Ride and Pick-up DBMS

Report

Course: **CPS 510**
Database Systems I
Semester: **F2024**
Instructor: Abdolreza Abhari

TA: Kiana Kheiri

Section: **03**

Prepared by-

Parnia Zare
Saanika Mahajan
Samantha Lo Papa

Abstract

This project aims to develop a robust database system using Oracle DB, focusing on advanced SQL queries, normalization, and Unix shell scripting for menu-driven operations. The design emphasizes consistency across schema, functional dependencies, and normalization stages. Additionally, optional UI integration demonstrates practical application development.

TABLE OF CONTENTS Page number

1. Introduction

The Ride and Pick-up DBMS is a robust and scalable solution to streamline transportation and delivery services. Ensuring secure transactions and efficient handling of high volume of requests, it empowers clients and drivers alike with a user-friendly interface. Key features such as a feedback management system, secure payment integration, and scalable architecture enable smooth and reliable operations, guaranteeing a seamless experience for both consumers and drivers. This system is not only efficient but also adaptable, making it a dependable choice for managing modern transportation and delivery needs.

---

2. ER/EER Diagram and Schema Design

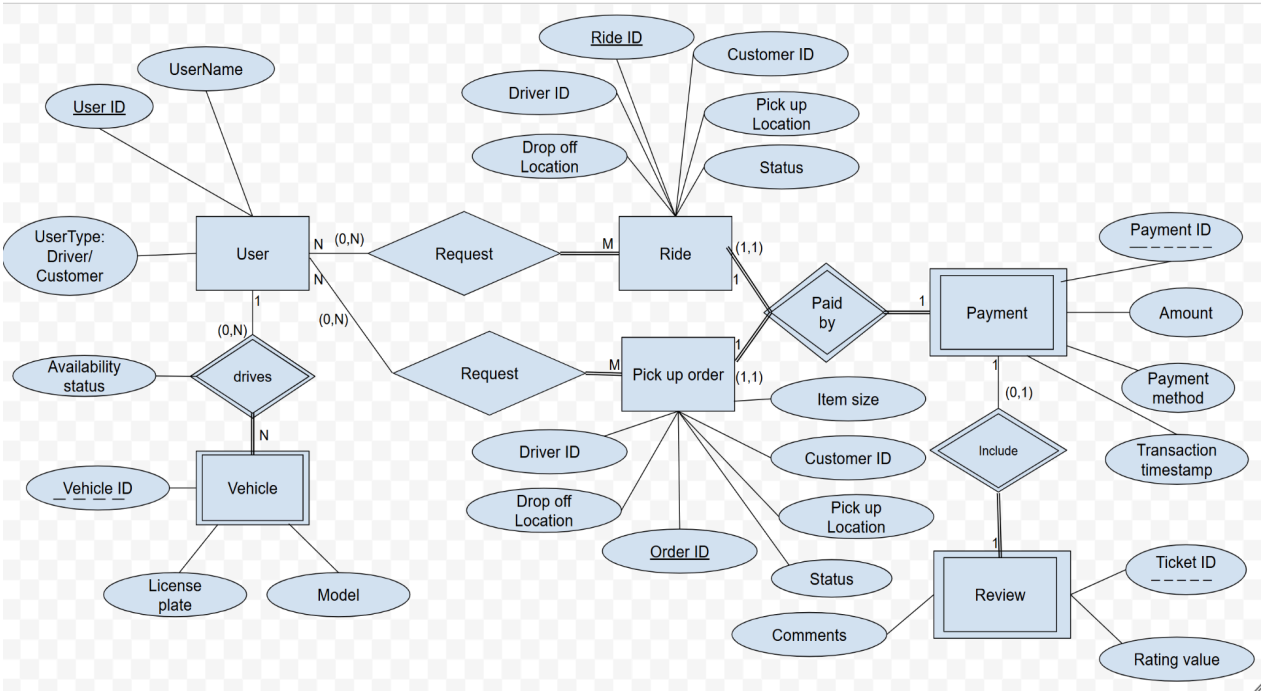


Fig 1- ER DIAGRAM

The Ride and Pick-up DBMS is a scalable and user-friendly system designed to streamline transportation and delivery services. The ER diagram illustrates key entities like **User**, **Vehicle**, **Ride**, **Pick-up Order**, **Payment**, and **Review**, highlighting their relationships. Users (drivers and customers) interact with vehicles, rides, and payments, while secure transactions and feedback management ensure efficiency and reliability. This system integrates all components to deliver smooth and scalable operations for clients and drivers.

- Schema Design:

User

UserID	UserName	UserType

Vehicle

VehicleID	License Plate	Model	DriverId

Ride

RideId	CustomerId	DriverId	PickUpLoc	DropOffLoc	Status

PickUpOrder

OrderId	CustomerId	DriverId	PickUpLoc	DropOff Loc	ItemSize	Status

Payment

PaymentId	Amount	PaymentMethod	TransactionTime Stamp

Review

TicketId	Comments	RatingValue	UserId

3. Normalization Process

➔ User Table

- **FDs**: `UserID → {UserType, UserName}`
- **1NF**: No repeating groups or multi-valued attributes.
- **2NF**: No partial dependencies as `UserID` is the primary key.
- **3NF**: No transitive dependencies. All non-prime attributes depend only on the primary key.
- **BCNF**: `UserID` is the sole determinant, satisfying BCNF.

---

2. Vehicle Table

- **FDs**: `VehicleID → {LicensePlate, Model, DriverID}`.
- **1NF**: No repeating groups.
- **2NF**: No partial dependencies as `VehicleID` is the primary key.
- **3NF**: No transitive dependencies; attributes depend directly on the primary key.
- **BCNF**: If `DriverID → UserName` is introduced, BCNF is violated as `DriverID` is not a superkey. To resolve:
    - Decompose:
        - `Vehicle(VehicleID, LicensePlate, Model, DriverID)`
        - `User(UserID, UserName)`.

---

3. Ride Table

- **FDs**: `RideID → {CustomerID, DriverID, PickupLocation, DropOffLocation, Status}`.
- **1NF**: No repeating groups
- **2NF**: No partial dependencies as `RideID` is the primary key.
- **3NF**: No transitive dependencies; all non-prime attributes depend directly on the primary key.
- **BCNF**: Satisfies BCNF as `RideID` is the sole determinant.

---

4. Payment Table

- **FDs**: `PaymentID → {Amount, PaymentMethod, TransactionTimeStamp}`.
- **1NF**: No repeating groups.
- **2NF**: No partial dependencies as `PaymentID` is the primary key.
- **3NF**: No transitive dependencies; all attributes depend only on `PaymentID`.
- **BCNF**: Satisfies BCNF as `PaymentID` is the only determinant.

---

5. PickUpOrder Table

- **FDs**: `OrderID → {DriverID, DropOffLocation, ItemSize, CustomerID, PickUpLocation}`.
- **1NF**: No repeating groups.
- **2NF**: No partial dependencies as `OrderID` is the primary key.
- **3NF**: Decompose into:
    - `PickUpOrder(OrderID, DriverID, CustomerID, ItemSize)`
    - `CustomerPickUp(CustomerID, PickUpLocation)`
    - `DriverDropOff(DriverID, DropOffLocation)`.
- **BCNF**: After decomposition, all relations satisfy BCNF.

---

6. Review Table

- **FDs**: `TicketID → {Comments, RatingValue}`.
- **1NF**: No repeating groups.
- **2NF**: No partial dependencies as `TicketID` is the primary key.
- **3NF**: No transitive dependencies; attributes depend only on `TicketID`.
- **BCNF**: Satisfies BCNF as `TicketID` is the sole determinant.

Algorithms Used for Normalization

➢ **Identify Functional Dependencies**:
  - List all FDs explicitly.

➢ **Check Candidate Keys**:
  - Identify the superkeys and primary keys for each table.

➢ **1NF Verification**:
  - Ensure no multi-valued attributes or repeating groups.

➢ **2NF Verification**:
  - Eliminate partial dependencies by ensuring all non-prime attributes are fully dependent on the primary key.

➢ **3NF Verification**:
  - Eliminate transitive dependencies; no non-prime attribute should depend on another non-prime attribute.

➢ **BCNF Verification**:
  - Ensure every determinant is a superkey. If violated, decompose the table.

This normalization process ensures the database is efficient, avoids redundancy, and preserves data integrity across all tables.

4. Database Construction

USER TABLE-

	USERID	USERNAME	USERTYPE
1	1234	Jane	Customer
2	1235	John	Customer
3	1236	Eve	Driver
4	1237	Adam	Driver
5	1238	Sarah	Customer
6	1239	Mike	Driver

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	USERID	VARCHAR2(10 BYTE)	No	(null)	1	(null)
2	USERNAME	VARCHAR2(25 BYTE)	Yes	(null)	2	(null)
3	USERTYPE	VARCHAR2(10 BYTE)	Yes	(null)	3	(null)

VEHICLE TABLE-

	VEHICLEID	LICENSEPLATE	Model	DRIVERID
1	4567	BUW6787	Toyota	1237
2	4568	LHY6875	Honda	1236

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	VEHICLEID	VARCHAR2(10 BYTE)	No	(null)	1	(null)
2	LICENSEPLATE	VARCHAR2(7 BYTE)	Yes	(null)	2	(null)
3	Model	VARCHAR2(10 BYTE)	Yes	(null)	3	(null)
4	DRIVERID	VARCHAR2(10 BYTE)	Yes	(null)	4	(null)

RIDE TABLE-

	RIDEID	CUSTOMERID	DRIVERID	PICKUPLOCATION	DROPOFFLOCATION	STATUS
1	R001	1238	1239	LocationA	LocationB	IN PRO...
2	R002	1235	1237	LocationC	LocationD	IN PRO...
3	R003	1234	1237	LocationA	LocationB	IN PRO...
4	R004	1235	1236	LocationA	LocationB	IN PRO...
5	R005	1238	1239	LocationF	LocationH	COMPLETE
6	R006	1234	1237	LocationS	LocationU	COMPLETE
7	R007	1235	1236	LocationA	LocationC	COMPLETE

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	RIDEID	VARCHAR2(10 BYTE)	No	(null)	1	(null)
2	CUSTOMERID	VARCHAR2(10 BYTE)	Yes	(null)	2	(null)
3	DRIVERID	VARCHAR2(10 BYTE)	Yes	(null)	3	(null)
4	PICKUPLOCATION	VARCHAR2(20 BYTE)	Yes	(null)	4	(null)
5	DROPOFFLOCATION	VARCHAR2(20 BYTE)	Yes	(null)	5	(null)
6	STATUS	VARCHAR2(20 BYTE)	Yes	'IN PROGRESS'	6	(null)

PICK UP ORDER TABLE-

	DRIVERID	DROPOFFLOCATION	ORDERID	ITEMSIZE	CUSTOMERID	PICKUPLOCATION	STATUS
1	1237	LocationF	3	Small	1234	LocationE	IN PROGRESS
2	1237	LocationB	1	Large	1235	LocationA	IN PROGRESS
3	1236	LocationD	2	Medium	1234	LocationC	IN PROGRESS

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	DRIVERID	VARCHAR2(10 BYTE)	Yes	(null)	1	(null)
2	DROPOFFLOCATION	VARCHAR2(100 BYTE)	Yes	(null)	2	(null)
3	ORDERID	VARCHAR2(10 BYTE)	No	(null)	3	(null)
4	ITEMSIZE	VARCHAR2(20 BYTE)	Yes	(null)	4	(null)
5	CUSTOMERID	VARCHAR2(10 BYTE)	Yes	(null)	5	(null)
6	PICKUPLOCATION	VARCHAR2(100 BYTE)	Yes	(null)	6	(null)
7	STATUS	VARCHAR2(20 BYTE)	Yes	'IN PROGRESS'	7	(null)

PAYMENT TABLE-

	PAYMENTID	AMOUNT	PAYMENTMETHOD	TRANSACTIONTIMESTAMP
1	4321	7	Credit	2024-09-19 07:52:41
2	4322	15.3	Debit	2024-09-19 11:36:16
3	4323	14.33	Credit	2024-09-03 09:32:34
4	4324	20	Debit	2022-10-01 11:00:00
5	4325	15	Debit	2022-10-02 07:30:00
6	4326	25	Mastercard	2022-10-03 10:50:00

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	PAYMENTID	VARCHAR2(10 BYTE)	No	(null)	1	(null)
2	AMOUNT	NUMBER	Yes	(null)	2	(null)
3	PAYMENTMETHOD	VARCHAR2(10 BYTE)	Yes	(null)	3	(null)
4	TRANSACTIONTIMESTAMP	VARCHAR2(20 BYTE)	Yes	(null)	4	(null)

REVIEW TABLE-

	TICKETID	COMMENTS	RATINGVALUE	USERID
1	T001	Good service	5	1234
2	T002	Good driver	5	1236
3	T003	Late driver	1	1237
4	T004	Good Passenger	1	1235

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	TICKETID	VARCHAR2(10 BYTE)	No	(null)	1	(null)
2	COMMENTS	VARCHAR2(20 BYTE)	Yes	(null)	2	(null)
3	RATINGVALUE	NUMBER	Yes	5	3	(null)
4	USERID	VARCHAR2(10 BYTE)	Yes	(null)	4	(null)

- SQL Script Sample:
  - ➔ CREATE TABLE PICK_UP_ORDER (
    DRIVER_ID REFERENCES "user"(UserID),
    DROP_OFF_LOCATION VARCHAR2(100),
    ORDER_ID NUMBER PRIMARY KEY,
    ITEM_SIZE VARCHAR2(20),
    CUSTOMER_ID REFERENCES "user"(UserID),
    PICK_UP_LOCATION VARCHAR2(100));
  - ➔ INSERT INTO PICK_UP_ORDER (DRIVER_ID, DROP_OFF_LOCATION, ORDER_ID, ITEM_SIZE, CUSTOMER_ID, PICK_UP_LOCATION) VALUES ('1237', 'LocationF', 3, 'Small', '1234', 'LocationE');

5. SQL Queries

- Simple Queries:

1. SELECT * FROM review WHERE RatingValue = 5;

	TICKETID	COMENTS	RATINGVALUE	USERID
1	T001	Good service	5	1234
2	T002	Good driver	5	1236

2. SELECT RatingValue, COUNT(*) AS ReviewCount FROM review GROUP BY RatingValue ORDER BY ReviewCount DESC;

	RATINGVALUE	REVIEWCOUNT
1	5	2
2	1	2

3. SELECT DISTINCT PickupLocation FROM ride ORDER BY PickupLocation;
4. SELECT * FROM ride WHERE CustomerID = ‘1234';
5. SELECT Status, COUNT(*) AS RideCount FROM ride GROUP BY Status ORDER BY RideCount DESC;
6. SELECT * FROM payment WHERE PaymentMethod = 'Credit';
7. SELECT SUM(Amount) AS TotalPayments FROM payment;
8. SELECT * FROM PICK_UP_ORDER WHERE DRIVER_ID = '1237';
9. SELECT * FROM PICK_UP_ORDER WHERE ITEM_SIZE = 'Large';
10. SELECT * FROM "user" WHERE UserType = 'Driver' AND RatingValue >= 4;
11. SELECT COUNT(*) AS TotalCustomers FROM "user" WHERE UserType = 'Customer';
12. SELECT * FROM vehicle WHERE "Model" = 'Toyota';
13. SELECT * FROM vehicle WHERE UserID = '1236';
14. SELECT DISTINCT "Model" FROM vehicle ORDER BY "Model";
15. SELECT D.USERNAME AS DRIVER_NAME FROM USER_TABLE D WHERE D.USERTYPE = 'Driver' AND EXISTS (SELECT 1 FROM REVIEW_TABLE R WHERE R.USERID = D.USERID);
16. SELECT C.USERNAME FROM USER_TABLE C JOIN RIDE_TABLE R ON C.USERID = R.CUSTOMERID MINUS SELECT U.USERNAME FROM USER_TABLE U JOIN REVIEW_TABLE V ON U.USERID = V.USERID;
17. SELECT D.USERNAME AS DRIVER_NAME, COUNT(R.RIDEID) AS TOTAL_RIDES FROM USER_TABLE D JOIN RIDE_TABLE R ON D.USERID = R.DRIVERID WHERE D.USERTYPE = 'Driver' GROUP BY D.USERNAME;

6. Views and Joins

```
CREATE VIEW DistinctDriverLocations AS
SELECT DISTINCT p.DRIVER_ID, p.PICK_UP_LOCATION, p.DROP_OFF_LOCATION
FROM PICK_UP_ORDER p;
```

	DRIVER_ID	PICK_UP_LOCATION	DROP_OFF_LOCATION
1	1237	LocationE	LocationF
2	1236	LocationC	LocationD
3	1237	LocationA	LocationB

```
CREATE VIEW RidesPerDriver AS
SELECT r.DriverID, COUNT(r.RideID) AS TotalRides
FROM ride r
WHERE r.DriverID IN (
    SELECT u.UserID
    FROM "user" u
    WHERE u.UserType = 'Driver'
)
GROUP BY r.DriverID;
```

	DRIVERID	TOTALRIDES
1	1236	4
2	1237	2

7. Unix Shell Scripting

- Menu Design:
    - Include shell scripts demonstrating:
        - Database creation.
        - Data insertion.
        - Executing simple and advanced queries.

Code for menu.sh:

```
#!/bin/s
MainMenu()
{
while [ "$CHOICE" != "START" ]
do
clear
echo
"============================================================="
echo "| Oracle All Inclusive Tool
|"
echo "| Main Menu - Select Desired Operation(s):
|"
echo "| <CTRL-Z Anytime to Enter Interactive CMD Prompt>
|"
echo "-------------------------------------------------------------
----"
echo " $IS_SELECTEDM M) View Manual"
echo " "
echo " $IS_SELECTED1 1) Drop Tables"
echo " $IS_SELECTED2 2) Create Tables"
echo " $IS_SELECTED3 3) Populate Tables"
echo " $IS_SELECTED4 4) Query Tables"
echo " "
echo " $IS_SELECTEDX X) Force/Stop/Kill Oracle DB"
echo " "
echo " $IS_SELECTEDE E) End/Exit"
echo "Choose: "
read CHOICE
if [ "$CHOICE" == "0" ]
then
echo "Nothing Here"
elif [ "$CHOICE" == "1" ]
then
bash drop_tables.sh
Pause
elif [ "$CHOICE" == "2" ]
then
bash create_tables.sh
Pause
elif [ "$CHOICE" == "3" ]
then
bash populate_tables.sh
Pause
elif [ "$CHOICE" == "4" ]
then
bash queries.sh
Pause
elif [ "$CHOICE" == "E" ]
then
exit
fi
done
}
#--COMMENTS BLOCK--
# Main Program
                                                    1,9           Top
```

```
#--COMMENTS BLOCK--
ProgramStart()
{
StartMessage
while [ 1 ]
do
MainMenu
done
}
ProgramStart
                                                    65,9          Bot
```

Code for drop_tables.sh:

```
#!/bin/sh


#export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
sqlplus64 "pzare/12072815@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.scs.r
yerson.ca)(Port=1521))(CONNECT_DATA=(SID=orcl)))" <<EOF

DROP TABLE PAYMENT CASCADE CONSTRAINTS;

DROP TABLE PICKUPORDER CASCADE CONSTRAINTS;

DROP TABLE REVIEW CASCADE CONSTRAINTS;

DROP TABLE RIDE CASCADE CONSTRAINTS;

DROP TABLE "user" CASCADE CONSTRAINTS;

DROP TABLE VEHICLE CASCADE CONSTRAINTS;

DROP TABLE PICKUPREQUEST CASCADE CONSTRAINTS;

DROP TABLE RIDEREQUEST CASCADE CONSTRAINTS;
exit;

EOF


~
```

Code for create_tables.sh:

```
#!/bin/sh

#export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
sqlplus64 "pzare/12072815@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.scs.r
yerson.ca)(Port=1521))(CONNECT_DATA=(SID=orcl)))" <<EOF

CREATE TABLE "user"(
    UserID VARCHAR2(10) NOT NULL UNIQUE,
    UserName VARCHAR2(25),
    UserType VARCHAR2(10) CHECK (UserType in ('Customer', 'Driver'))
);

CREATE TABLE review(
    TicketID VARCHAR2(10) NOT NULL UNIQUE,
    Comments VARCHAR2(20),
    RatingValue NUMBER DEFAULT 5,
    UserID VARCHAR2(10) REFERENCES "user"(UserID)
);

CREATE TABLE ride(
    RideID VARCHAR2(10) NOT NULL UNIQUE,
    CustomerID VARCHAR2(10) REFERENCES "user"(UserID),
    DriverID VARCHAR2(10) REFERENCES "user"(UserID),
    PickupLocation VARCHAR2(20),
    DropOffLocation VARCHAR2(20),
    Status VARCHAR2(20) DEFAULT 'AVAILABLE'
);

CREATE TABLE payment(
    PaymentID VARCHAR2(10) NOT NULL UNIQUE,
    Amount NUMBER,
    PaymentMethod VARCHAR2(10),
    TransactionTimeStamp VARCHAR2(20)
);

CREATE TABLE pickUpOrder (
    DriverID VARCHAR2(10) REFERENCES "user"(UserID),
    DropOffLocation VARCHAR2(100),
    OrderID VARCHAR2(10) PRIMARY KEY,
    ItemSize VARCHAR2(20),
    CustomerID VARCHAR2(10) REFERENCES "user"(UserID),
    PickUpLocation VARCHAR2(100)
);

CREATE TABLE vehicle (
    VehicleID VARCHAR2(10) NOT NULL UNIQUE,
    LicensePlate VARCHAR2(7),
    "Model" VARCHAR2(10),
    DriverID VARCHAR2(10) REFERENCES "user"(UserID)
);

CREATE TABLE rideRequest (
    CustomerID VARCHAR2(10) REFERENCES "user"(UserID),
    RideID VARCHAR2(10) REFERENCES ride(RideID)
);
                                                          1,1           Top
```

Code for populate_tables.sh:

```
#!/bin/sh


#export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
sqlplus64 "pzare/12072815@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.scs.ryerson.ca)(Port=1521))(CONNECT_DATA=(SID=orcl)))" <<EOF

INSERT INTO "user"(UserID, UserName, UserType) VALUES (1234, 'Jane', 'Customer');
INSERT INTO "user"(UserID, UserName, UserType) VALUES (1235, 'John', 'Customer');
INSERT INTO "user"(UserID, UserName, UserType) VALUES (1236, 'Eve', 'Driver');
INSERT INTO "user"(UserID, UserName, UserType) VALUES (1237, 'Adam', 'Driver');

INSERT INTO vehicle(VehicleID, LicensePlate, "Model", DriverID) VALUES (4567, 'BUW6787', 'Toyota', 1237);
INSERT INTO vehicle(VehicleID, LicensePlate, "Model", DriverID) VALUES (4568, 'LHY6875', 'Honda', 1236);

INSERT INTO payment(PaymentID, Amount, PaymentMethod, TransactionTimeStamp) VALUES (4321, 7.00, 'Credit', '2024-09-19 07:52:41');
INSERT INTO payment(PaymentID, Amount, PaymentMethod, TransactionTimeStamp) VALUES (4322, 15.30, 'Debit', '2024-09-19 11:36:16');
INSERT INTO payment(PaymentID, Amount, PaymentMethod, TransactionTimeStamp) VALUES (4323, 14.33, 'Credit', '2024-09-03 09:32:34');

INSERT INTO pickUpOrder (DriverID, DropOffLocation, OrderID, ItemSize, CustomerID, PickupLocation)
VALUES ('1237', 'LocationF', 3, 'Small', '1234', 'LocationE');

INSERT INTO pickUpOrder (DriverID, DropOffLocation, OrderID, ItemSize, CustomerID, PickupLocation)
VALUES ('1237', 'LocationB', 1, 'Large', '1235', 'LocationA');

INSERT INTO pickUpOrder (DriverID, DropOffLocation, OrderID, ItemSize, CustomerID, PickupLocation)
VALUES ('1236', 'LocationD', 2, 'Medium', '1234', 'LocationC');

INSERT INTO review(TicketID, Comments, RatingValue, UserID) VALUES ('T001', 'Good service', 5,1234);
INSERT INTO review(TicketID, Comments, RatingValue, UserID) VALUES ('T002', 'Good driver', 5,1236);
INSERT INTO review(TicketID, Comments, RatingValue, UserID) VALUES ('T003', 'Late driver', 1,1237);
INSERT INTO review(TicketID, Comments, RatingValue, UserID) VALUES ('T004', 'Good Passenger', 1,1235);

INSERT INTO ride(RideID, CustomerID, DriverID, PickupLocation, DropOffLocation, Status)
VALUES ('R001', '1236', '1234', 'LocationA', 'LocationB', 'Available');

INSERT INTO ride(RideID, CustomerID, DriverID, PickupLocation, DropOffLocation, Status)
VALUES ('R002', '1235', '1237', 'LocationC', 'LocationD', 'Available');

INSERT INTO ride(RideID, CustomerID, DriverID, PickupLocation, DropOffLocation, Status)
VALUES ('R003', '1234', '1237', 'LocationA', 'LocationB', 'Available');

INSERT INTO ride(RideID, CustomerID, DriverID, PickupLocation, DropOffLocation, Status)
VALUES ('R004', '1235', '1236', 'LocationA', 'LocationB', 'Available');

exit;
EOF
```

Code for queries_tables.sh:

```
#!/bin/sh
#export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
sqlplus64 "pzare/12072815@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.scs.ryerson.ca)(Port=1521))(CONNECT_DATA=(SID=orcl)))" <<EOF
SELECT * FROM "user" u, review r WHERE
    UserType = 'Driver' AND
    u.UserID = r.UserID AND
    RatingValue >= 4;
SELECT COUNT(*) AS TotalCustomers FROM "user" WHERE UserType = 'Customer';

SELECT * FROM vehicle WHERE "Model" = 'Toyota';
SELECT * FROM vehicle WHERE DriverID = '1236';
SELECT DISTINCT "Model" FROM vehicle ORDER BY "Model";

SELECT * FROM payment WHERE PaymentMethod = 'Credit';
SELECT SUM(Amount) AS TotalPayments FROM payment;

SELECT * FROM review WHERE RatingValue = 5;
SELECT RatingValue, COUNT(*) AS ReviewCount FROM review GROUP BY RatingValue ORDER BY ReviewCount DESC;

SELECT DISTINCT PickupLocation FROM ride ORDER BY PickupLocation;
SELECT * FROM ride WHERE CustomerID = '1234';
SELECT Status, COUNT(*) AS RideCount FROM ride GROUP BY Status ORDER BY RideCount DESC;

SELECT * FROM PickupOrder WHERE DriverID = '1237';
SELECT * FROM PickupOrder WHERE ItemSize = 'Large';
exit;
EOF
~
```

- Screenshots of shell menu and query execution results.

```
  M) View Manual

  1) Drop Tables
  2) Create Tables
  3) Populate Tables
  4) Query Tables

  X) Force/Stop/Kill Oracle DB

  E) End/Exit
Choose:
```

1) Drop Tables:

```
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SQL>
Table dropped.

SQL> SQL>
Table dropped.

SQL> SQL>
Table dropped.

SQL> SQL>
Table dropped.

SQL> SQL>
Table dropped.

SQL> SQL>
Table dropped.

SQL> SQL>
Table dropped.

SQL> SQL>
Table dropped.

SQL> Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0
 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
```

2) Create Tables :

```
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SQL>   2    3    4    5
Table created.

SQL> SQL>   2    3    4    5    6
Table created.

SQL> SQL>   2    3    4    5    6    7    8
Table created.

SQL> SQL>   2    3    4    5    6
Table created.

SQL> SQL>   2    3    4    5    6    7    8
Table created.

SQL> SQL>   2    3    4    5    6
Table created.

SQL> SQL>   2    3    4
Table created.

SQL> SQL>   2    3    4
Table created.

SQL> SQL> Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.
0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
```

3) Populate Tables

```
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL> SQL>
1 row created.

SQL>
1 row created.

SQL> SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL> SQL>   2
1 row created.

SQL> SQL>   2
1 row created.

SQL> SQL>   2
1 row created.

SQL> SQL>
1 row created.

SQL>
1 row created.

SQL>
1 row created.
```

4) Query Tables

```
SQL>   2    3    4
USERID     USERNAME                  USERTYPE   TICKETID   COMMENTS
---------- ------------------------- ---------- ---------- --------------------
RATINGVALUE USERID
----------- ----------
1236       Eve                       Driver     T002       Good driver
         5 1236


SQL>
TOTALCUSTOMERS
--------------
             2

SQL> SQL>
VEHICLEID  LICENSE Model      DRIVERID
---------- ------- ---------- ----------
4567       BUW6787 Toyota     1237

SQL>
VEHICLEID  LICENSE Model      DRIVERID
---------- ------- ---------- ----------
4568       LHY6875 Honda      1236

SQL>
Model
----------
Honda
Toyota

SQL> SQL>
PAYMENTID      AMOUNT PAYMENTMET TRANSACTIONTIMESTAMP
---------- ---------- ---------- --------------------
4321                7 Credit     2024-09-19 07:52:41
4323            14.33 Credit     2024-09-03 09:32:34

SQL>
TOTALPAYMENTS
-------------
        36.63

SQL> SQL>
TICKETID   COMMENTS             RATINGVALUE USERID
---------- -------------------- ----------- ----------
T001       Good service                   5 1234
T002       Good driver                    5 1236
```

```
SQL> SQL>
PICKUPLOCATION
--------------------
LocationA
LocationC

SQL>
RIDEID     CUSTOMERID DRIVERID   PICKUPLOCATION       DROPOFFLOCATION
---------- ---------- ---------- -------------------- --------------------
STATUS
--------------------
R003       1234       1237       LocationA            LocationB
Available


SQL>
STATUS               RIDECOUNT
-------------------- ----------
Available                     4

SQL> SQL>
DRIVERID
----------
DROPOFFLOCATION
--------------------------------------------------------------------------------
ORDERID    ITEMSIZE             CUSTOMERID
---------- -------------------- ----------
PICKUPLOCATION
--------------------------------------------------------------------------------
1237
LocationF
3          Small                1234
LocationE


DRIVERID
----------
DROPOFFLOCATION
--------------------------------------------------------------------------------
ORDERID    ITEMSIZE             CUSTOMERID
---------- -------------------- ----------
PICKUPLOCATION
--------------------------------------------------------------------------------
1237
LocationB
1          Large                1235
LocationA
```

```
SQL>
DRIVERID
----------
DROPOFFLOCATION
--------------------------------------------------------------------------------
ORDERID    ITEMSIZE             CUSTOMERID
---------- -------------------- ----------
PICKUPLOCATION
--------------------------------------------------------------------------------
1237
LocationB
1          Large                1235
LocationA
```

8. Relational Algebra for Queries

1. **SQL**: SELECT * FROM review WHERE RatingValue = 5;

RA: $\sigma_{Rating\ Value = 5}\ (REVIEW)$

2. **SQL**: SELECT RatingValue, COUNT(*) AS ReviewCount FROM review GROUP BY RatingValue ORDER BY ReviewCount DESC;

RA: $RatingValue\ F_{COUNT(*) \rightarrow ReviewCount}(REVIEW)$

3. **SQL**: SELECT DISTINCT PickupLocation FROM ride ORDER BY PickupLocation;

RA: $\Pi_{PickupLocation}(RIDE)$

4. **SQL**: SELECT * FROM ride WHERE CustomerID = '1234';

RA: $\sigma_{CustomerID='1234'}\ (RIDE)$

5. **SQL**: SELECT Status, COUNT(*) AS RideCount FROM ride GROUP BY Status ORDER BY RideCount DESC;

RA: $Status\ F_{COUNT(*) \rightarrow RideCount}(RIDE)$

6. **SQL**: SELECT * FROM payment WHERE PaymentMethod = 'Credit';

RA: $\sigma_{PaymentMethod='Credit'}\ (PAYMENT)$

7. **SQL**: SELECT SUM(Amount) AS TotalPayments FROM payment;

RA: $F_{SUM(Amount)\rightarrow TotalPayments}(PAYMENT)$

8. **SQL**: SELECT * FROM PICK_UP_ORDER WHERE DRIVER_ID = '1237';

RA: $\sigma_{DriverID='1237'}(PICK\_UP\_ORDER)$

9. **SQL**: SELECT * FROM PICK_UP_ORDER WHERE ITEM_SIZE = 'Large';

RA: $\sigma_{ItemSize='Large'}(PICK\_UP\_ORDER)$

10. **SQL**: SELECT * FROM "user" WHERE UserType = 'Driver' AND RatingValue >= 4;

RA: $\sigma_{UserType='Driver' \wedge RatingValue \geq 4}(USER)$

11. **SQL**: SELECT COUNT(*) AS TotalCustomers FROM "user" WHERE UserType = 'Customer';

RA: $F_{COUNT(*)\rightarrow TotalCustomers}(\sigma_{UserType='Customer'}(USER))$

12. **SQL**: SELECT * FROM vehicle WHERE "Model" = 'Toyota';

RA: $\sigma_{Model='Toyota'}(VEHICLE)$

13. **SQL**: SELECT * FROM vehicle WHERE UserID = '1236';

RA: $\sigma_{UserID='1236'}(VEHICLE)$

14. **SQL**: SELECT DISTINCT "Model" FROM vehicle ORDER BY "Model";

RA: $\Pi_{Model}(VEHICLE)$

15. **SQL**: SELECT D.USERNAME AS DRIVER_NAME FROM USER_TABLE D WHERE D.USERTYPE = 'Driver' AND EXISTS (SELECT 1 FROM REVIEW_TABLE R WHERE R.USERID = D.USERID);

RA: $\Pi_{USERNAME}(\sigma_{UserType='Driver'}(USER\_TABLE) \bowtie REVIEW\_TABLE)$

16. **SQL**: SELECT C.USERNAME FROM USER_TABLE C JOIN RIDE_TABLE R ON C.USERID = R.CUSTOMERID MINUS SELECT U.USERNAME FROM USER_TABLE U JOIN REVIEW_TABLE V ON U.USERID = V.USERID;

RA:
$\Pi_{USERNAME}(USER\_TABLE \bowtie RIDE\_TABLE) - \Pi_{USERNAME}(USER\_TABLE) \bowtie REVIEW\_TABLE)$

17. **SQL**: SELECT D.USERNAME AS DRIVER_NAME, COUNT(R.RIDEID) AS TOTAL_RIDES FROM USER_TABLE D JOIN RIDE_TABLE R ON D.USERID = R.DRIVERID WHERE D.USERTYPE = 'Driver' GROUP BY D.USERNAME;

RA: $USERNAME,\ COUNT(RIDEID)(\sigma_{UserType='Driver'}(USER\_TABLE) \bowtie REVIEW\_TABLE))$

---

9. User Interface Development

- UI Design:
  - Brief description of UI functionality and design process.
  - Screenshots of the text-based menus from the Unix shell.
- Optional GUI/Web Interface:
  - Description of the web/GUI interface (if implemented).
  - Screenshots and instructions for running the GUI.

Code for Python UI:

```
import oracledb

# Connect to Oracle Database
connection = oracledb.connect(
    user="system",
    password="510",
    dsn="localhost/xepdb1"
)
print("Successfully connected to Oracle Database")

cursor = connection.cursor()

create_statements = [
```

Here we are connecting to the Oracle XE database using oracledb.

The create_statements array contains all the CREATE statements for our tables.

There are similar arrays, drop_statements, inserts, and queries containing their corresponding sql statements as well.

```
def display_menu():
    print("\nMenu:")
    print("1. Create a Table")
    print("2. Drop a Table")
    print("3. Query Data")
    print("4. Insert Data")
    print("5. Update Data")
    print("6. Read Data")
    print("7. Exit")

def create_table():
    for create_statement in create_statements:
        try:
            cursor.execute(create_statement)
            print(f"Table '{create_statement}' created successfully.")
        except oracledb.DatabaseError as e:
            print(f"Error creating table '{create_statement}':", e)

def drop_table():
    for drop_statement in drop_statements:
        try:
            cursor.execute(drop_statement)
            print(f"Executed: {drop_statement}")
        except oracledb.DatabaseError as e:
            print(f"Error executing '{drop_statement}':", e)
```

We define the menu which simply displays the options to the user, we also have the create_table and drop_table functions which execute the sql statements in the arrays when called.

```
def query_data():
    for query in queries:
        try:
            # Execute the query
            cursor.execute(query)
            print(f"Executed: {query}")

            # Fetch and print the results
            rows = cursor.fetchall()
            if rows:
                for row in rows:
                    print(row)  # Print each row
            else:
                print("No results found.")

        except oracledb.DatabaseError as e:
            print(f"Error executing '{query}':", e)

def insert_data():
    for insert in inserts:
        try:
            cursor.execute(insert)
            connection.commit()  # Commit after each insert
            print(f"Executed: {insert}")
        except oracledb.DatabaseError as e:
            print(f"Error executing '{insert}':", e)
```

```
def update_data():
    try:
        # Example of updating a specific user's name
        user_id = input("Enter the UserID of the user to update: ").strip()
        new_username = input("Enter the new username: ").strip()

        # SQL statement to update the user's name based on UserID
        update_statement = """
            UPDATE users
            SET UserName = :1
            WHERE UserID = :2
        """

        # Execute the update
        cursor.execute(update_statement,  parameters: (new_username, user_id))

        # Commit the transaction
        connection.commit()
        print(f"User with UserID {user_id} updated to {new_username}.")
    except oracledb.DatabaseError as e:
        print(f"Error updating data:", e)
        connection.rollback()  # Rollback in case of error
```

```python
def read_data():
    try:
        # Ask the user for a table name to read data from
        table_name = input("Enter the table name you want to read data from: ").strip()

        # Ensure that table name is valid (not empty)
        if not table_name:
            print("Table name cannot be empty.")
            return

        # Check if the table exists in the database (basic query to check)
        cursor.execute(f"SELECT table_name FROM all_tables WHERE table_name = '{table_name.upper()}'")
        result = cursor.fetchone()

        if result is None:
            print(f"Table '{table_name}' does not exist.")
            return

        # Build the query dynamically to select all columns from the specified table
        query = f"SELECT * FROM {table_name}"

        # Execute the query
        cursor.execute(query)

        # Fetch and print the results
        rows = cursor.fetchall()
        if rows:
            print(f"Data from table {table_name}:")
            for row in rows:
                print(row)  # Print each row
```

```python
                print(row)  # Print each row
        else:
            print(f"No data found in table {table_name}.")

    except oracledb.DatabaseError as e:
        print(f"Error reading data from table '{table_name}':", e)

    except oracledb.DatabaseError as e:
        print(f"Error reading data", e)
```

Query, insert, update, and read functions that work similarly to create and drop.

```python
# Menu loop
while True:
    display_menu()
    choice = input("Enter your choice: ").strip()

    if choice == "1":
        create_table()
    elif choice == "2":
        drop_table()
    elif choice == "3":
        query_data()
    elif choice == "4":
        insert_data()
    elif choice == "5":
        update_data()
    elif choice == "6":
        read_data()
    elif choice == "7":
        print("Exiting program.")
        break
    else:
        print("Invalid choice, please try again.")

# Close resources
cursor.close()
connection.close()
print("Connection to Oracle Database closed.")
```

Function to display the menu until the user chooses to exit the program, then closes the connection afterwards.

Demo of the UI:

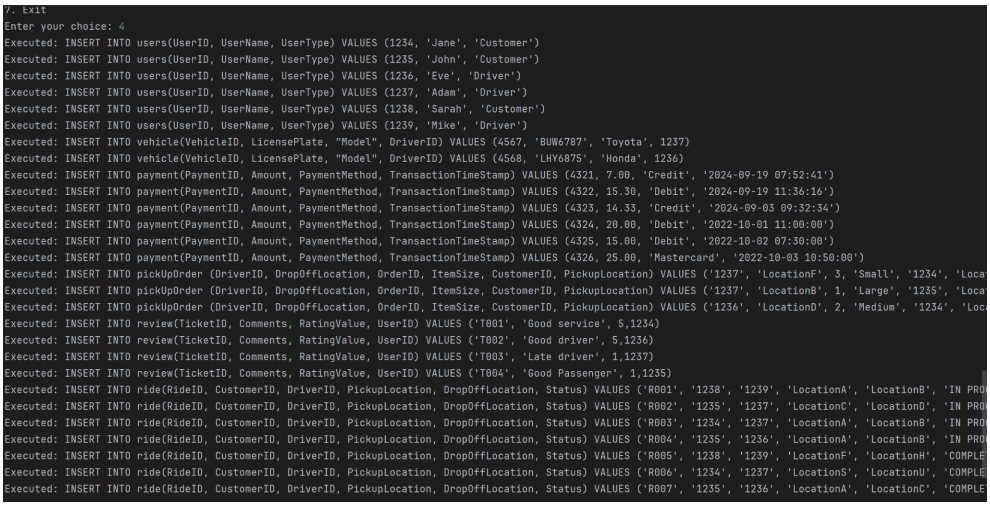
Dropping the tables

```
Menu:
1. Create a Table
2. Drop a Table
3. Query Data
4. Insert Data
5. Update Data
6. Read Data
7. Exit
Enter your choice: 2
Executed: DROP TABLE PAYMENT CASCADE CONSTRAINTS
Executed: DROP TABLE PICKUPORDER CASCADE CONSTRAINTS
Executed: DROP TABLE REVIEW CASCADE CONSTRAINTS
Executed: DROP TABLE RIDE CASCADE CONSTRAINTS
Executed: DROP TABLE users CASCADE CONSTRAINTS
Executed: DROP TABLE VEHICLE CASCADE CONSTRAINTS
Executed: DROP TABLE PICKUPREQUEST CASCADE CONSTRAINTS
Executed: DROP TABLE RIDEREQUEST CASCADE CONSTRAINTS
```

Creating the tables

```
Menu:
1. Create a Table
2. Drop a Table
3. Query Data
4. Insert Data
5. Update Data
6. Read Data
7. Exit
Enter your choice: 1
Table '
    CREATE TABLE users (
        UserID VARCHAR2(10) NOT NULL UNIQUE,
        UserName VARCHAR2(25),
        UserType VARCHAR2(10) CHECK (UserType in ('Customer', 'Driver'))
    )
    ' created successfully.
Table '
    CREATE TABLE review (
        TicketID VARCHAR2(10) NOT NULL UNIQUE,
        Comments VARCHAR2(20),
        RatingValue NUMBER DEFAULT 5,
        UserID VARCHAR2(10) REFERENCES users(UserID)
    )
    ' created successfully.
Table '
    CREATE TABLE ride (
        RideID VARCHAR2(10) NOT NULL UNIQUE,
        CustomerID VARCHAR2(10) REFERENCES users(UserID),
        DriverID VARCHAR2(10) REFERENCES users(UserID),
        PickupLocation VARCHAR2(20),
        DropOffLocation VARCHAR2(20)
```

Inserting data:



```
7. Exit
Enter your choice: 4
Executed: INSERT INTO users(UserID, UserName, UserType) VALUES (1234, 'Jane', 'Customer')
Executed: INSERT INTO users(UserID, UserName, UserType) VALUES (1235, 'John', 'Customer')
Executed: INSERT INTO users(UserID, UserName, UserType) VALUES (1236, 'Eve', 'Driver')
Executed: INSERT INTO users(UserID, UserName, UserType) VALUES (1237, 'Adam', 'Driver')
Executed: INSERT INTO users(UserID, UserName, UserType) VALUES (1238, 'Sarah', 'Customer')
Executed: INSERT INTO users(UserID, UserName, UserType) VALUES (1239, 'Mike', 'Driver')
Executed: INSERT INTO vehicle(VehicleID, LicensePlate, "Model", DriverID) VALUES (4567, 'BUW6787', 'Toyota', 1237)
Executed: INSERT INTO vehicle(VehicleID, LicensePlate, "Model", DriverID) VALUES (4568, 'LHY6875', 'Honda', 1236)
Executed: INSERT INTO payment(PaymentID, Amount, PaymentMethod, TransactionTimeStamp) VALUES (4321, 7.80, 'Credit', '2024-09-19 07:52:41')
Executed: INSERT INTO payment(PaymentID, Amount, PaymentMethod, TransactionTimeStamp) VALUES (4322, 15.30, 'Debit', '2024-09-19 11:36:16')
Executed: INSERT INTO payment(PaymentID, Amount, PaymentMethod, TransactionTimeStamp) VALUES (4323, 14.33, 'Credit', '2024-09-03 09:32:34')
Executed: INSERT INTO payment(PaymentID, Amount, PaymentMethod, TransactionTimeStamp) VALUES (4324, 20.00, 'Debit', '2022-10-01 11:00:00')
Executed: INSERT INTO payment(PaymentID, Amount, PaymentMethod, TransactionTimeStamp) VALUES (4325, 15.00, 'Debit', '2022-10-02 07:30:00')
Executed: INSERT INTO payment(PaymentID, Amount, PaymentMethod, TransactionTimeStamp) VALUES (4326, 25.00, 'Mastercard', '2022-10-03 10:50:00')
Executed: INSERT INTO pickUpOrder (DriverID, DropOffLocation, OrderID, ItemSize, CustomerID, PickupLocation) VALUES ('1237', 'LocationF', 3, 'Small', '1234', 'Loca
Executed: INSERT INTO pickUpOrder (DriverID, DropOffLocation, OrderID, ItemSize, CustomerID, PickupLocation) VALUES ('1237', 'LocationB', 1, 'Large', '1235', 'Loca
Executed: INSERT INTO pickUpOrder (DriverID, DropOffLocation, OrderID, ItemSize, CustomerID, PickupLocation) VALUES ('1236', 'LocationD', 2, 'Medium', '1234', 'Loc
Executed: INSERT INTO review(TicketID, Comments, RatingValue, UserID) VALUES ('T001', 'Good service', 5,1234)
Executed: INSERT INTO review(TicketID, Comments, RatingValue, UserID) VALUES ('T002', 'Good driver', 5,1236)
Executed: INSERT INTO review(TicketID, Comments, RatingValue, UserID) VALUES ('T003', 'Late driver', 1,1237)
Executed: INSERT INTO review(TicketID, Comments, RatingValue, UserID) VALUES ('T004', 'Good Passenger', 1,1235)
Executed: INSERT INTO ride(RideID, CustomerID, DriverID, PickupLocation, DropOffLocation, Status) VALUES ('R001', '1238', '1239', 'LocationA', 'LocationB', 'IN PRO
Executed: INSERT INTO ride(RideID, CustomerID, DriverID, PickupLocation, DropOffLocation, Status) VALUES ('R002', '1235', '1237', 'LocationC', 'LocationD', 'IN PRO
Executed: INSERT INTO ride(RideID, CustomerID, DriverID, PickupLocation, DropOffLocation, Status) VALUES ('R003', '1234', '1237', 'LocationA', 'LocationB', 'IN PRO
Executed: INSERT INTO ride(RideID, CustomerID, DriverID, PickupLocation, DropOffLocation, Status) VALUES ('R004', '1235', '1236', 'LocationA', 'LocationB', 'IN PRO
Executed: INSERT INTO ride(RideID, CustomerID, DriverID, PickupLocation, DropOffLocation, Status) VALUES ('R005', '1238', '1239', 'LocationF', 'LocationH', 'COMPLE
Executed: INSERT INTO ride(RideID, CustomerID, DriverID, PickupLocation, DropOffLocation, Status) VALUES ('R006', '1234', '1237', 'LocationS', 'LocationU', 'COMPLE
Executed: INSERT INTO ride(RideID, CustomerID, DriverID, PickupLocation, DropOffLocation, Status) VALUES ('R007', '1235', '1236', 'LocationA', 'LocationC', 'COMPLE
```

Querying data:

```
1. Create a Table
2. Drop a Table
3. Query Data
4. Insert Data
5. Update Data
6. Read Data
7. Exit
Enter your choice: 3
Executed: SELECT D.USERNAME AS DRIVER_NAME
    FROM users D
    WHERE D.USERTYPE = 'Driver'
    AND EXISTS (
        SELECT 1
        FROM review R
        WHERE R.USERID = D.USERID
    )
('Eve',)
('Adam',)
Executed: SELECT C.USERNAME
    FROM users C
    JOIN ride R ON C.USERID = R.CUSTOMERID
    MINUS
    SELECT U.USERNAME
    FROM users U
    JOIN review V ON U.USERID = V.USERID
('Sarah',)
Executed: SELECT USERNAME
    FROM users
    WHERE USERTYPE = 'Driver'
    UNION
    SELECT C.USERNAME
```

Updating data:

```
Menu:
1. Create a Table
2. Drop a Table
3. Query Data
4. Insert Data
5. Update Data
6. Read Data
7. Exit
Enter your choice: 5
Enter the UserID of the user to update: 1234
Enter the new username: fenjsfe
User with UserID 1234 updated to fenjsfe.
```

Reading data:

```
Menu:
1. Create a Table
2. Drop a Table
3. Query Data
4. Insert Data
5. Update Data
6. Read Data
7. Exit
Enter your choice: 6
Enter the table name you want to read data from: users
Data from table users:
('1234', 'fenjsfe', 'Customer')
('1235', 'John', 'Customer')
('1236', 'Eve', 'Driver')
('1237', 'Adam', 'Driver')
('1238', 'Sarah', 'Customer')
('1239', 'Mike', 'Driver')
```

Exiting the program:

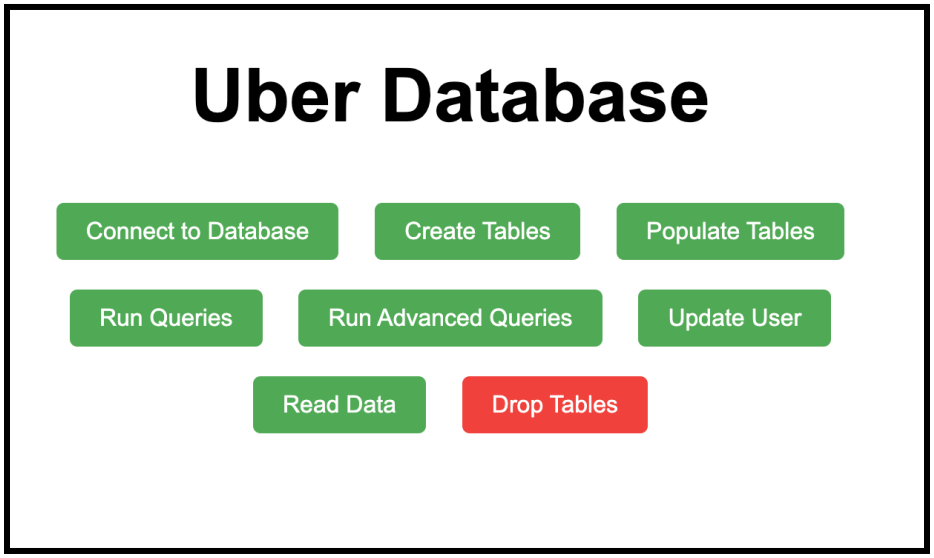
```
Menu:
1. Create a Table
2. Drop a Table
3. Query Data
4. Insert Data
5. Update Data
6. Read Data
7. Exit
Enter your choice: 7
Exiting program.
Connection to Oracle Database closed.

Process finished with exit code 0
```

---

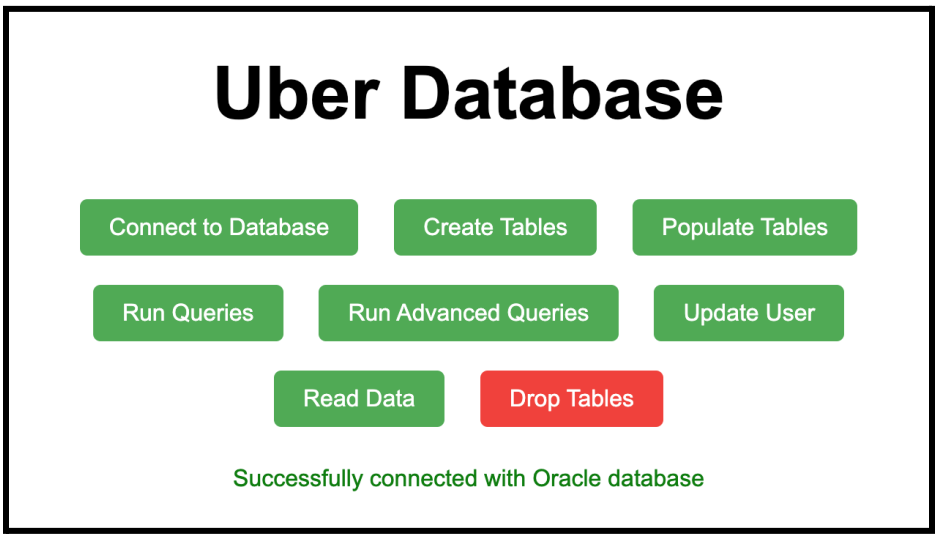
10. System Testing and Validation

Webpage link: https://webdev.scs.ryerson.ca/~slopapa/a9.php

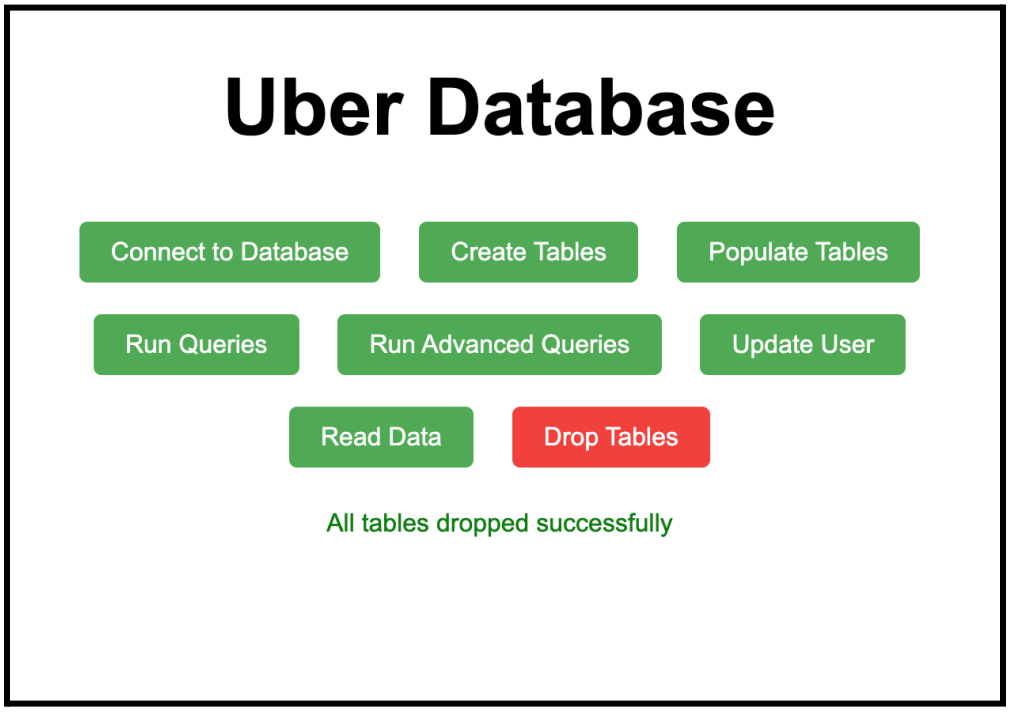
Screenshots of testing:



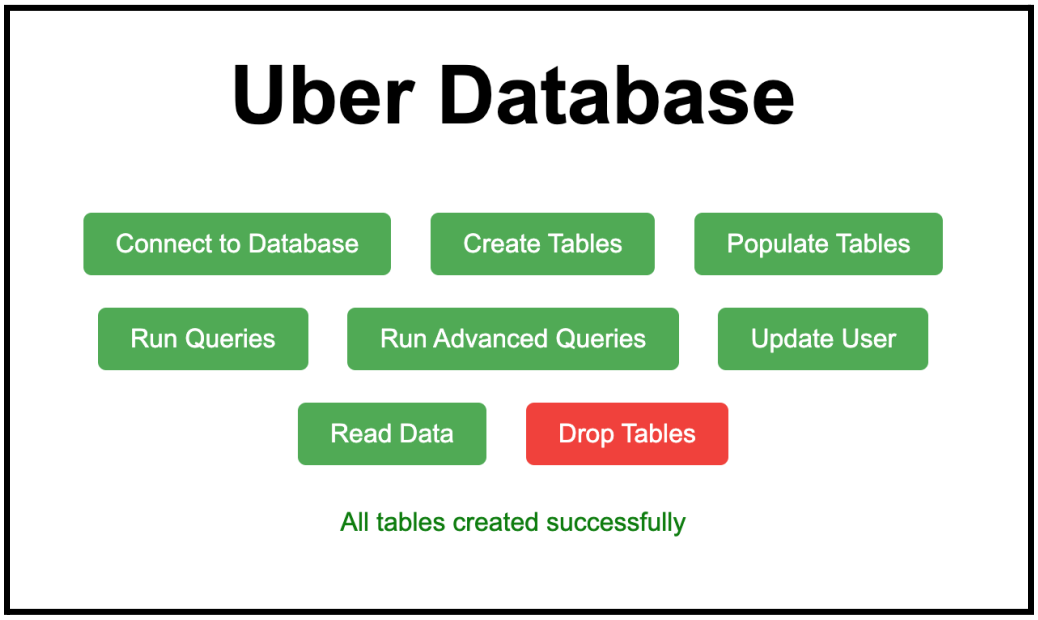
Connecting to Database:



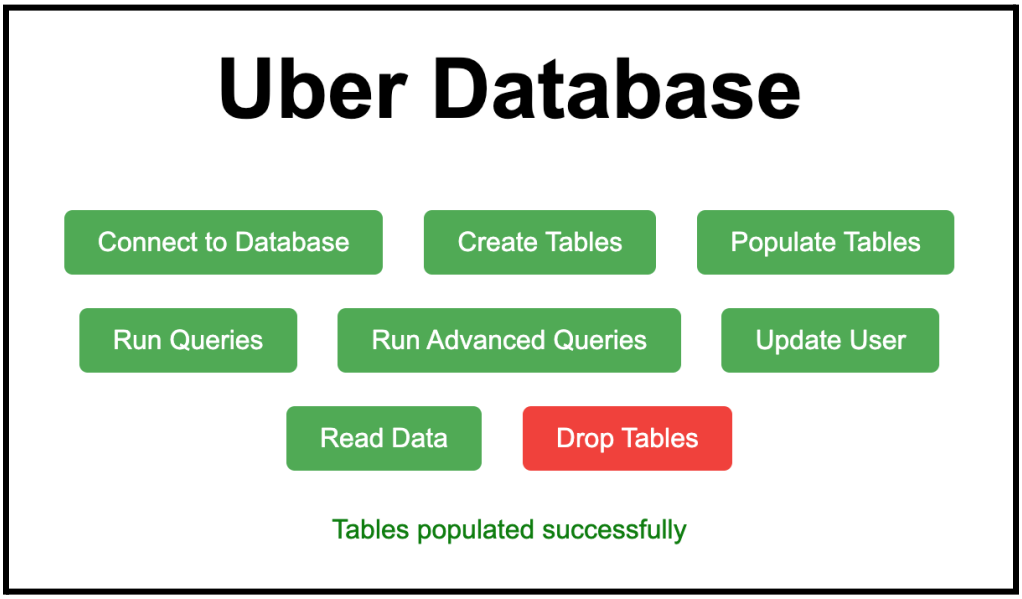
Drop Tables:



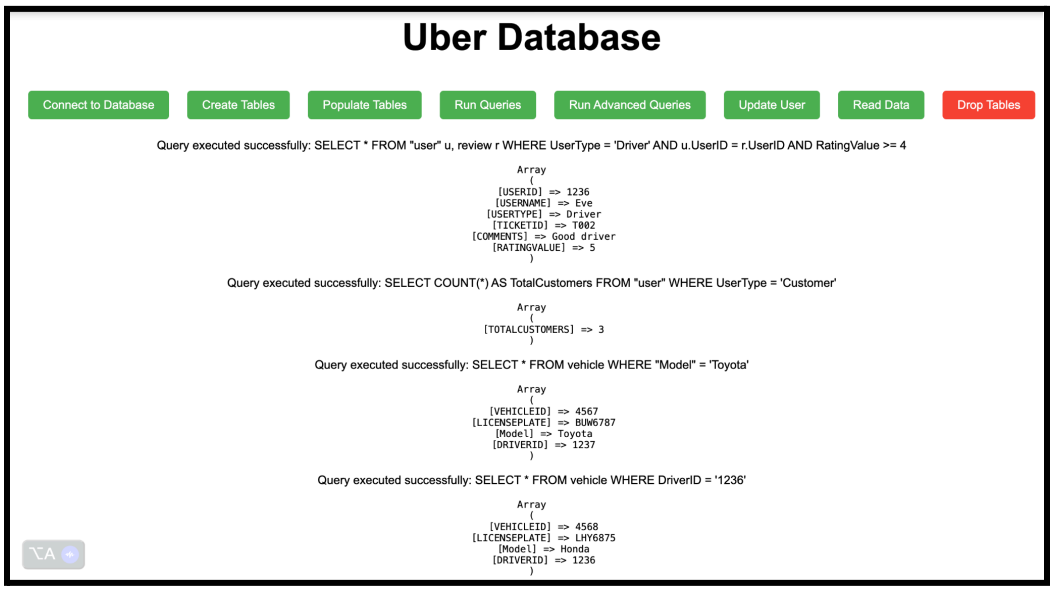
Creating tables:



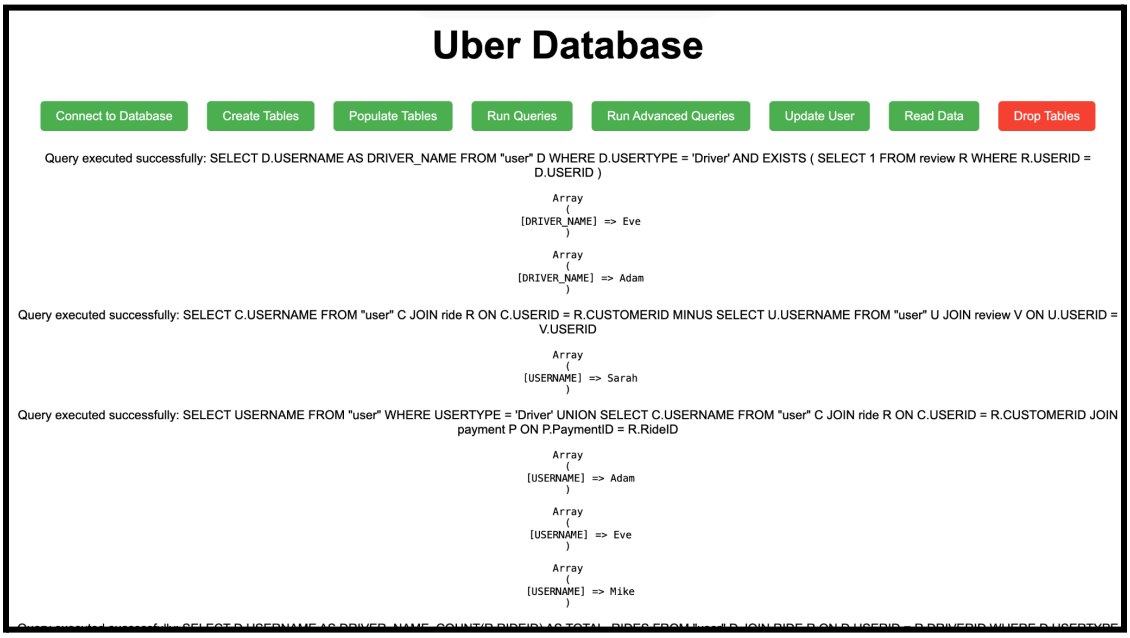
Populating Tables:



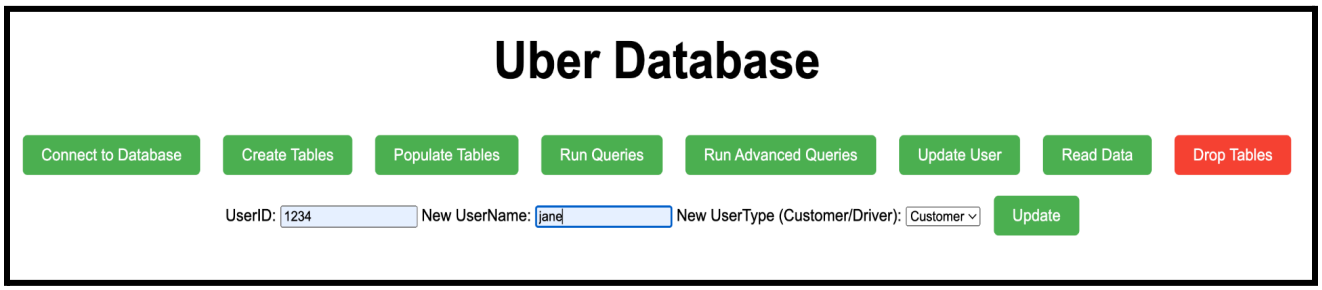
Running Queries:

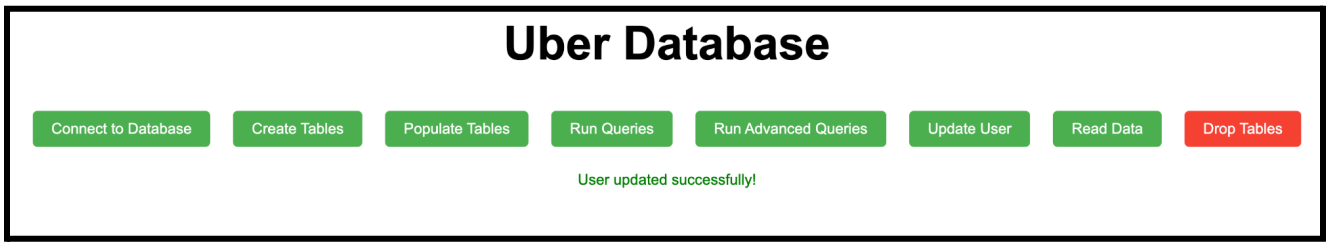


Running Advanced Queries:

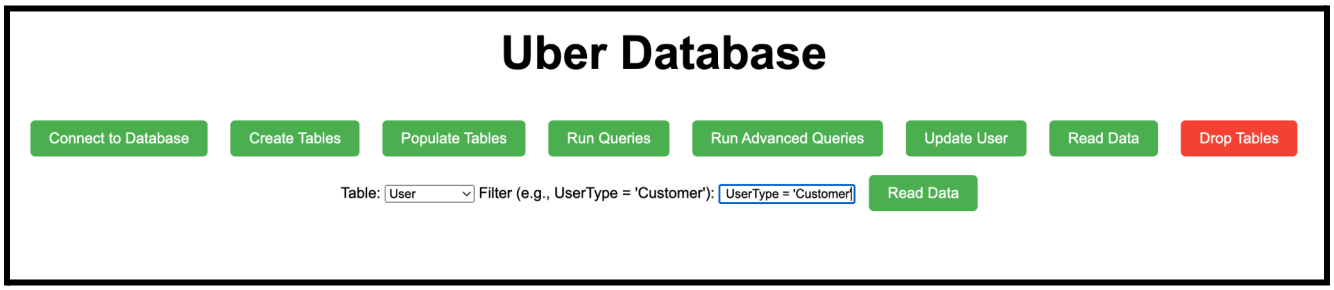


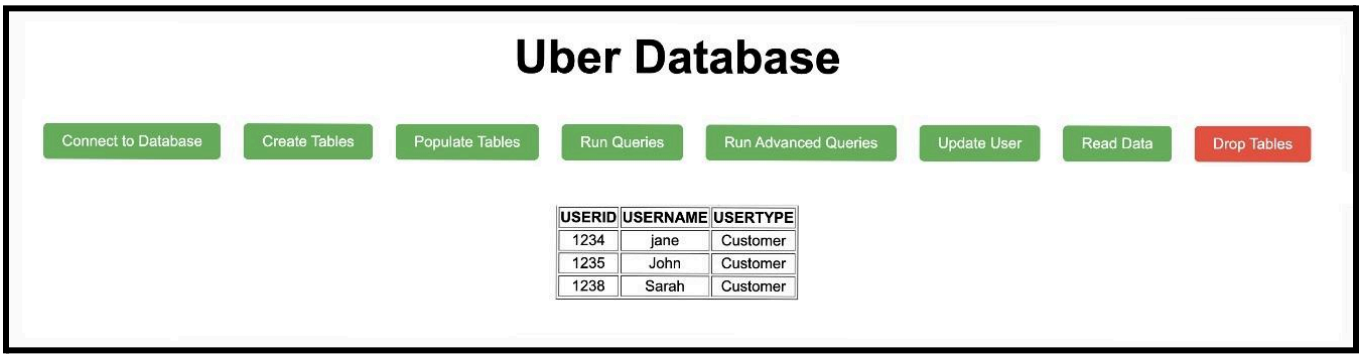
Updating Data:





Reading Data:





11. Concluding Remarks

During the design and normalization process for our ride and pick-up DBMS, we faced the challenge of ensuring every table met the criteria for 3NF and BCNF. This process required careful analysis of our data to eliminate redundancy and establish appropriate connections between tables. Despite the effort involved, the benefits were significant. Normalizing our database helped us avoid data duplication, enhance the efficiency of our DBMS, and achieve a deeper understanding of how our data is structured and managed.

Through this project, we gained valuable insights into the complexities of storing, managing, and working with data in larger projects. We enhanced our SQL skills, became proficient with Oracle Developer, and expanded our capabilities by integrating our DBMS into a web-based application using PHP.

For future improvements, we recommend starting with a well-thought-out ER diagram to provide a solid foundation for the database. Avoiding repetitive data, transitive dependencies, and storing multiple data points in a single cell are crucial lessons we would carry forward. We also learned that having more tables that are well-designed is preferable to fewer tables with formatting or dependency issues. This experience has highlighted the importance of careful planning and normalization in creating an effective and robust DBMS.

---

12. Installation and Operation Guide

The code requires Oracle Database and the Oracle Database Python driver.

First, install Oracle XE, then install the oracledb module using “pip install oracledb”.

---

13. Appendices

- Drive link for all codes.
- All previous assignments.

---

14. References

- CPS510 Project-information. (2024). *CPS510 Project: Information 2024*. Database Systems I, Fall 2024. [Retrieved from course materials.]
- CPS510 Course Notes. (2024). *Database Systems I - F2024 Lecture Notes*. [Retrieved from D2L.]