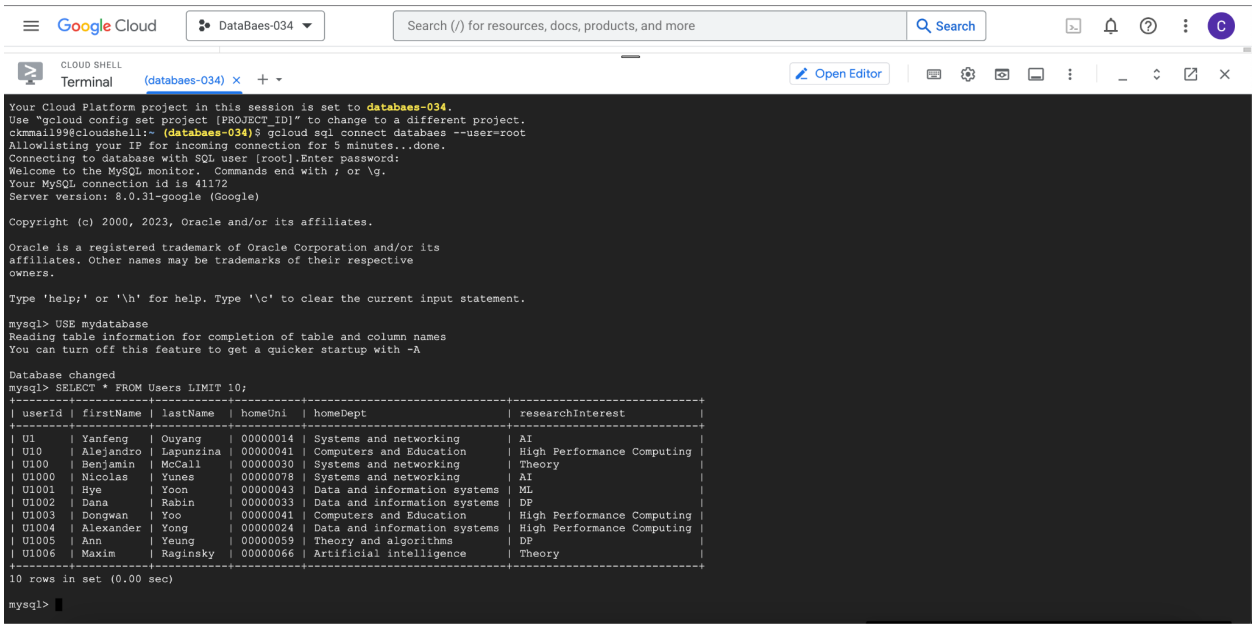
Team 034 DataBaes Project Track 1 Stage 3

Part 1

Proof of connection to database:



DDL Commands:

```
CREATE TABLE `Courses` (
  `courseId` varchar(16) NOT NULL,
  `courseName` varchar(255) DEFAULT NULL,
  `courseDesc` text,
  `num_students` int DEFAULT NULL,
  PRIMARY KEY (`courseId`)
)
```

```
mysql> SELECT COUNT(*) FROM Courses;
+----------+
| COUNT(*) |
+----------+
|      776 |
+----------+
1 row in set (0.00 sec)
```

```
CREATE TABLE `Experiences` (
  `expId` varchar(8) NOT NULL,
  `programId` varchar(8) DEFAULT NULL,
  `userId` varchar(8) DEFAULT NULL,
  `courseId` varchar(16) NOT NULL,
  `housingId` int DEFAULT NULL,
  `blog_exp` text,
  PRIMARY KEY (`expId`),
  KEY `userId` (`userId`),
  KEY `courseId_idx` (`courseId`),
  KEY `courseId` (`courseId`),
  KEY `programId` (`programId`),
  KEY `housingId_idx` (`housingId`),
  CONSTRAINT `housingId` FOREIGN KEY (`housingId`) REFERENCES `Housing`
(`housingId`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `programId` FOREIGN KEY (`programId`) REFERENCES `Programs`
(`programId`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `userId` FOREIGN KEY (`userId`) REFERENCES `Users` (`userId`) ON
DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `courseId` FOREIGN KEY (`courseId`) REFERENCES `Universities`
(`courseId`) ON DELETE CASCADE ON UPDATE CASCADE
);
```

```
mysql> SELECT COUNT(*) FROM Experiences;
+----------+
| COUNT(*) |
+----------+
|     1200 |
+----------+
1 row in set (0.00 sec)
```

```
CREATE TABLE `Housing` (
  `housingId` int NOT NULL,
  `housing_type` varchar(255) DEFAULT NULL,
  `sq_foot` int DEFAULT NULL,
  `bed_size` varchar(255) DEFAULT NULL,
```

```
  `kitchen_ind` tinyint DEFAULT NULL,
  `campus_ind` tinyint DEFAULT NULL,
  PRIMARY KEY (`housingId`)
)
```

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

```
CREATE TABLE `Programs` (
  `programId` varchar(8) NOT NULL,
  `universityId` varchar(8) DEFAULT NULL,
  `start_date` date DEFAULT NULL,
  `end_date` date DEFAULT NULL,
  `semester` varchar(10) DEFAULT NULL,
  `year` int DEFAULT NULL,
  PRIMARY KEY (`programId`),
  KEY `universityId_idx` (`universityId`),
  CONSTRAINT `universityId` FOREIGN KEY (`universityId`) REFERENCES
`Universities` (`universityId`) ON DELETE CASCADE ON UPDATE CASCADE
)
```

```
mysql> SELECT COUNT(*) FROM Programs;
+----------+
| COUNT(*) |
+----------+
|     1200 |
+----------+
1 row in set (0.00 sec)
```

```
CREATE TABLE `Universities` (
  `universityId` varchar(8) NOT NULL,
  `uniName` varchar(255) DEFAULT NULL,
  `attendance` int DEFAULT NULL,
  `countryName` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`universityId`)
)
```

```
mysql> SELECT COUNT(*) FROM Universities;
+----------+
| COUNT(*) |
+----------+
|     2048 |
+----------+
1 row in set (0.00 sec)
```

```
CREATE TABLE `Users` (
  `userId` varchar(8) NOT NULL,
  `firstName` varchar(255) DEFAULT NULL,
  `lastName` varchar(255) DEFAULT NULL,
  `homeUni` varchar(8) DEFAULT NULL,
  `homeDept` varchar(255) DEFAULT NULL,
  `researchInterest` text,
  PRIMARY KEY (`userId`),
```

```
  KEY `universityId_idx` (`homeUni`),
  CONSTRAINT `homeUni` FOREIGN KEY (`homeUni`) REFERENCES
`Universities` (`universityId`) ON DELETE CASCADE ON UPDATE CASCADE
)
```

```
mysql> SELECT COUNT(*) FROM Users;
+----------+
| COUNT(*) |
+----------+
|     1011 |
+----------+
1 row in set (0.01 sec)
```

SQL queries:

1. Find students who have had experiences in both semester "Fall" and "Spring":

Concepts used: Join of multiple relations, Set operations, Subqueries

```
SELECT DISTINCT U1.userId, U1.firstName, U1.lastName
FROM Users U1
JOIN Experiences E1 ON U1.userId = E1.userId
JOIN Programs P1 ON E1.programId = P1.programId
WHERE P1.semester = 'Fall'
AND U1.userId IN (
    SELECT U2.userId
    FROM Users U2
    JOIN Experiences E2 ON U2.userId = E2.userId
    JOIN Programs P2 ON E2.programId = P2.programId
    WHERE P2.semester = 'Spring'
);
```

Result Grid | Filter Rows: | Search

userId	firstName	lastName
U650	Charles	Gammie
U924	Chengxiang	Zhai
U594	Jonathan	Ebel
U583	Jennifer	Fraterrigo
U721	Justin	Rhodes
U603	Jason	Emmert
U571	Matthew	Finkin
U655	Matthew	Goldman
U359	Eugene	Avrutin
U638	Pamela	Hadley
U461	Aimo	Hinkkanen
U889	Rachel	Whitaker
U325	Stephanie	Craft
U548	Javier	Garcia
U484	Myung-Ja	Han
U363	Milan	Bagchi
U464	Jacqueline	Hitchon
U283	Carla	Caceres

2. Find the average number of students in courses for each program:

Concepts used: Join of multiple relations, Aggregation via GROUP BY

```
SELECT P.programId, P.semester, P.year, AVG(C.Num_students) AS
Avg_Students
FROM Programs P
JOIN Experiences E ON P.programId = E.programId
JOIN Courses C ON E.courseId = C.courseId
GROUP BY P.programId, P.semester, P.year
ORDER BY AVG(C.Num_students) DESC;
```

programId	semester	year	Avg_Students
P911	Fall	2020	399.0000
P929	Fall	2020	399.0000
P676	Summer	2018	399.0000
P841	Spring	2020	396.0000
P552	Spring	2017	396.0000
P1168	Summer	2023	395.0000
P376	Spring	2015	395.0000
P607	Fall	2017	393.0000
P1123	Spring	2023	392.0000
P989	Summer	2021	392.0000
P213	Spring	2013	390.0000
P255	Fall	2013	387.0000
P1003	Fall	2021	387.0000
P351	Fall	2014	386.0000
P1036	Spring	2022	386.0000

Part 2

Query 1:

Initial Performance:

`(cost=765.10..767.90 rows=31)`

`(actual time=9.081)`

```
-> Table scan on <temporary>  (cost=765.10..767.90 rows=31) (actual time=9.081..9.098 rows=104 loops=1)
    -> Temporary table with deduplication  (cost=765.01..765.01 rows=31) (actual time=9.079..9.079 rows=104 loops=1)
        -> Nested loop semijoin  (cost=761.90 rows=31) (actual time=0.204..8.936 rows=130 loops=1)
            -> Nested loop inner join  (cost=451.43 rows=185) (actual time=0.121..4.617 rows=427 loops=1)
                -> Nested loop inner join  (cost=386.83 rows=185) (actual time=0.115..3.917 rows=427 loops=1)
                    -> Nested loop inner join  (cost=322.24 rows=185) (actual time=0.096..2.960 rows=427 loops=1)
                        -> Filter: (P1.semester = 'Fall')  (cost=121.50 rows=120) (actual time=0.054..0.577 rows=432
loops=1)
                            -> Table scan on P1  (cost=121.50 rows=1200) (actual time=0.047..0.418 rows=1200 loops=1)
                        -> Filter: (E1.userId is not null)  (cost=1.52 rows=2) (actual time=0.005..0.005 rows=1
loops=432)
                            -> Index lookup on E1 using programId (programId=P1.programId)  (cost=1.52 rows=2) (actual
time=0.005..0.005 rows=1 loops=432)
                    -> Single-row index lookup on U1 using PRIMARY (userId=E1.userId)  (cost=0.25 rows=1) (actual
time=0.002..0.002 rows=1 loops=427)
                -> Single-row covering index lookup on U2 using PRIMARY (userId=E1.userId)  (cost=0.25 rows=1) (actual
time=0.001..0.001 rows=1 loops=427)
            -> Nested loop inner join  (cost=79.63 rows=0.2) (actual time=0.010..0.010 rows=0 loops=427)
                -> Filter: (E2.programId is not null)  (cost=1.67 rows=2) (actual time=0.005..0.006 rows=2 loops=427)
                    -> Index lookup on E2 using userId (userId=E1.userId)  (cost=1.67 rows=2) (actual
time=0.005..0.006 rows=2 loops=427)
                -> Limit: 1 row(s)  (cost=0.04 rows=0.1) (actual time=0.002..0.002 rows=0 loops=818)
                    -> Filter: (P2.semester = 'Spring')  (cost=0.04 rows=0.1) (actual time=0.002..0.002 rows=0
loops=818)
                        -> Single-row index lookup on P2 using PRIMARY (programId=E2.programId)  (cost=0.04 rows=1)
(actual time=0.001..0.001 rows=1 loops=818)
```

Indexes:

First Index:

`CREATE INDEX idx_users_userid ON Users(userId);`

The cost stayed the same (cost = 765.10) since the actual query itself stayed the same. However the time it took for the overall query to run decreased from `actual time=9.081` to `actual time=8.872.`

Which was slightly faster due to adding an index that we did not have before.

Then delete the index with:

```
DROP INDEX idx_users_userid ON Users;
```

Second Index:

`CREATE INDEX idx_programs_semester ON Programs(semester);`

The time got quite a bit faster to: `actual time=8.185`

This might be because with the focus on the semester from programs we were able to index based on the Programs Semester which is the most outer query in our SQL query. In addition the lack of the User Id might have made it a bit faster that we did not have to query onto, but the semester as the index increased the cost which you can see in the combined indexes below.

We can remove these indexes and prime it for the next one:

```
DROP INDEX idx_programs_semester ON Programs;
```

3rd Index (Adding both prior ones):

```
CREATE INDEX idx_users_userid ON Users(userId);
CREATE INDEX idx_programs_semester ON Programs(semester);
```

The actual time decreased to: `actual time=8.537`

However the cost increased significantly, this might be because indexing on the semester might create more concentrated indices (“Fall”, “Summer”...) leaving there not to be many options to index into, increasing the cost by around 3x.

```
cost=1719.14
```

Query 2:

Initial Performance:

```
SHOW INDEX FROM Programs;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
Programs	0	PRIMARY	1	programId	A	1201	NULL	NULL		BTREE			YES	NULL
Programs	1	universityId_idx	1	universityId	A	903	NULL	NULL	YES	BTREE			YES	NULL

```
EXPLAIN ANALYZE (SELECT P.programId, P.semester, P.year,
AVG(C.Num_students) AS Avg_Students
FROM Programs P
JOIN Experiences E ON P.programId = E.programId
JOIN Courses C ON E.courseId = C.courseId
GROUP BY P.programId, P.semester, P.year
ORDER BY AVG(C.Num_students) DESC)
```

```
-> Sort: Avg_Students DESC  (actual time=7.322..7.370 rows=554
loops=1)
    -> Table scan on <temporary>  (actual time=6.763..6.871 rows=554
loops=1)
        -> Aggregate using temporary table  (actual time=6.762..6.762
rows=554 loops=1)
            -> Nested loop inner join  (cost=1019.81 rows=1155)
```

```
(actual time=0.187..5.747 rows=741 loops=1)
                -> Nested loop inner join  (cost=615.56 rows=1155)
(actual time=0.166..4.560 rows=741 loops=1)
                    -> Filter: (E.programId is not null)
(cost=211.31 rows=1155) (actual time=0.127..0.838 rows=1200 loops=1)
                        -> Table scan on E  (cost=211.31 rows=1155)
(actual time=0.126..0.740 rows=1200 loops=1)
                    -> Single-row index lookup on C using PRIMARY
(courseId=E.courseId)  (cost=0.25 rows=1) (actual time=0.003..0.003
rows=1 loops=1200)
                -> Single-row index lookup on P using PRIMARY
(programId=E.programId)  (cost=0.25 rows=1) (actual time=0.001..0.001
rows=1 loops=741)
```

1. We created an index on “semester” because it is included in the GROUP BY.

```
CREATE INDEX idx_semester ON Programs (semester);
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
Programs	0	PRIMARY	1	programId	A	1201	NULL	NULL		BTREE			YES	NULL
Programs	1	universityId_idx	1	universityId	A	903	NULL	NULL	YES	BTREE			YES	NULL
Programs	1	idx_semester	1	semester	A	3	NULL	NULL	YES	BTREE			YES	NULL

```
EXPLAIN ANALYZE (SELECT P.programId, P.semester, P.year,
AVG(C.Num_students) AS Avg_Students
FROM Programs P
JOIN Experiences E ON P.programId = E.programId
JOIN Courses C ON E.courseId = C.courseId
GROUP BY P.programId, P.semester, P.year
ORDER BY AVG(C.Num_students) DESC)
```

```
-> Sort: Avg_Students DESC  (actual time=13.004..13.074 rows=554
loops=1)
    -> Table scan on <temporary>  (actual time=12.394..12.566
rows=554 loops=1)
        -> Aggregate using temporary table  (actual
time=12.392..12.392 rows=554 loops=1)
            -> Nested loop inner join  (cost=1019.81 rows=1155)
(actual time=0.192..10.268 rows=741 loops=1)
                -> Nested loop inner join  (cost=615.56 rows=1155)
(actual time=0.175..8.084 rows=741 loops=1)
```

```
                    -> Filter: (E.programId is not null)
(cost=211.31 rows=1155) (actual time=0.114..1.567 rows=1200 loops=1)
                        -> Table scan on E  (cost=211.31 rows=1155)
(actual time=0.111..1.421 rows=1200 loops=1)
                    -> Single-row index lookup on C using PRIMARY
(courseId=E.courseId)  (cost=0.25 rows=1) (actual time=0.005..0.005
rows=1 loops=1200)
                -> Single-row index lookup on P using PRIMARY
(programId=E.programId)  (cost=0.25 rows=1) (actual time=0.003..0.003
rows=1 loops=741)
```

Semester slows down the query significantly (7.3 seconds to 13.0 seconds).

2. Next, we created an index on "year" because it is also included in the GROUP BY. Dropping semester as an index.

```
DROP INDEX idx_semester ON Programs;
CREATE INDEX idx_year ON Programs (year);
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
Programs	0	PRIMARY	1	programId	A	1201	NULL	NULL		BTREE			YES	NULL
Programs	1	universityId_idx	1	universityId	A	903	NULL	NULL	YES	BTREE			YES	NULL
Programs	1	idx_year	1	year	A	13	NULL	NULL	YES	BTREE			YES	NULL

```
EXPLAIN ANALYZE (SELECT P.programId, P.semester, P.year,
AVG(C.Num_students) AS Avg_Students
FROM Programs P
JOIN Experiences E ON P.programId = E.programId
JOIN Courses C ON E.courseId = C.courseId
GROUP BY P.programId, P.semester, P.year
ORDER BY AVG(C.Num_students) DESC)
```

```
-> Sort: Avg_Students DESC  (actual time=8.312..8.361 rows=554
loops=1)
    -> Table scan on <temporary>  (actual time=7.786..7.988 rows=554
loops=1)
        -> Aggregate using temporary table  (actual time=7.784..7.784
rows=554 loops=1)
            -> Nested loop inner join  (cost=1019.81 rows=1155)
```

```
(actual time=0.150..6.581 rows=741 loops=1)
                -> Nested loop inner join  (cost=615.56 rows=1155)
(actual time=0.139..5.065 rows=741 loops=1)
                    -> Filter: (E.programId is not null)
(cost=211.31 rows=1155) (actual time=0.097..0.983 rows=1200 loops=1)
                        -> Table scan on E  (cost=211.31 rows=1155)
(actual time=0.096..0.874 rows=1200 loops=1)
                    -> Single-row index lookup on C using PRIMARY
(courseId=E.courseId)  (cost=0.25 rows=1) (actual time=0.003..0.003
rows=1 loops=1200)
                -> Single-row index lookup on P using PRIMARY
(programId=E.programId)  (cost=0.25 rows=1) (actual time=0.002..0.002
rows=1 loops=741)
```

Year speeds up the query compared to semester.

3. Now index on both semester and year because they are both included in the GROUP BY.

```
CREATE INDEX idx_semester ON Programs (semester);
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
Programs	0	PRIMARY	1	programId	A	1201	NULL	NULL		BTREE			YES	NULL
Programs	1	universityId_idx	1	universityId	A	903	NULL	NULL	YES	BTREE			YES	NULL
Programs	1	idx_year	1	year	A	13	NULL	NULL	YES	BTREE			YES	NULL
Programs	1	idx_semester	1	semester	A	3	NULL	NULL	YES	BTREE			YES	NULL

```
EXPLAIN ANALYZE (SELECT P.programId, P.semester, P.year,
AVG(C.Num_students) AS Avg_Students
FROM Programs P
JOIN Experiences E ON P.programId = E.programId
JOIN Courses C ON E.courseId = C.courseId
GROUP BY P.programId, P.semester, P.year
ORDER BY AVG(C.Num_students) DESC)
```

```
-> Sort: Avg_Students DESC  (actual time=7.908..7.957 rows=554
loops=1)
    -> Table scan on <temporary>  (actual time=7.509..7.606 rows=554
loops=1)
        -> Aggregate using temporary table  (actual time=7.507..7.507
rows=554 loops=1)
```

```
            -> Nested loop inner join  (cost=1019.81 rows=1155)
(actual time=0.127..6.377 rows=741 loops=1)
                -> Nested loop inner join  (cost=615.56 rows=1155)
(actual time=0.112..5.005 rows=741 loops=1)
                    -> Filter: (E.programId is not null)
(cost=211.31 rows=1155) (actual time=0.072..0.984 rows=1200 loops=1)
                        -> Table scan on E  (cost=211.31 rows=1155)
(actual time=0.071..0.877 rows=1200 loops=1)
                    -> Single-row index lookup on C using PRIMARY
(courseId=E.courseId)  (cost=0.25 rows=1) (actual time=0.003..0.003
rows=1 loops=1200)
                -> Single-row index lookup on P using PRIMARY
(programId=E.programId)  (cost=0.25 rows=1) (actual time=0.002..0.002
rows=1 loops=741)
```

Query speeds up again when both semester and year are added as an index. However, the total time is still greater than the time when only programId is the index.

For query 2, we chose to try semester, year, and both semester and year as indices in the Programs dataset. In this particular query, ProgramId, semester, and year are all a part of a GROUP BY clause, so we thought it would be appropriate to try them as indices. However, when we compare the times, the original run with only programId as an index was the fastest at 7.3 seconds. The last run with programId, semester, and year as an index ran in 7.9 seconds, which was the second fastest. Considering that creating an index on semester and year will also make write operations like INSERT and UPDATE take longer and the query degraded in performance based on increasing in time, we decided that adding this index was not useful in improving our database. Therefore, we will move forward with only programId as an index for the Programs table.