

CS 411 stage 3

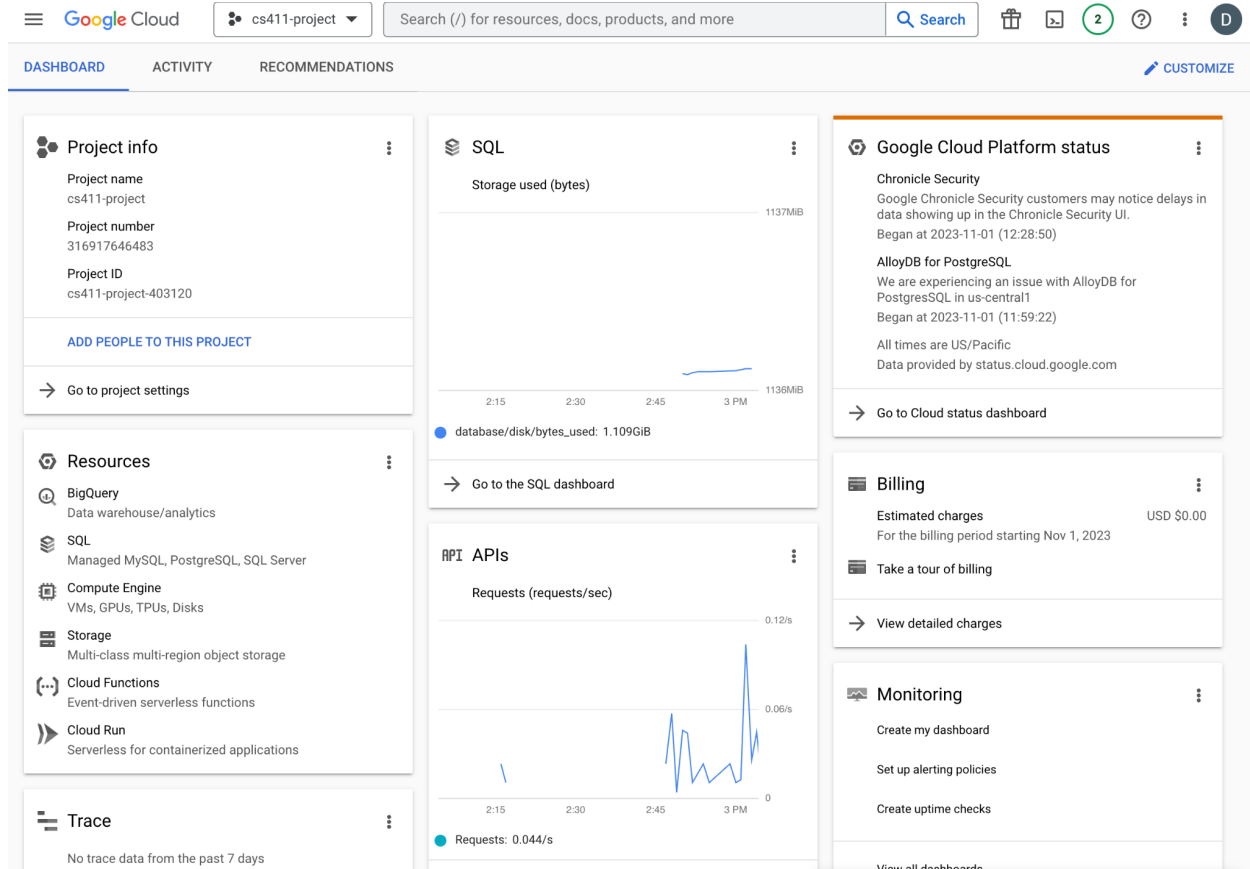
Database Implementation:



A screenshot of a Cloud Shell Terminal window. The window has a title bar with a terminal icon, the text "CLOUD SHELL", and a tab labeled "Terminal (cs411-project-403120)". The terminal content shows a MySQL prompt where the command "show tables" has been executed. The output lists five tables: "Tables_in_myDatabase", "Favorites", "History", "Rating", and "Users". The output is formatted with a header line and a footer line indicating 5 rows in the set.

```
Database changed
mysql> show tables
-> ;
+-----+
| Tables_in_myDatabase |
+-----+
| Favorites             |
| History               |
| Rating                |
| Restaurants           |
| Users                 |
+-----+
5 rows in set (0.00 sec)

mysql> 
```



DDL Commands:

```
CREATE DATABASE /*!32312 IF NOT EXISTS*/ `myDatabase` /*!40100 DEFAULT CHARACTER SET latin1 */;
USE `myDatabase`;
/* Table structure for table `Users` */
DROP TABLE IF EXISTS `History`;
DROP TABLE IF EXISTS `Rating`;
DROP TABLE IF EXISTS `Favorites`;
DROP TABLE IF EXISTS `Restaurants`;
DROP TABLE IF EXISTS `Users`;
CREATE TABLE `Users` (
  `userID` INT NOT NULL,
  `userName` VARCHAR(255) NOT NULL,
  `password` VARCHAR(255) NOT NULL,
  PRIMARY KEY (`userID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/* Table structure for table `Restaurants` */
DROP TABLE IF EXISTS `Rating`;
DROP TABLE IF EXISTS `Restaurants`;
CREATE TABLE `Restaurants` (
  `restaurantName` VARCHAR(255) NOT NULL,
```

```

        `style` VARCHAR(255) NOT NULL,
        `price` VARCHAR(255) NOT NULL,
        `address` VARCHAR(255) NOT NULL,
        `zip` VARCHAR(255) NOT NULL,
        PRIMARY KEY (`restaurantName`)
    ) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/* Table structure for table `History` */
DROP TABLE IF EXISTS `History`;
CREATE TABLE `History` (
    `historyID` INT NOT NULL,
    `userID` INT NOT NULL,
    `input` VARCHAR(255) NOT NULL,
    PRIMARY KEY (`historyID`),
    KEY `userID` (`userID`),
    CONSTRAINT `History_ibfk_1` FOREIGN KEY (`userID`) REFERENCES `Users` (`userID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/* Table structure for table `Favorite` */
DROP TABLE IF EXISTS `Favorites`;
CREATE TABLE `Favorites` (
    `favoriteID` INT NOT NULL,
    `userID` INT NOT NULL,
    `restaurantName` VARCHAR(255) NOT NULL,
    `note` VARCHAR(255) NOT NULL,
    PRIMARY KEY (`favoriteID`),
    KEY `userID` (`userID`),
    KEY `restaurantName` (`restaurantName`),
    CONSTRAINT `Favorite_ibfk_1` FOREIGN KEY (`userID`) REFERENCES `Users` (`userID`),
    CONSTRAINT `Favorite_ibfk_2` FOREIGN KEY (`restaurantName`) REFERENCES `Restaurants`
    (`restaurantName`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/* Table structure for table `Rating` */
DROP TABLE IF EXISTS `Rating`;
CREATE TABLE `Rating` (
    `ratingID` INT NOT NULL,
    `restaurantName` VARCHAR(255) NOT NULL,
    `score` INT NOT NULL,
    `comment` VARCHAR(10000) NOT NULL,
    PRIMARY KEY (`ratingID`),
    KEY `restaurantName` (`restaurantName`),
    CONSTRAINT `Rating_ibfk_1` FOREIGN KEY (`restaurantName`) REFERENCES `Restaurants`
    (`restaurantName`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

```

47 • `SELECT COUNT(historyID) from History;`

0% 38:47

Result Grid Filter Rows: Search Export:

COUNT(historyl...
▶ 1000

49 • `SELECT COUNT(favoriteID) from Favorites;`

100% 40:49

Result Grid Filter Rows: Search Export:

COUNT(favoritel...
▶ 1000

45 • `SELECT COUNT(ratingID) from Rating;`

100% 32:45

Result Grid Filter Rows: Search Export:

COUNT(ratingID)
▶ 2381

Advanced Queries:

```
SELECT restaurantName, AVG(score), style as average_rating
FROM Rating NATURAL JOIN Restaurants
WHERE price = '$$'
GROUP BY restaurantName
ORDER BY average_rating DESC;
```

	restaurantName	average_rating	style
►	Louders	5.0000	Asian Fusion, Chicken Shop
	Angry Chickz	5.0000	Southern, Comfort Food, Chicken Shop
	Honey Night	5.0000	Korean, Bars
	HanShik Express	5.0000	Korean, Fast Food, Tacos
	Jail Joa	5.0000	Gastropubs, Korean, Soup
	Good Goose Cafe	5.0000	Asian Fusion, Thai, Vegetarian
	Howlin' Ray's	5.0000	Southern, Chicken Shop, American (Traditional)
	Heng Heng Chicken Rice	5.0000	Chicken Shop, Thai, Smokehouse
	Ghost Sando Shop	5.0000	Sandwiches, Delis
	Egg Tuck - West Hollywood	5.0000	Breakfast & Brunch, Coffee & Tea, Sandwiches
	BROKEN MOUTH Lee's...	5.0000	Hawaiian, Korean, Comfort Food
	Cento	5.0000	Italian, Mediterranean, Tapas/Small Plates
	Chadolpoong	5.0000	Korean, Soup, Comfort Food
	CHD MANDU LA	5.0000	Korean
	Cooking Mom	5.0000	Korean, Specialty Food, Soup

```

SELECT DISTINCT restaurantName, style, price, address, zip
FROM Favorites NATURAL JOIN Restaurants
WHERE userID = '11' OR style IN (
SELECT input
FROM History
WHERE userID = '11');

```

```

10 SELECT DISTINCT restaurantName, style, price, address, zip
11 FROM Favorites NATURAL JOIN Restaurants
12 WHERE userID = '11' OR style IN (
13     SELECT input
14     FROM History
15     WHERE userID = '11')
16 LIMIT 15;

```

restaurantName	style	price	address	zip
Bacari Silverlake	Mediterranean, Breakfast & Brunch, Cocktail Bars	nan	3626 Sunset Blvd Los Angeles, CA 90026	90026
Chuncheon Dakgalbi Donghae Makguksu 2	Korean	nan	3601 W 6th St Los Angeles, CA 90020	90020
Girl & The Goat - Los Angeles	American (New)	nan	555-3 Mateo St Ste 300 Los Angeles, CA 90013	90013
HanEuem	Korean, Comfort Food, Gastropubs	nan	539 S Western Ave Los Angeles, CA 90020	90020
Kali Restaurant	American (New)	\$\$\$\$	5722 Melrose Ave Los Angeles, CA 90038	90038
Langer's Delicatessen	Delis, Sandwiches	\$\$	704 S Alvarado St Los Angeles, CA 90057	90057
Linden	American (New)	nan	5936 Sunset Blvd Los Angeles, CA 90028	90028
Little Sister	Vietnamese	\$\$	523 W 7th St Los Angeles, CA 90017	90017
Love Hour	Burgers, Food Stands, Bars	\$\$	532 S Western Ave Los Angeles, CA 90020	90020
Meteora	American (New)	\$\$\$\$	6703 Melrose Avenue Los Angeles, CA 90038	90038
Olivia	Vegetarian, Pizza, Wine Bars	\$\$	205 S Vermont Ave Los Angeles, CA 90004	90004
Otium	American (New)	\$\$\$	222 S Hope St Los Angeles, CA 90012	90012
Santo	Japanese, Coffee & Tea, Sushi Bars	nan	3822 Sunset Blvd Los Angeles, CA 90026	90026
Sidewalk Grill	Mediterranean, Wraps, Kebab	\$\$	1727 N Vermont Ave Ste 102 Los Angeles, CA...	90027
The Butcher, The Baker, The Cappuccino...	American (New)	\$\$	8653 W Sunset Blvd West Hollywood, CA 90069	90069

Indexing:

Query 1:

The evaluation of the EXPLAIN ANALYZE outputs from different indexing strategies shows that while all indices improve query performance, the index on the style column (idx_style) emerges as the most effective. This indexing approach yields the quickest sort and nested loop join times (actual time=3.370..3.370), indicating superior query performance. Although the idx_price index reduced join times slightly, it added some overhead to the Restaurants lookup. Conversely, the idx_score index didn't offer notable improvements over the no-index scenario. Therefore, for this specific query, the idx_style index is optimal. However, real-world performance can vary based on database workload and dataset size, necessitating ongoing monitoring and potential adjustments.

1. EXPLAIN ANALYZE with no index

```
'-> Sort: average_rating DESC (actual time=4.171..4.182 rows=123 loops=1)
  -> Table scan on <temporary> (actual time=4.029..4.096 rows=123 loops=1)
    -> Aggregate using temporary table (actual time=4.027..4.027 rows=123 loops=1)
      -> Nested loop inner join (cost=103.28 rows=226) (actual time=0.114..2.939
rows=1266 loops=1)
        -> Filter: (Restaurants.price = "$$") (cost=24.35 rows=24) (actual
time=0.078..0.216 rows=123 loops=1)
          -> Table scan on Restaurants (cost=24.35 rows=236) (actual
time=0.067..0.174 rows=236 loops=1)
            -> Index lookup on Rating using restaurantName
(restaurantName=Restaurants.restaurantName) (cost=2.43 rows=10) (actual
time=0.019..0.021 rows=10 loops=123)
'
```

2. CREATE INDEX idx_price ON Restaurants(price)

```
'-> Sort: average_rating DESC (actual time=3.754..3.764 rows=123 loops=1)
  -> Table scan on <temporary> (actual time=3.649..3.678 rows=123 loops=1)
    -> Aggregate using temporary table (actual time=3.645..3.645 rows=123 loops=1)
      -> Nested loop inner join (cost=425.94 rows=1175) (actual time=0.199..2.535
rows=1266 loops=1)
        -> Index lookup on Restaurants using idx_price (price="$") (cost=14.55
rows=123) (actual time=0.174..0.276 rows=123 loops=1)
          -> Index lookup on Rating using restaurantName
(restaurantName=Restaurants.restaurantName) (cost=2.40 rows=10) (actual
time=0.015..0.017 rows=10 loops=123)
'
```

3. CREATE INDEX idx_score ON Rating(score)
 - > Sort: average_rating DESC (actual time=3.749..3.759 rows=123 loops=1)
 - > Table scan on <temporary> (actual time=3.627..3.659 rows=123 loops=1)
 - > Aggregate using temporary table (actual time=3.625..3.625 rows=123 loops=1)
 - > Nested loop inner join (cost=103.28 rows=226) (actual time=0.119..2.543 rows=1266 loops=1)
 - > Filter: (Restaurants.price = '\$\$') (cost=24.35 rows=24) (actual time=0.077..0.201 rows=123 loops=1)
 - > Table scan on Restaurants (cost=24.35 rows=236) (actual time=0.075..0.172 rows=236 loops=1)
 - > Index lookup on Rating using restaurantName (restaurantName=Restaurants.restaurantName) (cost=2.43 rows=10) (actual time=0.016..0.018 rows=10 loops=123)

4. CREATE INDEX idx_style ON Restaurants(style);
 - > Sort: average_rating DESC (actual time=3.469..3.479 rows=123 loops=1)
 - > Table scan on <temporary> (actual time=3.372..3.401 rows=123 loops=1)
 - > Aggregate using temporary table (actual time=3.370..3.370 rows=123 loops=1)
 - > Nested loop inner join (cost=103.28 rows=226) (actual time=0.082..2.327 rows=1266 loops=1)
 - > Filter: (Restaurants.price = "\$\$") (cost=24.35 rows=24) (actual time=0.053..0.167 rows=123 loops=1)
 - > Table scan on Restaurants (cost=24.35 rows=236) (actual time=0.051..0.139 rows=236 loops=1)
 - > Index lookup on Rating using restaurantName (restaurantName=Restaurants.restaurantName) (cost=2.43 rows=10) (actual time=0.014..0.017 rows=10 loops=123)

Query 2:

The EXPLAIN ANALYZE outputs for Query 2 reveal that, among the different indexing strategies on the Restaurants table, the index on the price column (idx_price) significantly outperforms the others, including the scenarios with no index, an index on style (idx_style), and an index on zip (idx_zip). While the cost estimates for the indexed scenarios are identical (reduce 1184.04..1198.58 to 458.40..472.93), the actual time taken to execute the query is lowest with the price index, clocking in at 2.925..2.928 seconds compared to over 34 seconds without any

index. This marked improvement underscores the effectiveness of the price index in optimizing the query's performance.

1. EXPLAIN ANALYZE with no index

```
'-> Table scan on <temporary> (cost=1184.04..1198.58 rows=965) (actual
time=34.493..34.496 rows=16 loops=1)
  -> Temporary table with deduplication (cost=1184.03..1184.03 rows=965) (actual
time=34.489..34.489 rows=16 loops=1)
    -> Nested loop inner join (cost=1087.57 rows=965) (actual time=29.643..34.408
rows=42 loops=1)
      -> Table scan on Restaurants (cost=26.60 rows=236) (actual
time=19.414..20.219 rows=236 loops=1)
        -> Filter: ((Favorites.userID = 11) or
<in_optimizer>(Restaurants.style,Restaurants.style in (select #2))) (cost=4.09 rows=4)
(actual time=0.060..0.060 rows=0 loops=236)
          -> Index lookup on Favorites using restaurantName
(restaurantName=Restaurants.restaurantName) (cost=4.09 rows=4) (actual
time=0.042..0.044 rows=4 loops=236)
            -> Select #2 (subquery in condition; run only once)
              -> Filter: ((Restaurants.style = `<materialized_subquery>`.input))
(cost=15.30..15.30 rows=1) (actual time=0.015..0.015 rows=0 loops=230)
                -> Limit: 1 row(s) (cost=15.20..15.20 rows=1) (actual time=0.014..0.014
rows=0 loops=230)
                  -> Index lookup on <materialized_subquery> using
<auto_distinct_key> (input=Restaurants.style) (actual time=0.014..0.014 rows=0
loops=230)
                    -> Materialize with deduplication (cost=15.20..15.20 rows=16)
(actual time=3.082..3.082 rows=14 loops=1)
                      -> Index lookup on History using userID (userID=11) (cost=13.60
rows=16) (actual time=3.048..3.054 rows=16 loops=1)
                        ,
```

2. CREATE INDEX idx_style ON Restaurants(style);

```
'-> Table scan on <temporary> (cost=458.40..472.93 rows=965) (actual
time=4.718..4.722 rows=16 loops=1)
  -> Temporary table with deduplication (cost=458.38..458.38 rows=965) (actual
time=4.715..4.715 rows=16 loops=1)
    -> Nested loop inner join (cost=361.93 rows=965) (actual time=0.622..4.604
rows=42 loops=1)
      -> Table scan on Restaurants (cost=24.35 rows=236) (actual time=0.085..0.317
rows=236 loops=1)
        -> Filter: ((Favorites.userID = 11) or
<in_optimizer>(Restaurants.style,Restaurants.style in (select #2))) (cost=1.02 rows=4)
(actual time=0.018..0.018 rows=0 loops=236)
```


- > Index lookup on Favorites using restaurantName
(restaurantName=Restaurants.restaurantName) (cost=1.02 rows=4) (actual time=0.013..0.015 rows=4 loops=236)
- > Select #2 (subquery in condition; run only once)
- > Filter: ((Restaurants.style = `<materialized_subquery>`.input))
(cost=6.30..6.30 rows=1) (actual time=0.001..0.001 rows=0 loops=230)
- > Limit: 1 row(s) (cost=6.20..6.20 rows=1) (actual time=0.001..0.001 rows=0 loops=230)
- > Index lookup on <materialized_subquery> using
<auto_distinct_key> (input=Restaurants.style) (actual time=0.001..0.001 rows=0 loops=230)
- > Materialize with deduplication (cost=6.20..6.20 rows=16) (actual time=0.075..0.075 rows=14 loops=1)
- > Index lookup on History using userID (userID=11) (cost=4.60 rows=16) (actual time=0.055..0.059 rows=16 loops=1)

3. CREATE INDEX idx_price ON Restaurants(price)

- > Table scan on <temporary> (cost=458.40..472.93 rows=965) (actual time=2.925..2.928 rows=16 loops=1)
- > Temporary table with deduplication (cost=458.38..458.38 rows=965) (actual time=2.922..2.922 rows=16 loops=1)
- > Nested loop inner join (cost=361.93 rows=965) (actual time=0.355..2.865 rows=42 loops=1)
- > Table scan on Restaurants (cost=24.35 rows=236) (actual time=0.096..0.230 rows=236 loops=1)
- > Filter: ((Favorites.userID = 11) or
<in_optimizer>(Restaurants.style, Restaurants.style in (select #2))) (cost=1.02 rows=4) (actual time=0.011..0.011 rows=0 loops=236)
- > Index lookup on Favorites using restaurantName
(restaurantName=Restaurants.restaurantName) (cost=1.02 rows=4) (actual time=0.007..0.009 rows=4 loops=236)
- > Select #2 (subquery in condition; run only once)
- > Filter: ((Restaurants.style = `<materialized_subquery>`.input))
(cost=6.30..6.30 rows=1) (actual time=0.001..0.001 rows=0 loops=230)
- > Limit: 1 row(s) (cost=6.20..6.20 rows=1) (actual time=0.001..0.001 rows=0 loops=230)
- > Index lookup on <materialized_subquery> using
<auto_distinct_key> (input=Restaurants.style) (actual time=0.001..0.001 rows=0 loops=230)
- > Materialize with deduplication (cost=6.20..6.20 rows=16) (actual time=0.072..0.072 rows=14 loops=1)
- > Index lookup on History using userID (userID=11) (cost=4.60 rows=16) (actual time=0.055..0.058 rows=16 loops=1)

4. CREATE INDEX idx_zip ON Restaurants(zip)

- '-> Table scan on <temporary> (cost=458.40..472.93 rows=965) (actual time=5.218..5.223 rows=16 loops=1)
 - > Temporary table with deduplication (cost=458.38..458.38 rows=965) (actual time=5.215..5.215 rows=16 loops=1)
 - > Nested loop inner join (cost=361.93 rows=965) (actual time=0.462..5.121 rows=42 loops=1)
 - > Table scan on Restaurants (cost=24.35 rows=236) (actual time=0.111..0.437 rows=236 loops=1)
 - > Filter: ((Favorites.userID = 11) or <in_optimizer>(Restaurants.style, Restaurants.style in (select #2))) (cost=1.02 rows=4) (actual time=0.019..0.020 rows=0 loops=236)
 - > Index lookup on Favorites using restaurantName (restaurantName=Restaurants.restaurantName) (cost=1.02 rows=4) (actual time=0.013..0.015 rows=4 loops=236)
 - > Select #2 (subquery in condition; run only once)
 - > Filter: ((Restaurants.style = `<materialized_subquery>`.input)) (cost=6.30..6.30 rows=1) (actual time=0.002..0.002 rows=0 loops=230)
 - > Limit: 1 row(s) (cost=6.20..6.20 rows=1) (actual time=0.002..0.002 rows=0 loops=230)
 - > Index lookup on <materialized_subquery> using <auto_distinct_key> (input=Restaurants.style) (actual time=0.002..0.002 rows=0 loops=230)
 - > Materialize with deduplication (cost=6.20..6.20 rows=16) (actual time=0.090..0.090 rows=14 loops=1)
 - > Index lookup on History using userID (userID=11) (cost=4.60 rows=16) (actual time=0.071..0.076 rows=16 loops=1)