Cababa: A Cab Booking System

Akanksha Singal (2021008), Ananya Goyal (2021011)

1.1 Non-conflicting Transactions with Respect to Serialisation

Transaction 1:
This transaction involves selecting all ride records for a particular user, inserting a new one for the same user, and then updating the ride status. The transaction is non-conflict serialisable.

Schedule 1:
The schedule involves executing transaction 1 and transaction 2 in the following order:
t1r1, t1w1, t1w2, t2r1, t2w1, t2w2

Schedule 2:
The schedule involves executing transaction 2 and transaction 1 in the following order:
t2r1, t1r1, t1w2, t2w1, t2w2, t1w1

Transaction 2:
This transaction involves selecting all ride records for a particular user, inserting a new one for the same user, and then updating the ride status. The transaction is non-conflict serialisable.

```
--transaction 1
start TRANSACTION;
t1r1: select * from rides where u_email = 'kamille_kuphal@hotmail.com';
t1w1: INSERT INTO rides (receipt_no, pickup_time, pickup_loc, drop_time,
drop_loc, vehicle_no, d_email, u_email, distance, fare, rating, ongoing)
VALUES ('3#kamille_kuphal@hotmail.com', '2018-04-01 12:00:00', 'IIIT
Delhi', '2018-04-01 12:30:00', 'IIT Delhi', 'HR14IF9096',
'harmony.steuber@nikolaus.biz', 'kamille_kuphal@hotmail.com', 10, 100, 0,
1);
t1w2: update rides set ongoing = NULL where u_email =
'kamille_kuphal@hotmail.com';
commit;

--transaction 2
start TRANSACTION;
t2r1: select * from rides where u_email = 'krystina_fay@hotmail.com';
```

```
t2w1: INSERT INTO rides (receipt_no, pickup_time, pickup_loc, drop_time,
drop_loc, vehicle_no, d_email, u_email, distance, fare, rating, ongoing)
VALUES ('2#krystina_fay@hotmail.com', '2018-04-01 12:00:00', 'IIIT Delhi',
'2018-04-01 12:30:00', 'IIT Delhi', 'HR14IF9096',
'harmony.steuber@nikolaus.biz', 'krystina_fay@hotmail.com', 10, 100, 0,
1);
t2w2: update rides set ongoing = NULL where u_email =
'krystina_fay@hotmail.com';
COMMIT;

-- no conflict -> non conflict serialisable

--schedule 1
t1r1, t1w1, t1w2, t2r1, t2w1, t2w2

--schedule 2
t2r1, t1r1, t1w2, t2w1, t2w2, t1w1
```

Locking Statement

```
     BEGIN;
     LOCK TABLES vehicles WRITE;
     SELECT reg_no FROM vehicles WHERE type_id = %s order by rand() limit
1;
     COMMIT;
     UNLOCK TABLES;
```

BEGIN;
This statement starts a transaction. A transaction is a set of SQL statements executed as a single unit. If any statements in a transaction fail, the entire transaction will be rolled back.

LOCK TABLES vehicles WRITE;
This statement locks the vehicles' table in write mode. This means no other user can modify the table while it is locked.

SELECT reg_no FROM vehicles WHERE type_id = %s order by rand() limit 1;
This statement selects the registration number of the first vehicle that matches the specified type ID. The vehicles are ordered randomly before the first one is selected.

COMMIT;

This statement commits the transaction. This means that the changes made by the statements in the transaction are permanent.

UNLOCK TABLES;

This statement unlocks the vehicles table. This allows other users to modify the table.

This SQL statement can be used to book a cab in an Uber-like system. The user specifies the type of cab they want, and the system selects the first available cab of that type. The system then locks the vehicles table to prevent other users from booking the cab and selects the cab's registration number. The system then commits the transaction and unlocks the vehicles table.

1.2 Non-conflicting Transactions

1. The first transaction updates user information for a particular user based on their email address. The transaction ensures that the update query only affects one user's data by using the WHERE clause to filter by email address. This transaction is non-conflicting because it only modifies one row and does not depend on other transactions.

```
  "START TRANSACTION;"
  "UPDATE users SET user_email = %s, phno = %s, pswd = %s where user_email
= %s")
  "Commit" ELSE "ROLLBACK" (rollback in the event where the transaction
fails due to any reason)
```

2. The second transaction inserts a new user into the system with their email address, first name, last name, phone number, and password. This transaction is non-conflicting because it creates a new row in the database and does not depend on other transactions.

```
  "START TRANSACTION;"
  "INSERT INTO users (user_email, first_name, last_name, phno, pswd)
VALUES (%s, %s, %s, %s, %s)")
  "commit"
```

3. The third transaction updates the 'ongoing' field in the rides table to NULL for a particular user based on their email address. This transaction is non-conflicting because it only modifies one row and does not depend on other transactions.

```
  "START TRANSACTION;"
  "update rides set ongoing = NULL where u_email = %s"
  "commit"
```

4. The fourth transaction inserts a new ride into the system with a receipt number, pickup time, pickup location, drop time, location, vehicle number, driver email, user email, distance, fare, rating, and ongoing status. This transaction is non-conflicting because it creates a new row in the database and does not depend on other transactions.

```
  "START TRANSACTION;"
  "INSERT INTO rides (receipt_no, pickup_time, pickup_loc, drop_time,
drop_loc, vehicle_no, d_email, u_email, distance, fare, rating, ongoing)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)")
  "commit”
```

2. Conflicting Transactions

Transaction 3:
This transaction updates the first name and password of a user with the email 'kamille_kuphal@hotmail.com' in the 'users' table. It then selects and returns the updated user's record using the updated email address. Finally, it commits the changes made in the transaction.

Transaction 4:
This transaction selects and returns all the rides a user with the email 'kamille_kuphal@hotmail.com' has taken from the 'rides' table. It then updates the last name of the user in the 'users' table and commits the changes made in the transaction.

Conflict serialisable schedule: Schedule 2
Non-conflict serialisable schedule: Schedule 1
In the non-conflict serialisable schedule (Schedule 1), transaction 4 reads from the 'rides' table before transaction 3 has been committed, violating the conflict serializability property. This can lead to dirty reads or non-repeatable reads.

In the conflict serialisable schedule (Schedule 2), the transactions are arranged so that the read and write operations on the 'users' table are interleaved, ensuring that the transaction that commits last has access to the latest data. This schedule ensures that the conflict serializability property is maintained.

```
--transaction 3
start TRANSACTION;
t3wa1: update users set first_name = 'kam' where user_email =
'kamille_kuphal@hotmail.com';
t3ra1: select * from users where u_email = 'kamille_kuphal@hotmail.com';
t3wa2: update users set pswd = 'abcdefgh' where user_email =
'kamille_kuphal@hotmail.com';
t3ra2: select * from users where u_email = 'kamille_kuphal@hotmail.com';
commit;

start transaction;
```

```
--transaction 4
t4rb: select * from rides where u_email = 'kamille_kuphal@hotmail.com';
t4wa: update users set last_name = 'kuphal' where user_email =
'kamille_kuphal@hotmail.com';
commit;

--schedule 1 - Non-conficting serialisable
t3wa1, t3ra1, t4rb, t4wa, t3wa2, t3ra2

--schedule 2 - conflict serialisable
t3wa1, t3ra1, t4rb, t3wa2, t3ra2, t4wa
```

Transaction 5:
This transaction represents the process of a user booking a ride and a driver being assigned to the ride. The first statement in this transaction inserts a new ride record into the "rides" table, and the second statement selects all rides with the same user email as the one who just booked the ride. The third statement updates the driver's availability status to "UNAVAILABLE". The fourth statement again selects all rides with the same user email as the one who booked the ride. Finally, the transaction is committed.

Transaction 6:
This transaction represents completing a ride and updating the ride and driver information accordingly. The first statement in this transaction updates the total number of trips for the driver who just completed the ride. The second statement updates the "ongoing" field in the ride record to indicate that the ride has been completed. The third statement selects the most recent completed ride for the user who just completed the ride, and the fourth statement selects the driver information for the driver who just completed the ride. Finally, the transaction is committed.

Schedule 1 - Conflict Serializable:
This schedule represents a conflict-serializable execution of the two transactions. In this schedule, Transaction 6 completes before Transaction 5 begins, so there is no conflict between them. The order of statements in each transaction is preserved, but the order in which the transactions execute differs from the order in which they were initiated.

Schedule 2 - Non-Conflict Serializable:
This schedule represents a non-conflict-serializable execution of the two transactions. In this schedule, Transaction 6 updates the driver's trip count before Transaction 5 assigns the driver to a ride, which means that

the driver's trip count may be updated before it should be. This schedule violates the conflict-serializability property of the system.

```
--transaction 5

start transaction;
t5wa: INSERT INTO rides (receipt_no, pickup_time, pickup_loc, drop_time,
drop_loc, vehicle_no, d_email, u_email, distance, fare, rating, ongoing)
VALUES ('2#kamille_kuphal@hotmail.com', '2018-04-01 12:00:00', 'IIIT
Delhi', '2018-04-01 12:30:00', 'IIT Delhi', 'HR14IF9096',
'harmony.steuber@nikolaus.biz', 'kamille_kuphal@hotmail.com', 10, 100, 0,
1);
t5ra1: select * from rides where u_email = 'kamille_kuphal@hotmail.com';
t5wb: update drivers set driver_availibity = 'UNAVAILABLE' where
driver_email = 'harmony.steuber@nikolaus.biz';
t5ra2: select * from rides where u_email = 'kamille_kuphal@hotmail.com';
commit;

--transaction 6

start transaction;
t6wb: update drivers set total_trips = total_trips + 1 where driver_email
= 'harmony.steuber@nikolaus.biz';
t6wa: update rides set ongoing = NULL where u_email =
'kamille_kuphal@hotmail.com';
t6ra: select * from rides where u_email = 'kamille_kuphal@hotmail.com' and
ongoing is NULL order by drop time desc limit 1;
t6rb: select * from drivers where driver_email =
'harmony.steuber@nikolaus.biz';
commit;

--schedule 1 - conflict serialisable
t6wb, t6wa, t6ra, t5wa, t5ra1, t6rb, t5wb, t5ra2

--schedule 2 - non-conflict serialisable
t6wb, t5wa, t5ra1, t6wa, t6ra, t5wb, t5ra2, t6rb
```