Final Report: Construction of Database

CSI2132 - Databases 1

Winter 2023

School of Electrical Engineering and Computer Science

University of Ottawa

Professor: Wail Mardini

Yash Jain 300245571

Submission Date: April 12th , 2023

Introduction

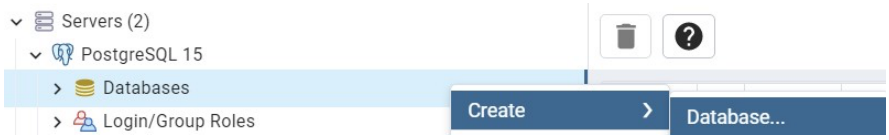
During the course of the database project, we gained invaluable insights into the importance of organizing and managing data effectively. As a large-scale project, it required significant planning and coordination to design a database that would be both efficient and user-friendly. We learned the importance of considering the needs of all stakeholders in the design process, including end-users and developers. This report will discuss the languages used in the project, a practical installation guide, then the DDL used to set up the schema.

Languages Used

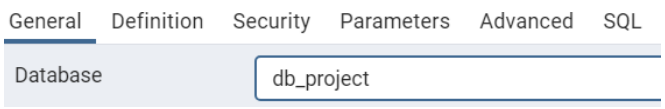
- The Database Management System used is PostgreSQL(pg Admin GUI)
- The programming languages used was HTML and php

Installation Guide

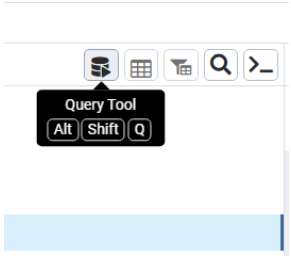
1. Open pgAdmin and create new database



2. Name the database: “db_project” and save



3. Open Query Tool, paste DDL, then execute



4. Open Apache24\htdoc file location(for windows) or /usr/local/var/www file(for Mac) and paste the provided files
5. Open the files, and edit the following lines to match pg Admin system properties:

```
$conn_string = "host=localhost port = 5432 dbname=db_project user=postgres password = root";
```

a. Updates the following files:
   i. employeeRecords.php
   ii. employeeLogin.php
   iii. customerView.php
   iv. customerRegister.php
   v. customerRecords.php
   vi. customerLogin.php
   vii. cancelRoom.php
   viii. roomBooking.php
   ix. rentInPerson.php

6. Start apache server: (refer to Lab 9)

```
apachectl start // for Mac
```

7. Visit the following URL address:
   http://localhost:8080/databases_project/customerOrEmployee.php

8. Use the provided hardcoded logins to access the employee login

name	employee_id	hotel_id
Aleckson Townsend	111111111	987654123
Tommy Green	123456789	987654321
Vicky Patel	666666666	357910111
Wassim Ahmar	777777777	741369852
Theo James	888888888	951753258
Von Right	999999999	654789123
Vikas Dune	999666999	159753246
Rex Grimes	909109381	789123456
John Stone	183901909	741852963
James King	647524689	123789456

Yope Mare	909890689	258741963
Payal Patel	222222222	852369741
Mohammed Klink	444444444	234567890
Morgan Smith	333333333	987654121

Hotel by Area

- Both hotels located in Cherrywood

```
INSERT INTO hotel VALUES (234567890, '9876 Country Lane, Pineville, IT', 1, 5, '234567890@gmail.com', '456-123-789', 'Cherrywood', '789 Oak Avenue', 'L5N 8C9');

INSERT INTO hotel VALUES (234567110, '9896 Country Lane, Pineville, IT', 1, 5, '234567450@gmail.com', '456-113-789', 'Cherrywood', '780 Oak Avenue', 'L5N 8D9');
```

Queries & Triggers

Query 1

```
SELECT * FROM room WHERE price > 250;
```

Query 2

```
SELECT * FROM room WHERE views = 'mountain_view';
```

Query 3

```
SELECT * FROM hotel WHERE stars = 5;
```

Query 4

```
SELECT * FROM Room WHERE extension = 'yes';
```

Trigger 1

```
CREATE FUNCTION delete_workers()
RETURNS TRIGGER AS $$
BEGIN
        IF EXISTS (SELECT * FROM employee WHERE hotel_id = NEW.hotel_id) THEN
        DELETE FROM employee WHERE hotel_id = NEW.hotel_id;
        RAISE NOTICE 'EMPLOYEE DELETE FROM EMPLOYEE TABLE';
        END IF;
        RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER delete_employee
AFTER DELETE ON hotel
FOR EACH ROW
EXECUTE FUNCTION delete_workers();
```

Trigger 2

```
CREATE FUNCTION update_housing()
RETURNS TRIGGER AS $$
BEGIN
        IF NOT EXISTS (SELECT * FROM customer WHERE customer_id =
NEW.customer_id) THEN
        RAISE EXCEPTION 'INVALID CUSTOMER ID. CANNOT UPDATE ROOM.';
        END IF;
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_room
BEFORE UPDATE ON room
FOR EACH ROW
EXECUTE FUNCTION update_housing();
```

Indexes

```
CREATE INDEX index_room_amenities ON room (amenities);
```

This index can be useful in accelerating queries that involve filtering or searching for rooms based on specific amenities. For example, if a user wants to find all rooms with a specific amenity (e.g., a room with a microwave), the database engine can use this index to quickly locate all the rooms that have that amenity, without having to scan the entire table.

```
CREATE INDEX index_room_view ON room (views);
```

This index can be useful in accelerating queries that involve sorting or filtering rooms based on the view they offer. For example, if a user wants to find all rooms with an sea view, the database engine can use this index to quickly locate all the rooms with that view, without having to scan the entire table.

```
CREATE INDEX index_room_avalibility ON room (booked);
```

This index can be useful in accelerating queries that involve checking the availability of rooms for specific dates. For example, if a user wants to find all the available rooms for a particular date range, the database engine can use this index to quickly locate all the rooms that are not booked during that time, without having to scan the entire table.

Overall, the usefulness of these indexes depends on the specific needs of the database and the queries being executed. By creating these indexes, the database can efficiently locate the required data without having to perform full table scans, thereby improving query performance. However, it's important to note that creating indexes comes at a cost in terms of disk space and maintenance overhead, so it's important to carefully consider the benefits and trade-offs before creating indexes.