CSI2132 - Project report

Angus Young | 300078935

Kian Salehi | 300056262

Melissa Wu |300060261

1.

a. DBMS and languages used in the production of this application:
DBMS - postgres with pgAdmin
Languages used: Python

b.

```
-- Table: public.branch

-- DROP TABLE public.branch;

CREATE TABLE public.branch
(
    branch_id integer NOT NULL DEFAULT nextval('branch_branch_id_seq'::regclass),
    employee_id integer NOT NULL,
    associatedcountry character varying(30) COLLATE pg_catalog."default",
    CONSTRAINT branch_pkey PRIMARY KEY (branch_id),
    CONSTRAINT branch_employee_id_fkey FOREIGN KEY (employee_id)
        REFERENCES public.employee (employee_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```

```
-- Table: public.employee

-- DROP TABLE public.employee;

CREATE TABLE public.employee
(
    employee_id integer NOT NULL DEFAULT nextval('employee_employee_id_seq'::regclass),
    "position" character varying(20) COLLATE pg_catalog."default" NOT NULL,
    salary numeric(10,2),
    branch_id integer,
    CONSTRAINT employee_pkey PRIMARY KEY (employee_id),
    CONSTRAINT branch_id_fk FOREIGN KEY (branch_id)
        REFERENCES public.branch (branch_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```

```sql
-- Table: public.guest

-- DROP TABLE public.guest;

CREATE TABLE public.guest
(
    guest_id integer NOT NULL DEFAULT nextval('guest_guest_id_seq'::regclass),
    user_id integer NOT NULL,
    branch_id integer,
    CONSTRAINT guest_pkey PRIMARY KEY (guest_id),
    CONSTRAINT branch_id_fk FOREIGN KEY (branch_id)
        REFERENCES public.branch (branch_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT guest_user_id_fkey FOREIGN KEY (user_id)
        REFERENCES public.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)
```

```sql
-- Table: public.host

-- DROP TABLE public.host;

CREATE TABLE public.host
(
    host_id integer NOT NULL DEFAULT nextval('host_host_id_seq'::regclass),
    user_id integer NOT NULL,
    CONSTRAINT host_pkey PRIMARY KEY (host_id),
    CONSTRAINT host_user_id_fkey FOREIGN KEY (user_id)
        REFERENCES public.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)
```

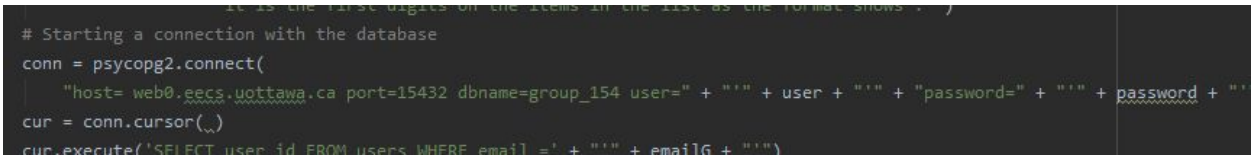
```sql
-- Table: public.payment

-- DROP TABLE public.payment;

CREATE TABLE public.payment
(
    payment_id integer NOT NULL DEFAULT nextval('payment_payment_id_seq'::regclass),
    rentalagreement_id integer NOT NULL,
    paymenttype character varying(50) COLLATE pg_catalog."default" NOT NULL,
    amount integer NOT NULL,
    status boolean,
    CONSTRAINT payment_pkey PRIMARY KEY (payment_id),
    CONSTRAINT payment_rentalagreement_id_fkey FOREIGN KEY (rentalagreement_id)
        REFERENCES public.rentalagreement (rentalagreement_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)
```

```sql
-- Table: public.rentalagreement

-- DROP TABLE public.rentalagreement;

CREATE TABLE public.rentalagreement
(
    rentalagreement_id integer NOT NULL DEFAULT nextval('rentalagreement_rentalagreement_id_seq'::regclass),
    property_id integer NOT NULL,
    guest_id integer NOT NULL,
    host_id integer NOT NULL,
    signingdate date NOT NULL,
    startdate date NOT NULL,
    enddate date NOT NULL,
    CONSTRAINT rentalagreement_pkey PRIMARY KEY (rentalagreement_id),
    CONSTRAINT rentalagreement_guest_id_fkey FOREIGN KEY (guest_id)
        REFERENCES public.guest (guest_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT rentalagreement_host_id_fkey FOREIGN KEY (host_id)
        REFERENCES public.host (host_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT rentalagreement_property_id_fkey FOREIGN KEY (property_id)
        REFERENCES public.property (property_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)
```

```sql
-- Table: public.review

-- DROP TABLE public.review;

CREATE TABLE public.review
(
    review_id integer NOT NULL DEFAULT nextval('review_review_id_seq'::regclass),
    property_id integer NOT NULL,
    rating integer NOT NULL,
    guest_id integer NOT NULL,
    communication integer,
    cleanliness integer,
    CONSTRAINT review_pkey PRIMARY KEY (review_id),
    CONSTRAINT review_guest_id_fkey FOREIGN KEY (guest_id)
        REFERENCES public.guest (guest_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT review_property_id_fkey FOREIGN KEY (property_id)
        REFERENCES public.property (property_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

-- Table: public.users

-- DROP TABLE public.users;

CREATE TABLE public.users
(
    user_id integer NOT NULL DEFAULT nextval('users_user_id_seq'::regclass),
    firstname character varying(30) COLLATE pg_catalog."default" NOT NULL,
    lastname character varying(30) COLLATE pg_catalog."default" NOT NULL,
    email character varying(30) COLLATE pg_catalog."default" NOT NULL,
    phonenumber character varying(20) COLLATE pg_catalog."default" NOT NULL,
    address character varying(100) COLLATE pg_catalog."default" NOT NULL,
    provinceorstate character varying(30) COLLATE pg_catalog."default" NOT NULL,
    country character varying(30) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT users_pkey PRIMARY KEY (user_id)
)
```

2. Installation instructions

In order to run this program Python 3 with libraries psycopg2, datetime, and prettytable need to be installed. To install the libraries type ‘pip install LIBRARYNAME’ in the python virtual environment interpreter. Some IDEs such as Pycharm give the option of automatic install to the user when the code is put in the IDE. A database is required to connect with the app, for example, the University of Ottawa database is used to run and test this program. The DBMS used in this app is PostgreSQL, which can be downloaded from the PostgreSQL official website.



As shown in the image host would be the host address of the database server, port would be the port number of the database server, and dbname would be the database name. User and password would not need to be changed, because those values can be placed by the user input when the program is executed. The explanation for making the tables in the database are give at the first page of the report. In order to access the database for all the users other than the database owner, the permission to the database needs to be granted by the following codes in the query tool:

```
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO USER_NAME;
```

And

```
GRANT ALL ON TABLE table_name TO USER_NAME;
```

A database admin would need to use pgAdmin4 as a SQL developer in order to execute the SQL commands in the query tool.
Users would need to use the command line in order to access the data and use the app, or use an IDE console to use the app. Employees would use the same method as the users but with a different login option inside the app.

3.

Queries

1. Give the details of all the guests who rented properties. Please display the columns as guest name, rental type, rental price, signing date, branch, payment type and payment status. Sort by the payment type in ascending order and signing date in descending order.

```
SELECT*FROM
   guest
INNER JOIN rentalagreement ON guest.guest_id = rentalagreement.guest_id
INNER JOIN payment ON payment.rentalagreement_id =
rentalagreement.rentalagreement_id
order by paymenttype ASC, signingdate DESC;
```

2. Create a view named GuestListView that gives the details of all the guests. Please, sort the guests by the branch id and then by guest id.

made the assumption that sort meant sort but ascending order
CREATE VIEW guest.GuestListView() AS SELECT * FROM guest ORDER ASC branch_id, guest_id ASC

3. Display the details of the cheapest (completed) rental.

```
SELECT rentalagreement.rentalagreement_id,
rentalagreement.property_id, guest_id, rentalagreement.host_id,
signingdate, startdate, enddate, amount FROM payment
INNER JOIN rentalagreement ON rentalagreement.rentalagreement_id =
payment.rentalagreement_id
WHERE rentalagreement.enddate < NOW()
order by amount ASC
LIMIT 1;
```

4. List all the properties rented and sort based on the branch id and review rating.
note: not all properties will have a review rating

```
Select (p.property_id , p.branch_id, r.review  ) from property p left
join review r on p.property_id = r.property_id where p.booked  = true
```

5. Find the properties that are already listed but not yet rented. Please, avoid duplications.

```
SELECT * FROM property WHERE property_id NOT IN (SELECT property_id
FROM rentalagreement)
```

6. List all the details of all properties rented on the 10th day of any month. Ensure to insert dates in your table that correspond in order to run your query.

```
Select *from properties where day(startdate) = 10
```

7. List all the managers and the employees with salary greater than or equal to $15000 by their ids, names, branch ids, branch names and salary. Sort by manager id and then by employee id.
branch name is the country it is in

```
SELECT (e.employee_id,e.branch_id,e.salary,b.associatedcountry) From
employee e JOIN branch b ON e.employee_id = b.employee_id WHERE
e.salary >= 15000
```

8. Consider creating a simple bill for a guest stating the property type, host, address, amount paid and payment type.
assume the guest id is known

```
SELECT(p.propertytype,u.firstname ,p.streetname, p.housenumber,
p.city , r.amount, c.paymenttype)
FROM payment r JOIN rentalagreement t ON
r.rentalagreement_id=t.rentalagreement_id JOIN property p ON
r.property_id = p.property_id JOIN host h ON
p.host_id = h.host_t  JOIN User u ON
h.user_id = u.user_id
WHERE u.user_ID =”guest_ID”
```

9. Update the phone number of a guest.
assume the guest_id is known

```
UPDATE u SET u.phonenumber = 1111111111 FROM users u INNER JOIN
Guest g ON u.user_id = g.user_id where g.guest_id = 35
```

10. Create and test a user-defined function named FirstNameFirst that combines two attributes of the guest named firstName and lastName into a concatenated value named fullName [e.g., James and Brown will be combined to read James Brown].

```
CREATE FUNCTION FirstNameFirst(firstName VARCHAR(50), lastName
VARCHAR(50)) RETURNS VARCHAR AS $$
BEGIN
RETURN firstName + lastName;
END; $$
LANGUAGE PLPGSQL;
```