



# 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:	
<i>inactiveCustomers</i> .....	18
<i>unorderedFood</i> .....	19
<i>perksLocations</i> .....	20
Stored Procedures:	
<i>searchCustomerName</i> .....	21
<i>getCalories</i> .....	23
<i>storeLocation</i> .....	25
Triggers:	
<i>checkFood</i> .....	26
<i>checkDrink</i> .....	28
<i>priceCeiling</i> .....	30
Reports.....	32
User Roles/Security .....	33
Problems/Enhancements.....	34

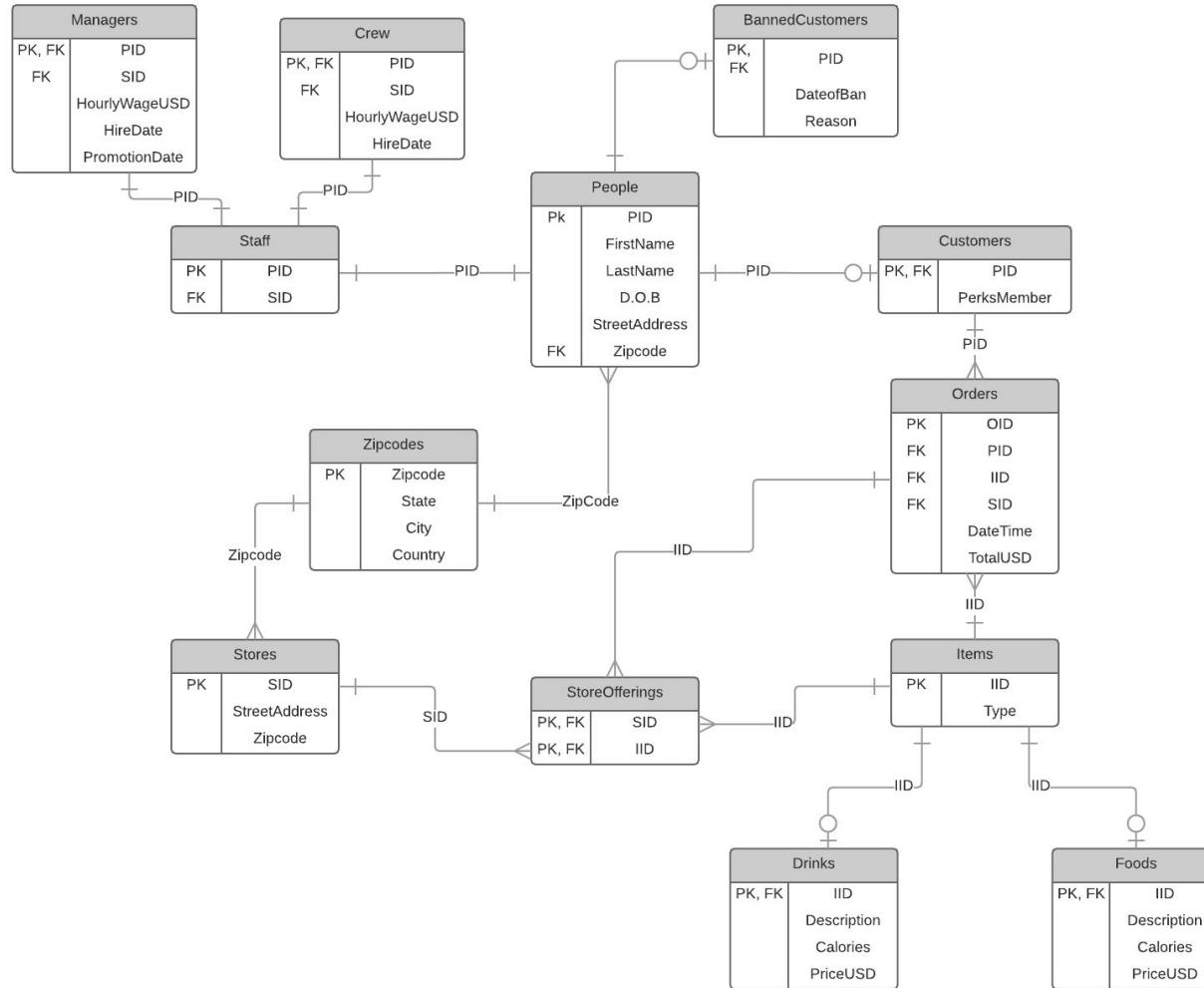
# Executive Summary

---

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.*



# 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)
);
```

	<b>zipcode</b> <b>integer</b>	<b>city</b> <b>text</b>	<b>state</b> <b>text</b>	<b>country</b> <b>text</b>
1	6475	Old Saybrook	CT	USA
2	7446	Ramsey	NJ	USA
3	12601	Poughkeepsie	NY	USA
4	12538	Hyde Park	NY	USA
5	1524	Leicester	MA	USA

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)
);
```

	pid character(4) text	firstname text	lastname text	dob date	streetaddress text	zipcode integer
<b>1</b>	p001	Tien	Liengtiraphan	1996-05-26	3399 North Road	12601
<b>2</b>	p002	Alan	Labouseur	2017-04-01	3399 North Road	12601
<b>3</b>	p003	Mary	Morrison	1996-01-21	11 Fox Hollow Road	6475
<b>4</b>	p004	Rachel	Wheaton	1996-04-30	52 Sun Valley Road	7446
<b>5</b>	p005	Bridget	Leahy	1995-09-08	5 Pryor Road	1524
<b>6</b>	p006	John	Doe	1990-05-06	3399 North Road	12601
<b>7</b>	p007	Jane	Deer	1989-09-22	3399 North Road	12601
<b>8</b>	p008	Data	Base	1970-01-01	3399 North Road	12601
<b>9</b>	p009	Ronald	McDonald	1960-11-04	3399 North Road	12601
<b>10</b>	p010	Star	Buck	1971-03-31	3434 North Road	12601
<b>11</b>	p011	Jeff	Kortina	1989-12-13	34 Main Street	7446
<b>12</b>	p012	Spoon	Wotherspoon	1993-04-01	83 Bee Way	7446
<b>13</b>	p013	Alex	Carlin	1990-06-05	4 Pleasant Street	7446
<b>14</b>	p014	Hannah	Youseff	1990-06-05	3399 North Road	12601
<b>15</b>	p015	Emma	Leahey	1996-08-08	3 Britney Drive	1524
<b>16</b>	p016	Michael	Leahy	1993-11-07	5 Pryor Road	1524
<b>17</b>	p017	Anna	Young	1998-02-26	3399 North Road	12601
<b>18</b>	p018	Erin	Concannon	1996-02-14	13 Irish Lane	7446
<b>19</b>	p019	Michael	Leahy	1993-11-07	5 Orange Road	7446
<b>20</b>	p020	Anna	Young	1998-02-26	98 Sun Road	6475

# 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

	<b>pid character(4)</b>	<b>dateofban date</b>	<b>reason text</b>
1	p002	2017-04-21	Fan of Starbucks
2	p010	1971-05-01	Brought Starbucks into Dunkin

# Customers 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')
);
```

	pid character(4)	perksmember text
1	p009	No
2	p004	Yes
3	p006	Yes
4	p018	No
5	p019	Yes
6	p020	Yes

Functional Dependency: PID → PerksMember

# Staff Table

---

The staff table contains the ID of each person and the ID of the store in which they work. A person cannot be added to the staff table unless they already are in the person table. A store ID cannot be assigned to a staff member unless that ID already exists in the store table.

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

Functional Dependency:

PID → SID

	<b>pid character(4)</b>	<b>sid character(4)</b>
<b>1</b>	p001	s003
<b>2</b>	p003	s004
<b>3</b>	p005	s001
<b>4</b>	p007	s005
<b>5</b>	p011	s005
<b>6</b>	p012	s002
<b>7</b>	p013	s001
<b>8</b>	p014	s003
<b>9</b>	p015	s001
<b>10</b>	p017	s004
<b>11</b>	p016	s005

# Managers Table

---

The managers table contains the store 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),
    SID char(4) not null references Stores(SID),
    HireDate date,
    PromotionDate date,
    HourlyWageUSD numeric(10,2) default 20.00,
    primary key(PID),
    check (PromotionDate > HireDate)
);
```

Functional Dependencies:

PID → SID, HireDate, PromotionDate,  
HourlyWageUSD

	pid character(4)	sid character(4)	hiredate date	promotiondate date	hourlywageusd numeric(10,2)
1	p001	s003	2016-02-03	2017-02-03	20.00
2	p007	s002	2015-10-15	2016-12-31	25.00
3	p015	s001	2013-04-19	2014-06-13	27.00
4	p016	s005	2012-09-01	2014-12-01	25.00
5	p017	s004	2015-03-15	2017-02-01	30.00

# Crew Table

---

The crew table contains the store 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 → SID, HireDate, HourlyWageUSD

	<b>pid character(4)</b>	<b>sid character(4)</b>	<b>hiredate date</b>	<b>hourlywageusd numeric(10,2)</b>
<b>1</b>	p005	s001	2010-06-18	10.00
<b>2</b>	p003	s004	2010-06-18	10.00
<b>3</b>	p011	s005	2010-06-18	10.00
<b>4</b>	p012	s002	2010-06-18	10.00
<b>5</b>	p013	s001	2010-06-18	10.00
<b>6</b>	p014	s003	2016-12-22	11.00

# 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.

```
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

	<b>sid character(4)</b>	<b>streetaddress text</b>	<b>zipcode integer</b>
<b>1</b>	s001	101 Main Street	1524
<b>2</b>	s002	3979 Albany Post Road	12538
<b>3</b>	s003	1 Dunkin Way	12601
<b>4</b>	s004	22 Sunset Road	6475
<b>5</b>	s005	92 Flower Lane	7446

# 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)
);
```

There are no functional dependencies for this table

A portion of the sample data:

	sid character(4)	iid character(4)
<b>1</b>	s001	i001
<b>2</b>	s001	i002
<b>3</b>	s001	i003
<b>4</b>	s001	i004
<b>5</b>	s001	i005
<b>6</b>	s001	i006
<b>7</b>	s001	i007
<b>8</b>	s001	i008
<b>9</b>	s001	i009
<b>10</b>	s001	i010
<b>11</b>	s001	i011
<b>12</b>	s001	i014
<b>13</b>	s002	i002
<b>14</b>	s002	i006
<b>15</b>	s002	i007
<b>16</b>	s002	i008
<b>17</b>	s002	i010
<b>18</b>	s003	i001
<b>19</b>	s003	i002
<b>20</b>	s003	i003
<b>21</b>	s003	i004
<b>22</b>	s003	i005
<b>23</b>	s003	i007
<b>24</b>	s003	i008
<b>25</b>	s003	i009
<b>26</b>	s003	i010
<b>27</b>	s003	i011
<b>28</b>	s003	i014
<b>29</b>	s004	i001
<b>30</b>	s004	i002
<b>31</b>	s004	i005

# Items Table

---

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:  
IID → Type

	<b>iid</b> character(4)	<b>type</b> text
<b>1</b>	i001	food
<b>2</b>	i002	drink
<b>3</b>	i003	food
<b>4</b>	i004	food
<b>5</b>	i005	food
<b>6</b>	i006	drink
<b>7</b>	i007	drink
<b>8</b>	i008	drink
<b>9</b>	i009	food
<b>10</b>	i010	drink
<b>11</b>	i011	food

# 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 → Description, Calories, PriceUSD

	<b>iid character(4)</b>	<b>description text</b>	<b>calories integer</b>	<b>priceusd numeric(10,2)</b>
<b>1</b>	i002	Medium Dunkaccino	350	3.00
<b>2</b>	i006	Medium Hot Chocolate	330	2.75
<b>3</b>	i007	Small Hot Coffee	5	1.75
<b>4</b>	i008	Espresso	5	1.99
<b>5</b>	i010	Espresso with Sugar	30	2.25
<b>6</b>	i002	Medium Dunkaccino	350	3.00
<b>7</b>	i006	Medium Hot Chocolate	330	2.75
<b>8</b>	i007	Small Hot Coffee	5	1.75
<b>9</b>	i008	Espresso	5	1.99
<b>10</b>	i010	Espresso with Sugar	30	2.25

# Food Table

---

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.

```
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

	iid character(4)	description text	calories integer	priceusd numeric(10,2)
<b>1</b>	i001	Plain Bagel	310	2.50
<b>2</b>	i003	Chocolate Coconut Donut	400	1.00
<b>3</b>	i004	Lemon Stick	430	1.50
<b>4</b>	i005	Blueberry Bagel	310	3.00
<b>5</b>	i009	Poppy Seed Bagel	350	3.00
<b>6</b>	i011	Blueberry Crumb Cake Donut	420	1.25
<b>7</b>	i001	Plain Bagel	310	2.50
<b>8</b>	i003	Chocolate Coconut Donut	400	1.00
<b>9</b>	i004	Lemon Stick	430	1.50
<b>10</b>	i005	Blueberry Bagel	310	3.00
<b>11</b>	i009	Poppy Seed Bagel	350	3.00
<b>12</b>	i011	Blueberry Crumb Cake Donut	420	1.25

# 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

	oid character(4)	pid character(4)	sid character(4)	iid character(4)	dateordered date
1	o001	p004	s002	i002	2017-04-26
2	o002	p018	s001	i006	2017-02-02
3	o003	p009	s005	i003	2016-05-22
4	o005	p020	s003	i010	2017-01-15
5	o006	p006	s005	i009	2014-12-26
6	o004	p004	s004	i002	2017-04-14
7	o007	p004	s005	i011	2017-01-31
8	o008	p020	s001	i007	2016-12-25
9	o009	p009	s002	i002	2017-03-17
10	o010	p004	s003	i001	2017-04-26

# *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;
```

	<b>pid character(4)</b>	<b>firstname text</b>	<b>lastname text</b>
1	p019	Michael	Leahy

# *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;
```

	iid character(4)	description text	calories integer	priceusd numeric(10,2)
1	i004	Lemon Stick	430	1.50
2	i005	Blueberry Bagel	310	3.00

# **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.

```
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;
```

	city text	state text
1	Ramsey	NJ
2	Poughkeepsie	NY
3	Old Saybrook	CT

# 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.

```
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;
```

	pid character(4)	firstname text	lastname text	dob date	streetaddress text	zipcode integer
1	p002	Alan	Labouseur	2017-04-01	3399 North Road	12601

	pid character(4)	firstname text	lastname text	dob date	streetaddress text	zipcode integer
1	p005	Bridget	Leahy	1995-09-08	5 Pryor Road	1524
2	p015	Emma	Leahy	1996-08-08	3 Britney Drive	1524
3	p016	Michael	Leahy	1993-11-07	5 Pryor Road	1524
4	p019	Michael	Leahy	1993-11-07	5 Orange Road	7446

	pid character(4)	firstname text	lastname text	dob date	streetaddress text	zipcode integer
1	p001	Tien	Liengtiraphan	1996-05-26	3399 North Road	12601
2	p002	Alan	Labouseur	2017-04-01	3399 North Road	12601
3	p006	John	Doe	1990-05-06	3399 North Road	12601
4	p012	Spoon	Wotherspoon	1993-04-01	83 Bee Way	7446
5	p018	Erin	Concannon	1996-02-14	13 Irish Lane	7446

# 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*

---

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

	pid character(4)	firstname text	lastname text	calories bigint
1	p004	Rachel	Wheaton	1430

```
SELECT getCalories('%%', 'M%', 'ref4');  
FETCH ALL FROM ref4;
```

	pid character(4)	firstname text	lastname text	calories bigint
1	p009	Ronald	McDonald	750

# Stored Procedure: Store Location Information

---

This procedure takes input of a city name and finds all store location information using LIKE.

```
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;
```

```
SELECT storeLocation('%e%', 'ref1');
FETCH ALL FROM ref1;
```

```
SELECT storeLocation('%o%', 'ref1');
FETCH ALL FROM ref1;
```

	zipcode integer	city text	state text	country text
1	6475	Old Saybrook	CT	USA
2	12601	Poughkeepsie	NY	USA

	zipcode integer	city text	state text	country text
1	7446	Ramsey	NJ	USA
2	12601	Poughkeepsie	NY	USA
3	12538	Hyde Park	NY	USA
4	1524	Leicester	MA	USA

# 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

```
insert into items(IID, Type) values ('i013', 'drink');
```

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

items:

	Data Output		Explain	Message
	iid character(4)	type text		
6	i006	drink		
7	i007	drink		
8	i008	drink		
9	i009	food		
10	i010	drink		
11	i011	food		
12	i013	drink		

food:

	iid character(4)	description text	calories integer	priceusd numeric(10,2)
1	i001	Plain Bagel	310	2.50
2	i003	Chocolate Coconut Donut	400	1.00
3	i004	Lemon Stick	430	1.50
4	i005	Blueberry Bagel	310	3.00
5	i009	Poppy Seed Bagel	350	3.00
6	i011	Blueberry Crumb Cake Donut	420	1.25

drinks:

	iid character(4)	description text	calories integer	priceusd numeric(10,2)
1	i002	Medium Dunkaccino	350	3.00
2	i006	Medium Hot Chocolate	330	2.75
3	i007	Small Hot Coffee	5	1.75
4	i008	Espresso	5	1.99
5	i010	Espresso with Sugar	30	2.25
6	i013	Testing	310	3.00

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:

```
insert into items(IID, Type) values ('i012', 'food');
```

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

Items:

	iid character(4)	type text
7	i007	drink
8	i008	drink
9	i009	food
10	i010	drink
11	i011	food
12	i012	food

drinks:

	iid character(4)	description text	calories integer	priceusd numeric(10,2)
1	1002	Medium Dunkaccino	350	3.00
2	1006	Medium Hot Chocolate	330	2.75
3	1007	Small Hot Coffee	5	1.75
4	1008	Espresso	5	1.99
5	1010	Espresso with Sugar	30	2.25

food:

	iid character(4)	description text	calories integer	priceusd numeric(10,2)
1	i001	Plain Bagel	310	2.50
2	i003	Chocolate Coconut Donut	400	1.00
3	i004	Lemon Stick	430	1.50
4	i005	Blueberry Bagel	310	3.00
5	i009	Poppy Seed Bagel	350	3.00
6	i011	Blueberry Crumb Cake Donut	420	1.25
7	i012	Testing food	352	2.29

Item 12 is classified as a food, so it is removed from the drinks table and inserted into the food table.

# Trigger: *priceCeiling*

---

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 *priceCeiling*

---

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

```
insert into items(IID, Type) values ('i014', 'food');
```

```
insert into food(IID, description, calories, priceUSD) values ('i014', 'Too  
Expensive', 310, 6);
```

```
select * from food;
```

	iid character(4)	description text	calories integer	priceusd numeric(10,2)
<b>1</b>	i001	Plain Bagel	310	2.50
<b>2</b>	i003	Chocolate Coconut Donut	400	1.00
<b>3</b>	i004	Lemon Stick	430	1.50
<b>4</b>	i005	Blueberry Bagel	310	3.00
<b>5</b>	i009	Poppy Seed Bagel	350	3.00
<b>6</b>	i011	Blueberry Crumb Cake Donut	420	1.25

```
Select * from items;
```

	iid character(4)	type text
<b>7</b>	i007	drink
<b>8</b>	i008	drink
<b>9</b>	i009	food
<b>10</b>	i010	drink
<b>11</b>	i011	food

# Reports on Sales in 2017

---

Total money collected in 2017:

```
select sum(totalUSD)
from orders
where dateordered >= '2017-01-01';
```

sum	numeric
	17.75

Total number of orders in 2017:

```
select count(oid) from orders where dateordered >= '2017-01-01';
```

count	bigint
	7

Average calories from every item:

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

round	numeric
	257

Average price from every item:

```
Select round((round(avg(d.priceUSD), 2) + round(avg(f.priceUSD), 2))/2, 2)
from drinks d, food f;
```

round	numeric
	2.20

# 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 on bannedCustomers to CEO;
grant SELECT, INSERT, UPDATE 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 database was originally created in separate .sql files for creating the tables, inserting the data, creating the views, stored procedures, triggers, and roles. When all of these scripts were combined as one .sql file and executed at once, it does not implement correctly. The code must be run in sections.
- The way in which items are presented could be improved. Rather than the preset way for items to be inserted and ordered, we aim to establish more of a “cookbook” systems in which items can be completely customized.
- Leading zeros are removed in zipcodes