Database Design Proposal
Bridget Leahy
Table of Contents

Executive Summary_________________3
ER Diagram________________________4
Create Table Statements:
  Zipcode Table_________________5
  People Table__________________6
  Banned Customers Table________7
  Customers Table_______________8
  Staff Table___________________9
  Managers Table________________10
  Crew Table___________________11
  Stores Table__________________12
  StoreOfferings Table__________13
  Items Table___________________14
  Drinks Table__________________15
  Food Table___________________16
  Orders Table__________________17

Create View Statements:
  inactiveCustomers_______________18
  unorderedFood____________________19
  perksLocations__________________20

Stored Procedures:
  searchCustomerName_______________21
  getCalories______________________23
  storeLocation__________________25

Triggers:
  checkFood_______________________26
  checkDrink______________________28
  priceCeiling_____________________30

Reports__________________________32
User Roles/Security______________33
Problems/Enhancements____________34
Executive Summary

The Dunkin Donuts database has been created to keep track of all aspects of the operation of this successful company.

The following paper provides an information regarding the database as well as indications of its uses. The ER diagram, create statements for tables, and their sample data are first presented. Following this, uses include creating queries, views, stored procedures, reports, and triggers are designed and tested. The function of the database is to provide accurate and relevant information on all bases of the company – from customers, to food, to store locations, to managers and staff.

America runs on Dunkin and Dunkin runs on its database.
E/R Diagram
Zipcode Table

The table containing all zipcodes for stores and people, whether they are customers, managers, or crew. There is a city, state, and country for each unique zipcode.

DROP TABLE IF EXISTS ZipCode;
CREATE TABLE ZipCode (  
    ZipCode int not null,  
    City text,  
    State text,  
    Country text,  
    primary key(ZipCode)
);

<table>
<thead>
<tr>
<th>Zipcode</th>
<th>City</th>
<th>State</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>26475</td>
<td>CT</td>
<td>USA</td>
</tr>
<tr>
<td>2</td>
<td>37446</td>
<td>NJ</td>
<td>USA</td>
</tr>
<tr>
<td>3</td>
<td>12601</td>
<td>NY</td>
<td>USA</td>
</tr>
<tr>
<td>4</td>
<td>12538</td>
<td>NY</td>
<td>USA</td>
</tr>
<tr>
<td>5</td>
<td>11524</td>
<td>MA</td>
<td>USA</td>
</tr>
</tbody>
</table>

Functional Dependencies:
Zipcode → city, state, country
People Table

Table containing information on any customer, banned customer, crew, or manager. Gives the person’s name, date of birth, street address, and zip code based on their unique ID.

Functional Dependencies:
PID → FirstName, LastName, DOB, StreetAddress, ZipCode

DROP TABLE IF EXISTS People;
CREATE TABLE People (PID char(4) not null, FirstName text, LastName text, DOB date, StreetAddress text, ZipCode int not null references ZipCode(ZipCode), primary key(PID) );
Banned Customers Table

Any person who has been banned from Dunkin Donuts will be listed here, along with the date of their ban and the reason for the ban. The person must already exist in the People table to be banned.

DROP TABLE IF EXISTS BannedCustomers;
CREATE TABLE BannedCustomers (  
  PID char(4) not null references People(PID),  
  DateofBan date,  
  Reason text,  
  primary key (PID) 
);

Functional Dependencies:
  PID → DateofBan, Reason

<table>
<thead>
<tr>
<th>pid character(4)</th>
<th>dateofban date</th>
<th>reason text</th>
</tr>
</thead>
<tbody>
<tr>
<td>p002</td>
<td>2017-04-21</td>
<td>Fan of Starbucks</td>
</tr>
<tr>
<td>p010</td>
<td>1971-05-01</td>
<td>Brought Starbucks into Dunkin</td>
</tr>
</tbody>
</table>
The customers table contains all of Dunkin’s customers and whether it is true or false that they are a DD Perks Member. A person cannot be in the customer table unless they exist in the people table. If information regarding whether or not the customers is a perks member is not given, the default value will be false.

```
DROP TABLE IF EXISTS Customers;
CREATE TABLE Customers (  
  PID char(4) not null references People(PID),  
  PerksMember text DEFAULT 'No',  
  primary key(PID),  
  CONSTRAINT CHK_Member CHECK  
    (PerksMember='Yes' OR PerksMember='No')  
);  
```

<table>
<thead>
<tr>
<th>pid character(4)</th>
<th>perksmember text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 p009</td>
<td>No</td>
</tr>
<tr>
<td>2 p004</td>
<td>Yes</td>
</tr>
<tr>
<td>3 p006</td>
<td>Yes</td>
</tr>
<tr>
<td>4 p018</td>
<td>No</td>
</tr>
<tr>
<td>5 p019</td>
<td>Yes</td>
</tr>
<tr>
<td>6 p020</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Functional Dependency: PID \( \rightarrow \) PerksMember
Staff Table

The staff table contains the ID of each staff member. A person cannot be added to the staff table unless they already are in the person table.

DROP TABLE IF EXISTS Staff;
CREATE TABLE Staff (
    PID char(4) not null references People(PID),
    primary key(PID)
);

Functional Dependency:
    PID →

<table>
<thead>
<tr>
<th>pid</th>
<th>character(4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p001</td>
</tr>
<tr>
<td>2</td>
<td>p003</td>
</tr>
<tr>
<td>3</td>
<td>p005</td>
</tr>
<tr>
<td>4</td>
<td>p007</td>
</tr>
<tr>
<td>5</td>
<td>p011</td>
</tr>
<tr>
<td>6</td>
<td>p012</td>
</tr>
<tr>
<td>7</td>
<td>p013</td>
</tr>
<tr>
<td>8</td>
<td>p014</td>
</tr>
<tr>
<td>9</td>
<td>p015</td>
</tr>
<tr>
<td>10</td>
<td>p017</td>
</tr>
<tr>
<td>11</td>
<td>p016</td>
</tr>
</tbody>
</table>
Managers Table

---

The managers table contains the person ID, hourly wage in U.S. dollars, original hire date, promotion to manager date, and hourly wage for each manager. A person cannot be added to the manager table unless he or she already exists in the staff table. The person must be a manager for the same store as in the staff table. A person cannot be promoted at a date earlier than he or she is hired and if salary is not entered, the default value will be $20 based on company policy.

```
DROP TABLE IF EXISTS Managers;
CREATE TABLE Managers (  
    PID char(4) not null references Staff(PID),  
    HireDate date,  
    PromotionDate date,  
    HourlyWageUSD numeric(10,2) default 20.00,  
    primary key(PID),  
    check (PromotionDate > HireDate)
);
```

Functional Dependencies:

PID → HireDate, PromotionDate, HourlyWageUSD

<table>
<thead>
<tr>
<th>pid</th>
<th>character(4)</th>
<th>hiredate</th>
<th>promotiondate</th>
<th>hourlywageUSD numeric(10,2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p001</td>
<td>2016-02-03</td>
<td>2017-02-03</td>
<td>20.00</td>
</tr>
<tr>
<td>2</td>
<td>p007</td>
<td>2015-10-15</td>
<td>2016-12-31</td>
<td>25.00</td>
</tr>
<tr>
<td>3</td>
<td>p015</td>
<td>2013-04-19</td>
<td>2014-06-13</td>
<td>27.00</td>
</tr>
<tr>
<td>4</td>
<td>p016</td>
<td>2012-09-01</td>
<td>2014-12-01</td>
<td>25.00</td>
</tr>
<tr>
<td>5</td>
<td>p017</td>
<td>2015-03-15</td>
<td>2017-02-01</td>
<td>30.00</td>
</tr>
</tbody>
</table>
Crew Table

The crew table contains the person ID, hourly wage in U.S. dollars, original hire date, and hourly wage for each crew member. A person cannot be added to the crew table unless he or she already exists in the staff table. The person must be a crew member for the same store as in the staff table. The default hourly wage is $10.

DROP TABLE IF EXISTS Crew;
CREATE TABLE Crew (  
  PID char(4) not null references Staff(PID),  
  SID char(4) not null references Staff(SID),  
  HireDate date,  
  HourlyWageUSD numeric(10,2) default 10.00,  
  primary key(PID)
);

Functional Dependencies:
  PID → HireDate, HourlyWageUSD
Stores Table

The stores table contains the ID, street address, and zipcode for each store. A zipcode cannot be given to a store unless it already exists in the zipcode table.

```sql
DROP TABLE IF EXISTS Stores;
CREATE TABLE Stores (  
    SID char(4) not null,
    StreetAddress text,
    Zipcode int not null references Zipcode(Zipcode),
    primary key(SID)
);
```

Functional Dependencies:

SID → StreetAddress, Zipcode
StoreOfferings Table

The store offerings table contains which items are offered at each store. A store ID cannot be listed unless it is already in the stores table. An item ID cannot be listed unless it is already in the items table.

DROP TABLE IF EXISTS StoreOfferings;
CREATE TABLE StoreOfferings (  
    SID char(4) not null references Stores(SID),
    IID char(4) not null references Items(IID),
    primary key(SID, IID)
);

Functional Dependencies:  
SID, IID →
The items table contains the id and type of item for any food or drink sold at any Dunkin location.

```
DROP TABLE IF EXISTS Items;
CREATE TABLE Items (  
    IID char(4) not null,
    Type text,
    primary key(IID),
    CONSTRAINT CHK_Type CHECK (Type='food' OR Type='drink')
);
```

Functional Dependency:
- \( \text{IID} \rightarrow \text{Type} \)

<table>
<thead>
<tr>
<th>id</th>
<th>character(4)</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1001</td>
<td>food</td>
</tr>
<tr>
<td>2</td>
<td>1002</td>
<td>drink</td>
</tr>
<tr>
<td>3</td>
<td>1003</td>
<td>food</td>
</tr>
<tr>
<td>4</td>
<td>1004</td>
<td>food</td>
</tr>
<tr>
<td>5</td>
<td>1005</td>
<td>food</td>
</tr>
<tr>
<td>6</td>
<td>1006</td>
<td>drink</td>
</tr>
<tr>
<td>7</td>
<td>1007</td>
<td>drink</td>
</tr>
<tr>
<td>8</td>
<td>1008</td>
<td>drink</td>
</tr>
<tr>
<td>9</td>
<td>1009</td>
<td>food</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>drink</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>food</td>
</tr>
</tbody>
</table>
Drinks Table

The drinks table contains each drink’s ID, description, calorie count, and price in U.S. dollars. An item cannot be added to the drinks table unless it already exists as a drink item in the items table.

DROP TABLE IF EXISTS Drinks;
CREATE TABLE Drinks (  
    IID char(4) not null references items(IID),  
    Description text,  
    Calories integer,  
    PriceUSD numeric(10,2)
);

Functional Dependencies:  
  IID \rightarrow Description, Calories, PriceUSD
The food table contains each food item’s ID, description, calorie count, and price in U.S. dollars. An item cannot be added to the food table unless it already exists as a food item in the items table.

```sql
DROP TABLE IF EXISTS Food;
CREATE TABLE Food(
    IID char(4) not null references items(IID),
    Description text,
    Calories integer,
    PriceUSD numeric(10,2)
);
```

Functional Dependencies:
IID → Description, Calories, PriceUSD
Orders Table

The orders table lists the order id, customer who ordered, item that was ordered, store at which the item was ordered, and the date of the order. An item cannot be in the orders table unless it is already in the items table. A customer cannot be in the orders table unless he or she is already in the customers table. An store cannot be in the orders table unless it is already in the stores table.

DROP TABLE IF EXISTS Orders;
CREATE TABLE Orders (  
   OID char(4) not null,
   PID char(4) not null references Customers(PID),
   SID char(4),
   IID char(4),
   DateOrdered date,
   totalUSD numeric(10,2),
   foreign key(SID, IID) references StoreOfferings(SID, IID),
   primary key(OID)
);

Functional Dependencies:  
   OID → PID, SID, IID, DateOrdered
inactiveCustomers view

---

Gets the id, first name, and last name of any customer who has not placed an order

create or replace view inactiveCustomers
    as select distinct p.pid, p.firstname, p.lastname
    from customers c, orders o, people p
    where c.pid not in
        (select pid from orders)
        and c.pid = p.pid
    order by p.pid ASC;

Select * from inactiveCustomers;

<table>
<thead>
<tr>
<th>pid</th>
<th>firstname</th>
<th>lastname</th>
</tr>
</thead>
<tbody>
<tr>
<td>p019</td>
<td>Michael</td>
<td>Leahy</td>
</tr>
</tbody>
</table>
unorderedFood view

Retrieves information about each food that has never been ordered

create or replace view unorderedFood
as select * from food f
where f.iid in (select distinct i.iid
    from items i
    where i.iid not in (select iid from orders))
order by f.iid ASC;

select * from unorderedFood;
View: *perksLocations*

A view designed to get the city and states of customers who are perks members. This is to analyze the locations where the new perks membership is being used.

```sql
create or replace view perksLocations as
  select z.city, z.state
  from zipcode z, customers c, people p
  where z.zipcode = p.zipcode
  and c.pid = p.pid and c.perksmember = 'Yes'
  group by z.city, z.state;

Select * from perksLocations;
```
Stored Procedure: `searchCustomerName`

This procedure allows users to search for people based on an element of their first name, last name, or both. This is also easily adaptable to be able to search only staff, customers, managers, crew, or banned customers instead of every person.

```sql
CREATE OR REPLACE FUNCTION searchCustomerName(TEXT, TEXT, REFCURSOR) RETURNS refcursor AS $$
DECLARE
    searchFirst TEXT := $1;
    searchLast TEXT := $2;
    resultSet REFCURSOR := $3;
BEGIN
    OPEN resultSet FOR
        SELECT *
        FROM people
        WHERE firstname LIKE searchFirst
        AND lastname LIKE searchLast;
    return resultSet;
end;
$$
LANGUAGE plpgsql;
```
searchCustomerName examples

SELECT searchCustomerName('A%', 'L%', 'ref');
FETCH ALL FROM ref;

SELECT searchCustomerName('%', 'Leah%', 'ref1');
FETCH ALL FROM ref1;

SELECT searchCustomerName('%n', '%', 'ref2');
FETCH ALL FROM ref2;
Stored Procedure: **getCalories**

This allows for users to search for a section of a customer’s first, last, or both names. The result of the search includes not only the customer’s full name, but his or her ID and total calories consumed from every order.

```
CREATE OR REPLACE FUNCTION getCalories(TEXT, TEXT, REFCURSOR) RETURNS refcursor AS $$
DECLARE
    searchFirst TEXT := $1;
    searchLast TEXT := $2;
    resultSet REFCURSOR := $3;
BEGIN
    OPEN resultSet FOR
        select p.pid, p.firstname, p.lastname, sum(f.calories) + sum(d.calories) as calories
        from orders o left outer join drinks d on o.iid = d.iid
            left outer join food f on o.iid = f.iid
            left outer join customers c on o.pid = c.pid
            left outer join people p on o.pid = p.pid
        where o.pid=c.pid and
            p.firstname LIKE searchFirst
        AND p.lastname LIKE searchLast
        group by p.pid, p.firstname, p.lastname;
    return resultSet;
END;
$$
LANGUAGE plpgsql;
```
Testing `getCalories`

```sql
SELECT getCalories('R%', 'W%', 'ref3');
FETCH ALL FROM ref3;
```

```sql
SELECT getCalories('%%', 'M%', 'ref4');
FETCH ALL FROM ref4;
```
This procedure takes input of a city name and finds all store location information using LIKE.

```sql
CREATE OR REPLACE FUNCTION storeLocation(TEXT, REFCURSOR) RETURNS refcursor AS $$
DECLARE
    searchZip TEXT := $1;
    resultSet REFCURSOR := $2;
BEGIN
    OPEN resultSet FOR
    select * from zipcode z
    where z.city like searchZip
    and z.zipcode in (select zipcode from stores);
    return resultSet;
end;
$$
LANGUAGE plpgsql;
```
Trigger: checkFood

This trigger ensures that a drink is not added to the food table. If an item is listed as a drink in the item table and information regarding the drink is incorrectly added to the food table, the row containing that item’s ID is deleted from the food table and added to the drink table.

```
CREATE OR REPLACE FUNCTION checkFood()
RETURNS TRIGGER AS
$$
BEGIN
    IF (select i.type from items i where i.iid=NEW.IID ) = 'drink'
    THEN
        delete from food where iid = NEW.IID;
        insert into drinks(IID, description, calories, priceUSD) values (NEW.IID, NEW.Description, NEW.CALORIES, NEW.PriceUSD);
    END IF;
    RETURN NEW;
END;
$$
language plpgsql;

CREATE TRIGGER checkFood
AFTER INSERT ON Food
FOR EACH ROW
EXECUTE PROCEDURE checkFood();
```
Testing *checkFood*

---

Add a drink to the item table, then try to insert more information on it into the food table:

```sql
insert into items(IID, Type) values ('i013', 'drink');
insert into food(IID, description, calories, priceUSD) values ('i013', 'Testing', 310, 3);
```

<table>
<thead>
<tr>
<th>id</th>
<th>character(4)</th>
<th>description</th>
<th>calories</th>
<th>priceUSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>i006</td>
<td>drink</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>i007</td>
<td>drink</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>i008</td>
<td>food</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>i009</td>
<td>food</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>i010</td>
<td>drink</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>i011</td>
<td>food</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>i013</td>
<td>drink</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>character(4)</th>
<th>description</th>
<th>calories</th>
<th>priceUSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>i001</td>
<td>Plain Bagel</td>
<td>310</td>
<td>2.50</td>
</tr>
<tr>
<td>2</td>
<td>i003</td>
<td>Chocolate Coconut Donut</td>
<td>400</td>
<td>1.00</td>
</tr>
<tr>
<td>3</td>
<td>i004</td>
<td>Lemon Stick</td>
<td>430</td>
<td>1.50</td>
</tr>
<tr>
<td>4</td>
<td>i005</td>
<td>Blueberry Bagel</td>
<td>310</td>
<td>3.00</td>
</tr>
<tr>
<td>5</td>
<td>i009</td>
<td>Poppy Seed Bagel</td>
<td>350</td>
<td>3.00</td>
</tr>
<tr>
<td>6</td>
<td>i011</td>
<td>Blueberry Crumb Cake Donut</td>
<td>420</td>
<td>1.25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>character(4)</th>
<th>description</th>
<th>calories</th>
<th>priceUSD(10,2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1102</td>
<td>Medium Dunkaccino</td>
<td>350</td>
<td>3.00</td>
<td></td>
</tr>
<tr>
<td>1106</td>
<td>Medium Hot Chocolate</td>
<td>330</td>
<td>2.75</td>
<td></td>
</tr>
<tr>
<td>1107</td>
<td>Small Hot Coffee</td>
<td>5</td>
<td>1.75</td>
<td></td>
</tr>
<tr>
<td>1108</td>
<td>Espresso</td>
<td>5</td>
<td>1.99</td>
<td></td>
</tr>
<tr>
<td>1109</td>
<td>Espresso with Sugar</td>
<td>30</td>
<td>2.25</td>
<td></td>
</tr>
<tr>
<td>1113</td>
<td>Testing</td>
<td>310</td>
<td>3.00</td>
<td></td>
</tr>
</tbody>
</table>

Item 13 is classified as a drink, so it is removed from the food table and inserted into the drinks table.
Trigger: checkDrink

This trigger ensures that a food is not added to the drink table. If an item is listed as a food in the item table and information regarding the food is incorrectly added to the drink table, the row containing that item’s ID is deleted from the drink table and added to the food table.

```
CREATE OR REPLACE FUNCTION checkDrink()
RETURNS TRIGGER AS
$$
BEGIN
  IF (select i.type from items i where i.iid=NEW.IID ) = 'food'
    THEN
      delete from drinks where iid = NEW.IID;
      insert into food(IID, description, calories, priceUSD) values (NEW.IID, NEW.Description, NEW.CALORIES, NEW.PriceUSD);
      END IF;
  RETURN NEW;
END;
$$
language plpgsql;

CREATE TRIGGER checkDrink
AFTER INSERT ON Drinks
FOR EACH ROW
EXECUTE PROCEDURE checkDrink();
```
Testing `checkDrink`

---

Insert a new item (‘i012’) into the items table as a food:

```sql
insert into items(IID, Type) values ('i012', 'food');
insert into drinks(IID, description, calories, priceUSD) values ('i012', 'Testing food', 352, 2.29);
```

<table>
<thead>
<tr>
<th>Items:</th>
<th>drinks:</th>
<th>food:</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Items table" /></td>
<td><img src="image2" alt="Drinks table" /></td>
<td><img src="image3" alt="Food table" /></td>
</tr>
</tbody>
</table>

Item 12 is classified as a food, so it is removed from the drinks table and inserted into the food table.
Dunkin has enacted a new policy in which food items sold must remain under $5 to attract customers. If a food item is added that costs $5 or more, it is immediately deleted from the food and items tables.

CREATE OR REPLACE FUNCTION foodCeiling()
RETURNS TRIGGER AS
$$
BEGIN
    IF NEW.priceUSD >= 5 THEN
        delete from food where priceUSD = NEW.priceUSD;
        delete from items where iid = NEW.iid;
    END IF;
    RETURN NEW;
END;
$$
language plpgsql;

CREATE TRIGGER foodCeiling
AFTER INSERT ON Food
FOR EACH ROW
EXECUTE PROCEDURE foodCeiling();
Testing \texttt{priceCeiling}

---

Try to insert an item that is too expensive to check that it gets removed:

\begin{verbatim}
insert into items(IID, Type) values ('i014', 'food');
insert into food(IID, description, calories, priceUSD) values ('i014', 'Too Expensive', 310, 6);
\end{verbatim}

\begin{verbatim}
select * from food;
select * from items;
\end{verbatim}
Reports on Sales in 2017

---

Total money collected in 2017:
```
select sum(totalUSD)
from orders
where dateordered >= '2017-01-01';
```

Total number of orders in 2017:
```
select count(oid) from orders where dateordered >= '2017-01-01';
```

Average calories from every item:
```
Select round((avg(d.calories) + avg(f.calories))/2) from drinks d, food f;
```

Average price from every item:
```
Select round((round(avg(d.priceUSD), 2) + round(avg(f.priceUSD), 2))/2, 2)
from drinks d, food f;
```
User Roles

There are three user roles: Admin, CEO, and Managers

create role admin;
create role CEO;
create role managers;

ADMIN: Admin has administrative power over the entirety of the Dunkin database.
grant all on all tables in schema public to admin;

MANAGERS: Have complete power over the crew. They cannot delete people, customers, or banned customers. They can only select orders, items, food, drinks, and store offerings.
grant SELECT, INSERT, UPDATE, DELETE on crew to managers;
grant SELECT, INSERT, UPDATE, DELETE on staff to managers;
grant SELECT, INSERT, UPDATE on people to managers;
grant SELECT, INSERT, UPDATE on bannedCustomers to managers;
grant SELECT, INSERT, UPDATE on customers to managers;
grant SELECT on orders to managers;
grant SELECT on items to managers;
grant SELECT on food to managers;
grant SELECT on drinks to managers;
grant SELECT on storeofferings to managers;

CEO: Has ability to make any select, insert, update, or view the the majority of tables. However cannot delete customers nor banned customers.
grant SELECT, INSERT, UPDATE, DELETE on managers to CEO;
grant SELECT, INSERT, UPDATE, DELETE on crew to CEO;
grant SELECT, INSERT, UPDATE, DELETE on staff to CEO;
grant SELECT, INSERT, UPDATE, DELETE on people to CEO;
grant SELECT, INSERT, UPDATE, DELETE on bannedCustomers to CEO;
grant SELECT, INSERT, UPDATE, DELETE on customers to CEO;
grant SELECT, INSERT, UPDATE, DELETE on orders to CEO;
grant SELECT, INSERT, UPDATE, DELETE on items to CEO;
grant SELECT, INSERT, UPDATE, DELETE on food to CEO;
grant SELECT, INSERT, UPDATE, DELETE on drinks to CEO;
grant SELECT, INSERT, UPDATE, DELETE on storeofferings to CEO;
grant SELECT, INSERT, UPDATE, DELETE on stores to CEO;
grant SELECT, INSERT, UPDATE, DELETE on zipcode to CEO;

Chaos breaks loose and the CEO tries to sabotage the company. Immediately the admin will immediately:
revoke all on all tables in schema public from CEO;
Known Problems/Future Enhancements

- The way in which items are stored could be improved. Rather than the preset way for items to be inserted and ordered, in the future there will be a more “cookbook” style design, in which items can be completely customized. There will be drink subtype tables for types of drink flavors, dairy options, sweetener options, drink size, etc.

- Currently an order only consists of one item. In the future, the order table will only consist of the order ID, store ID, person ID, and date. There will be an additional table added called OrderDetails, in which multiple items can be added to each order while maintaining 3NF.

- When the user roles and privileges are added to the main .sql file containing create statements, inserts, reports, triggers, views, and stored procedures, it cannot correctly be run in its entirety. Parts of the script get discarded. Therefore, there are two separate .sql files.