

ACCA7002 Database Systems

Tutorial 9

Tutorial 9: Perform transactions with sample data

The aim of this exercise is to help you practice including creating tables, inserting records, indexing, grouping data, and managing transactions in PostgreSQL

Task 1: Create a Database and Connect

```
CREATE DATABASE taxi_management;
```

Task 2: Create Tables

Assuming you want to track drivers, passengers, and trips, create the following tables:

```
CREATE TABLE drivers (  
    driver_id SERIAL PRIMARY KEY,  
    name VARCHAR(100) NOT NULL,  
    license_number VARCHAR(20) UNIQUE  
);
```

```
CREATE TABLE passengers (  
    passenger_id SERIAL PRIMARY KEY,  
    name VARCHAR(100) NOT NULL  
);
```

```
CREATE TABLE trips (  
    trip_id SERIAL PRIMARY KEY,  
    driver_id INT REFERENCES drivers(driver_id),  
    passenger_id INT REFERENCES  
passengers(passenger_id),  
    fare DECIMAL(10, 2) NOT NULL,  
    date timestamp  
);
```

Task 3: Insert Records

Insert 10 records into the drivers, passengers, and trips tables.

```
-- Insert 5 drivers
INSERT INTO drivers (name, license_number) VALUES
    ('John Doe', 'ABC123'),
    ('Alice Smith', 'XYZ456'),
    ('Bob Johnson', 'DEF789'),
    ('Eva Wilson', 'GHI123'),
    ('Mike Brown', 'JKL456')
    .
    .
    .
    ;

-- Insert 5 passengers
INSERT INTO passengers (name) VALUES
    ('Mary Johnson'),
    ('David Lee'),
    ('Linda Harris'),
    ('James Wilson'),
    ('Sarah Davis')
    .
    .
    .
    ;

-- Insert 10 trips
INSERT INTO trips (driver_id, passenger_id, fare,
date) VALUES
    (1, 1, 25.50, NOW()),
    (2, 2, 18.75, NOW()),
    (3, 3, 12.00, NOW()),
    (4, 4, 30.25, NOW()),
    (5, 5, 22.80, NOW()),
    (1, 6, 16.90, NOW()),
    (2, 7, 14.25, NOW()),
    (3, 8, 20.50, NOW()),
    (4, 9, 28.75, NOW()),
    (5, 10, 19.40, NOW());
```

Task 4: Create Index

You can create an index on the `driver_id` and `passenger_id` columns of the `trips` table for faster query performance.

```
CREATE INDEX driver_index ON trips(driver_id);  
CREATE INDEX passenger_index ON trips(passenger_id);
```

Task 5: Grouping and Aggregation

Let's find the total earnings for each driver.

```
SELECT drivers.name, SUM(trips.fare) AS  
total_earnings  
FROM drivers  
LEFT JOIN trips ON drivers.driver_id =  
trips.driver_id  
GROUP BY drivers.name;
```

Task 8: JOINS

An **INNER JOIN** retrieves only the rows with matching values in both tables. Here, we can join `drivers` and `trips` to find which driver made each trip:

```
SELECT drivers.name AS driver_name, trips.trip_id  
FROM drivers  
INNER JOIN trips ON drivers.driver_id =  
trips.driver_id;
```

A **LEFT JOIN** retrieves all rows from the left table and the matching rows from the right table. If there's no match in the right table, `NULL` values are returned. You can use this to find all drivers and their trips, including drivers with no trips

```
SELECT drivers.name AS driver_name, trips.trip_id  
FROM drivers  
LEFT JOIN trips ON drivers.driver_id =  
trips.driver_id;
```

A **RIGHT JOIN** retrieves all rows from the right table and the matching rows from the left table. If there's no match in the left table, `NULL` values are returned. This can be used to find all trips and their corresponding drivers, including trips with no drivers

```
SELECT drivers.name AS driver_name, trips.trip_id
FROM drivers
RIGHT JOIN trips ON drivers.driver_id =
trips.driver_id;
```

A FULL JOIN retrieves all rows when there is a match in either the left or the right table. If there's no match in either table, NULL values are returned. This can be used to find all drivers and trips, including unmatched records:

```
SELECT drivers.name AS driver_name, trips.trip_id
FROM drivers
FULL JOIN trips ON drivers.driver_id =
trips.driver_id;
```

Task 7: Transaction Management

Now, let's perform transaction management by simulating a situation where you need to commit or rollback a transaction.

```
BEGIN; -- Start a transaction

-- Insert a new trip
INSERT INTO trips (driver_id, passenger_id, fare,
date) VALUES
    (1, 7, 14.75, NOW());

-- Check if the trip was successful
SELECT * FROM trips WHERE driver_id = 1 AND
passenger_id = 7;

-- If the trip is successful, commit the transaction
COMMIT;

-- If there was an issue, rollback the transaction
-- ROLLBACK;
```

Uncomment either the COMMIT or ROLLBACK statement to see how the transaction behaves.