UB Lyft

Sai Chandra Rachiraju
Pheonix
University at Buffalo
Buffalo, USA
srachira, srachira@buffalo.edu

Venkat Kaushik Vadlamudi
Pheonix
vvadlamu@buffalo.edu
Buffalo, USA
vvadlamu, vvadlamu@buffalo.edu

Indeevara Kodam
Pheonix
University at Buffalo
Buffalo, USA
indeevar, indeevar@buffalo.edu

***Abstract*—UBLyft is an imaginary idea to replace the existing Stampede with an imaginary cab system inside of UB. (But the data is made up of all the countries and locations).**

I. Introduction

There would be riders looking for cab services to be booked, cab drivers waiting for a booking. Riders can pay through Cash/Card or Bitcoin. Drivers have to register themselves with their valid license id and would have a rating for their ride performance.

Bookings would be mocked with riders mapped to drivers, their location, the payment done, the driver rating and the rider rating. Payments are also recorded in the system in a different table, so that we can analyze the revenue our company gets in a location,country and city.

The scope for phase 1 is limited to utmost 6-8 tables, and down the lane more features such as ride sharing, dynamic pricing and ride support can be provided. None of these can be easily managed through excel. Cross table queries need to be performed that calculates demand and supply, costs and profits, which cannot be done using excel sheets. The number of users are limited to 1000 as of now,but can be scaled infinitely to accommodate more user data, a DB can easily do this.

II. Target User and Access

The database would be consumed by a ride booking UI through authenticated APIs. Different tables would have its corresponding role access. Customer support teams can access the rides, riders and providers tables.Rider feedback and provider feedback can be used by Customer support and Escalation teams. Regional admins can access Providers in a location or city. Country admins can access by country.

Payments table would only be viewed by raising a privilege request access, by the customer support in case of payment escalations. Only view access has to be given, hiding the card details and codes, with access to base_fare,surge_fare,total_amount and mode_of_payment.

A user can access providers in his/her location, reviews of the existing rides and their respective trip details from the trips database.

Table Structure and Details

Relationships

Relationships explained in chronology- First we have created a Rider table with the following schema - id - Not null, primary key of this table

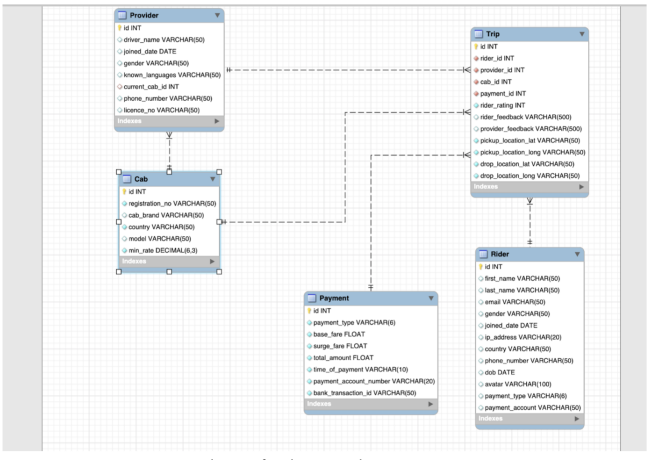

Fig. 1. ER Diagram

first_name - Mandatory first_name which is not null of a rider (customer) with 50 as its limit. Also allowed varchar to support funny names.

last_name - Mandatory ,last name which is not null of a rider(customer) with 50 characters as limit.Also allowed varchar to support funny names.

email - Can be empty, if the customer doesn't want to share his email. Limit 50 varchar.

gender - To support communities of different orientations, all types of genders are supported with a 50 varchar limit. If someone doesn't want to share this information, we support that too by making it a non not null field.

joined_date - Date type field which stores the onboarded date of a customer, this can be automatically generated by the API. Default date is today's date.

ip_address IP Address from where the user has signed up, this can be used to track users in a location and resource allocations. Can be null, but this is generally auto generated during the server request. User consent is required hence not a mandatory field.

country Country of the customer, so that customer analytics can be drawn. User consent is required hence not a mandatory field.

phone_number Mandatory field, so as to ensure customer safety, which is a varchar(50).

dob - Date type field, which can be null and needs customer consent.
avatar Random image URL of the user generated as of now which is not mandatory with 100 as character limit.
payment_type Default ‘cash’ while it can take ‘Credit’,’Debt’ Data type is varchar(6)

Details such as first name, last name, email, gender, joined_date, ip_address, country, phone_number, dob, avatar, payment_type and payment_account.

```
CREATE TABLE IF NOT EXISTS Rider (
    id INT NOT NULL UNIQUE PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(50),
    gender VARCHAR(50),
    joined_date DATE DEFAULT (CURRENT_DATE),
    ip_address VARCHAR(20),
    country VARCHAR(50),
    phone_number VARCHAR(50) NOT NULL,
    dob DATE,
    avatar VARCHAR(100),
    payment_type VARCHAR(6) DEFAULT 'cash'
);
```

Fig. 2. Rider Schema

The table is filled with 1000 fake people with fake details and 10% of the data which allow null values are filled with null, so as to replicate a real life database and to learn exception handling.

Once users are mocked in the database, we created a Cab table where we store the registration_number, cab brand, country, model and the minimum rate of the operation.

```
CREATE TABLE IF NOT EXISTS Cab (
    id INT NOT NULL UNIQUE PRIMARY KEY,
    registration_no VARCHAR(50) NOT NULL UNIQUE,
    cab_brand VARCHAR(50),
    country VARCHAR(50) not null,
    model VARCHAR(50),
    min_rate NUMERIC(6,3) NOT NULL
);
```

Fig. 3. Cab Schema

id - Not null primary key for this table
registration_no Cabs registration number uniquely used to identify the vehicle. This is a mandatory field and can take varchar upto 50.
cab_brand Not a mandatory field which can take a max varchar of 50.
country Country where the cab is licensed to operate, which is mandatory.
model Model of the cab to display it to the user, when he requests for a ride. Not mandatory.
min_rate The minimum rate of this car, so that it doesn’t have a booking level below this price. Also this field is mandatory.

Once a provider onboards his cab details first including his/her license number and registration details, we then collect provider details that are connected to the Cab table by current_cab_id FOREIGN KEY. Any delete on current_cab_id from Cab table is Cascaded into this table. For providers privacy, a driver might request to delete his data and hence cascade is used.

```
CREATE TABLE IF NOT EXISTS Provider (
    id INT NOT NULL UNIQUE PRIMARY KEY,
    driver_name VARCHAR(50) NOT NULL,
    joined_date DATE NOT NULL DEFAULT (CURRENT_DATE),
    gender VARCHAR(50) NOT NULL,
    known_languages VARCHAR(50),
    current_cab_id INT NOT NULL,
    phone_number VARCHAR(50) NOT NULL,
    licence_no VARCHAR(50) NOT NULL,
    FOREIGN KEY (current_cab_id) REFERENCES Cab(id) ON DELETE CASCADE
);
```

Fig. 4. Provider Schema

id - Primary key of this table which cannot be null
driver_name Mandatory so that a rider knows the providers name
joined_date Date type field which stores the onboarded date of a provider, this can be automatically generated by the API. Default date is today’s date.
gender To support communities of different orientations, all types of genders are supported with a 50 varchar limit. If someone doesn’t want to share this information, we support that too by making it a non not null field.
known_languages All languages known by the provider, so that it’s useful for the rider to communicate. Not mandatory.
current_cab_id The id of the cab present in the cab table, which is now used by this provider. This is connected to the Cab table by Foreign key reference.
phone_number Mandatory field, so as to ensure provider safety, which is a varchar(50).
licence_no The license number of the provider, so that we can get a history of their driving skills. Mandatory field.

```
CREATE TABLE IF NOT EXISTS Payment (
    id int not null unique primary key,
    payment_type VARCHAR(6) not null,
    base_fare float not null,
    surge_fare float not null,
    total_amount float not null,
    time_of_payment varchar(10) not null,
    payment_account_number VARCHAR(20) not null,
    bank_transaction_id VARCHAR(50) not null
);
```

Fig. 5. Payment Schema

id Not null primary key of every payment
payment_type
The payment type used by the rider. Credit/Debit/Cash/Bitcoin
base_fare
Base fare which doesn’t include surcharge, this can be used to calculate profits/loss. This is a mandatory field
surge_fee Providers are incentivised when they ride late nights and on special demand situations. Hence the surge_fee, which is a mandatory field.
total_amount Sum of base_fare and surge_fee, this is the fee paid by the customer and is a mandatory field in the table.
time_of_payment Used to record when the transaction has occurred, so as to prevent any transaction disputes. Mandatory

field.
payment_account_number Account number from where the payment has arrived for example the credit card of the customer.
bank_transaction_id The payment ID which is generated by the payment gateway and is used for settling transactions incase of a dispute and hence is mandatory.

```
CREATE TABLE IF NOT EXISTS Trip (
    id int not null unique primary key,
    rider_id int not null,
    provider_id int not null,
    cab_id int not null,
    payment_id int not null,
    rider_rating int not null,
    rider_feedback varchar(500),
    provider_feedback varchar(500),
    pickup_location_lat varchar(50) not null,
    pickup_location_long varchar(50) not null,
    drop_location_lat varchar(50) not null,
    drop_location_long varchar(50) not null,
    FOREIGN KEY (rider_id) REFERENCES Rider(id) ON DELETE CASCADE,
    FOREIGN KEY (provider_id) REFERENCES Provider(id) ON DELETE CASCADE,
    FOREIGN KEY (payment_id) REFERENCES Payment(id) ON DELETE CASCADE,
    FOREIGN KEY (cab_id) REFERENCES Cab(id) ON DELETE CASCADE
);
```

Fig. 6. Trip Schema

payment_id The payment id which can be referenced from the Payment Table. This is a mandatory field to know the payment details
rider_rating Feedback rating provided by the rider, which is mandated, so that drivers are assessed. This is an int field
provider_rating Feedback rating provided by the driver, which is mandated, so that riders are assessed. This is an integer field.
rider_feedback Text field to record riders feedback which can be analyzed to improve the platform. Max of 500 characters are supported and this is not mandatory.
provider_feedback Text field to drivers riders feedback which can be analyzed to improve the platform. Max of 500 characters are supported and this is not mandatory.
pickup_location_lat Latitude of the pick up where the ride started. This is mandatory and supports 50 varchar.
pickup_location_long Longitude of the pick up where the ride started. This is mandatory and supports 50 varchar.
drop_location_lat Latitude of the drop where the ride ended. This is mandatory and supports 50 varchar.
drop_location_long Longitude of the drop where the ride ended. This is mandatory and supports 50 varchar.
When a rider or a driver requests to delete his information from the system for privacy concerns, we cascade the delete from the parent table to the Trip Schema too.

Every Trip is done by a rider and a provider, which are referenced from their tables and stored in Trip by a Foreign key connection. Rider_id, provider_id, payment_id and cab_id are the relations from other tables.

Normalization and Functional Dependency

Rider (id, first_name, last_name,email,gender,joined_date, _address,country,phone_number,dob,avatar, payment_type,payment_account)

Functional dependencies in Rider relation: Id ->first_name, Id ->last_name, Id ->gender, Id ->joined_date, Id -> ip_address, Id ->country, Id ->phone_number, Id ->dob, Id ->avatar, Id ->payment_type, Id ->payment_account

Provider:(id,driver_name,joined_date,gender,known_languages,current_c phone_number,licence_number).

Functional dependencies in Provider relation: Id ->driver_name Id ->joined_date Id ->gender Id ->known_languages Id ->current_cab_id Id ->phone_number Id ->licence_number.

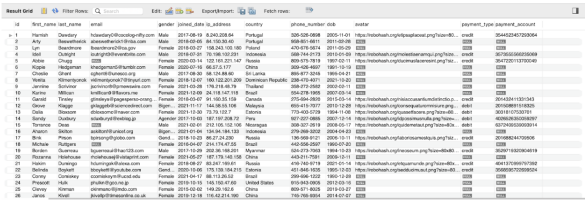

Fig. 7. Rider Table

id	registration_...	cab_brand	country	model	min_rate
1	6256049835	Isuzu	Argentina	Axiom	43.000
2	9932266684	Infiniti	Czech Republic	M	24.000
3	5533366389	Porsche	Russia	Boxster	8.000
4	4437614827	Toyota	Chile	Matrix	65.000
5	3151556673	NULL	France	NULL	35.000
6	0505399607	Chevrolet	China	2500	54.000
7	3768484352	GMC	Japan	1500	67.000
8	6547389927	Toyota	Croatia	Camry Hybrid	16.000
9	5769520901	Mazda	Russia	626	19.000
10	9157440823	Acura	Philippines	TSX	46.000
11	7910233879	Daewoo	Taiwan	Nubira	3.000
12	0020316415	Mercury	Palestinian Ter...	Cougar	10.000
13	6782620791	Buick	Thailand	LeSabre	97.000
14	7029433170	Mazda	Kosovo	929	22.000
15	1241513745	Dodge	Japan	Ram 1500	1.000
16	4319046520	Hummer	Portugal	H1	81.000
17	3324394811	Mitsubishi	Czech Republic	Eclipse	31.000
18	0713329275	Mazda	Moldova	RX-7	45.000

Fig. 8. Cabs Table

Cab(id, registration_no, cab_brand, country, model, min_rate)
Functional dependencies in Cab relation: Id ->registration_no Id ->cab_brand Id ->country Id ->model Id ->min_rate

id	payment_type	base_fare	surge_fare	total_amou...	time_of_payme...	payment_account_num...	bank_transaction_id
1	debit	5914.93	7359	13273.9	7:08 PM	3531449707501962	9535281922
2	credit	5721.04	206	5927.04	5:37 PM	3580549331283760	2268757118
3	credit	7065.4	2348	9413.4	9:13 AM	5482752723319697	7719697570
4	debit	5495.06	3322	8817.06	2:50 AM	357890003878278	5437254081
5	credit	9536.91	9452	18988.9	8:41 PM	560224138659515 95	5280561486
6	credit	8042.23	9493	17535.2	5:40 AM	589377816685135531	0339095005
7	debit	9282.37	988	10270.4	3:13 AM	633319159006323222	8716215626
8	credit	8915.31	2985	11900.3	9:52 AM	560222617391952286	3464158069
9	debit	2742.62	3458	6200.62	9:26 PM	503856815997955 79	5317791728
10	debit	8483.31	8223	16706.3	12:35 PM	357897007416530 9	2509687429
11	debit	1138.41	7394	8532.41	7:09 AM	559901821410860 5	3780945360
12	debit	3955.77	1720	5675.77	6:28 AM	374283667116172	2564611486
13	debit	2863.38	5682	8545.38	9:38 PM	353572655476704 4	1747333776
14	credit	580.24	53	633.24	6:59 AM	404159706039367 3	2442364234
15	debit	3506.41	622	4128.41	5:23 PM	357380414386964 8	4552477441

Fig. 9. Payments Table

Payment (id, payment_type, base_fare,surge_fare,total_amount, ,time_of_payment,payment_account_number, ,bank_transaction_id)

Functional dependencies in Payment relation: Id ->payment_type Id ->base_fare Id ->surge_fare

Id ->total_amount Id ->time_of_payment Id ->payment_account_number Id ->bank_transaction_id

Trip (id, rider_id,provider_id, payment_id,rider_rating,rider_feedback, provider_feedback,pickup_location_lat, pickup_location_long,drop_location_lat, drop_location_long)

Functional dependencies in Trip relation: Id ->rider_id, Id ->provider_id Id ->payment_id Id ->rider_rating Id ->rider_feedback Id ->provider_feedback Id ->pickup_location_lat Id ->pickup_location_long Id ->drop_location_lat Id ->drop_location_long

id	driver_name	joined_date	gender	known_languag...	current_cab...	phone_number	licence_no
1	Dermot Scurman	2021-10-04	Female	Malay	53	102-784-3707	480918653-9
2	Nathalia Undrell	2021-06-27	Male	French	811	689-708-9714	061387319-X
3	Erskine Binyon	2015-09-30	Female	Bislama	729	736-642-4882	069250070-7
4	Isacco Mainstone	2017-12-11	Male	Hungarian	293	180-735-5118	388917997-5
5	Darline Bram	2016-07-23	Female	NULL	796	454-194-0951	637301300-6
6	Currie Ilyasov	2020-01-13	Female	Indonesian	325	334-349-4646	444877258-6
7	Gnni Mingardi	2021-04-14	Agender	Kazakh	202	103-632-9073	448214856-3
8	Kessiah Faier	2018-05-24	Male	Luxembourgish	899	726-526-5801	857266018-6
9	Pepi Jerams	2021-11-05	Male	Norwegian	702	496-477-2671	138657373-6
10	Haily Rendbaek	2019-12-14	Female	Bislama	338	346-718-5892	710195376-X
11	Jacinda MacNei...	2016-08-01	Female	Khmer	627	772-363-0763	515503667-4
12	Kingston Tolfrey	2018-07-15	Male	Afrikaans	88	169-972-3075	125114948-0
13	Tiebout Belle	2021-06-05	Male	Hungarian	89	122-857-5900	452043476-9
14	Merrili Crosston	2015-06-13	Female	Tetum	565	240-946-4152	438686568-3
15	Opal Diehn	2018-03-17	Female	English	853	845-408-7495	703252988-7
16	Germaine Holbury	2018-07-15	Male	Mongolian	303	415-938-6240	959041546-6
17	Davida Prender...	2020-12-25	Male	Japanese	690	390-348-9225	692419571-7
18	Saleem Halewood	2016-09-21	Female	French	670	644-684-9997	306982821-3
19	Valentina Gittens	2019-02-17	Female	West Frisian	484	517-168-6162	528473284-7
20	Dukey Greenhalf	2020-09-04	Female	West Frisian	614	531-992-0233	429325236-3
21	Carolyne Gaenor	2021-06-15	Female	Georgian	823	920-628-4675	042456413-0
22	Shaine Ewings	2018-11-28	Female	Swahili	23	729-617-8952	627454766-5
23	Norbie Ollier	2018-08-12	Male	Hungarian	382	360-588-0123	735090457-6
24	Zora Gillion	2017-08-20	Male	NULL	785	726-346-4681	148123786-1
25	Brose O'Deoran	2018-12-11	Male	Bengali	539	957-890-4773	920822728-6
26	Paige Merrell	2016-01-21	Male	Swati	209	146-535-5117	814107201-3

Fig. 10. Riders Table

Since all the attributes are single valued they are atomic. Therefore this table is in 1NF. Since there is no partial dependencies, this table is in 2NF Since there is no transitive dependencies, this table is in 3NF By using id all the attributes of a relation can be determined. Thus id is our candidate key for this relation. All the FD's are non-trivial as well. Therefore the Trip table satisfies/ does not violate BCNF.

We made sure all of our relations are in Boyce-Codd Normal Form.

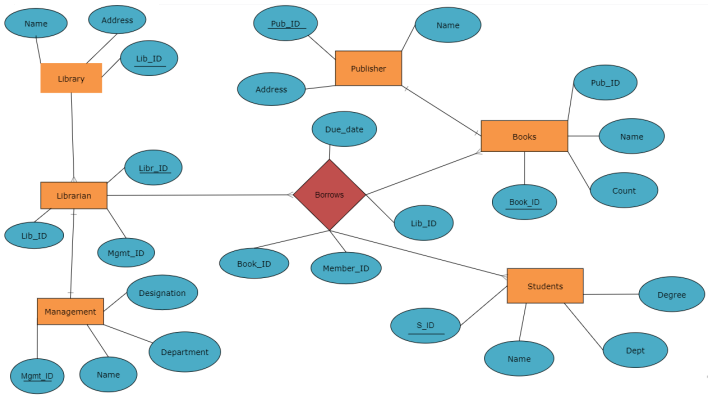

Fig. 11. ER Diagram

III. Create and Insert

```
CREATE TABLE IF NOT EXISTS Payment (
        id INT NOT NULL UNIQUE PRIMARY KEY,
        payment_type VARCHAR(7),
        base_fare INT,
        surge_fare INT,
        total_amount INT,
        time_of_payment DATE,
        payment_account_number VARCHAR(50),
        bank_transaction_id VARCHAR(50)
);


create table Trip (
        id INT,
        rider_id INT,
        provider_id INT,
        payment_id INT,
        rider_rating INT,
        rider_feedback TEXT,
        provider_feedback TEXT,
        pickup_location_lat VARCHAR(50),
        pickup_location_long VARCHAR(50),
        drop_location_lat VARCHAR(50),
        drop_location_long VARCHAR(50),
    FOREIGN KEY (rider_id) REFERENCES Rider(id),
    FOREIGN KEY (provider_id) REFERENCES Provider(
    FOREIGN KEY (payment_id) REFERENCES Payment(id
);

insert into Rider (id, first_name, last_name, emai

insert into Cab (id, registration_no, cab_brand, c

insert into Provider (id, driver_name, joined_date
```

IV. Other Analytical Queries

A.

The first query is to Find riders who made more than 100 rides in the application.

Here we get the riders who made more than 100 rides in the application whom we consider as loyal customers. And this information helps us to give specific deals to them. This is achieved by this query.

```
SELECT Trip.rider_id, Rider.first_name,
Rider.last_name, COUNT(Trip.id)
from Trip, Rider
WHERE Rider.id = Trip.rider_id
GROUP BY Trip.rider_id
HAVING count(Trip.id) > 100
ORDER BY count(Trip.id) DESC;
```

B.

The second Query is to get providers with the highest ratings given by the riders. This will help us find the potential providers who are eligible for incentives. So this is achieved by this query.

Result Grid | Filter Rows: Search | Export:

rider_id	first_name	last_name	COUNT(Trip.id)
9	Jennine	Scrivinor	114
7	Cheslie	Ghent	108
8	Venita	Klimentyonok	106
10	Karine	Millican	105

Fig. 12. Query 1 Result

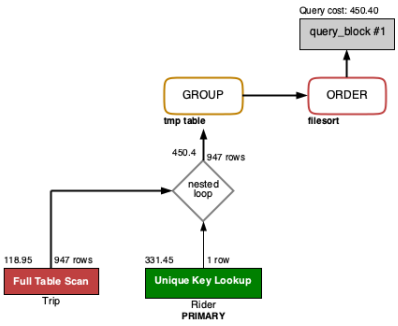

Fig. 13. Query 1 Cost

```
SELECT Provider.driver_name,
AVG(Trip.rider_rating)
from Trip, Provider
WHERE Provider.id = Trip.provider_id
GROUP BY Trip.provider_id
ORDER By AVG(Trip.rider_rating) DESC
LIMIT 1;
```

The provider who got the highest ratings is Tiebout Belle with an average rating of 3.3182.The results can be seen in the below screenshot.

driver_name	AVG(Trip.rider_rati...
Tiebout Belle	3.3182

Fig. 14. Query 2 Result

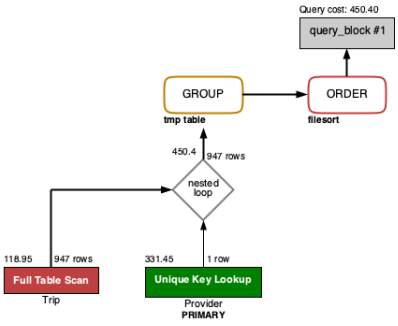

Fig. 15. Query 2 Cost

C.

The Third Query is to get the most used mode of payment by the riders to pay for the ride. This will help us find the mode of payment most preferred and give some exclusive deals to people who pay through this mode. This is achieved by this query.

```
SELECT Payment.payment_type,
COUNT(Payment.id)
FROM Payment
GROUP BY Payment.payment_type
ORDER BY COUNT(Payment.id)
DESC
LIMIT 1;
```

D.

The fourth query is to get total monthly earnings through the rides. This helps us find the income of the company monthly. This helps to see whether the company is on track. So this is achieved by this query.

```
SELECT month(time_of_payment)
as mon,
year(time_of_payment)
as yyyy,
sum(total_amount)
from Payment
group by 1,2
order by sum(total_amount) desc;
```

From this query we got the earnings from the website each month.The results can be seen in the below screenshot.

mon	yyyy	sum(total_amou...
1	2022	48714
4	2022	48539
2	2022	44625
11	2021	42273
12	2021	41698
3	2022	41164
10	2021	40686
6	2021	38746
5	2021	38286
9	2021	38234
7	2021	38131
8	2021	29823
5	2022	4958

Fig. 16. Query 4 Result

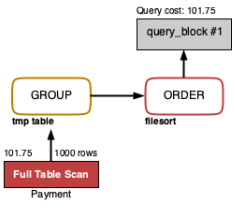

Fig. 17. Query 4 Cost

E.

Similar to the above query we have taken quarterly earnings of the company as well in this query. This is achieved by this query.

```
select sum(earnings) total_earning from (
SELECT month(time_of_payment) as mon,
year(time_of_payment) as yyyy,
sum(total_amount) earnings
from Payment
group by 1,2
HAVING mon = '01'
OR mon = '02'
OR mon = '03'
order by sum(total_amount) desc)
as monthly_earnings;
```

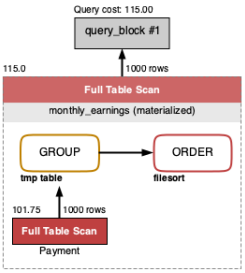

Fig. 18. Query 5 Cost

F.

In the sixth query we get the top 10 people who spent the highest amount on the application. This helps us to find the premium customers and give them exclusive deals. This is achieved by this query.

```
SELECT Trip.rider_id, Rider.first_name,
SUM(Payment.total_amount)
From Payment, Trip, Rider
WHERE Payment.id = Trip.payment_id AND
Trip.rider_id = Rider.id
GROUP BY Trip.rider_id
ORDER BY SUM(Payment.total_amount) DESC
LIMIT 10;
```

From this query we got the top ten customers who spent the highest amount on the rides. The results can be seen in the below screenshot.

More queries such as Top 100 people with how spent the highest amount on the application are tried. Indexing helped in optimising the query of the operations.

V. Optimization

To increase the performance we performed indexing by using a different data structure which captures the field values and pointer to record that is linked to it. This data structure is sorted which helps in executing binary search that results in log2N time complexity.

By using this indexing method, we increased the performance of the relations that are frequently used.

We have noticed a significant difference in the cost of the query after performing indexing on the relation. The cost of the query drastically reduced after applying indexing on the query.

```
SELECT Trip.rider_id, Rider.first_name,
Rider.last_name, COUNT(Trip.id)
from Trip, Rider
WHERE Rider.id = Trip.rider_id
and first_name = \'Abbie'
GROUP BY Trip.rider_id
HAVING count(Trip.id) > 50
ORDER BY count(Trip.id) \DESC;
```

```
create index idx_name on Rider(first_name);
```

Query Cost

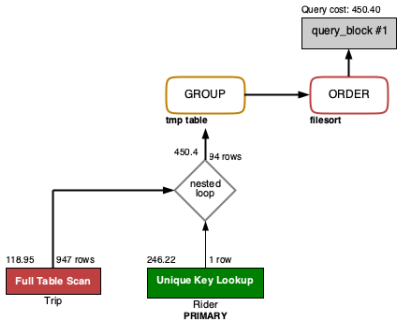

Fig. 19. Cost before optimization

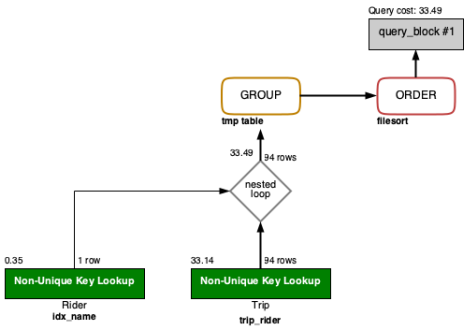

Fig. 20. Cost after optimization

Improving the query We have selected a query which performed poorly in terms of cost of the query and wrote it in a different format. We used product to join the tables in the first format which performed very poorly.

```
SELECT Trip.rider_id, Rider.*
from Trip, Provider, Rider
WHERE Provider.id = Trip.provider_id
and Rider.id = Trip.rider_id
GROUP BY Trip.rider_id
ORDER By AVG(Trip.rider_rating) \DESC
LIMIT 1;
```

Query Cost

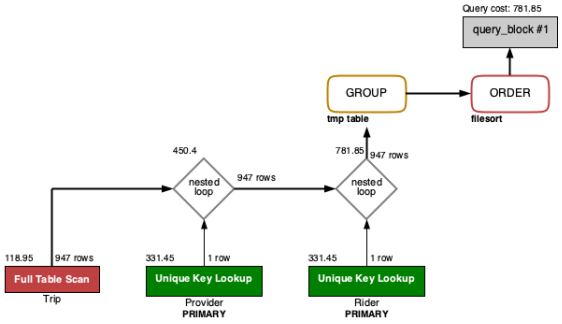

Fig. 21. Cost before optimization

So, now in the second format of the same query we have optimized it by using . This has improved the performance of the query drastically. That is the cost got reduced significantly.

```
select * from rider where id in (
select rider_id from Trip
group by rider_id
having AVG(Trip.rider_rating)
>= ALL(select AVG(Trip.rider_rating)
from trip group by rider_id));
```

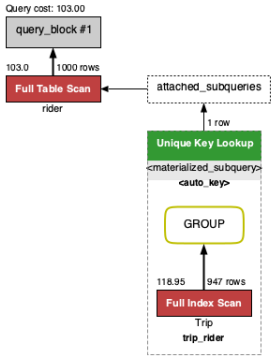

Fig. 22. Cost after optimization

VI. OF THE APPLICATION

Using Python as the back end language we create a few GET to fetch data from the DB. Flask API structure has been maintained. SQL Library from has been used to send cursor to

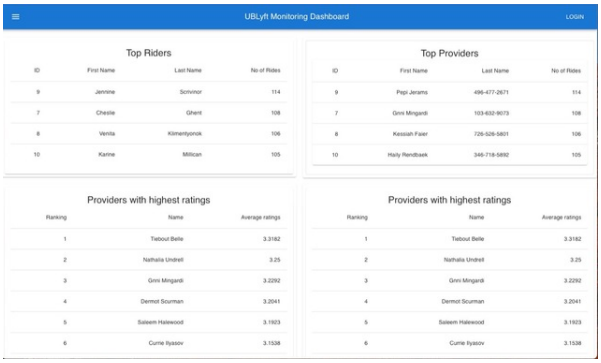

Fig. 23. of the application

the DB.All this has been tested using Postman API tester.Once done, a React JS application has been made using Material UI as CSS interface to load the data from the APIs. Axios library has been used to make these API calls. Once the data is loaded, React JS Dashboard renders the data in Material UI tables.

VII. CONTRIBUTION

Team member	Contribution %
Sai Chandra Rachiraju (srachira)	33.3%
Venkat Kaushik Vadlamudi (vvadlamu)	33.3%
Indeevara Kodam (indeevar)	33.3%

Fig. 24. Contribution