RYERSON UNIVERSITY

Faculty of Engineering, Architecture and Science

Course Number	CPS 510
Course Title	Database Systems I
Semester/Year	F2021

Instructor	Abdolreza Abhari

ASSIGNMENT No. 10

Assignment Title	CPS Final Project Report

Submission Date	December 2nd, 2021
Due Date	December 2nd, 2021

Student Name	Marvy Shaker	Youssef El Mahallaway	Kirolos Youssef
Student ID	501002035	500973262	500968175
Signature*	M.S.	Y.E.M	K.Y

**By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: www.ryerson.ca/senate/current/pol60.pdf.*

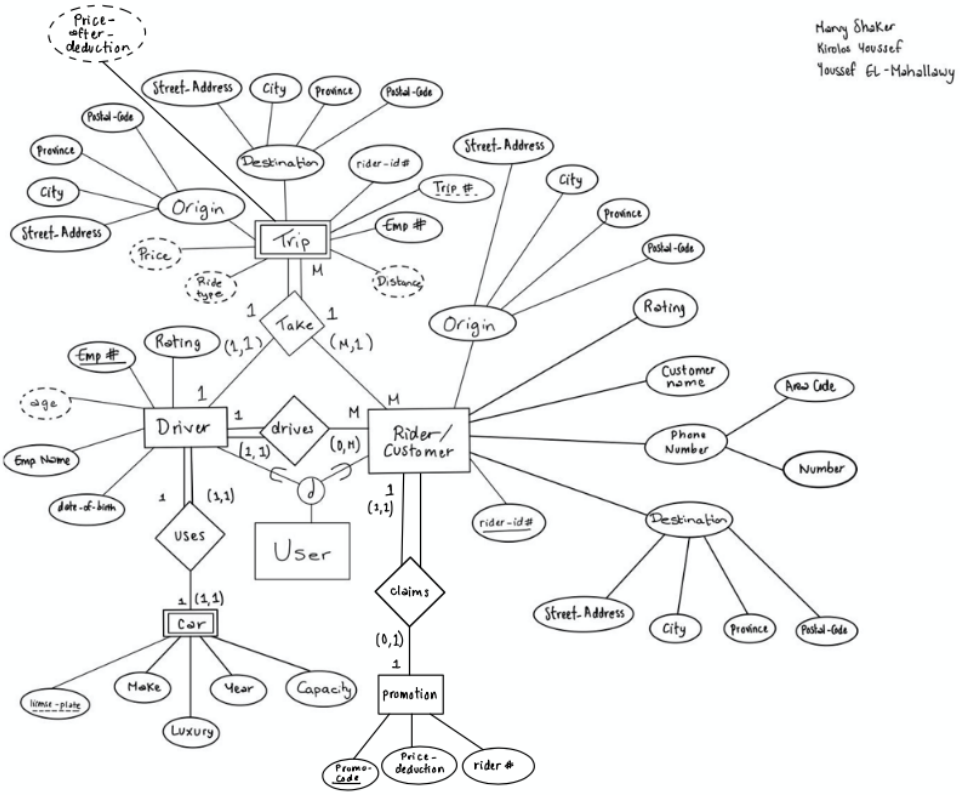
Table of Contents

Assignment Breakdown:
Assignment 1:

In assignment 1, the group discussed in detail the plan for the project ahead, its description, how it will be implemented, what methods will be used, and lastly the purpose and functionality of this Ride and Pickup Service database with the addition of some sample tables.

Assignment 2:

In assignment 2, the group designed the ER Diagram for the database. We used all types of relationships between our tables and entities and avoided redundancy from the beginning of the project.



Assignment 3:

In assignment 3, we were tasked to start using SQL and building tables based on the ER diagram provided in assignment 2. We then created all the tables with their respective data types as it would be as if we created the actual database and application. (All of the SQL code screenshots are at the end). We also used reference variables as it would be depicted in the ER Diagram.

```
-- Start of Assignment 3
DROP TABLE Trips;
DROP TABLE Promotion;
DROP TABLE Rider;
DROP TABLE CARS;
DROP TABLE Driver;
DROP TABLE AppUser;

CREATE TABLE AppUser(
UserID INT PRIMARY KEY
);

CREATE TABLE Driver(
Rating DECIMAL(2,1),
DOB VARCHAR2(10),
Age INT CHECK (Age >= 21),
EmployeeNum INT PRIMARY KEY REFERENCES AppUser(UserID) ON DELETE CASCADE,
EmployeeName VARCHAR2(30)
);

CREATE TABLE Cars(
LicensePlate VARCHAR2(8) PRIMARY KEY,
Make VARCHAR2(20),
Luxury VARCHAR2(1),
CONSTRAINT CHK_BOOLEAN_VAL CHECK (Luxury IN ('1','0')),
YearMade INTEGER CHECK(YearMade BETWEEN 2012 and 2022),
CarCapacity INT,
UsedBy INT REFERENCES Driver(EmployeeNum) ON DELETE CASCADE
);

CREATE TABLE Rider(
RiderID INT PRIMARY KEY REFERENCES AppUser(UserID) ON DELETE CASCADE,
RiderName VARCHAR2(20),
RiderRating DECIMAL(2,1),
PhoneNumber VARCHAR2(20)
);

CREATE TABLE Promotion(
PromoCode VARCHAR2(10) PRIMARY KEY,
PriceReduce INT,
ClaimedBy REFERENCES Rider(RiderID) ON DELETE CASCADE
);

CREATE TABLE Trips(
TripID INT PRIMARY KEY,
Rider INT REFERENCES Rider(RiderID) ON DELETE CASCADE,
Driver INT References Driver(EmployeeNum) ON DELETE CASCADE,
PriceReduction VARCHAR2(10) DEFAULT 0 REFERENCES Promotion(PromoCode),
Price INT,
RideType VARCHAR(10),
DistanceInKM DECIMAL(4,1),
PromotionCode VARCHAR2(10) DEFAULT 0,

originStreetAddress VARCHAR2(40),
originCity VARCHAR2(20),
originProvince VARCHAR2(20),
originPostalCode VARCHAR2(10),

finalStreetAddress VARCHAR2(40),
finalCity VARCHAR2(20),
finalProvince VARCHAR2(20),
finalPostalCode VARCHAR2(10)
);

-- End of Assignment 3
```

Assignment 4:

For our fourth assignment, we had to start using queries on our database and test our data. We then started using simple queries which would populate our tables with data, then selecting data from the populated tables. Instead of specific selective queries, we decided to return all data in the tables using the universal notation '*' (asterisk).

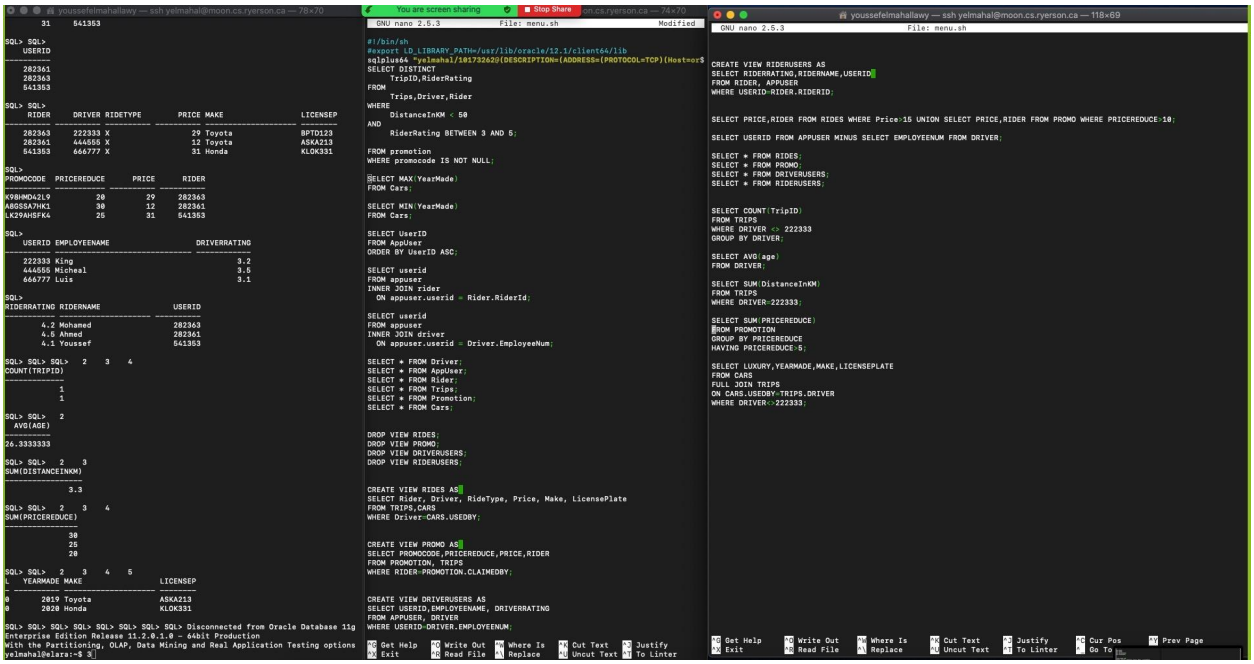
```
INSERT INTO AppUser(UserID) VALUES (282363);
INSERT INTO AppUser(UserID) VALUES (282361);
INSERT INTO AppUser(UserID) VALUES (541353);

INSERT INTO AppUser(UserID) VALUES (222333);
INSERT INTO AppUser(UserID) VALUES (444555);
INSERT INTO AppUser(UserID) VALUES (666777);

INSERT INTO Rider(RiderID,RiderName,PhoneNumber,RiderRating) VALUES (282363,'Mohamed','416-666-6666',4.2);
INSERT INTO Rider(RiderID,RiderName,PhoneNumber,RiderRating) VALUES (282361,'Ahmed','416-666-5555',4.5);
INSERT INTO Rider(RiderID,RiderName,PhoneNumber,RiderRating) VALUES (541353,'Youssef','416-666-4444',4.1);

INSERT INTO Driver(EmployeeNum,EmployeeName,DOB,Rating,Age) VALUES (222333,'King','04/20/2001',3.2,31);
INSERT INTO Driver(EmployeeNum,EmployeeName,DOB,Rating,Age) VALUES (444555,'Micheal','01/20/2001',3.5,25);
INSERT INTO Driver(EmployeeNum,EmployeeName,DOB,Rating,Age) VALUES (666777,'Luis','05/21/2008',3.1,23);

INSERT INTO Cars(LicensePlate,Make,Luxury,YearMade,CarCapacity,UsedBy) VALUES('BPTD123','Toyota','1',2018,5,222333);
INSERT INTO Cars(LicensePlate,Make,Luxury,YearMade,CarCapacity,UsedBy) VALUES('ASKA213','Toyota','0',2019,5,444555);
INSERT INTO Cars(LicensePlate,Make,Luxury,YearMade,CarCapacity,UsedBy) VALUES('KLOK331','Honda','0',2020,5,666777);

INSERT INTO Promotion(PromoCode,PriceReduce,ClaimedBy) VALUES ('K98HMD42L9',20,282363);
INSERT INTO Promotion(PromoCode,PriceReduce,ClaimedBy) VALUES ('A8GSSA7HK1',30,282361);
INSERT INTO Promotion(PromoCode,PriceReduce,ClaimedBy) VALUES ('LK29AHSFK4',25,541353);

INSERT INTO Trips(TripID,Rider,Driver,PriceReduction,Price,RideType,DistanceInKM,originStreetAddress,originCity,originProvince,originPostalCode,finalStreetAddress,
finalCity,finalProvince,finalPostalCode) VALUES
(817624,282363,222333,'K98HMD42L9',29,'X',3.32,'9102 Bird Lane','Oakville','ON','L5S 5K2','83 Saga Lane','Burlington','ON','L5F 5L3');
INSERT INTO Trips(TripID,Rider,Driver,PriceReduction,Price,RideType,DistanceInKM,originStreetAddress,originCity,originProvince,originPostalCode,finalStreetAddress,
finalCity,finalProvince,finalPostalCode) VALUES
(817617,282361,444555,'A8GSSA7HK1',12,'X',3.42,'7812 Stokes Lane','Oakville','ON','L5S 5K2','8123 Stokes Lane','Burlington','ON','L5F 5L3');
INSERT INTO Trips(TripID,Rider,Driver,PriceReduction,Price,RideType,DistanceInKM,originStreetAddress,originCity,originProvince,originPostalCode,finalStreetAddress,
finalCity,finalProvince,finalPostalCode) VALUES
(817626,541353,666777,'LK29AHSFK4',31,'X',3.12,'72  Lane','Oakville','ON','L5S 5K2','2491 Jok Lane','Burlington','ON','L5F 5L3');

SELECT * FROM Driver;
SELECT * FROM AppUser;
SELECT * FROM Rider;
SELECT * FROM Trips;
SELECT * FROM Promotion;
SELECT * FROM Cars;
```

Assignment 5:

For this assignment, we had to start using more advanced queries where we can selectively query data from joined tables. We also created views, which are basically selective sub-tables of several tables. For example; we can create a ‘view’ which is basically a temporary table that stores all user entities with their respective data, and all we have to do is select certain fields from Drivers and Riders. The example specified is the first view we created down below.

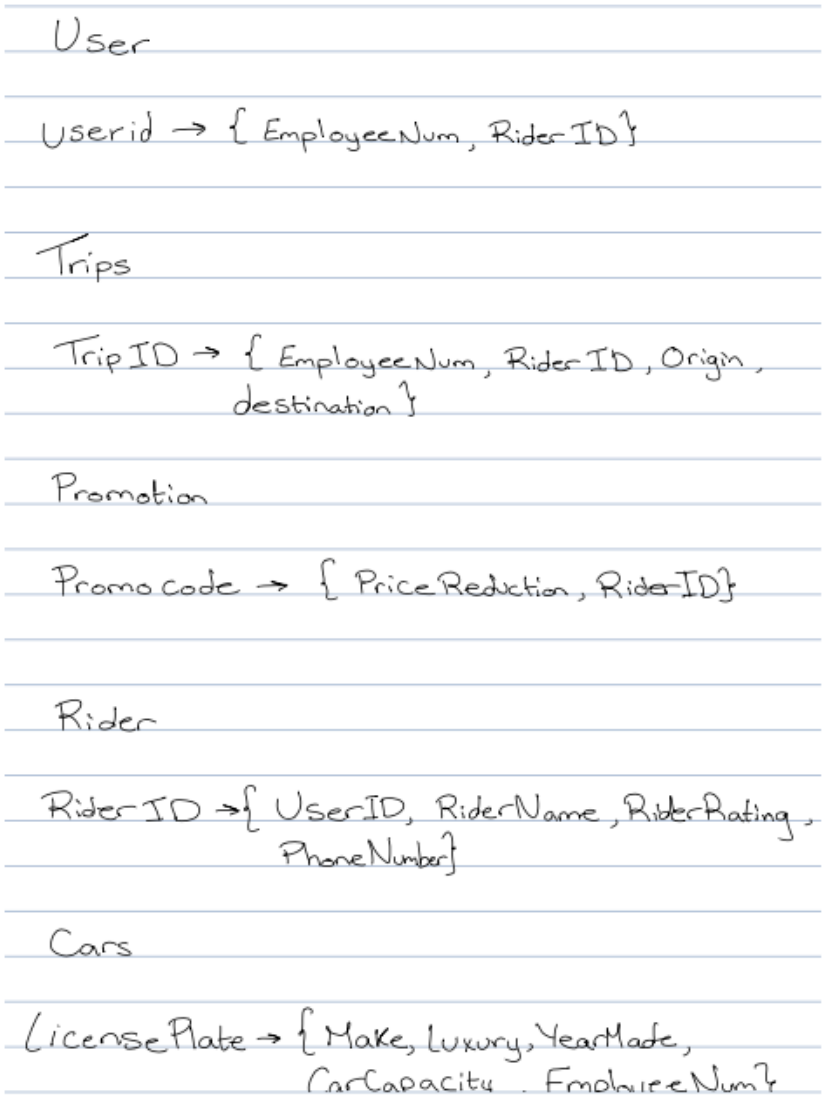
```
SELECT PromoCode,PriceReduce
FROM promotion
WHERE promocode IS NOT NULL;

SELECT MAX(YearMade)
FROM Cars;

SELECT MIN(YearMade)
FROM Cars;

SELECT UserID
FROM AppUser
ORDER BY UserID ASC;

SELECT userid
FROM appuser
INNER JOIN rider
  ON appuser.userid = Rider.RiderId;

SELECT userid
FROM appuser
INNER JOIN driver
  ON appuser.userid = Driver.EmployeeNum;



DROP VIEW RIDES;
DROP VIEW PROMO;
DROP VIEW DRIVERUSERS;
DROP VIEW RIDERUSERS;



CREATE VIEW UserEntity AS
SELECT RiderID, EmployeeNum, RiderName, EmployeeName
FROM Rider,Driver;


CREATE VIEW RIDES AS
SELECT Rider, Driver, RideType, Price, Make, LicensePlate
FROM TRIPS,CARS
WHERE Driver=CARS.USEDBY;


CREATE VIEW PROMO AS
SELECT PROMOCODE,PRICEREDUCE,PRICE,RIDER
FROM PROMOTION, TRIPS
WHERE RIDER=PROMOTION.CLAIMEDBY;


CREATE VIEW DRIVERUSERS AS
SELECT USERID,EMPLOYEENAME, DRIVERRATING
FROM APPUSER, DRIVER
WHERE USERID=DRIVER.EMPLOYEENUM;


CREATE VIEW RIDERUSERS AS
SELECT RIDERRATING,RIDERNAME,USERID
FROM RIDER, APPUSER
WHERE USERID=RIDER.RIDERID;


SELECT PRICE,RIDER FROM RIDES WHERE Price>15 UNION SELECT PRICE,RIDER FROM PROMO WHERE PRICEREDUCE>10;

SELECT USERID FROM APPUSER MINUS SELECT EMPLOYEENUM FROM DRIVER;
```

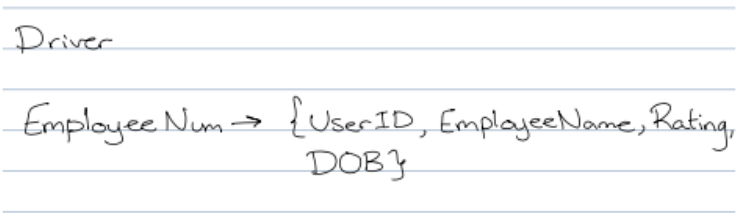
UNIX REPRESENTATION:



Assignment 6:

For this assignment, we show the functional dependencies that our database schema uses. We displayed these dependencies by writing out each one of our primary keys and what is dependent on it.



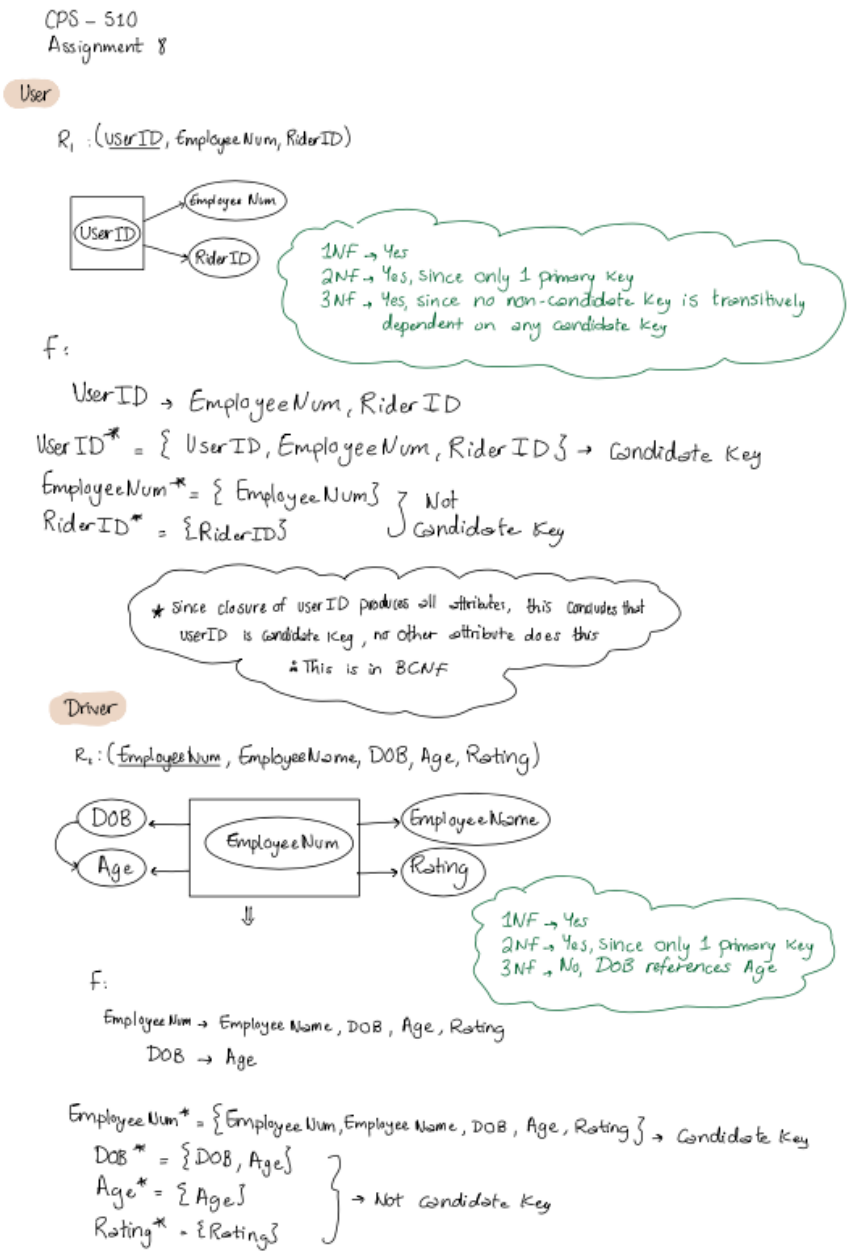


Assignment 7:

In this assignment, we changed each of our tables in order to be in 3NF format. This was done by ensuring that all tables were in 1NF and 2NF first. Afterwards, we had to make sure there were no transitive dependencies between any non-candidate keys and candidate keys. The 3NF tables were displayed in diagrams representing each table. Additionally, the Bernstein algorithm was used in order to make sure all proper steps were followed as we were transitioning each table into 3NF.s

Assignment 8:

Since Assignment 7 was executed correctly, this assignment was quite simple to complete since all tables were already in 1NF, 2NF, and 3NF. Therefore, putting each table into BCNF afterwards, which was the goal of this assignment, was a quick task to do. The BCNF and Bernstein Algorithms were both used to ensure the end results were correct.



∴ Not in BCNF
Since Not everything on left hand side is a candidate key

Using Bernstein's Algorithm

$R_{2.1}$ (<u>Employee Num</u>, Employee Name, Rating, DOB)
AND
$R_{2.2}$ (DOB, Age)

Step 1: We want the employee number to determine all the other attributes & date of birth to determine age

Employee Num → Employee Name, Rating, DOB
DOB → Age

Step 2: Ensuring there is no redundancy in the fDs

Step 3: Finding the keys by applying Closure on all the attributes

* Employee Num* = {Employee Name, Rating, DoB, Age}

Employee Name* = {Employee Name}

Rating* = {Rating}

* DOB* = {DOB, Age}

Age* = {Age}

Step 4: Decompose the table into 2 tables while maintaining the desired functional dependancies

$R_{2.1}$ (Employee Num, Employee Name, Rating, DOB)

AND

$R_{2.2}$ (DOB, Age)

Car

R_3 (License Plate, Make, Year, Luxury, Capacity)

1NF → Yes
2NF → Yes, since only 1 primary key
3NF → Yes, since no non-candidate key is transitively dependent on any candidate key

licensePlate* = { licensePlate, Make, Year, Luxury, Capacity }

↓

Candidate Key

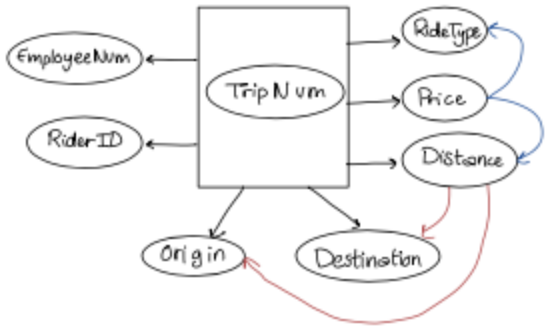
Luxury* = { luxury }
Make* = { Make }
Year* = { Year }
Capacity* = { Capacity }

★ Since closure of licensePlate produces all attributes, this concludes that licensePlate is candidate key, no other attribute does this

∴ This is in BCNF

Trip

R_4 : (TripNum, EmployeeNum, RiderID, Distance, Price, RideType, Destination, Origin)



1NF → No, since Origin and Destination are multi-valued attributes, it must be split up into respective parts

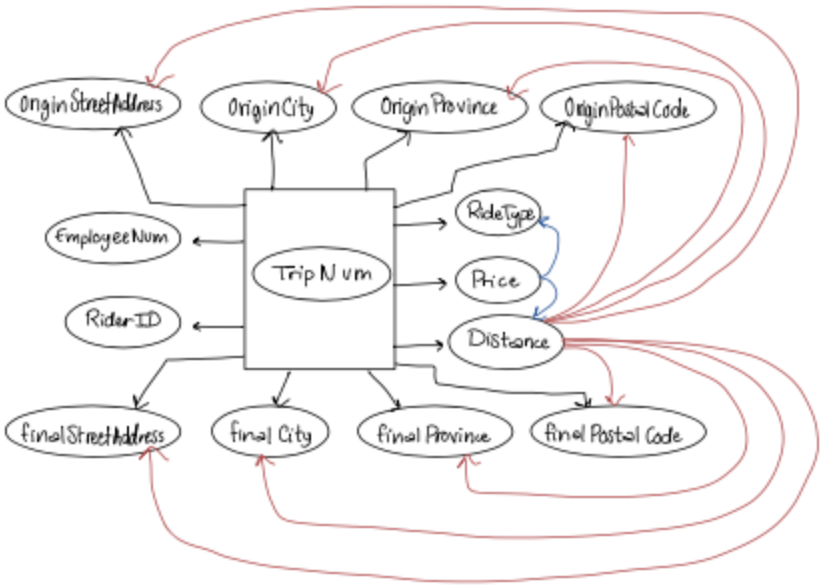
⇓

R_4 : (TripNum, EmployeeNum, RiderID, Distance, Price, RideType, originStreetAddress, originCity, originProvince, originPostalCode, finalStreetAddress, finalCity, finalProvince, finalPostalCode)

TripNum* = { TripNum, EmployeeNum, RiderID, Distance, Price, RideType, originStreetAddress, originCity, originProvince, originPostalCode, finalStreetAddress, finalCity, finalProvince, finalPostalCode }

Not Candidate Keys:
- Price* = { Price, RideType, Distance }
- Distance* = { Distance, originStreetAddress, originCity, originProvince, originPostalCode, finalStreetAddress, finalCity, finalProvince, finalPostalCode }

↳ Not BCNF



√2NF: Yes, since there's only 1 primary key

X 3NF: No, since price depends on ridetype and distance
AND
distance depends on origin and destination

Split R's into the following 3 Tables

$R_{4.1}$: (TripNum, EmployeeNum, RiderID, Price)

$R_{4.2}$: (Price, RideType, Distance)

$R_{4.3}$: (Distance, originStreetAddress, originCity, originProvince, originPostalCode, finalStreetAddress, finalCity, finalProvince, finalPostalCode)

$R_{4.1}$:

TripNum* = {TripNum, EmployeeNum, RiderID, Price}
EmployeeNum* = {EmployeeNum}
RiderID* = {RiderID}
Price* = {Price}

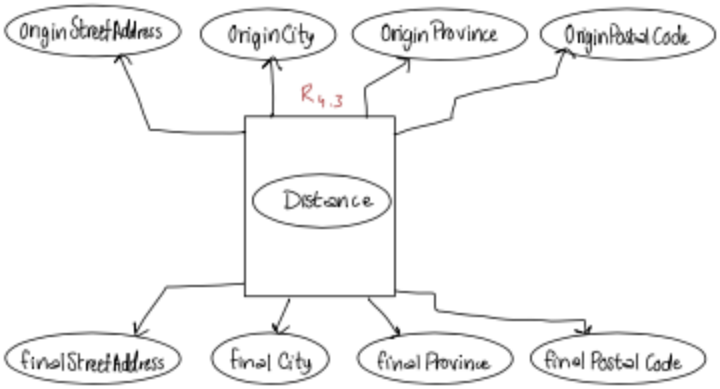
$R_{4.2}$:

Price* = {Price, RideType, Distance}
RideType* = {RideType}
Distance* = {Distance}

$R_{4.3}$:

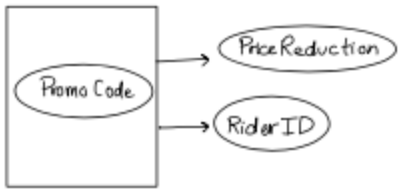
Distance* = {Distance, originStreetAddress, originCity, originProvince, originPostalCode, finalStreetAddress, finalCity, finalProvince, finalPostalCode}

∴ Now it's in BCNF since each lefthand side is a candidate key, including all attributes in the table



Promotion

R_6 : (<u>Promo Code</u>, Price Reduction, Rider ID)

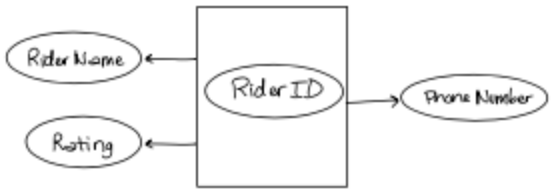


PromoCode* = {Promo Code, Price Reduction, Rider ID}
PriceReduction* = { Price Reduction}
RiderID * = { Rider ID}

1NF → Yes
2NF → Yes, since only 1 primary key
3NF → Yes, since no non-candidate key is transitively dependent on any candidate key
BCNF → Yes, since left hand side is a candidate key, including all attributes in the table

Rider

R_6 : (<u>RiderID</u> , RiderName, Rating, Phone Number)



RiderID* = {RiderID , RiderName, Rating, Phone Number}
RiderName* = {RiderName}
Rating* = {Rating}
Phone Number* = {PhoneNumber}

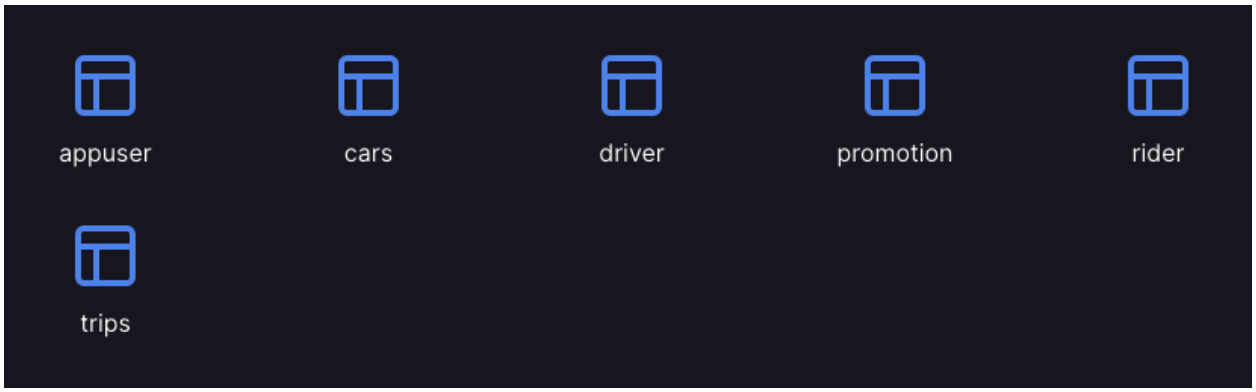
1NF → Yes
2NF → Yes, since only 1 primary key
3NF → Yes, since no non-candidate key is transitively dependent on any candidate key

BCNF → Yes, since lefthand side is ⇒ candidate key, including all attributes in the table

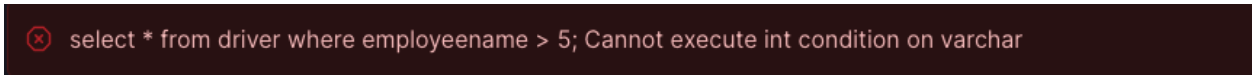
Assignment 9:

For this assignment, we were to demonstrate the queries using a GUI, but rather, we decided to go for the bonus marks and created a web UI using React.js (Javascript framework). We used Prisma with Typescript as our ORM to communicate with the backend database where we were able to make queries using raw SQL. We decided to use Prisma since it allows simple queries and provides great flexibility as our middleware between the client and server. Prisma client allowed us to run our query and have our updates show almost instantly. We decided to use React.js to build our front-end rather than Oracle Java or Python since it allowed for instantaneous rerenders whenever the changes in our database had fully registered, and because it was the language all group members know how to use. We used Typescript to explicitly specify datatypes of all properties in each table, so that during runtime, the code is checked and makes sure that users can't run queries that would result in an error. When the user enters a query that would return an error, we wouldn't query the database, but instead we would purposely throw an error to our client and alert the user of the error.

Tables in DB



Error handling



Assignment 10 (Relational Algebra Notation):

$$\Pi_{tripID,\ riderRating}\left(\sigma_{distanceInKm < 50\ AND\ (3 <= riderRating\ AND\ riderRating <= 5)}(trips \times driver \times rider)\right)$$

$$\Pi_{promoCode,\ priceReduce}\left(\sigma_{NOT(promocode = NULL)}(Promotion)\right)$$

$$\Pi_{MAX(yearMade)}\left(\gamma_{MAX(yearMade)}(Cars)\right)$$

$$\Pi_{MIN(yearMade)}\left(\gamma_{MIN(yearMade)}(Cars)\right)$$

$$\tau_{UserID}\left(\Pi_{UserID}\ (AppUser)\right)$$

$$\Pi_{userID}\left(AppUser \bowtie_{AppUser.UserID = Rider.RiderID} Rider\right)$$

$$\Pi_{userID}\left(AppUser \bowtie_{AppUser.UserID = Driver.employeeNum} Driver\right)$$

$$\Pi_{Rider,\ Driver,\ RideType,\ Price,\ Make,\ licensePlate}\left(\sigma_{Driver = Cars.usedby}(Trips \times Cars)\right)$$

$$\Pi_{PromoCode,\ PriceReduce,\ Price,\ Rider}\left(\sigma_{Rider = Promotion.Claimedby}(Promotion \times Trips)\right)$$

$$\pi_{UserID, EmployeeName, DriverRating}(\sigma_{userID = Driver.EmployeeNum}(AppUser \times Driver))$$

$$\pi_{RiderRating, RiderName, UserID}(\sigma_{UserID = Rider.RiderID}(Rider \times Appuser))$$

$$\pi_{Price, Rider}(\sigma_{Price>15}(Rides)) \cup \pi_{Price, Rider}(\sigma_{PriceReduce>10}(Promo))$$

$$\pi_{UserID}(AppUser) - \pi_{UserID}(\sigma_{userID = Driver.EmployeeNum}(AppUser \times Driver))$$

$$\pi_{COUNT(TripID)}(\gamma_{Driver, COUNT(TripID)}(\sigma_{Driver <> 222333}(Trips)))$$

$$\pi_{AVG(age)}(\gamma_{AVG(age)}(Driver))$$

$$\pi_{SUM(distanceInKm)}(\gamma_{SUM(distanceInKm)}(\sigma_{Driver = 222333}(Trips)))$$

$$\pi_{SUM(PriceReduce)}(\sigma_{PriceReduce>5}(\gamma_{PriceReduce, SUM(PriceReduce)}(Promotion)))$$

$$\pi_{luxury, YearMade, Make, LicensePlate}(\sigma_{Driver <> 222333}(Cars \bowtie_{Cars.usedby = Trips.Driver} Trips))$$

Concluding Remarks:

Overall, this project has been a great learning experience for all members of the group. We learnt how to work together in order to achieve the desired database with the useful input of our TA. We ran into some difficulties with the implementation of certain assignments and some confusion. Assignment 5 was one of the more challenging assignments for us as we were not entirely familiar with the Linux implementation of our SQL code. After discussing this and getting the help needed from our TA, we were able to execute it successfully as instructed. As for the remainder of the assignments, most of the instructions were fairly clear and we were able to finish them without many problems. The assignments being on track with the content learnt in lecture allowed us to have the perfect practice outlet in order to apply the information we were learning, in a way we can easily visualize and participate in.

Complete SQL Code

```
DROP TABLE Trips;
DROP TABLE Promotion;
DROP TABLE Rider;
DROP TABLE CARS;
DROP TABLE Driver;
DROP TABLE AppUser;

CREATE TABLE AppUser(
UserID INT PRIMARY KEY
);

CREATE TABLE Driver(
Rating DECIMAL(2,1),
DOB VARCHAR2(10),
Age INT CHECK (Age >= 21),
EmployeeNum INT PRIMARY KEY REFERENCES AppUser(UserID) ON
DELETE CASCADE,
EmployeeName VARCHAR2(30)
```

```
);

CREATE TABLE Cars(
LicensePlate VARCHAR2(8) PRIMARY KEY,
Make VARCHAR2(20),
Luxury VARCHAR2(1),
CONSTRAINT CHK_BOOLEAN_VAL CHECK (Luxury IN ('1','0')),
YearMade INTEGER CHECK(YearMade BETWEEN 2012 and 2022),
CarCapacity INT,
UsedBy INT REFERENCES Driver(EmployeeNum) ON DELETE
CASCADE
);

CREATE TABLE Rider(
RiderID INT PRIMARY KEY REFERENCES AppUser(UserID) ON
DELETE CASCADE,
RiderName VARCHAR2(20),
RiderRating DECIMAL(2,1),
PhoneNumber VARCHAR2(20)
);

CREATE TABLE Promotion(
PromoCode VARCHAR2(10) PRIMARY KEY,
PriceReduce INT,
ClaimedBy REFERENCES Rider(RiderID) ON DELETE CASCADE
);

CREATE TABLE Trips(
```

```
TripID INT PRIMARY KEY,
Rider INT REFERENCES Rider(RiderID) ON DELETE CASCADE,
Driver INT References Driver(EmployeeNum) ON DELETE
CASCADE,
PriceReduction VARCHAR2(10) DEFAULT 0 REFERENCES
Promotion(PromoCode),
Price INT,
RideType VARCHAR(10),
DistanceInKM DECIMAL(4,1),
PromotionCode VARCHAR2(10) DEFAULT 0,

originStreetAddress VARCHAR2(40),
originCity VARCHAR2(20),
originProvince VARCHAR2(20),
originPostalCode VARCHAR2(10),

finalStreetAddress VARCHAR2(40),
finalCity VARCHAR2(20),
finalProvince VARCHAR2(20),
finalPostalCode VARCHAR2(10)
);

-- End of Assignment 3

INSERT INTO AppUser(UserID) VALUES (282363);
INSERT INTO AppUser(UserID) VALUES (282361);
INSERT INTO AppUser(UserID) VALUES (541353);
INSERT INTO AppUser(UserID) VALUES (222333);
```

```
INSERT INTO AppUser(UserID) VALUES (444555);
INSERT INTO AppUser(UserID) VALUES (666777);

INSERT INTO
Rider(RiderID,RiderName,PhoneNumber,RiderRating) VALUES
(282363,'Mohamed','416-666-6666',4.2);
INSERT INTO
Rider(RiderID,RiderName,PhoneNumber,RiderRating) VALUES
(282361,'Ahmed','416-666-5555',4.5);
INSERT INTO
Rider(RiderID,RiderName,PhoneNumber,RiderRating) VALUES
(541353,'Youssef','416-666-4444',4.1);

INSERT INTO
Driver(EmployeeNum,EmployeeName,DOB,Rating,Age) VALUES
(222333,'King','04/20/2001',3.2,31);
INSERT INTO
Driver(EmployeeNum,EmployeeName,DOB,Rating,Age) VALUES
(444555,'Micheal','01/20/2001',3.5,25);
INSERT INTO
Driver(EmployeeNum,EmployeeName,DOB,Rating,Age) VALUES
(666777,'Luis','05/21/2008',3.1,23);

INSERT INTO
Cars(LicensePlate,Make,Luxury,YearMade,CarCapacity,UsedBy
) VALUES('BPTD123','Toyota','1',2018,5,222333);
```

```
INSERT INTO
Cars(LicensePlate,Make,Luxury,YearMade,CarCapacity,UsedBy
) VALUES('ASKA213','Toyota','0',2019,5,444555);
INSERT INTO
Cars(LicensePlate,Make,Luxury,YearMade,CarCapacity,UsedBy
) VALUES('KLOK331','Honda','0',2020,5,666777);

INSERT INTO Promotion(PromoCode,PriceReduce,ClaimedBy)
VALUES ('K98HMD42L9',20,282363);
INSERT INTO Promotion(PromoCode,PriceReduce,ClaimedBy)
VALUES ('A8GSSA7HK1',30,282361);
INSERT INTO Promotion(PromoCode,PriceReduce,ClaimedBy)
VALUES ('LK29AHSFK4',25,541353);

INSERT INTO
Trips(TripID,Rider,Driver,PriceReduction,Price,RideType,D
istanceInKM,originStreetAddress,originCity,originProvince
,originPostalCode,finalStreetAddress,finalCity,finalProvi
nce,finalPostalCode) VALUES
(817624,282363,222333,'K98HMD42L9',29,'X',3.32,'9102 Bird
Lane','Oakville','ON','L5S 5K2','83 Saga
Lane','Burlington','ON','L5F 5L3');
INSERT INTO
Trips(TripID,Rider,Driver,PriceReduction,Price,RideType,D
istanceInKM,originStreetAddress,originCity,originProvince
,originPostalCode,finalStreetAddress,finalCity,finalProvi
nce,finalPostalCode) VALUES
```

```
(817617,282361,444555,'A8GSSA7HK1',12,'X',3.42,'7812
Stokes Lane','Oakville','ON','L5S 5K2','8123 Stokes
Lane','Burlington','ON','L5F 5L3');
INSERT INTO
Trips(TripID,Rider,Driver,PriceReduction,Price,RideType,D
istanceInKM,originStreetAddress,originCity,originProvince
,originPostalCode,finalStreetAddress,finalCity,finalProvi
nce,finalPostalCode) VALUES
(817626,541353,666777,'LK29AHSFK4',31,'X',3.12,'72
Lane','Oakville','ON','L5S 5K2','2491 Jok
Lane','Burlington','ON','L5F 5L3');

SELECT DISTINCT
     TripID,RiderRating
FROM
     Trips,Driver,Rider
WHERE
     DistanceInKM < 50
AND
     RiderRating BETWEEN 3 AND 5;

ALTER TABLE DRIVER RENAME COLUMN RATING TO DriverRating;

SELECT PromoCode,PriceReduce
FROM promotion
WHERE promocode IS NOT NULL;

SELECT MAX(YearMade)
```

```
FROM Cars;

SELECT MIN(YearMade)
FROM Cars;

SELECT UserID
FROM AppUser
ORDER BY UserID ASC;

SELECT userid
FROM appuser
INNER JOIN rider
  ON appuser.userid = Rider.RiderId;

SELECT userid
FROM appuser
INNER JOIN driver
  ON appuser.userid = Driver.EmployeeNum;

SELECT * FROM Driver;
SELECT * FROM AppUser;
SELECT * FROM Rider;
SELECT * FROM Trips;
SELECT * FROM Promotion;
SELECT * FROM Cars;

DROP VIEW RIDES;
DROP VIEW PROMO;
```

```
DROP VIEW DRIVERUSERS;
DROP VIEW RIDERUSERS;

CREATE VIEW RIDES AS
SELECT Rider, Driver, RideType, Price, Make, LicensePlate
FROM TRIPS,CARS
WHERE Driver=CARS.USEDBY;

CREATE VIEW PROMO AS
SELECT PROMOCODE,PRICEREDUCE,PRICE,RIDER
FROM PROMOTION, TRIPS
WHERE RIDER=PROMOTION.CLAIMEDBY;

CREATE VIEW DRIVERUSERS AS
SELECT USERID,EMPLOYEENAME, DRIVERRATING
FROM APPUSER, DRIVER
WHERE USERID=DRIVER.EMPLOYEENUM;

CREATE VIEW RIDERUSERS AS
SELECT RIDERRATING,RIDERNAME,USERID
FROM RIDER, APPUSER
WHERE USERID=RIDER.RIDERID;

SELECT PRICE,RIDER FROM RIDES WHERE Price>15 UNION SELECT
PRICE,RIDER FROM PROMO WHERE PRICEREDUCE>10;

SELECT USERID FROM APPUSER MINUS SELECT EMPLOYEENUM FROM
DRIVER;
```

```sql
SELECT * FROM RIDES;
SELECT * FROM PROMO;
SELECT * FROM DRIVERUSERS;
SELECT * FROM RIDERUSERS;

SELECT COUNT(TripID)
FROM TRIPS
WHERE DRIVER <> 222333
GROUP BY DRIVER;

SELECT AVG(age)
FROM DRIVER;

SELECT SUM(DistanceInKM)
FROM TRIPS
WHERE DRIVER=222333;

SELECT SUM(PRICEREDUCE)
FROM PROMOTION
GROUP BY PRICEREDUCE
HAVING PRICEREDUCE>5;

SELECT LUXURY,YEARMADE,MAKE,LICENSEPLATE
FROM CARS
FULL JOIN TRIPS
ON CARS.USEDBY=TRIPS.DRIVER
WHERE DRIVER<>222333;
```