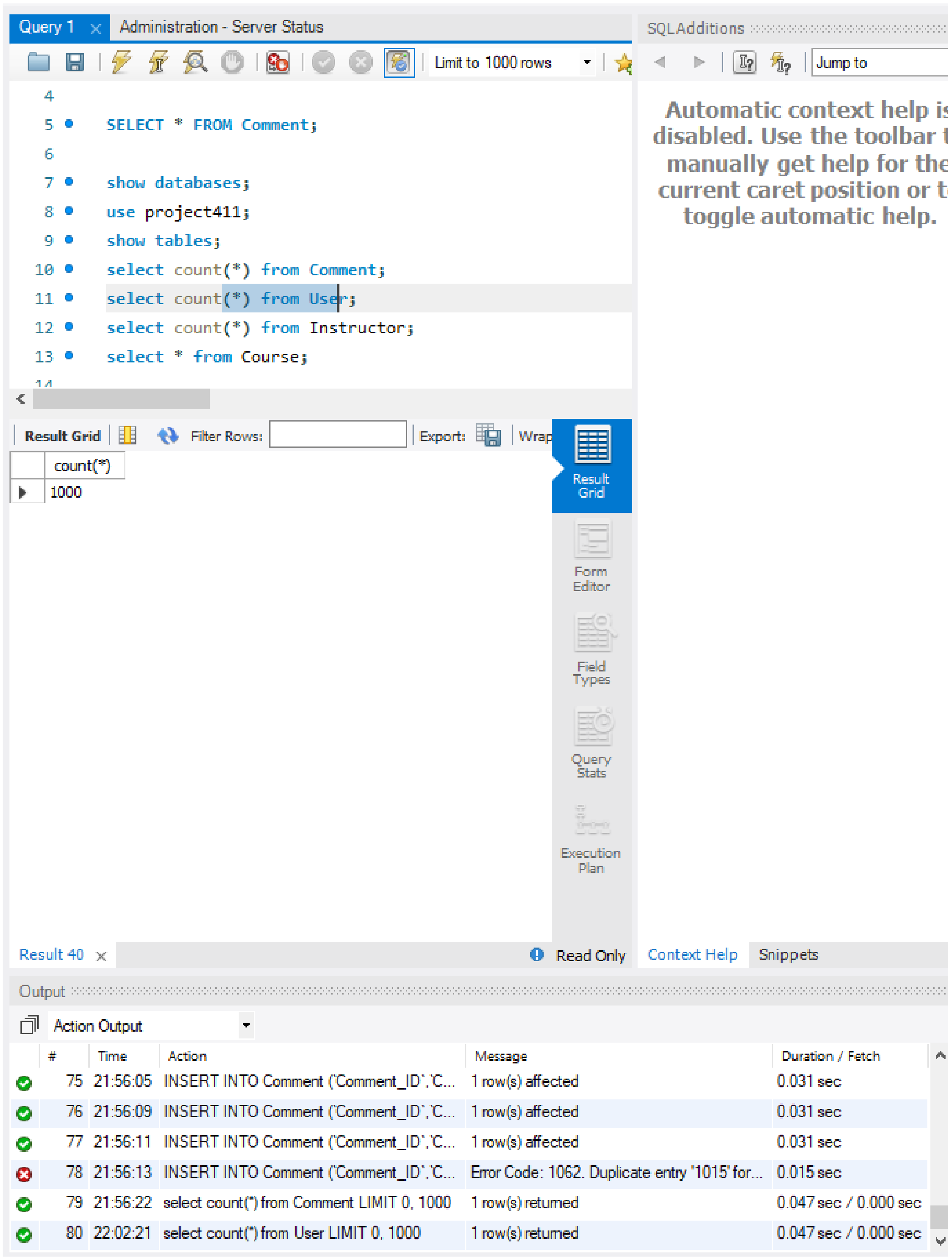


```
select count(*) from Comment;
select count(*) from User;
```

```
count(*)
1000
```

```
#   Time      Action                                        Message                                      Duration / Fetch
74  21:56:00  INSERT INTO Comment ('Comment_ID','C...       1 row(s) affected                            0.031 sec
75  21:56:05  INSERT INTO Comment ('Comment_ID','C...       1 row(s) affected                            0.031 sec
76  21:56:09  INSERT INTO Comment ('Comment_ID','C...       1 row(s) affected                            0.031 sec
77  21:56:11  INSERT INTO Comment ('Comment_ID','C...       1 row(s) affected                            0.031 sec
78  21:56:13  INSERT INTO Comment ('Comment_ID','C...       Error Code: 1062. Duplicate entry '1015' for...  0.015 sec
79  21:56:22  select count(*) from Comment LIMIT 0, 1000    1 row(s) returned                            0.047 sec / 0.000 sec
```

User



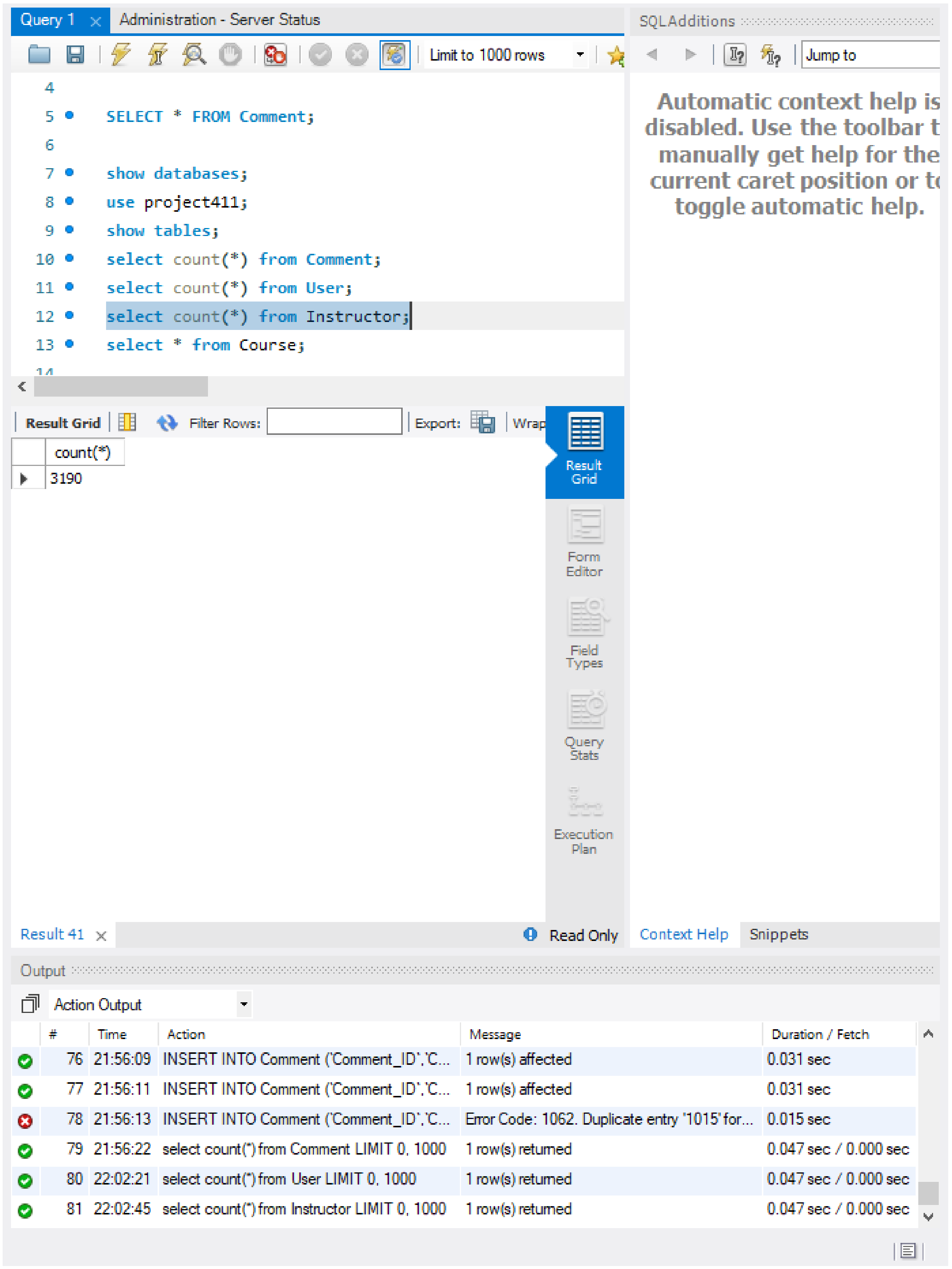
```

SELECT * FROM Comment;

show databases;
use project411;
show tables;
select count(*) from Comment;
select count(*) from User;
select count(*) from Instructor;
select * from Course;
```

Automatic context help is disabled. Use the toolbar t manually get help for the current caret position or t toggle automatic help.

	count(*)
▶	1000

```
  #   Time      Action                                      Message                                  Duration / Fetch
  75  21:56:05  INSERT INTO Comment ('Comment_ID','C...      1 row(s) affected                        0.031 sec
  76  21:56:09  INSERT INTO Comment ('Comment_ID','C...      1 row(s) affected                        0.031 sec
  77  21:56:11  INSERT INTO Comment ('Comment_ID','C...      1 row(s) affected                        0.031 sec
  78  21:56:13  INSERT INTO Comment ('Comment_ID','C...      Error Code: 1062. Duplicate entry '1015' for...  0.015 sec
  79  21:56:22  select count(*) from Comment LIMIT 0, 1000   1 row(s) returned                        0.047 sec / 0.000 sec
  80  22:02:21  select count(*) from User LIMIT 0, 1000      1 row(s) returned                        0.047 sec / 0.000 sec
```

Instructor



```
SELECT * FROM Comment;

show databases;
use project411;
show tables;
select count(*) from Comment;
select count(*) from User;
select count(*) from Instructor;
select * from Course;
```

Limit to 1000 rows

Automatic context help is disabled. Use the toolbar t manually get help for the current caret position or t toggle automatic help.

count(*)
3190

Result 41 · Read Only

Output · Action Output

	#	Time	Action	Message	Duration / Fetch
✓	76	21:56:09	INSERT INTO Comment ('Comment_ID','C...	1 row(s) affected	0.031 sec
✓	77	21:56:11	INSERT INTO Comment ('Comment_ID','C...	1 row(s) affected	0.031 sec
✗	78	21:56:13	INSERT INTO Comment ('Comment_ID','C...	Error Code: 1062. Duplicate entry '1015' for...	0.015 sec
✓	79	21:56:22	select count(*) from Comment LIMIT 0, 1000	1 row(s) returned	0.047 sec / 0.000 sec
✓	80	22:02:21	select count(*) from User LIMIT 0, 1000	1 row(s) returned	0.047 sec / 0.000 sec
✓	81	22:02:45	select count(*) from Instructor LIMIT 0, 1000	1 row(s) returned	0.047 sec / 0.000 sec

3.Two advanced SQL queries

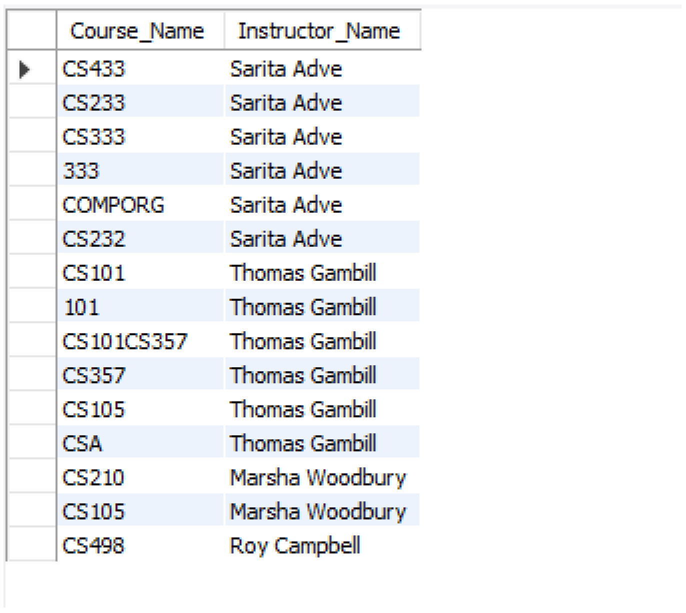
We created 2 sql queries, The first one is used to show all course which belongs to computer science department and their instructor's information. This would be used as a guidance for user to see the catalog. Another is to show all the course and their average score which got more than 4 points. This query would be used in filter course.

The first 15 lines of each sql are listed below:

1.Query commands

```
SELECT DISTINCT Course_Name, i.Instructor_Name

FROM Instructor i JOIN Course c ON i.Instructor_ID = c.Course_Instructor_ID

WHERE c.Course_ID IN (

        SELECT Course_ID

   FROM Course

   WHERE Course_Department = "Computer Science"

)

limit 15;
```

screenshot:



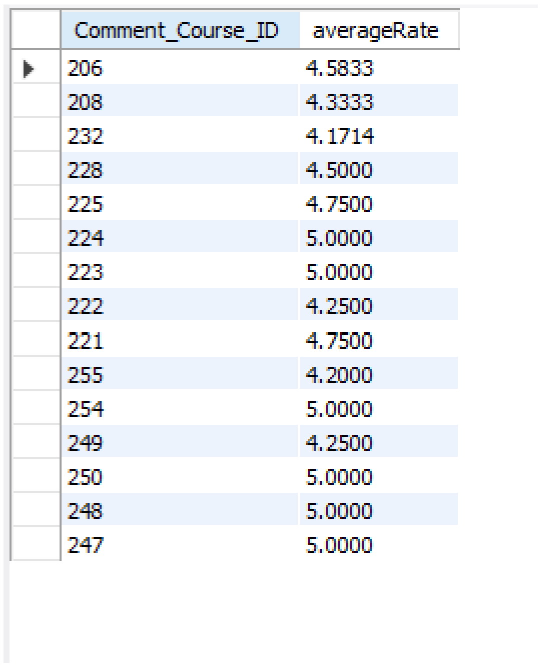
	Course_Name	Instructor_Name
▶	CS433	Sarita Adve
	CS233	Sarita Adve
	CS333	Sarita Adve
	333	Sarita Adve
	COMPORG	Sarita Adve
	CS232	Sarita Adve
	CS101	Thomas Gambill
	101	Thomas Gambill
	CS101CS357	Thomas Gambill
	CS357	Thomas Gambill
	CS105	Thomas Gambill
	CSA	Thomas Gambill
	CS210	Marsha Woodbury
	CS105	Marsha Woodbury
	CS498	Roy Campbell

2. Query commands

```
SELECT Comment_Course_ID, AVG(c.Comment_Score) as averageRate
```

```
FROM Comment c NATURAL JOIN Course co
WHERE co.Course_Department = "Computer Science"
GROUP BY Comment_Course_ID
HAVING AVG(c.Comment_Score) > 4
limit 15;
```

screenshot:



Comment_Course_ID	averageRate
206	4.5833
208	4.3333
232	4.1714
228	4.5000
225	4.7500
224	5.0000
223	5.0000
222	4.2500
221	4.7500
255	4.2000
254	5.0000
249	4.2500
250	5.0000
248	5.0000
247	5.0000

4. Indexing

For the first query, the three index and EXPLAIN ANALYZE are as followed:

The First Query:

Compare with the original cost, all these three Index shorten the actual running time. We create Index courseName because the first query needs a subquery which will check all record if they have a specific courseName. Using this index, query can access less records than original. This part make the query consume less time than original. Then we try to use index based on Course_Department. The main time reduce in on Single row index lookup on I am using primary part. The third index using Instructor_Name. The main saving part is the Temporary table with deduplication part. Here the actual time is half less than the original. However, the coast did not change. This might because this index changes the order of original array. This makes SQL more easily finding out the duplicate records.

Original

```
'-> Limit: 15 row(s)  (cost=45.72..47.28 rows=15) (actual time=0.561..0.564 rows=15 loops=1)\n
-> Table scan on <temporary>  (cost=0.11..2.81 rows=25) (actual time=0.001..0.002 rows=15
loops=1)\n        -> Temporary table with deduplication  (cost=45.72..48.42 rows=25) (actual
time=0.560..0.562 rows=15 loops=1)\n            -> Limit table size: 15 unique row(s)\n                ->
Nested loop inner join  (cost=43.09 rows=25) (actual time=0.087..0.522 rows=15 loops=1)\n
-> Nested loop inner join  (cost=34.27 rows=25) (actual time=0.072..0.114 rows=15 loops=1)\n
-> Filter: (Course.Course_Department = \'Computer Science\')  (cost=25.45 rows=25) (actual
time=0.055..0.063 rows=15 loops=1)\n                         -> Table scan on Course  (cost=25.45
rows=252) (actual time=0.049..0.052 rows=15 loops=1)\n                     -> Filter:
(c.Course_Instructor_ID is not null)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1
loops=15)\n                         -> Single-row index lookup on c using PRIMARY
(Course_ID=Course.Course_ID)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1
loops=15)\n                  -> Single-row index lookup on i using PRIMARY
(Instructor_ID=c.Course_Instructor_ID)  (cost=0.25 rows=1) (actual time=0.027..0.027 rows=1
loops=15)\n'
```

```
CREATE INDEX courseName

ON Course(Course_Department);
```

```
'-> Limit: 15 row(s)  (cost=174.39..174.75 rows=15) (actual time=0.197..0.201 rows=15
loops=1)\n    -> Table scan on <temporary>  (cost=0.03..4.84 rows=187) (actual
time=0.001..0.003 rows=15 loops=1)\n        -> Temporary table with deduplication
(cost=174.39..179.20 rows=187) (actual time=0.196..0.199 rows=15 loops=1)\n            -> Limit
table size: 15 unique row(s)\n                -> Nested loop inner join  (cost=155.66 rows=187)
(actual time=0.089..0.163 rows=15 loops=1)\n                   -> Nested loop inner join  (cost=90.21
rows=187) (actual time=0.080..0.143 rows=15 loops=1)\n                     -> Index lookup on
Course using courseName (Course_Department=\'Computer Science\')  (cost=24.76 rows=187)
(actual time=0.053..0.057 rows=15 loops=1)\n                     -> Filter: (c.Course_Instructor_ID is
not null)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=15)\n                         ->
Single-row index lookup on c using PRIMARY (Course_ID=Course.Course_ID)  (cost=0.25
rows=1) (actual time=0.004..0.004 rows=1 loops=15)\n                  -> Single-row index lookup on
i using PRIMARY (Instructor_ID=c.Course_Instructor_ID)  (cost=0.25 rows=1) (actual
time=0.001..0.001 rows=1 loops=15)\n'
```

```
CREATE INDEX courseName2
```

```
ON Course(Course_Name);
```

```
'-> Limit: 15 row(s)  (cost=45.72..47.28 rows=15) (actual time=0.215..0.217 rows=15 loops=1)\n
-> Table scan on <temporary>  (cost=0.11..2.81 rows=25) (actual time=0.001..0.002 rows=15
loops=1)\n        -> Temporary table with deduplication  (cost=45.72..48.42 rows=25) (actual
time=0.214..0.216 rows=15 loops=1)\n            -> Limit table size: 15 unique row(s)\n                ->
Nested loop inner join  (cost=43.09 rows=25) (actual time=0.087..0.132 rows=15 loops=1)\n
-> Nested loop inner join  (cost=34.27 rows=25) (actual time=0.075..0.111 rows=15 loops=1)\n
-> Filter: (Course.Course_Department = \'Computer Science\')  (cost=25.45 rows=25) (actual
time=0.060..0.068 rows=15 loops=1)\n                        -> Table scan on Course  (cost=25.45
rows=252) (actual time=0.053..0.055 rows=15 loops=1)\n                    -> Filter:
(c.Course_Instructor_ID is not null)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1
loops=15)\n                        -> Single-row index lookup on c using PRIMARY
(Course_ID=Course.Course_ID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1
loops=15)\n                  -> Single-row index lookup on i using PRIMARY
(Instructor_ID=c.Course_Instructor_ID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1
loops=15)\n'
```

```
CREATE INDEX instName

ON Instructor(Instructor_Name);
```

```
'-> Limit: 15 row(s)  (cost=45.72..47.28 rows=15) (actual time=0.151..0.154 rows=15 loops=1)\n
-> Table scan on <temporary>  (cost=0.11..2.81 rows=25) (actual time=0.001..0.002 rows=15
loops=1)\n        -> Temporary table with deduplication  (cost=45.72..48.42 rows=25) (actual
time=0.151..0.152 rows=15 loops=1)\n            -> Limit table size: 15 unique row(s)\n                ->
Nested loop inner join  (cost=43.09 rows=25) (actual time=0.077..0.119 rows=15 loops=1)\n
-> Nested loop inner join  (cost=34.27 rows=25) (actual time=0.069..0.102 rows=15 loops=1)\n
-> Filter: (Course.Course_Department = \'Computer Science\')  (cost=25.45 rows=25) (actual
time=0.053..0.061 rows=15 loops=1)\n                        -> Table scan on Course  (cost=25.45
rows=252) (actual time=0.047..0.050 rows=15 loops=1)\n                    -> Filter:
(c.Course_Instructor_ID is not null)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1
loops=15)\n                        -> Single-row index lookup on c using PRIMARY
(Course_ID=Course.Course_ID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1
loops=15)\n                  -> Single-row index lookup on i using PRIMARY
(Instructor_ID=c.Course_Instructor_ID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1
loops=15)\n'
```

The Second Query:

For the second query, we also design three index. One is using Commment_Score, one is using Comment_Course_ID from comment table. Another is using Course_Department from course table. However, all these three-index failed to shorten the actual running time. The third one makes the query spend twice as much time as the original one. The reason causes such a situation may be due to the group clause. No matter how we design the index and shorten the search time, sql still need time to sort and make calculation. For the last index, such a sort make the inner join more hard to finish. All these reasons make the designed index failed to reduce the actual running time.

Original

```
'-> Limit: 15 row(s)  (actual time=89.319..89.329 rows=15 loops=1)\n    -> Filter:
(avg(c.Comment_Score) > 4)  (actual time=89.318..89.327 rows=15 loops=1)\n        -> Table scan
on <temporary>  (actual time=0.002..0.004 rows=38 loops=1)\n            -> Aggregate using
temporary table  (actual time=89.276..89.280 rows=38 loops=1)\n                -> Inner hash join
(no condition)  (cost=2375.16 rows=2258) (actual time=0.439..13.572 rows=184382 loops=1)\n
-> Filter: (co.Course_Department = \'Computer Science\')  (cost=0.03 rows=25) (actual
time=0.036..0.260 rows=187 loops=1)\n                    -> Table scan on co  (cost=0.03 rows=252)
(actual time=0.030..0.154 rows=255 loops=1)\n                  -> Hash\n                    -> Table scan
on c  (cost=93.60 rows=896) (actual time=0.039..0.313 rows=986 loops=1)\n'
```

```
CREATE INDEX commentScore

ON Comment(Comment_Score);
```

```
'-> Limit: 15 row(s)  (actual time=90.832..90.841 rows=15 loops=1)\n    -> Filter:
(avg(c.Comment_Score) > 4)  (actual time=90.831..90.839 rows=15 loops=1)\n        -> Table scan
on <temporary>  (actual time=0.001..0.003 rows=38 loops=1)\n            -> Aggregate using
temporary table  (actual time=90.822..90.826 rows=38 loops=1)\n                -> Inner hash join
(no condition)  (cost=2375.16 rows=2258) (actual time=0.429..12.557 rows=184382 loops=1)\n
-> Filter: (co.Course_Department = \'Computer Science\')  (cost=0.03 rows=25) (actual
time=0.042..0.301 rows=187 loops=1)\n                    -> Table scan on co  (cost=0.03 rows=252)
(actual time=0.037..0.176 rows=255 loops=1)\n                  -> Hash\n                    -> Table scan
on c  (cost=93.60 rows=896) (actual time=0.035..0.297 rows=986 loops=1)\n'
```

```
CREATE INDEX commentId

ON Comment(Comment_Course_ID);
```

```
'-> Limit: 15 row(s)  (actual time=91.350..91.359 rows=15 loops=1)\n    -> Filter:
(avg(c.Comment_Score) > 4)  (actual time=91.349..91.357 rows=15 loops=1)\n        -> Table scan
```

```
on <temporary>  (actual time=0.001..0.003 rows=38 loops=1)\n            -> Aggregate using
temporary table  (actual time=91.336..91.340 rows=38 loops=1)\n                -> Inner hash join
(no condition)  (cost=2375.16 rows=2258) (actual time=0.448..13.879 rows=184382 loops=1)\n
-> Filter: (co.Course_Department = \'Computer Science\')  (cost=0.03 rows=25) (actual
time=0.036..0.378 rows=187 loops=1)\n                        -> Table scan on co  (cost=0.03 rows=252)
(actual time=0.029..0.231 rows=255 loops=1)\n                    -> Hash\n                        -> Table scan
on c  (cost=93.60 rows=896) (actual time=0.052..0.317 rows=986 loops=1)\n'
```

```
CREATE INDEX courseName

ON Course(Course_Department);
```

```
'-> Limit: 15 row(s)  (actual time=175.163..175.183 rows=15 loops=1)\n    -> Filter:
(avg(c.Comment_Score) > 4)  (actual time=175.162..175.181 rows=15 loops=1)\n        -> Table
scan on <temporary>  (actual time=0.002..0.006 rows=56 loops=1)\n            -> Aggregate using
temporary table  (actual time=175.155..175.162 rows=56 loops=1)\n                -> Nested loop
inner join  (cost=22280.80 rows=167552) (actual time=0.105..98.879 rows=184382 loops=1)\n
-> Table scan on c  (cost=93.60 rows=896) (actual time=0.032..0.413 rows=986 loops=1)\n
-> Index lookup on co using courseName (Course_Department=\'Computer Science\')
(cost=6.08 rows=187) (actual time=0.024..0.088 rows=187 loops=986)\n'
```