Task 1: Creating database, tables (Tuomas Pasanen)

Note: Text marked in *italics* means a variable name, replace it with the desired name of your own.

Note: psql supports multiline commands, so remember to use a semicolon to end them.

Note: psql is not case sensitive, but it is commonplace to type commands in uppercase.

Note: In strings (text), single quotes -> ‘ ‘ are used instead of double quotes -> “ “.

A database can be created with the command “CREATE DATABASE *dbname*[*options*]”:

```
postgres=> create database mybusiness;
WARNING:  could not flush dirty data: Function not implemented
CREATE DATABASE
postgres=> \l
                                 List of databases
    Name    |  Owner   | Encoding  | Collate | Ctype |   Access privileges
------------+----------+-----------+---------+-------+-----------------------
 mybusiness | dev      | SQL_ASCII | C       | C     |
 noobs      | dev      | SQL_ASCII | C       | C     |
 postgres   | postgres | SQL_ASCII | C       | C     |
 template0  | postgres | SQL_ASCII | C       | C     | =c/postgres          +
            |          |           |         |       | postgres=CTc/postgres
 template1  | postgres | SQL_ASCII | C       | C     | =c/postgres          +
            |          |           |         |       | postgres=CTc/postgres
```

“\l” lists the databases, where we can see that the “mybusiness” database has been created.

```
postgres=> \c mybusiness
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
You are now connected to database "mybusiness" as user "dev".
mybusiness=>
```

With “\c *dbname*”, we can connect to the database. Notice the reticle changing in the terminal, indicating the database we are currently in.

Doing “\dt” (short for “\dtables”) lists the tables in the database, which we have none of currently.

```
mybusiness=> \dt
Did not find any relations.
```

Let’s create the first table in the task, the command to do it is “CREATE TABLE *tablename(data1 datatype, data2 datatype)*”.

Inside the parentheses we have our columns

```
mybusiness=> CREATE TABLE salesman(
mybusiness(> salesman_id SERIAL PRIMARY KEY,
mybusiness(> name TEXT,
mybusiness(> city TEXT,
mybusiness(> commission DECIMAL
mybusiness(> );
CREATE TABLE
```

Notice the multiline command to make the command easier to type and read.

Now doing “\dt” shows our table:

```
mybusiness=> \dt
         List of relations
 Schema |   Name   | Type  | Owner
--------+----------+-------+-------
 public | salesman | table | dev
(1 row)
```

Furthermore, doing “SELECT * FROM *salesman*” shows the data inside our table, and how it is structured:

```
mybusiness=> SELECT * FROM salesman;
 salesman_id | name | city | commission
-------------+------+------+------------
(0 rows)
```

Now we want to alter the columns a bit, to make them fit the requirements of the task.

First is constricting the commission -column to values between 0 and 1. We can alter the column and do a check everytime something is inserted, giving an error if our check returns false:

```
mybusiness=> ALTER TABLE salesman
mybusiness-> ADD CHECK (commission > 0 and commission < 1);
```

Second is having the serial salesman_id start from 5000 instead of 0, this can be done with “ALTER SEQUENCE *table_column_seq* RESTART WITH *integer*“:

```
mybusiness=> ALTER SEQUENCE salesman_salesman_id_seq RESTART WITH 5001;
```

Finally, we can start adding rows to our table with the syntax: “INSERT INTO *tablename*(*data1 datatype, data2 datatype…*) VALUES (*value1, value2…*)”. The data and values in this command have to be in the same order, but the data does not have to be in the same order as the columns in the table.

Since the salesman_id is a serial type, it is automatically incremented and we don’t (shouldn’t) give it a value when inserting, as it affects the incrementing and can create unwanted results. Serial can also be given the value of “default”, this may be advisable.

```
mybusiness=> INSERT INTO salesman(name, city, commission)
mybusiness-> VALUES('James Hoow', 'New York', 0.15);
INSERT 0 1
mybusiness=> SELECT * FROM salesman;
 salesman_id |    name    |   city   | commission
-------------+------------+----------+------------
        5001 | James Hoow | New York |       0.15
(1 row)
```

I will next insert the rest of the rows using a single multiline command by just giving multiple values () fields.

```
mybusiness=> INSERT INTO salesman(salesman_id, name, city, commission)
mybusiness-> VALUES (5002, 'Nail Knite', 'Paris', 0.13),
mybusiness-> (5005, 'Pit Alex', 'London', 0.11),
mybusiness-> (5006, 'Mc Lyon', 'Paris', 0.14),
mybusiness-> (5007, 'Paul Adam', 'Rome', 0.13),
mybusiness-> (5003, 'Lauson Hen', 'San Jose', 0.12),
mybusiness-> (5010, 'Ben Johnson', 'San Jose', 0.13),
mybusiness-> (5011, 'Sam Lawson', 'Santiago', 0.11);
INSERT 0 7
```

```
mybusiness=> SELECT * FROM salesman;
 salesman_id |    name     |   city   | commission
-------------+-------------+----------+------------
        5001 | James Hoow  | New York |       0.15
        5002 | Nail Knite  | Paris    |       0.13
        5005 | Pit Alex    | London   |       0.11
        5006 | Mc Lyon     | Paris    |       0.14
        5007 | Paul Adam   | Rome     |       0.13
        5003 | Lauson Hen  | San Jose |       0.12
        5010 | Ben Johnson | San Jose |       0.13
        5011 | Sam Lawson  | Santiago |       0.11
(8 rows)
```

Despite the salesman_id field being serial, in the interest of the task I gave the id's manually. It might be better practice to just not specify a salesman_id field, or give it the value of "default".

We can try out our checks we implemented by trying to insert a row which breaks the check:

```
mybusiness=> INSERT INTO salesman(salesman_id, name, city, commission)
mybusiness-> VALUES(54, 'Billy Fish', 'Rovaniemi', 23);
ERROR:  new row for relation "salesman" violates check constraint "salesman_commission_check"
DETAIL:  Failing row contains (54, Billy Fish, Rovaniemi, 23).
```

An error is thrown and the row is not added (Also an id value was specified, so it does not follow the increment). However if we change commission to follow our constraint:

```
mybusiness=> INSERT INTO salesman(salesman_id, name, city, commission)
VALUES(54, 'Billy Fish', 'Rovaniemi', 0.8);
INSERT 0 1
mybusiness=> SELECT * FROM salesman;
 salesman_id |    name     |   city    | commission
-------------+-------------+-----------+------------
        5001 | James Hoow  | New York  |       0.15
        5002 | Nail Knite  | Paris     |       0.13
        5005 | Pit Alex    | London    |       0.11
        5006 | Mc Lyon     | Paris     |       0.14
        5007 | Paul Adam   | Rome      |       0.13
        5003 | Lauson Hen  | San Jose  |       0.12
        5010 | Ben Johnson | San Jose  |       0.13
        5011 | Sam Lawson  | Santiago  |       0.11
          54 | Billy Fish  | Rovaniemi |        0.8
(9 rows)
```

It gets added. I will remove this row in the interest of the task, with “DELETE FROM *tablename* WHERE *variable* = *somethingelse*”:

```
mybusiness=> DELETE FROM salesman
mybusiness-> WHERE name = 'Billy Fish';
DELETE 1
```

Now to create the second table, which introduces foreign keys.

The command is basically the same: create a new table, specify the columns, add the rows.

```
mybusiness=> CREATE TABLE customer (
mybusiness(> customer_id SERIAL PRIMARY KEY,
mybusiness(> cust_name text,
mybusiness(> city text,
mybusiness(> grade INT,
mybusiness(> salesman_id INT REFERENCES salesman(salesman_id));
```

```
mybusiness=> SELECT * FROM customer;
 customer_id | cust_name | city | grade | salesman_id
-------------+-----------+------+-------+-------------
(0 rows)
```

Notice that the salesman_id references the salesman_id -column of the salesman table. This means that it references the salesman table, if for example we wanted to find out more about the salesmen who’ve done the most sales or information about an individual salesman, we can easily reference the id’s.

We can again start the incrementation from a different number:

```
mybusiness=> ALTER SEQUENCE customer_customer_id_seq RESTART WITH 3001;
```

Now to just insert the rows inside the table:

```
mybusiness=> INSERT INTO customer(customer_id, cust_name, city, grade, salesman_id)
mybusiness-> VALUES( 3002, 'Nick Rimando', 'New York', 100, 5001),
mybusiness-> (3007, 'Brad Davis', 'New York', 200, 5001),
mybusiness-> (3005, 'Graham Zusi', 'California', 200, 5002),
mybusiness-> (3008, 'Julian Green', 'London', 300, 5002),
mybusiness-> (3004, 'Fabian Johnson', 'Paris', 300, 5006),
mybusiness-> (3009, 'Geoff Cameron', 'Berlin', 100, 5003),
mybusiness-> (3003, 'Jozy Altidor', 'Moscow', 200, 5007),
mybusiness-> (3001, 'Brad Guzan', 'London', 300, 5005),
mybusiness-> (3010, 'Marion Cameron', 'San Jose', 300, 5010);
```

Finally, we can use the affromentioned commands to create the last table, orders:

```
mybusiness=> CREATE TABLE orders(
mybusiness(> ord_no INT PRIMARY KEY,
mybusiness(> purch_amt DECIMAL,
mybusiness(> ord_date DATE,
mybusiness(> customer_id INT REFERENCES customer(customer_id),
mybusiness(> salesman_id INT REFERENCES salesman(salesman_id));
```

```
mybusiness=> SELECT * FROM orders;
 ord_no | purch_amt | ord_date | customer_id | salesman_id
--------+-----------+----------+-------------+-------------
(0 rows)
```

Again, customer_id and salesman_id are foreign keys, representing the id fields of their respective tables. DATE is a new datatype, but it just represents a date (duh), where it is recommended to use the ISO 8601 format (YYYY-MM-DD), wrapped in single quotes (‘ ‘).

```
mybusiness=> INSERT INTO orders(ord_no, purch_amt, ord_date, customer_id, salesman_id) VALUES(
70001, 150.5, '2012-10-05', 3005, 5002),
(70009, 270.65, '2012-09-10', 3001, 5005),
(70002, 65.26, '2012-10-05', 3002, 5001),
(70004, 110.5, '2012-08-17', 3009, 5003),
(70007, 948.5, '2012-09-10', 3005, 5002),
(70005, 2400.6, '2012-07-27', 3007, 5001),
(70008, 5760, '2012-09-10', 3002, 5001),
(70010, 1983.43, '2012-10-10', 3004, 5006),
(70003, 2480.4, '2012-10-10', 3009, 5003),
(70012, 250.45, '2012-06-27', 3008, 5002),
(70011, 75.29, '2012-08-17', 3003, 5007),
(70013, 3045.6, '2012-04-25', 3002, 5001),
(70014, 1786.4, '2012-06-25', 3004, 5006);
```

```
mybusiness=> SELECT * FROM orders;
 ord_no | purch_amt |  ord_date  | customer_id | salesman_id
--------+-----------+------------+-------------+-------------
  70001 |     150.5 | 2012-10-05 |        3005 |        5002
  70009 |    270.65 | 2012-09-10 |        3001 |        5005
  70002 |     65.26 | 2012-10-05 |        3002 |        5001
  70004 |     110.5 | 2012-08-17 |        3009 |        5003
  70007 |     948.5 | 2012-09-10 |        3005 |        5002
  70005 |    2400.6 | 2012-07-27 |        3007 |        5001
  70008 |      5760 | 2012-09-10 |        3002 |        5001
  70010 |   1983.43 | 2012-10-10 |        3004 |        5006
  70003 |    2480.4 | 2012-10-10 |        3009 |        5003
  70012 |    250.45 | 2012-06-27 |        3008 |        5002
  70011 |     75.29 | 2012-08-17 |        3003 |        5007
  70013 |    3045.6 | 2012-04-25 |        3002 |        5001
  70014 |    1786.4 | 2012-06-25 |        3004 |        5006
(13 rows)
```