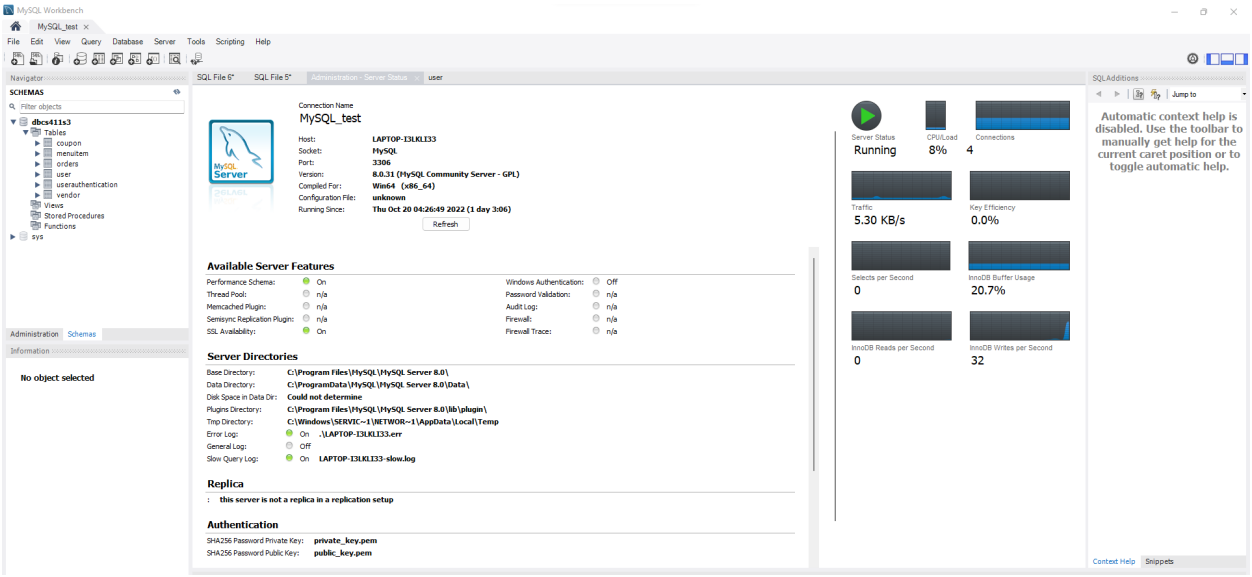
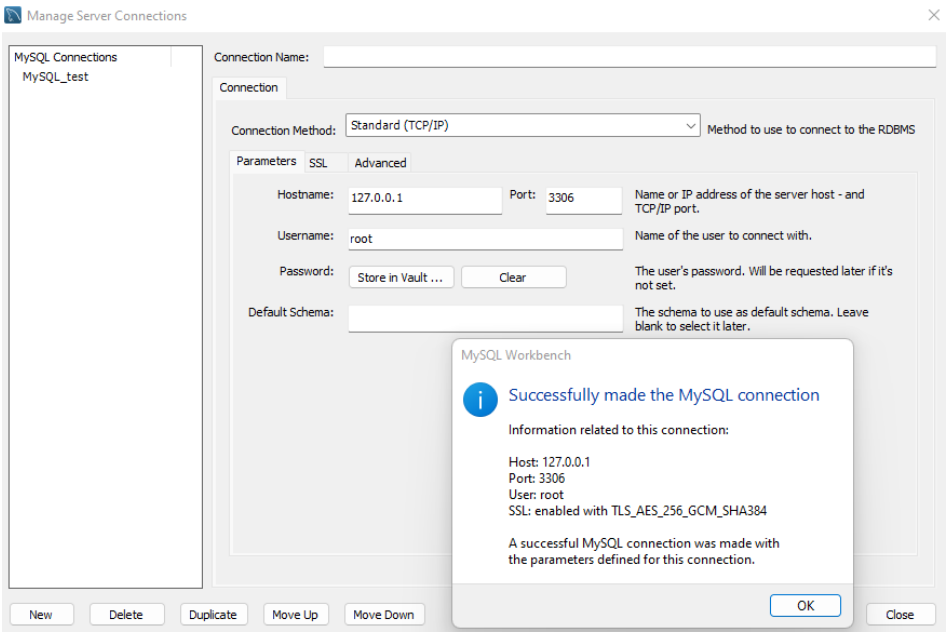
Project Track 1: Stage 3

Team 065

2. Database Implementation

2.1. The database was implemented locally. The connection is:





2.2. The DDL commands are:

```
CREATE TABLE User (
  UserId INT NOT NULL,
  FirstName VARCHAR(255) NOT NULL,
  LastName VARCHAR(255) NOT NULL,
  PhoneNumber VARCHAR(15) NOT NULL,
  Address VARCHAR(255) NOT NULL,
  PRIMARY KEY (UserId));
```

```
CREATE TABLE UserAuthentication (
  UserId INT NOT NULL,
  EmailAddress VARCHAR(255) NOT NULL,
  Password VARCHAR(255) NOT NULL,
  PRIMARY KEY (UserId));
```

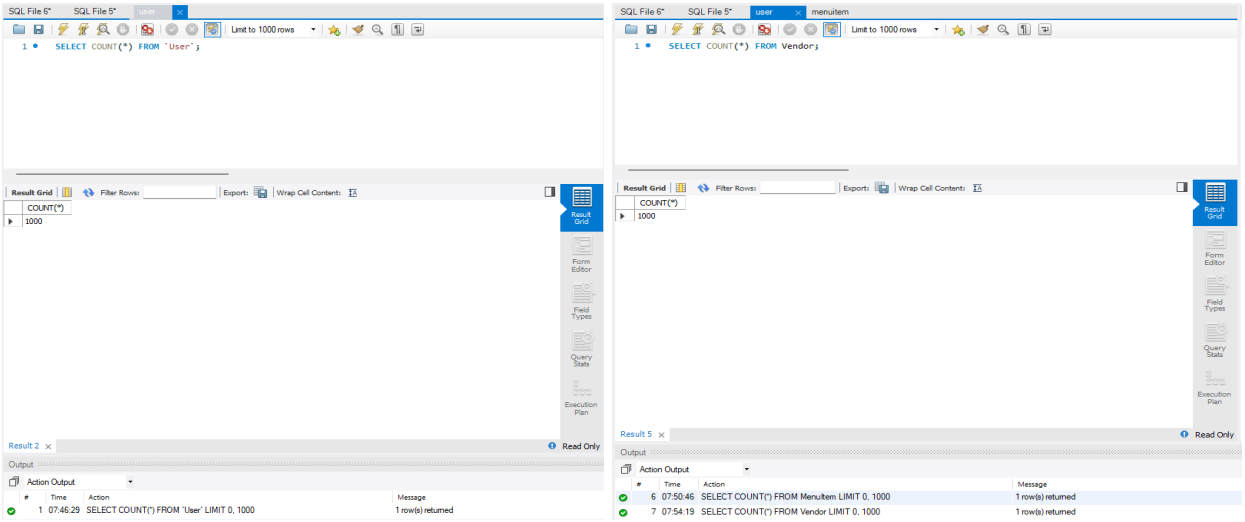
```
CREATE TABLE Vendor (
  VendorId INT NOT NULL,
  PhoneNumber VARCHAR(15) NOT NULL,
  Address VARCHAR(255) NOT NULL,
  FirstName VARCHAR(255) NOT NULL,
  LastName VARCHAR(255) NOT NULL,
  VendorName VARCHAR(255) NOT NULL,
  Type VARCHAR(255) NOT NULL,
  PRIMARY KEY (VendorId));
```

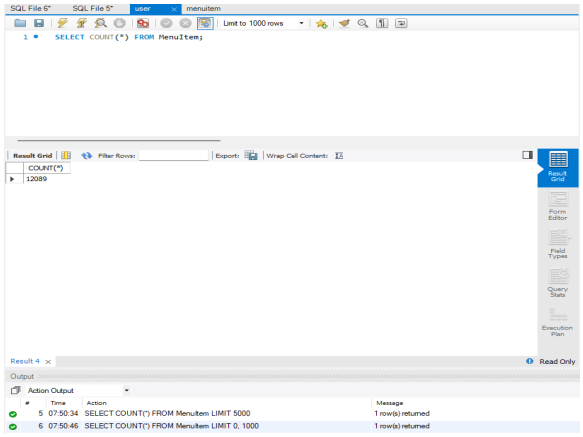
```
CREATE TABLE Coupon (
  CouponId INT NOT NULL,
  DiscountPercentage INT NOT NULL,
  Active BOOLEAN NOT NULL,
  UserId INT NULL,
  VendorId INT NULL,
   Foreign Key (UserId) References UserAuthentication(UserId),
   Foreign Key (VendorId) References Vendor(VendorId);
```

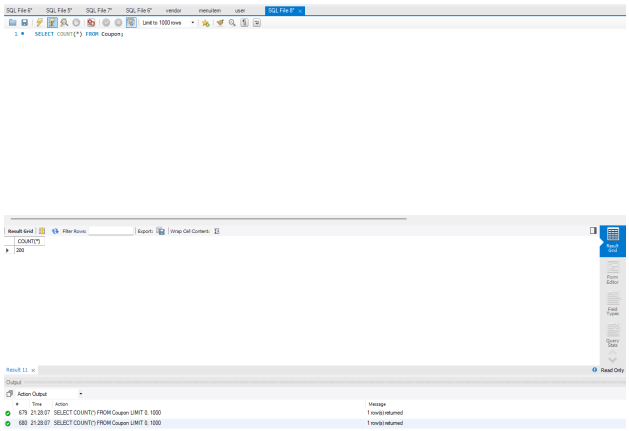
```
CREATE TABLE Orders (
  OrderId INT NOT NULL,
  VendorId INT NOT NULL,
  UserId INT NOT NULL,
  Item VARCHAR(255) NOT NULL,
  OrderDate VARCHAR(255) NOT NULL,
  Ratings DOUBLE NOT NULL,
  Total DOUBLE NOT NULL,
  TransactionId INT NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (TransactionId),
  FOREIGN KEY (UserId) REFERENCES User (UserId)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
    FOREIGN KEY (VendorId)
    REFERENCES Vendor (VendorId)
    ON DELETE CASCADE
    ON UPDATE CASCADE);
```

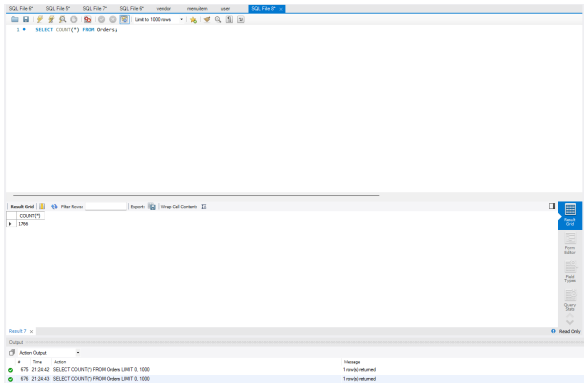
```
CREATE TABLE MenuItem (
  VendorId INT NOT NULL,
  ItemName VARCHAR(255) NOT NULL,
  Price DOUBLE NOT NULL,
  PRIMARY KEY (VendorId, ItemName),
    FOREIGN KEY (VendorId) REFERENCES Vendor(VendorId)
    ON DELETE CASCADE
    ON UPDATE CASCADE);
```

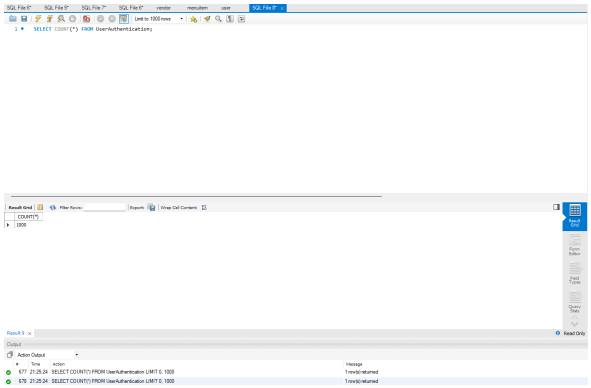
2.3. Three of the tables have at least 1000 rows, namely User, Vendor, and MenuItem. This is depicted below. Along with these tables, the counts of the rest of the tables are presented:



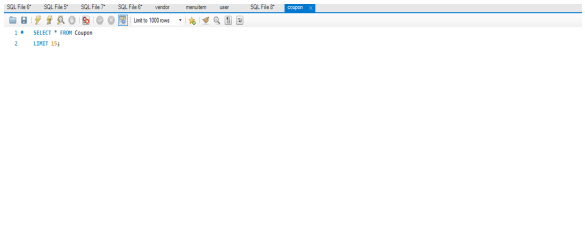


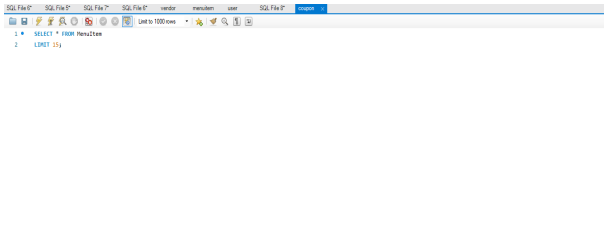


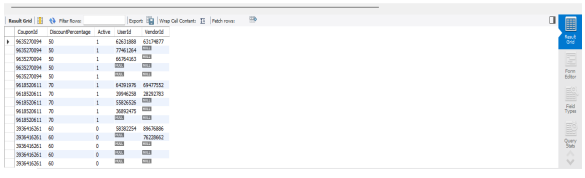


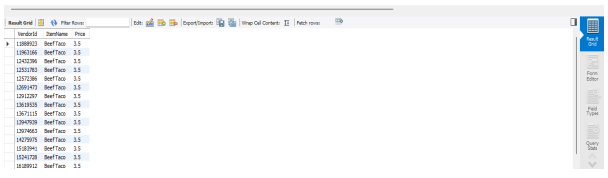


We also provide the first 15 rows for all tables:









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

OrderId	VendorId	UserId	Item	OrderDate	Ratings	Total	TransactionId
86303966	86165862	55948466	Risotto	2022-12-17	2.03	44.96	1
86303966	86165862	55948466	Gnocchi	2022-12-17	2.03	44.96	2
86303966	86165862	55948466	Lasagne	2022-12-17	2.03	44.96	3
86303966	86165862	55948466	Ribollita	2022-12-17	2.03	44.96	4
80494083	72199619	75944734	Butter Chicken	2022-03-07	4.27	39.97	5
80494083	72199619	75944734	Matar Paneer	2022-03-07	4.27	39.97	6
80494083	72199619	75944734	Tandoori Chicken	2022-03-07	4.27	39.97	7
31184019	28916447	47592165	Chicken Fried Rice	2021-06-17	2.78	57.1	8
31184019	28916447	47592165	Orange Chicken	2021-06-17	2.78	57.1	9
31184019	28916447	47592165	Kung Pao Chicken	2021-06-17	2.78	57.1	10
31184019	28916447	47592165	General Tso Tofu	2021-06-17	2.78	57.1	11
31184019	28916447	47592165	Szechuan Shrimp	2021-06-17	2.78	57.1	12
31184019	28916447	47592165	Ma Po Tofu	2021-06-17	2.78	57.1	13
79310448	98184216	78517394	Chicken Fried Rice	2018-08-14	3.17	35.6	14
79310448	98184216	78517394	Orange Chicken	2018-08-14	3.17	35.6	15

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

UserId	FirstName	LastName	PhoneNumber	Address
11229886	MAGDA	SHANNAN	1415597598	1769 W SUNNYSIDE AVE
11238832	FLORINDA	DIEDRA	8448574610	3308 N HARLEM AVE
11571321	SHANE	SHARLEEN	6515668233	3015 W LAWRENCE AVE
11614283	KELSIE	THEO	5427505111	1012 N WESTERN AVE
11619734	VERNETTA	SCOTT	2406234334	115 S CLINTON ST
11669837	KINDRA	BARBERA	1111278967	7575 S KOSTNER AVE
11978392	TRINITY	KATIA	1168827130	233 N MICHIGAN AVE
11979129	GIGI	TYLER	7857254444	6001 W ADDISON ST
12317642	VANNESSA	HEE	6825575500	3600-3602 N CLARK ST
12374623	CHLOE	SHAMIKA	2519805949	1050 N STATE ST
12765952	ALENE	MARVEL	3578534254	3310 W BRYN MAWR AVE
12995214	SAMMIE	LANETTE	3150830030	1470 N MILWAUKEE AVE
13182727	ZULMA	KYM	6970924704	131 N CLINTON ST
13191845	CAMILLA	LORENZA	7021954094	231 S LA SALLE ST
13297382	SHONNA	GAYLENE	5999211625	8903 S SOUTH CHICAG...

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

UserId	EmailAddress	Password
11229886	mash1288@gmail.com	8dcWgsVMr6
11238832	fldi3798@gmail.com	*$vjX1ckqO
11571321	shsh636@gmail.com	bvQW3K65pm
11614283	keth373@gmail.com	FaIUWeqgNO
11619734	vesc262@gmail.com	68PgHmqEKw
11669837	kiba244@gmail.com	v9Eeop1jsr
11978392	trka4754@gmail.com	iUus95BNjw
11979129	gity865@gmail.com	S&WRet6LUa
12317642	vahe1045@gmail.com	il%TwapLrz
12374623	chsh1594@gmail.com	Wx7YKBeC*s
12765952	alma1311@gmail.com	v8UrOZe0Vm
12995214	sala3098@gmail.com	wqX$eVB%Dn
13182727	zuky3436@gmail.com	pZJ0t!sBg&
13191845	calo3839@gmail.com	HJhq1TItLs
13297382	shga1373@gmail.com	U9jzmap2nX

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

VendorId	PhoneNumber	Address	FirstName	LastName	VendorName	Type
11187321	6256346234	2825 W CERMAK RD	WM	ARON	CHAN'S GARDEN CHINESE RESTAURANT	Indian
11291989	7344729758	4859 N MILWAUKEE AVE	LUIS	ELWOOD	KAWAYAN	Indian
11331766	5234858695	233 E ONTARIO ST	JULIA	TRUDY	IVY HOTEL	Indian
11373698	5366799647	1008 W 79TH ST	VIOLA	CLARISSA	BREWER COFFEE AND CUSTARD	Indian
11573981	9717147624	2311 W HOWARD ST	RICARDO	DANTE	KIZIN CREOLE	Chinese
11658965	3428626951	840 N ORLEANS ST	HOMER	RICH	TRIO	Greek
11679725	8279318397	1025 W LAWRENCE AVE	LENA	MARYLOU	DIB SUSHI BAR AND THAI CUISINE	Indian
11835862	4825167928	2331 N CALIFORNIA AVE	OMAR	SANDY	DON CHEMA	Indian
11888923	3463695417	8740 S LAFAYETTE AVE	JULIAN	RAYMUNDO	WENDY'S	Mexican
11932542	6535679238	1059-1065 W ADDISON ST	PHILLIP	XAVIER	CUBBY BEAR LOUNGE	Indian
11942589	6198475498	4048 S ARCHER AVE	SHERMAN	DINO	THE SUBMARINE PIER	Greek
11963166	7567295617	3518 W 25TH ST	HOWARD	JOAQUIN	TAQUERIA LOS BARRILITOS	Mexican
12179477	4375939467	3046-3050 W 63RD ST	MICHELE	SHAUNA	LA MIXTECA POBLANA 2 LLC	Chinese
12235931	1947385984	2302 W 95TH ST	LOLA	OPHELIA	CHIPOTLE MEXICAN GRILL	Indian
12242314	8231563847	1154-1156 N DEARBORN ST	SONIA	MALINDA	LA STORIA	Chinese

3. Advanced Queries

3.1. The first advanced query finds the users with the most and least expenditures on food, whose last name starts with A. It contains a SET operation and a JOIN command. This query can be modified to lookup users who match other criteria as well (e.g. users whose names contain different characters, or users with different phone number area codes), and therefore will be part of the final application.:

```
(SELECT Total, u.FirstName, u.LastName
FROM Orders NATURAL JOIN User u
WHERE Total = (SELECT MAX(Total) FROM Orders) AND u.LastName LIKE "A%"
)
UNION
(SELECT Total, u.FirstName, u.LastName
FROM Orders NATURAL JOIN User u
WHERE Total = (SELECT MIN(Total) FROM Orders) AND u.LastName LIKE "A%"
)
ORDER BY Total DESC;
```

The second advanced query returns the menu items which were ordered fewer times than the average, along with a vendor that cooked this item in at least one of the orders. It contains a JOIN operation and Aggregation via GROUP BY. This means that this query can help in knowing which menu items are ordered fewer times than the average, so that causes, or correlations, can be found for that. Therefore, this query will be a part of the final application, possibly adding/changing the extra information, which is the vendor name:

```
SELECT COUNT(*) as  freq, Item, VendorName FROM Orders NATURAL JOIN Vendor
GROUP BY Item HAVING freq <= (SELECT AVG(temp.num_count) FROM (SELECT COUNT(*) as num_count FROM Orders GROUP BY  Item) as temp)
ORDER BY Item;
```

3.2. The first 15 results of the first advanced query are the following. Because of the nature of the query, which looks specifically for last names starting with “A”, only two results are returned:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Total	FirstName	LastName
83.94	JOSETTE	ASHELY
5.95	FILOMENA	ALBERTHA

Result 66

Output

Action Output

#	Time	Action	Message
342	11:24:17	(SELECT Total, u.FirstName, u.LastName FROM Orders NATURAL JOIN User u WHERE Total = (SELECT MAX(Total) FROM Orders) AND u....	2 row(s) returned
343	11:24:38	(SELECT Total, u.FirstName, u.LastName FROM Orders NATURAL JOIN User u WHERE Total = (SELECT MAX(Total) FROM Orders) AND u....	2 row(s) returned

The first 15 results of the second advanced query are:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

freq	Item	VendorName
10	Arrosticini	EXPLORER CAFE
28	Beef Taco	MSI TRACK END
4	Chana Masala	ONE ELEVEN FOOD HALL
20	Chicken Burrito	MSI TRACK END
37	Chicken Korma	NAMI
9	Chicken Wings	LITTLE CAESARS
5	Chimichangas	LOS POLLITOS
41	Fried Chicken	EASTERN STYLE PIZZA II LT
19	Ma Po Tofu	Individual Home-Cook
13	Meatloaf	EASTERN STYLE PIZZA II LT
5	Mongolian Beef	CLEVER RABBIT
37	Papoutsakia	LITTLE CAESAR'S PIZZA
34	Pastitsio	LITTLE CAESAR'S PIZZA
37	Pizza	EASTERN STYLE PIZZA II LT
32	Ravioli	EXPLORER CAFE

Result 109

Output

Action Output

#	Time	Action
663	14:14:54	EXPLAIN ANALYZE SELECT COUNT(*) as freq, Item, VendorName FROM Orders NATURAL JOIN Vendor GROUP BY Item HAVING freq <...
664	14:15:01	SELECT COUNT(*) as freq, Item, VendorName FROM Orders NATURAL JOIN Vendor GROUP BY Item HAVING freq <= (SELECT AVG(temp...

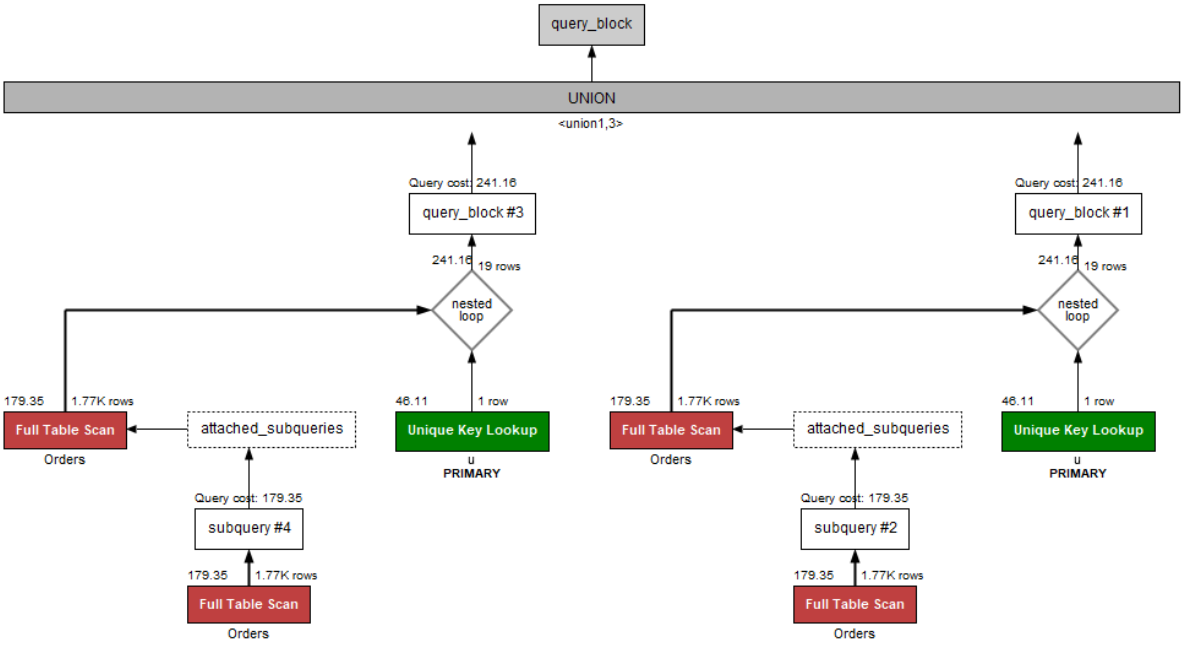
4. Indexing results

4.1. First advanced query:

```
CREATE INDEX idx_LastName ON User(LastName);
CREATE INDEX idx_Total ON Orders(Total);

EXPLAIN ANALYZE
(SELECT Total, u.FirstName, u.LastName
FROM Orders NATURAL JOIN User u
WHERE Total = (SELECT MAX(Total) FROM Orders) AND u.LastName LIKE "A%"
)
UNION
(SELECT Total, u.FirstName, u.LastName
FROM Orders NATURAL JOIN User u
WHERE Total = (SELECT MIN(Total) FROM Orders) AND u.LastName LIKE "A%"
)
ORDER BY Total DESC;
```

For the default Index, the tree structure is depicted below:



It can be seen that there are four full table scans on Orders which concern the variable Total. Moreover, there is a Unique Key Lookup that concerns the filtering of the user's last name. Therefore, three indexing schemas will be created: the first will be on the Total variables of the Orders table, the second on the LastName variable, and the third on both.

4.1.A) The Default Index schema results are shown below:

```
# NO EXTRA INDEX

-> Sort: Total DESC  (cost=513.93..513.93 rows=39) (actual time=1.717..1.717 rows=2 loops=1)
    -> Table scan on <union temporary>  (cost=486.32..489.23 rows=39) (actual time=1.708..1.708 rows=2 loops=1)
        -> Union materialize with deduplication  (cost=486.24..486.24 rows=39) (actual time=1.707..1.707 rows=2 loops=1)
            -> Nested loop inner join  (cost=241.16 rows=20) (actual time=0.648..0.910 rows=6 loops=1)
                -> Filter: (orders.Total = (select #2))  (cost=179.35 rows=177) (actual time=0.560..0.855 rows=72 loops=1)
                    -> Table scan on Orders  (cost=179.35 rows=1766) (actual time=0.041..0.375 rows=1766 loops=1)
                    -> Select #2 (subquery in condition; run only once)
                        -> Aggregate: max(orders.Total)  (cost=355.95 rows=1) (actual time=0.388..0.388 rows=1 loops=1)
                            -> Table scan on Orders  (cost=179.35 rows=1766) (actual time=0.009..0.315 rows=1766 loops=1)
                -> Filter: (u.LastName like 'A%')  (cost=0.25 rows=0.1) (actual time=0.001..0.001 rows=0 loops=72)
                    -> Single-row index lookup on u using PRIMARY (UserId=orders.UserId)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=72)
            -> Nested loop inner join  (cost=241.16 rows=20) (actual time=0.518..0.787 rows=1 loops=1)
                -> Filter: (orders.Total = (select #4))  (cost=179.35 rows=177) (actual time=0.423..0.771 rows=10 loops=1)
                    -> Table scan on Orders  (cost=179.35 rows=1766) (actual time=0.012..0.350 rows=1766 loops=1)
                    -> Select #4 (subquery in condition; run only once)
                        -> Aggregate: min(orders.Total)  (cost=355.95 rows=1) (actual time=0.342..0.342 rows=1 loops=1)
                            -> Table scan on Orders  (cost=179.35 rows=1766) (actual time=0.008..0.274 rows=1766 loops=1)
                -> Filter: (u.LastName like 'A%')  (cost=0.25 rows=0.1) (actual time=0.001..0.002 rows=0 loops=10)
                    -> Single-row index lookup on u using PRIMARY (UserId=orders.UserId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=10)
```

These results will be compared to the index-schemas that follow.

4.1.B) Index on Last Name:

```
## LAST NAME

-> Sort: Total DESC  (cost=433.20..433.20 rows=81) (actual time=1.447..1.447 rows=2 loops=1)
    -> Table scan on <union temporary>  (cost=369.96..373.43 rows=81) (actual time=1.438..1.438 rows=2 loops=1)
        -> Union materialize with deduplication  (cost=369.92..369.92 rows=81) (actual time=1.437..1.437 rows=2 loops=1)
            -> Nested loop inner join  (cost=180.89 rows=41) (actual time=0.697..0.766 rows=6 loops=1)
                -> Index range scan on u using idx_LastName over ('A' <= LastName <= 'A'), with index condition: (u.LastName like 'A%')  (cost=38.51 rows=85) (actual time=
0.013..0.137 rows=85 loops=1)
                -> Filter: (orders.Total = (select #2))  (cost=1.20 rows=0.5) (actual time=0.007..0.007 rows=0 loops=85)
                    -> Index lookup on Orders using UserId_order_idx (UserId=u.UserId)  (cost=1.20 rows=5) (actual time=0.003..0.003 rows=2 loops=85)
                    -> Select #2 (subquery in condition; run only once)
                        -> Aggregate: max(orders.Total)  (cost=355.95 rows=1) (actual time=0.339..0.339 rows=1 loops=1)
                            -> Table scan on Orders  (cost=179.35 rows=1766) (actual time=0.012..0.270 rows=1766 loops=1)
            -> Nested loop inner join  (cost=180.89 rows=41) (actual time=0.390..0.663 rows=1 loops=1)
                -> Index range scan on u using idx_LastName over ('A' <= LastName <= 'A'), with index condition: (u.LastName like 'A%')  (cost=38.51 rows=85) (actual time=
0.005..0.098 rows=85 loops=1)
                -> Filter: (orders.Total = (select #4))  (cost=1.20 rows=0.5) (actual time=0.007..0.007 rows=0 loops=85)
                    -> Index lookup on Orders using UserId_order_idx (UserId=u.UserId)  (cost=1.20 rows=5) (actual time=0.002..0.003 rows=2 loops=85)
                    -> Select #4 (subquery in condition; run only once)
                        -> Aggregate: min(orders.Total)  (cost=355.95 rows=1) (actual time=0.328..0.328 rows=1 loops=1)
                            -> Table scan on Orders  (cost=179.35 rows=1766) (actual time=0.008..0.275 rows=1766 loops=1)
```

It can be seen that the actual times and costs of the operation are lower or similar to indexing schema A). The index helped with that since as seen in the report, some lines changed after using

the index, e.g. the fifth line in A) does not use an index, but in B) it does, i.e. it becomes an index range scan. However, the full table scans on Orders still exist.

4.1.C) Index on Total:

```
## TOTAL
-> Sort: Total DESC  (cost=54.99..54.99 rows=9) (actual time=0.199..0.199 rows=2 loops=1)
    -> Table scan on <union temporary>  (cost=48.85..51.17 rows=9) (actual time=0.189..0.190 rows=2 loops=1)
        -> Union materialize with deduplication  (cost=48.56..48.56 rows=9) (actual time=0.188..0.188 rows=2 loops=1)
            -> Nested loop inner join  (cost=40.65 rows=8) (actual time=0.102..0.146 rows=6 loops=1)
                -> Filter: (orders.Total = (select #2))  (cost=15.45 rows=72) (actual time=0.078..0.099 rows=72 loops=1)
                    -> Index lookup on Orders using idx_Total (Total=(select #2))  (cost=15.45 rows=72) (actual time=0.076..0.093 rows=72 loops=1)
                    -> Select #2 (subquery in condition; run only once)
                        -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
                -> Filter: (u.LastName like 'A%')  (cost=0.25 rows=0.1) (actual time=0.001..0.001 rows=0 loops=72)
                    -> Single-row index lookup on u using PRIMARY (UserId=orders.UserId)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=72)
            -> Nested loop inner join  (cost=7.00 rows=1) (actual time=0.025..0.033 rows=1 loops=1)
                -> Filter: (orders.Total = (select #4))  (cost=3.50 rows=10) (actual time=0.015..0.017 rows=10 loops=1)
                    -> Index lookup on Orders using idx_Total (Total=(select #4))  (cost=3.50 rows=10) (actual time=0.015..0.016 rows=10 loops=1)
                    -> Select #4 (subquery in condition; run only once)
                        -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
                -> Filter: (u.LastName like 'A%')  (cost=0.25 rows=0.1) (actual time=0.001..0.001 rows=0 loops=10)
                    -> Single-row index lookup on u using PRIMARY (UserId=orders.UserId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=10)
```

By incorporating an index on the Total variable, the costs and and actual times of the operations have dropped considerably when compared to A). It is noted that the “Table scan on Orders” now became “Index lookup on Orders” using the total-variable index. This is what gives the significant improvement in costs and actual times.

4.1.D) Index on both Total and LastName:

```
## TOTAL PLUS LAST NAME
-> Sort: Total DESC  (cost=53.57..53.57 rows=7) (actual time=0.237..0.237 rows=2 loops=1)
    -> Table scan on <union temporary>  (cost=48.72..50.92 rows=7) (actual time=0.227..0.228 rows=2 loops=1)
        -> Union materialize with deduplication  (cost=48.35..48.35 rows=7) (actual time=0.226..0.226 rows=2 loops=1)
            -> Nested loop inner join  (cost=40.65 rows=6) (actual time=0.135..0.182 rows=6 loops=1)
                -> Filter: (orders.Total = (select #2))  (cost=15.45 rows=72) (actual time=0.111..0.134 rows=72 loops=1)
                    -> Index lookup on Orders using idx_Total (Total=(select #2))  (cost=15.45 rows=72) (actual time=0.110..0.128 rows=72 loops=1)
                    -> Select #2 (subquery in condition; run only once)
                        -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
                -> Filter: (u.LastName like 'A%')  (cost=0.25 rows=0.09) (actual time=0.001..0.001 rows=0 loops=72)
                    -> Single-row index lookup on u using PRIMARY (UserId=orders.UserId)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=72)
            -> Nested loop inner join  (cost=7.00 rows=1) (actual time=0.027..0.036 rows=1 loops=1)
                -> Filter: (orders.Total = (select #4))  (cost=3.50 rows=10) (actual time=0.017..0.020 rows=10 loops=1)
                    -> Index lookup on Orders using idx_Total (Total=(select #4))  (cost=3.50 rows=10) (actual time=0.017..0.019 rows=10 loops=1)
                    -> Select #4 (subquery in condition; run only once)
                        -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
                -> Filter: (u.LastName like 'A%')  (cost=0.25 rows=0.09) (actual time=0.001..0.001 rows=0 loops=10)
                    -> Single-row index lookup on u using PRIMARY (UserId=orders.UserId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=10)
```

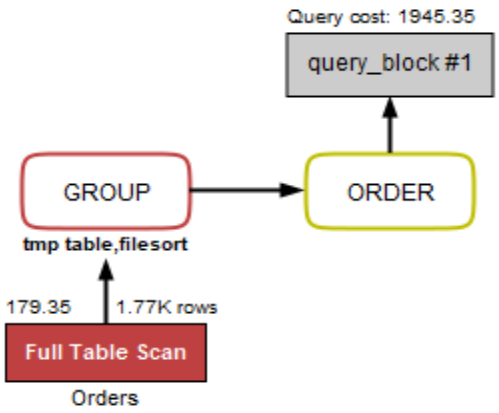
By incorporating indices on both the Total and LastName variables, the speedup is not significantly better than C) which uses an index only on the Total variable. This is because the main reason for slow operation was the four full table scans on the Total variable.

4.2. Second advanced query:

```
CREATE INDEX idx_item ON Orders(item);
CREATE INDEX idx_VendorName ON Vendor(VendorName);

EXPLAIN ANALYZE SELECT COUNT(*) as  freq, Item, VendorName FROM Orders NATURAL JOIN Vendor
GROUP BY Item HAVING freq <= (SELECT AVG(temp.num_count) FROM (SELECT COUNT(*) as num_count FROM Orders GROUP BY  Item) as temp)
ORDER BY Item;
```

For the default Index, the tree structure is depicted below:



It can be seen that there is one full table scan. The only variable that is associated with the scan and is not a primary key, i.e., it does not have an index already, is Item. Therefore an index on Item will be used. Moreover, an index on Vendor Name will be used since Vendor Name is another attribute without an index already. Lastly, indices on both of the previous variables will be used.

4.2.A) The Default Index schema results are shown below:

```
## NO EXTRA INDEX
-> Sort: orders.Item  (actual time=3.328..3.329 rows=23 loops=1)
    -> Filter: (freq <= (select #2))  (actual time=3.301..3.307 rows=23 loops=1)
        -> Table scan on <temporary>  (actual time=2.213..2.218 rows=43 loops=1)
            -> Aggregate using temporary table  (actual time=2.212..2.212 rows=43 loops=1)
                -> Nested loop inner join  (cost=804.88 rows=1766) (actual time=0.044..1.395 rows=1766 loops=1)
                    -> Table scan on Orders  (cost=186.78 rows=1766) (actual time=0.033..0.531 rows=1766 loops=1)
                    -> Single-row index lookup on Vendor using PRIMARY (VendorId=orders.VendorId)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=1766)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(temp.num_count)  (cost=2.50..2.50 rows=1) (actual time=1.067..1.067 rows=1 loops=1)
                -> Table scan on temp  (cost=2.50..2.50 rows=0) (actual time=1.057..1.060 rows=43 loops=1)
                    -> Materialize  (cost=0.00..0.00 rows=0) (actual time=1.057..1.057 rows=43 loops=1)
                        -> Table scan on <temporary>  (actual time=1.044..1.048 rows=43 loops=1)
                            -> Aggregate using temporary table  (actual time=1.044..1.044 rows=43 loops=1)
                                -> Table scan on Orders  (cost=186.78 rows=1766) (actual time=0.015..0.429 rows=1766 loops=1)
```

These results will be compared to the index-schemas that follow.

4.2.B) Index on Item:

```
## ITEM
-> Sort: orders.Item  (actual time=2.707..2.707 rows=23 loops=1)
    -> Filter: (freq <= (select #2))  (actual time=2.684..2.690 rows=23 loops=1)
        -> Table scan on <temporary>  (actual time=2.147..2.152 rows=43 loops=1)
            -> Aggregate using temporary table  (actual time=2.147..2.147 rows=43 loops=1)
                -> Nested loop inner join  (cost=804.88 rows=1766) (actual time=0.045..1.339 rows=1766 loops=1)
                    -> Table scan on Orders  (cost=186.78 rows=1766) (actual time=0.034..0.479 rows=1766 loops=1)
                    -> Single-row index lookup on Vendor using PRIMARY (VendorId=orders.VendorId)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=1766)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(temp.num_count)  (cost=741.15..741.15 rows=1) (actual time=0.520..0.520 rows=1 loops=1)
                -> Table scan on temp  (cost=539.99..564.55 rows=1766) (actual time=0.513..0.516 rows=43 loops=1)
                    -> Materialize  (cost=539.98..539.98 rows=1766) (actual time=0.512..0.512 rows=43 loops=1)
                        -> Group aggregate: count(0)  (cost=363.38 rows=1766) (actual time=0.020..0.504 rows=43 loops=1)
                            -> Covering index scan on Orders using idx_item  (cost=186.78 rows=1766) (actual time=0.013..0.273 rows=1766 loops=1)
```

It can be observed that by using an index on Item, the table scan on Orders (the last line in A) now successfully uses the index to provide faster results. This is associated with the observed decrease in costs in some of the query-execution stages as well.

4.2.C) Index on Vendor Name:

```
## VENDOR NAME
-> Sort: orders.Item  (actual time=3.302..3.304 rows=23 loops=1)
    -> Filter: (freq <= (select #2))  (actual time=3.274..3.282 rows=23 loops=1)
        -> Table scan on <temporary>  (actual time=2.165..2.172 rows=43 loops=1)
            -> Aggregate using temporary table  (actual time=2.165..2.165 rows=43 loops=1)
                -> Nested loop inner join  (cost=804.88 rows=1766) (actual time=0.044..1.381 rows=1766 loops=1)
                    -> Table scan on Orders  (cost=186.78 rows=1766) (actual time=0.034..0.521 rows=1766 loops=1)
                    -> Single-row index lookup on Vendor using PRIMARY (VendorId=orders.VendorId)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=1766)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(temp.num_count)  (cost=2.50..2.50 rows=1) (actual time=1.090..1.090 rows=1 loops=1)
                -> Table scan on temp  (cost=2.50..2.50 rows=0) (actual time=1.080..1.083 rows=43 loops=1)
                    -> Materialize  (cost=0.00..0.00 rows=0) (actual time=1.080..1.080 rows=43 loops=1)
                        -> Table scan on <temporary>  (actual time=1.069..1.073 rows=43 loops=1)
                            -> Aggregate using temporary table  (actual time=1.065..1.065 rows=43 loops=1)
                                -> Table scan on Orders  (cost=186.78 rows=1766) (actual time=0.013..0.433 rows=1766 loops=1)
```

By indexing Vendor Name no significant speedup is increased. This is because this variable only appears in the SELECT part of the query. By the time of the SELECT statements of this particular query, the operations that define the selected Vendor Name values (and the rest of the SELECT attributes) are completed, therefore the scan on the Vendor Name variables is limited.

4.2.D) Index on both Item and VendorName:

```
## BOTH ITEM AND VENDOR NAME
-> Sort: orders.Item  (actual time=2.748..2.749 rows=23 loops=1)
    -> Filter: (freq <= (select #2))  (actual time=2.725..2.732 rows=23 loops=1)
        -> Table scan on <temporary>  (actual time=2.174..2.178 rows=43 loops=1)
            -> Aggregate using temporary table  (actual time=2.173..2.173 rows=43 loops=1)
                -> Nested loop inner join  (cost=804.88 rows=1766) (actual time=0.048..1.372 rows=1766 loops=1)
                    -> Table scan on Orders  (cost=186.78 rows=1766) (actual time=0.035..0.474 rows=1766 loops=1)
                    -> Single-row index lookup on Vendor using PRIMARY (VendorId=orders.VendorId)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=1766)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(temp.num_count)  (cost=741.15..741.15 rows=1) (actual time=0.535..0.535 rows=1 loops=1)
                -> Table scan on temp  (cost=539.99..564.55 rows=1766) (actual time=0.528..0.531 rows=43 loops=1)
                    -> Materialize  (cost=539.98..539.98 rows=1766) (actual time=0.527..0.527 rows=43 loops=1)
                        -> Group aggregate: count(0)  (cost=363.38 rows=1766) (actual time=0.020..0.517 rows=43 loops=1)
                            -> Covering index scan on Orders using idx_item  (cost=186.78 rows=1766) (actual time=0.013..0.283 rows=1766 loops=1)
```

By indexing on both Item and VendorName the speed is similar to that of B), i.e., the speedup of the index on Item alone. This is directly related to the explanation of 4.2.C), i.e., to the fact that indexing on VendorName does not yield any increase in query performance.