COMP-421 Database Systems

Term: Winter 2018

Title: Project 3: writing your application

Group: 22

Question 1:

We have written a stored procedure to iterates through all customers and saves the highest spending customer to a new weak entity of customer called bestcustomer on our project database.
(refer to storedproceedure.sql if needed)

```
CREATE OR REPLACE FUNCTION updateBestCustomer() RETURNS void AS $$

DECLARE custid INT DEFAULT 0;
DECLARE cust_sumspent FLOAT DEFAULT 0;
DECLARE max_custid_sofar INT DEFAULT 0;
DECLARE max_cust_sumspent FLOAT DEFAULT 0;

DECLARE c1 CURSOR FOR
SELECT customerid, SUM(caseprice)
FROM purchase p, beercase bc
WHERE p.caseid=bc.caseid
GROUP BY customerid;

BEGIN

OPEN c1;
LOOP
FETCH C1 INTO custid, cust_sumspent;
EXIT WHEN NOT FOUND;

IF (cust_sumspent > max_cust_sumspent) THEN
max_cust_sumspent = cust_sumspent;
max_custid_sofar = custid;
END IF;
END LOOP;
CLOSE c1;

DELETE FROM bestcustomer;
INSERT INTO bestcustomer VALUES (max_custid_sofar, max_cust_sumspent);
-- UPDATE bestcustomer SET customerid = max_custid_sofar, totalspent = max_cust_sumspent;
END; $$

LANGUAGE plpgsql;
```

```
-- DROP TABLE bestcutomer;

CREATE TABLE bestcustomer (
customerid INTEGER,
totalspent FLOAT,
PRIMARY KEY(customerid),
FOREIGN KEY(customerid) REFERENCES customer(customerid)
);
```

To demonstrate that the programs had their intended effect:
Before:

```
cs421=> SELECT customerid, SUM(caseprice)
cs421-> FROM purchase p, beercase bc
cs421-> WHERE p.caseid=bc.caseid
cs421-> GROUP BY customerid;
 customerid |  sum
------------+--------
   83158536 | 130.95
  656656298 |  24.99
  997759725 |  55.98
  210307036 |  50.97
  944957552 |  50.97
    8788967 | 173.94
(6 rows)
```

After:

```
cs421=> select * from bestcustomer;
 customerid | totalspent
------------+------------
    8788967 |     173.94
```

Question 2:

We have written a java program for our database. Eclipse was used with the jdbc.jar file added as an external library.
(refer to BeerCompany.java)

```
Please enter the number for the option you want:
1) Get all beer names in the database
2) Look up customer information
3) Look up all the types of beer a given customer has purchased
4) Insert a new ingredient
5) Delete an ingredient
6) Exit
```

Here is how the program works when compile and run:

1) Get all beer names in the database
   SELECT DISTINCT beerName FROM beer;

```
1
name:  LukeSkyWalkersGreenNippleJuice
name:  GaseousVolcano
name:  LiquidIceberg
name:  TearsOfAVirgin
name:  SaltySeaWater
name:  ToiletWater
Closing Connection.
```

2) Look up customer information
   SELECT * FROM customer WHERE customername='Morty';

```
2
Please enter customer name:
Morty
customerid:  997759725
customername:  Morty
emailadress:  dropitlikeitscold@hotmail.com
deliveraddress:  Hoth Rebel Base
employeeid:  438421923
Closing Connection.
```

3) Look up all the types of beer Morty has purchased
   SELECT DISTINCT beername FROM beer b, purchase p, customer c WHERE p.customerid=c.customerid AND c.customername='Rick' AND b.beerid=p.beerid;

```
3
Please enter customer name:
Rick
beername:  LukeSkyWalkersGreenNippleJuice
beername:  ToiletWater
beername:  SaltySeaWater
beername:  LiquidIceberg
beername:  TearsOfAVirgin
Closing Connection.
```

4) Add a new ingredients ‘swag’
INSERT INTO ingredients VALUES('Chips');

```
4
Please enter new ingredient name:
Chips
Ingredient added.
Closing Connection.
```

5) Delete a ingredients ‘swag’
DELETE FROM ingredients WHERE ingredients.ingredientname='Chips';

```
5
Please enter name of ingredient to delete:
Chips
Ingredient removed.
Closing Connection.
```

6) Quit

```
6
Exiting and closing conections.
```

Question 3:

We have created two indexes that help to speed up queries.
(refer to dropindex.sql, index.sql if needed)

DROP INDEX dates;
DROP INDEX alccontent;

CREATE INDEX dates ON batch(datemade);
CREATE INDEX alccontent ON beer(abv);

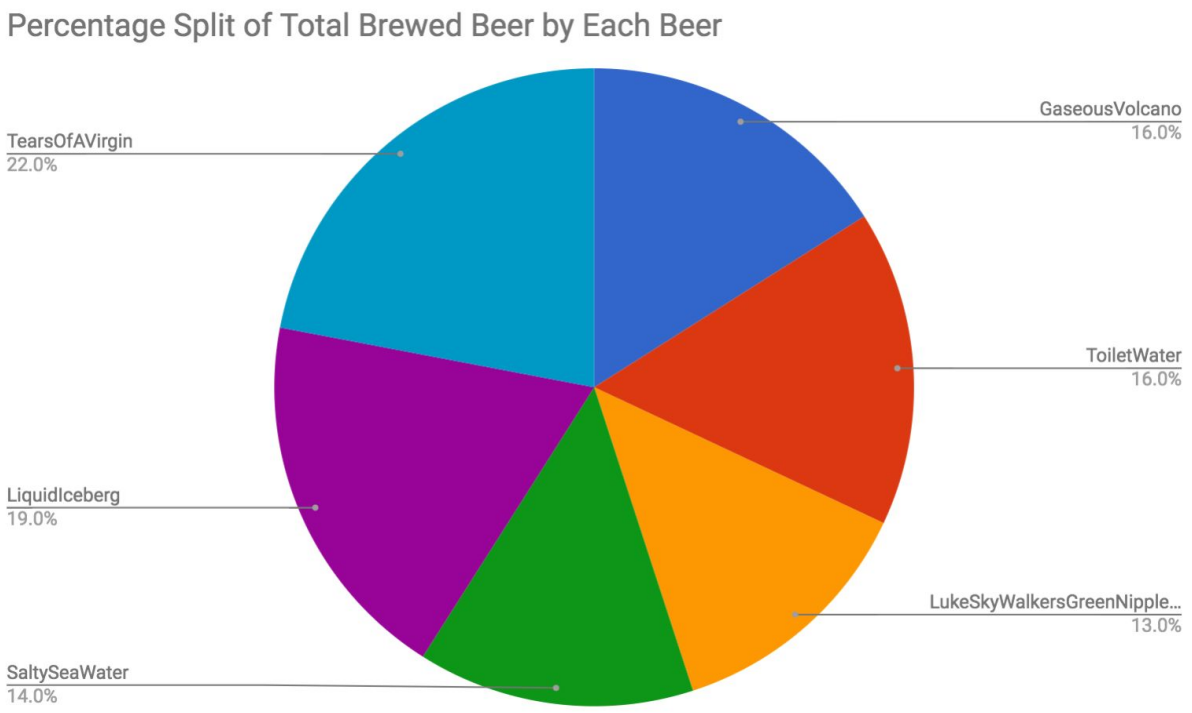
/* descriptions */
Index on datemade:
We want to create the index on the datemade on the batches so that we can quicker access all the batches of beer made on a specific date.
This is useful when we want to find out what was most recently made, what is potentially expiring soon and we would want to get rid of sooner.

Index on abv:
We want to create an index on the abv so that when we want to search or pull up all the beers that fall under a certain alcoholic content or strength it is faster.

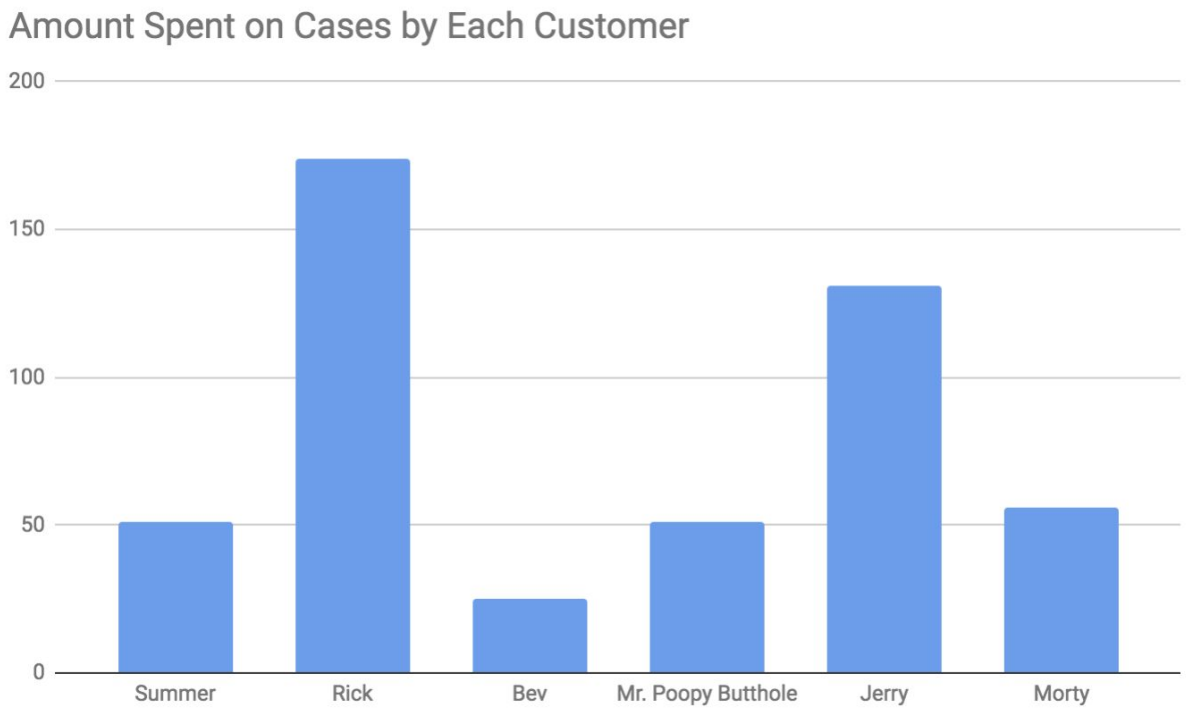
Question 4:

We have written two SQL queries and exported the data into CSV format and visualized them with separate plotting techniques. (refer to Project3-Q4.xlsx, q4p1.csv, q4p2.csv if needed)

```
cs421=> select beer.beername, batch.beerid, count(batch.beerid)
cs421-> from batch, beer
cs421-> where beer.beerid = batch.beerid group by beer.beername,batch.beerid;
           beername             |  beerid   | count
--------------------------------+-----------+-------
 GaseousVolcano                 | 185824422 |    16
 ToiletWater                    | 638776299 |    16
 LukeSkyWalkersGreenNippleJuice | 935127238 |    13
 SaltySeaWater                  | 403235613 |    14
 LiquidIceberg                  | 423785392 |    19
 TearsOfAVirgin                 | 962383855 |    22
(6 rows)
```




```
cs421=> SELECT customername, p.customerid, SUM(caseprice) as BeerBought
cs421-> FROM purchase p, beercase bc, customer c
cs421-> WHERE p.caseid=bc.caseid AND c.customerid=p.customerid
cs421-> GROUP BY customername,p.customerid;
    customername     | customerid | beerbought
---------------------+------------+------------
 Summer              |  210307036 |      50.97
 Rick                |    8788967 |     173.94
 Bev                 |  656656298 |      24.99
 Mr. Poopy Butthole  |  944957552 |      50.97
 Jerry               |   83158536 |     130.95
 Morty               |  997759725 |      55.98
(6 rows)
```



Question 5:

We have written a Triggers that shows the price different every time you update the price on the beercase in the inventory.
(refer to Q5.sql in needed)

```
CREATE TRIGGER display_price_changes
BEFORE UPDATE ON beercase
FOR EACH ROW
WHEN (NEW.caseid > 0)
  DECLARE
     price_diff number;

BEGIN
   price_diff = NEW.caseprice  - OLD.caseprice;
   raise notice 'Old Case Price: ', OLD.caseprice;
   raise notice 'New Case Price: ', NEW.caseprice;
   raise notice 'Case Price difference: ', price_diff;
END;
```

When the above code is executed at the SQL prompt, it produces the following result:

Trigger created.

When executed the following SQL query:
INSERT INTO beercase (caseid, caseprice, casesize, batchnum)
VALUES(12341111, 12.99, 10, 916992581)
We get the **INSERT 0 1** message.

When we UPDATE this record with the follow SQL query:
UPDATE beercase SET caseprice=13.99 WHERE caseid=12341111;
We get the **UPDATE 1** message.

And the above create trigger, display_price_changes will be fired and it will display the following result:
Old Case Price: 12.99
New Case Price: 13.99
Case Price difference: 1