Databases Report

Phillip Brule
Richard Xiong

Instructions for using User Interface

1. Download the zip file from brightspace
2. Unzip the file
3. Open the folder
    a. For windows users, double click the file CSI2132.exe to run the programming
    b. For linux users, use the terminal to go into that directory and type "./CSI2132". This maybe the same for MAC OSX users
4. Open your browser (preferably google chrome) and paste in http://localhost:8000/

Programming Languages

Since the purpose of this class was to focus on the design and manipulation of databases, we were told that we could build the user interface with whichever technologies we felt comfortable with. Thus, the main programming languages used were Go (Version 1.14), Javascript (ECMAScript 6), CSS3 and HTML5. Go was a viable option because its standard library came with out-of-the-box tools that made it extremely easy to set up a web server that could connect to and interact with the remote database (named group_153). However, various 3rd party go libraries and a Go sql driver were also used (see list below) The JQuery library, which is built from javascript, was used for some minimal button animation designs and form logic. Bootstrap (Version 4.4.1) is the main CSS3 library used. A Bootstrap template was acquired from startbootstrap.com. A lot of customized CSS was programmed in by us to enhance the look and feel of the user interface. The application was programmed with the Model View Controller (MVC) design pattern.

List of 3rd Party Go Libraries

gorilla/mux	Used mainly for routing
gorilla/schema	Used to convert form data to be used by go
lib/pq	The main postgres driver for Go

List of DDLs

Following is our list of DDLs in which we used to create our database

---

```
-- Custom Defined Types
CREATE TYPE valid_property_type AS ENUM ('apartment', 'bed and
breakfast', 'unique home', 'vacation home');

CREATE TYPE valid_payment_type AS ENUM ('debit', 'credit', 'cash',
'cheque', 'paypal');

CREATE TYPE valid_payment_status AS ENUM ('approved', 'pending',
'completed', 'declined', 'incomplete');

CREATE TYPE address AS (
      house_number INTEGER,
      city character varying(90),
      street character varying(90),
      province character varying(90),
      postal_code character varying(6),
      country character varying(90)
);

-- All Queries used to create tables
CREATE TABLE employees (
    user_id INTEGER NOT NULL,
    salary INTEGER,
    manager BOOLEAN DEFAULT false,
    branch_id INTEGER,
    employee_id SERIAL PRIMARY KEY
);


CREATE TABLE branch (
    branch_id SERIAL,
    manager_id INTEGER NOT NULL,
    country character varying(20) COLLATE pg_catalog."default",
    CONSTRAINT branch_pkey PRIMARY KEY (branch_id),
    CONSTRAINT branch_manager_id_fkey FOREIGN KEY (manager_id)
        REFERENCES employees (employee_id) MATCH SIMPLE
```

```
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT chk_manager_id CHECK (check_manager(manager_id) = true)
);

-- Add constraint to employee table after creating branch table
ALTER TABLE employees
ADD CONSTRAINT branch_id FOREIGN KEY (branch_id)
REFERENCES branch (branch_id) MATCH SIMPLE
ON UPDATE NO ACTION ON DELETE NO ACTION;


CREATE TABLE users
(
    user_id INTEGER NOT NULL GENERATED BY DEFAULT AS IDENTITY (
INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),
    user_address address,
    first_name character varying(30) COLLATE pg_catalog."default",
    last_name character varying(30) COLLATE pg_catalog."default",
    email character varying(90) COLLATE pg_catalog."default" NOT NULL,
    phone_number character(10) COLLATE pg_catalog."default",
    host boolean DEFAULT false,
    guest boolean DEFAULT false,
    middle_name character varying(50) COLLATE pg_catalog."default",
    password character varying(30) COLLATE pg_catalog."default" NOT
NULL,
    branch_id INTEGER,
    CONSTRAINT users_pkey PRIMARY KEY (user_id),
    CONSTRAINT unq_email UNIQUE (email),
    CONSTRAINT chk_email CHECK (email::text ~
'^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'::text),
    CONSTRAINT chk_phone CHECK (phone_number !~~ '%[^0-9]%'::text)
)
CREATE TABLE pricing (
    price_id INTEGER NOT NULL GENERATED BY DEFAULT AS IDENTITY (
INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),
    rate_per_day numeric(12,2),
    rate_per_week numeric(12,2),
    CONSTRAINT pricing_pkey PRIMARY KEY (price_id)
```

```
);

CREATE TABLE properties (
    property_id INTEGER NOT NULL GENERATED BY DEFAULT AS IDENTITY (
INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),
    pricing_id INTEGER NOT NULL,
    property_type valid_property_type NOT NULL,
    accommodates INTEGER,
    amenities character varying(800) COLLATE pg_catalog."default",
    bathrooms INTEGER,
    bedrooms INTEGER,
    property_address address,
    host_id INTEGER NOT NULL,
    CONSTRAINT properties_pkey PRIMARY KEY (property_id),
    CONSTRAINT fk_host_id FOREIGN KEY (host_id)
        REFERENCES users (user_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT properties_pricing_id_fkey FOREIGN KEY (pricing_id)
        REFERENCES pricing (price_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
);

-- FUNCTION guest_check needs to be created before the TABLE
rental_agreement is created
CREATE OR REPLACE FUNCTION check_guest(
      guest_id INTEGER)
    RETURNS boolean
    LANGUAGE 'plpgsql'

    COST 100
    VOLATILE
AS $BODY$
 DECLARE result BOOLEAN;
BEGIN
 SELECT guest into result FROM users WHERE user_id = guest_id;
return result;
END
```

```
$BODY$;

CREATE TABLE payments (
    payment_id SERIAL PRIMARY KEY,
    payment_method valid_payment_type NOT NULL,
    payment_status valid_payment_status NOT NULL,
    CONSTRAINT payment_pkey PRIMARY KEY (payment_id)
);

CREATE TABLE rental_agreement (
    rental_id INTEGER NOT NULL GENERATED BY DEFAULT AS IDENTITY (
INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),
    property_id INTEGER NOT NULL,
    guest_id INTEGER NOT NULL,
    signing_date timestamp without time zone DEFAULT now(),
    start_date date NOT NULL,
    end_date date NOT NULL,
    price_of_stay numeric(12,2),
    payment_id INTEGER,
    CONSTRAINT rental_agreement_pkey PRIMARY KEY (rental_id),
    CONSTRAINT guest_key FOREIGN KEY (guest_id)
        REFERENCES users (user_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT rental_agreement_payment_id_fkey FOREIGN KEY
(payment_id)
        REFERENCES payments (payment_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT rental_agreement_property_id_fkey FOREIGN KEY
(property_id)
        REFERENCES properties (property_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT chk_guest CHECK (check_guest(guest_id) = true)
);


CREATE TABLE reviews(
```

```
    review_id INTEGER NOT NULL GENERATED BY DEFAULT AS IDENTITY (
INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),
    rental_id INTEGER NOT NULL,
    rating INTEGER,
    review_comments character varying(800) COLLATE
pg_catalog."default",
    cleanliness character varying(200) COLLATE pg_catalog."default",
    rent_value INTEGER,
    CONSTRAINT review_pkey PRIMARY KEY (review_id),
    CONSTRAINT review_rental_id_fkey FOREIGN KEY (rental_id)
        REFERENCES rental_agreement (rental_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT valid_review CHECK (rating <= 10 AND rating > 0)
);




-- Functions

CREATE OR REPLACE FUNCTION check_manager(
      manager_id_x INTEGER)
    RETURNS boolean
    LANGUAGE 'plpgsql'

    COST 100
    VOLATILE

AS $BODY$
 DECLARE result BOOLEAN;
BEGIN
 SELECT manager into result FROM employees WHERE manager_id_x =
employee_id;
return result;
END
$BODY$;

CREATE OR REPLACE FUNCTION calculate_price(
```

```
      property_id_x INTEGER,
      start_date date,
      end_date date)
    RETURNS numeric
    LANGUAGE 'plpgsql'

    COST 100
    VOLATILE
AS $BODY$
DECLARE price NUMERIC(12,2);
DECLARE id_of_price int; DECLARE day_price NUMERIC(12,2); DECLARE
week_price NUMERIC(12,2);
DECLARE rental_time int; DECLARE weeks int; DECLARE days int;
BEGIN
SELECT pricing_id into id_of_price FROM properties WHERE property_id =
property_id_x;
SELECT rate_per_day into day_price FROM pricing WHERE price_id =
id_of_price;
SELECT rate_per_week into week_price FROM pricing WHERE price_id =
id_of_price;
SELECT (end_date - start_date) INTO rental_time;
SELECT FLOOR(rental_time / 7) INTO weeks;
SELECT rental_time % 7 INTO days;
price = (weeks*week_price) + (days*day_price);
return price;
END
$BODY$;



CREATE FUNCTION price_trigger()
RETURNS trigger AS '
BEGIN
  IF NEW.price_of_stay IS NULL THEN
    NEW.price_of_stay := calculate_price(NEW.property_id,
NEW.start_date, NEW.end_date);
  END IF;
  RETURN NEW;
END' LANGUAGE 'plpgsql';
```

```
--- Triggers

CREATE TRIGGER price_trigger
BEFORE INSERT ON rental_agreement
FOR EACH ROW
EXECUTE PROCEDURE price_trigger();

CREATE TRIGGER user_branch_trigger
    BEFORE INSERT
    ON users
    FOR EACH ROW
    EXECUTE PROCEDURE user_branch_trigger();
```

Mandatory 10 Queries

```
--1.
SELECT concat(u.first_name, ' ', u.last_name) AS guest_name,
prop.property_type AS rental_type,
r.price_of_stay AS rental_price, r.signing_date AS signing_date,
(property_address).country AS branch, pay.payment_method,
pay.payment_status
FROM users u, rental_agreement r, properties prop, payments pay
WHERE r.guest_id = u.user_id AND prop.property_id = r.property_id AND
r.payment_id = pay.payment_id;

--2.
CREATE VIEW GuestListView AS SELECT * FROM users WHERE guest = TRUE
ORDER BY branch_id ASC, user_id ASC ;


--3.

(SELECT MIN(price_of_stay), signing_date, start_date, end_date,
property_id
FROM rental_agreement AS R
       FULL OUTER JOIN
```

```
       payments AS PAY
       ON R.payment_id = PAY.payment_id
         WHERE PAY.payment_status = 'completed'
     OR PAY.payment_status = 'approved' GROUP BY  signing_date,
start_date, end_date, property_id ) ;

--4.
SELECT prop.*, rev.rating, rev.review_comments FROM properties prop,
rental_agreement r, reviews rev
WHERE r.start_date <= NOW()::DATE AND r.end_date >= NOW()::DATE AND
r.property_id = prop.property_id AND rev.review_id = r.review_id
     ORDER BY rev.rating ASC;


--5
SELECT * FROM properties WHERE property_id NOT IN(SELECT property_id
FROM rental_agreement);

--6
SELECT prop.*, r.signing_date FROM properties prop, rental_agreement r
WHERE DATE_PART('DAY', r.signing_date) = 10;

--7
SELECT * FROM employees WHERE salary >= 15000 ORDER BY manager ASC,
employee_id ASC;


--8
SELECT prop.property_type, concat(u.first_name, ' ', u.last_name) AS
host_name, prop.property_address, r.price_of_stay AS amount_paid,
     pay.payment_method FROM properties prop, users u,
rental_agreement r, payments pay WHERE r.guest_id = 3 AND
     r.property_id = prop.property_id AND u.user_id = r.guest_id AND
r.payment_id = pay.payment_id;


--9
UPDATE users SET phone_number = 1824736293 WHERE user_id = 1 AND guest
= TRUE;
```

```
--10
CREATE OR REPLACE FUNCTION public.FirstNameFirst(
      guest_id integer)
    RETURNS VARCHAR(MAX)
    LANGUAGE 'plpgsql'

    COST 100
    VOLATILE
AS $result$
DECLARE result VARCHAR(90));
DECLARE firsName VARCHAR(90); DECLARE lastName VARCHAR(90);
BEGIN
SELECT first_name into firstName FROM users WHERE guest_id = user_id;
SELECT last_name into lastName FROM users WHERE guest_id = user_id;
result = firstName + ‘ ‘ + lastName;
Return result;
END
$BODY$;
```