Database Project

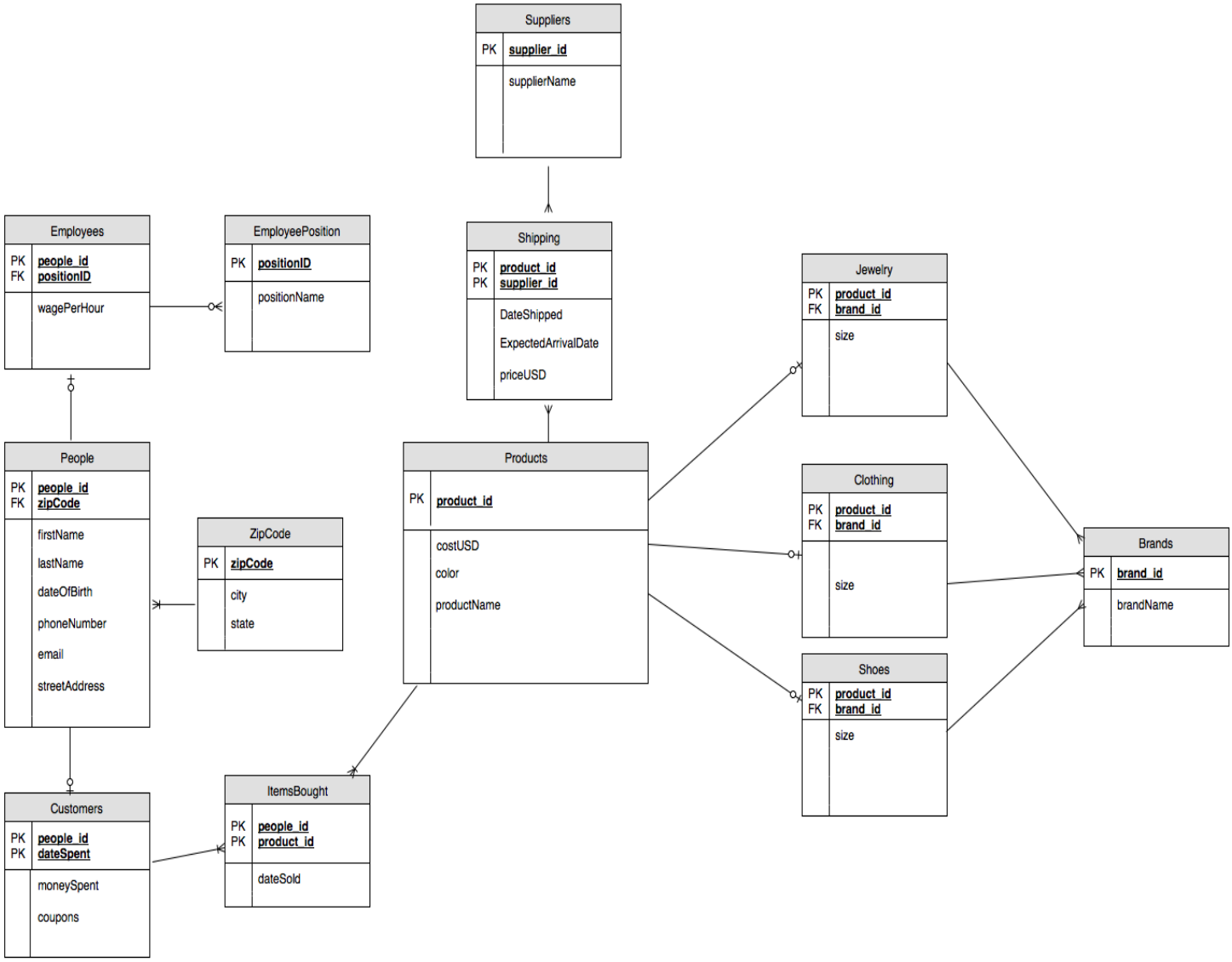
By
Carlie Maxwell

Table of Contents:

Executive Summary:

Being one of the most well-known and successful department stores in the country, Macy's needs a database that can keep track of its employees, its customers, the items sold there (and whether they sell or not), along with the general shipping information. The following pages contain information regarding the database design for the infamous Macy's department store. This design will have ER diagrams that will outline exactly how the store functions. Additionally, the document will show the tables created from the ER diagram, along with sample data inserted into them in order to test queries to make sure they are helpful and useful. It will go on to show possible views, reports, stored procedures, and triggers that can be created in order to hide complexity from the user, create data to help with decision making, and create overall efficient data. In big companies such as Macy's, security is a huge issue. Not all employees can have every privilege, and customers presumably have none. Understandably so, this model can't be perfect, so the document will outline known problems with this design and implementation. It will, however, go on to explain how these can be fixed and what future enhancements can be done.

ER Diagram:


Suppliers
PK supplier_id
supplierName
Employees
PK people_id
FK positionID
wagePerHour
EmployeePosition
PK positionID
positionName
Shipping
PK product_id
PK supplier_id
DateShipped
ExpectedArrivalDate
priceUSD
Jewelry
PK product_id
FK brand_id
size
People
PK people_id
FK zipCode
firstName
lastName
dateOfBirth
phoneNumber
email
streetAddress
ZipCode
PK zipCode
city
state
Products
PK product_id
costUSD
color
productName
Clothing
PK product_id
FK brand_id
size
Brands
PK brand_id
brandName
Shoes
PK product_id
FK brand_id
size
Customers
PK people_id
PK dateSpent
moneySpent
coupons
ItemsBought
PK people_id
PK product_id
dateSold

Address Table: Allows the person to include their full address with zip code, city, and state instead of just the street name.

```
CREATE TABLE IF NOT EXISTS Address (
zipCode char(5) NOT NULL,
city text NOT NULL,
state char(2) NOT NULL,
PRIMARY KEY (zipCode)
 );
```

Functional Dependencies:

zipCode → {city, state}

Sample Data:

	zipcode character(5)	city text	state character(2)
1	12342	Latham	NY
2	12110	Cohoes	NY
3	15434	Loudonville	NY
4	19426	Collegeville	PA
5	11255	Pittsburgh	PA

People Table: Includes all necessary information about a person that could either work or shop at Macy's

```
CREATE TABLE IF NOT EXISTS People (
people_id int NOT NULL UNIQUE,
fName text NOT NULL,
lName text NOT NULL,
DOB date NOT NULL,
phoneNumber char(12) NOT NULL,
emailAddress varchar(320),
streetAddress text NOT NULL,
zipCode char(5) NOT NULL,
PRIMARY KEY (people_id),
FOREIGN KEY (zipCode) REFERENCES Address(zipCode)
);
```

Functional Dependencies:

people_id → {fName, lName, DOB, phoneNumber, emailAddress, streetAddress, zipCode}

Sample Data:

	people_id integer	fname text	lname text	dob date	phonenumber character(12)	emailaddress character varying(320)	streetaddress text	zipcode character(5)
1	100	Carlie	Maxwell	1996-06-27	111-123-2343	database@aol.com	12 database lane	12342
2	101	Tim	Reese	1994-02-12	112-256-2980	desktop@aol.com	11 notebook lane	12110
3	102	Rob	Bush	1992-01-11	125-246-2250	robbie@aol.com	164 shoes drive	15434

Employee Position Table: Lists the positions that an employee can have. The employee in my database can only have one position.

```
CREATE TABLE IF NOT EXISTS EmployeePosition (
position_id int NOT NULL UNIQUE,
positionName text NOT NULL UNIQUE,
PRIMARY KEY (position_id)
 );
```

Functional Dependencies:

position_id → positionName

Sample Data:

	position_id integer	positionname text
1	200	General Manager
2	201	Assistant Manager
3	202	Cashier
4	203	Floor Leader
5	204	Regional Manager
6	205	Customer Service

Employee Table: The employee table will specify what that listed person does as an employee, including their position and their hourly wage.

```
CREATE TABLE IF NOT EXISTS Employees (
people_id int NOT NULL,
wagePerHour decimal NOT NULL,
position_id int NOT NULL,
PRIMARY KEY (people_id),
FOREIGN KEY (position_id) REFERENCES EmployeePosition (position_id)
);
```

Functional Dependencies:

people_id → {wagePerHour, position_id}

Sample Data:

	people_id integer	wageperhour numeric	position_id integer
1	100	14.50	200
2	102	30.00	201
3	103	25.67	202
4	106	11.50	208
5	108	10.00	207

Customers Table: Lists all people that are customers and specific information regarding what they did that day at the store

```
CREATE TABLE IF NOT EXISTS Customers (
people_id int NOT NULL,
coupons boolean NOT NULL,
moneySpent decimal NOT NULL,
dateSpent date NOT NULL,
PRIMARY KEY (people_id, dateSpent)
);
```

Functional Dependencies:

{people_id, dateSpent} → {coupons, moneySpent}

Sample Data:

	people_id integer	coupons boolean	moneyspent numeric	datespent date
1	101	t	1320.45	2016-02-21
2	104	t	88.22	2016-05-08
3	105	f	450.11	2015-03-09
4	107	t	269.09	2015-12-25
5	109	f	1245.67	2016-08-10

ItemsBought Table: Lists which people have bought what items and the date in which they were sold; helps to show what items are selling and when they are the most popular

```
CREATE TABLE IF NOT EXISTS ItemsBought (
people_id int NOT NULL,
product_id int NOT NULL,
dateSold date NOT NULL,
PRIMARY KEY (people_id, product_id)
);
```

Functional Dependencies:

{people_id, product_id} → dateSold

Sample Data:

	people_id integer	product_id integer	datesold date
1	101	306	2016-01-14
2	104	303	2016-08-10
3	105	300	2015-03-09
4	107	309	2016-12-10
5	109	301	2016-04-21

Brands Table: Shows which brands Macy's carries, as it is a department store and it doesn't just have one brand like Gap or American Eagle.

```
CREATE TABLE IF NOT EXISTS BrandsHeld (
brand_id int NOT NULL UNIQUE,
name text NOT NULL UNIQUE,
PRIMARY KEY (brand_id)
 );
```

Functional Dependencies:

brand_id → name

Sample Data:

	brand_id integer	name text
1	400	Fossil
2	401	Prada
3	402	Nautica
4	403	Ralph Lauren
5	404	Inc.

Products Table: Lists the products held in the store, as the store has many different sections to it such as shoes, clothes, watches, and perfume

```
CREATE TABLE IF NOT EXISTS Products (
product_id int NOT NULL UNIQUE,
productName text NOT NULL,
color text NOT NULL,
costUSD decimal NOT NULL,
PRIMARY KEY (product_id),
);
```

Functional Dependencies:

product_id → {productName, color, costUSD}

Sample Data:

	product_id integer	productname text	color text	costusd numeric
1	300	Watch	Black	145.60
2	301	Ring	Silver	400.60
3	302	Wind-breaker	Blue	95.10
4	303	Earrings	Rose Gold	65.00
5	304	Tanktop	Red/White	35.12

Suppliers Table: Shows the suppliers that ship products to Macy's stores

```
CREATE TABLE IF NOT EXISTS Suppliers (
supplier_id int NOT NULL,
supplierName text NOT NULL,
PRIMARY KEY (supplier_id)
);
```

Functional Dependencies:

supplier_id → supplierName

Sample Data:

	supplier_id integer	suppliername text
1	500	FedEx
2	501	UPS
3	502	DHL Express
4	503	USPS

Shipping Table: Shows the date shipped and expected arrival date along with the price that the suppliers sell it at.

```
CREATE TABLE IF NOT EXISTS Shipping (
product_id int NOT NULL,
supplier_id int NOT NULL,
dateShipped date NOT NULL,
priceUSD decimal NOT NULL,
expectedArrivalDate date NOT NULL,
PRIMARY KEY (product_id, dateShipped)
 );
```

Functional Dependencies:

supplier_id → supplierName

Sample Data:

	product_id integer	supplier_id integer	dateshipped date	priceusd numeric	expectedarrivaldate date
1	300	503	2016-05-25	140.00	2016-05-28
2	301	502	2016-05-30	395.00	2016-06-05
3	302	500	2016-02-06	78.00	2016-02-10
4	303	500	2016-04-10	50.00	2016-04-15

Jewelry Table: Takes the jewelry items from the products table and lists specific attributes about them.

```
CREATE TABLE IF NOT EXISTS Jewelry (
 product_id int NOT NULL,
brand_id int NOT NULL,
size text NOT NULL,
PRIMARY KEY (product_id),
FOREIGN KEY (brand_id) REFERENCES Brands
);
```

Functional Dependencies:

product_id → {brand_id, size}

Sample Data:

	product_id integer	brand_id integer	size text
1	300	400	50 mm
2	303	403	3.8 mm
3	301	409	7
4	306	406	30 in

Clothing Table: Takes the clothing items from the products table and lists specific attributes about them.

```
CREATE TABLE IF NOT EXISTS Clothing (
 product_id int NOT NULL,
brand_id int NOT NULL,
size text NOT NULL,
PRIMARY KEY (product_id),
FOREIGN KEY (brand_id) REFERENCES Brands
 );
```

Functional Dependencies:

product_id → {brand_id, size}

Sample Data:

	product_id integer	brand_id integer	size text
1	302	402	Medium
2	304	401	Small
3	305	407	Large
4	307	409	X-Small
5	308	400	Small

Shoes Table: Takes the shoe items from the products table and lists specific attributes about them.

```
CREATE TABLE IF NOT EXISTS Shoes (
product_id int NOT NULL,
brand_id int NOT NULL,
 size decimal NOT NULL,
 PRIMARY KEY (product_id),
 FOREIGN KEY (brand_id) REFERENCES Brands
);
```

Functional Dependencies:

product_id → {brand_id, size}

Sample Data:

	product_id integer	brand_id integer	size numeric
1	309	403	8
2	310	405	7
3	311	412	5
4	312	411	10
5	313	409	9

MostProfitableProducts View: Lists the top 10 most profitable products along with the amount Macy's makes by selling it

```
CREATE VIEW MostProfitableProducts AS
        SELECT Products.productName, Products.costUSD -
        Shipping.priceUSD AS difference
        FROM Products, Shipping
        WHERE Products.product_id = Shipping.product_id
        ORDER BY difference DESC
        LIMIT 10;
```

Sample Data:

	productname text	difference numeric
1	Wind-breaker	17.10
2	Earrings	15.00
3	Rain Boots	14.70
4	Sneakers	12.10
5	Ripped Jeans	10.12
6	Boat Shoes	6.99
7	Watch	5.60

CustomersInNewYork View: Outputs the number of customers that live in NY incase Macy's wants to open up another store there. It will show if it's worth it to Macy's to open another location there.

```
CREATE VIEW CustomersInNewYork AS
        SELECT Address.state, COUNT(Address.state)
        FROM Address, Customers, people
        WHERE People.zipCode = Address.zipCode
        AND Customers.people_id = People.people_id
        AND Address.state = 'NY'
        GROUP BY Address.state;
```

Sample Data:

	state character(2)	count bigint
1	NY	1

Reports:

Why is this useful? Find out which city and state the products are selling most at. This way you can know if you need to stock up on or discontinue certain products in that specific area.

```
SELECT address.city, address.state, products.productName,
Count(products.product_id)
FROM products, itemsBought, customers, people, address
WHERE products.product_id = itemsBought.product_id
AND itemsBought.people_id = customers.people_id
AND customers.people_id = people.people_id
AND people.zipCode = address.zipCode
GROUP BY address.city, address.state, products.product_id,
products.productName
ORDER BY products.product_id DESC;
```

	city text	state character(2)	productname text	count bigint
1	Miami	FL	Sneakers	1
2	Cohoes	NY	Belt	1
3	Pittsburgh	PA	Earrings	1
4	Beverly Hills	CA	Ring	1
5	Seattle	WA	Watch	1

Reports:

Why is this useful? Find out which supplier ships the most products. This will show to Macy's the suppliers they may need to cut back on in the future or which suppliers they should continue to employ.

```
SELECT suppliers.supplierName, Count(suppliers.supplierName)
FROM suppliers, shipping, products, itemsBought
WHERE suppliers.supplier_id = shipping.supplier_id
AND shipping.product_id = products.product_id
AND products.product_id = itemsBought.product_id
GROUP BY suppliers.supplierName
ORDER BY suppliers.supplierName DESC;
```

	suppliername text	count bigint
1	USPS	1
2	UPS	1
3	FedEx	2
4	DHL Express	1

Stored Procedures:

Why is this useful? Find out which products were sold on a certain date. This can help with future data analysis, such as how the sales were that day or if certain items need to continue to be on/be taken off of the shelves.

```
CREATE OR REPLACE FUNCTION datesOfProducts(date, refcursor)
RETURNS refcursor AS
$$
DECLARE
   dateOfProducts date          :=$1;
  resultset refcursor           :=$2;
BEGIN
   open resultset FOR
      SELECT itemsBought.dateSold, itemsBought.product_id,
products.productName,   products.color, products.costUSD
      FROM itemsBought, Products
      WHERE itemsBought.product_id = products.product_id
     AND dateOfProducts = itemsBought.dateSold;
RETURN resultset;
END
$$
Language plpgsql;

SELECT datesOfProducts('2016-01-14' , 'results');
FETCH ALL FROM results;
```

	datesold date	product_id integer	productname text	color text	costusd numeric
1	2016-01-14	304	Tanktop	Red/White	35.12
2	2016-01-14	306	Belt	Silver	500.60

Why is this useful? When a product is out of stock in the store, people like to simply have the employees put in an order for them. This stored procedure allows the employees to type in the person's ID number and check when they're product will be arriving so they can either call them or send an email out to alert them.

```
CREATE OR REPLACE FUNCTION peopleProductArrivalDate(INT, refcursor)
RETURNS refcursor AS
$$
DECLARE
  person INT                    :=$1;
  resultset refcursor           :=$2;
BEGIN
  open resultset FOR
     SELECT  people.people_id, people.fName, people.lName,
products.productName, shipping.expectedArrivalDate
     FROM people, customers, itemsBought, products, shipping
     WHERE people.people_id = customers.people_id
     AND customers.people_id = itemsBought.people_id
     AND itemsBought.product_id = products.product_id
     AND products.product_id = shipping.product_id
    AND person = people.people_id;
RETURN resultset;
END
$$
Language plpgsql;

SELECT peopleProductArrivalDate(101 , 'results'); FETCH ALL FROM results;
```

	people_id integer	fname text	lname text	productname text	expectedarrivaldate date
1	101	Tim	Reese	Belt	2016-11-18

Why is this useful? Products are constantly being added to the database, as Macy's is an ever-growing store. This stored procedure allows any product with a color of sterling silver to be automatically entered into the jewelry table as well, as the only thing that can be sterling silver is jewelry.

```
CREATE OR REPLACE FUNCTION addToJewelry() RETURNS TRIGGER AS
$$
BEGIN
   IF NEW.color = 'sterling silver' THEN
        INSERT INTO jewelry(product_id)
        VALUES (NEW.product_id);
  END IF;
RETURN NEW;
 END;
$$
LANGUAGE plpgsql;
```

Triggers:

Why is this useful? This trigger updates the jewelry table and places a new product that can only be categorized as jewelry based on its color into it.

```
CREATE TRIGGER addToJewelry
AFTER INSERT ON Products
FOR EACH ROW
EXECUTE PROCEDURE addToJewelry();

INSERT INTO products (Product_ID, costUSD, color, productname)
VALUES (314, 132.40, 'sterling silver', 'tiffany necklace');
```

Products Table (before/after):

	product_id integer	productname text	color text	costusd numeric
1	300	Watch	Black	145.60
2	301	Ring	Silver	400.60
3	302	Wind-breaker	Blue	95.10
4	303	Earrings	Rose Gold	65.00
5	304	Tanktop	Red/White	35.12
6	305	V-neck	Brown	115.10
7	306	Belt	Silver	500.60
8	307	Oversized Sweater	Cream	85.90
9	308	Ripped Jeans	Black	78.12
10	309	Sneakers	Gold/Black	92.10
11	310	Flats	Blue/White	52.10
12	311	Heels	Black	102.50
13	312	Boat Shoes	Beige	49.99
14	313	Rain Boots	Yellow	115.70

	product_id integer	productname text	color text	costusd numeric
1	314	tiffany necklace	sterling silver	132.40
2	300	Watch	Black	145.60
3	301	Ring	Silver	400.60
4	302	Wind-breaker	Blue	95.10
5	303	Earrings	Rose Gold	65.00
6	304	Tanktop	Red/White	35.12
7	305	V-neck	Brown	115.10
8	306	Belt	Silver	500.60

Jewelry Table (before/after):

	product_id integer	brand_id integer	size text
1	300	400	50 mm
2	303	403	3.8 mm
3	301	409	7
4	306	406	30 in

	product_id integer	brand_id integer	size text
1	300	400	50 mm
2	303	403	3.8 mm
3	301	409	7
4	306	406	30 in
5	314		

Security:

CREATE ROLE employeeManager;
GRANT SELECT ON employees, people, customers, address, products, itemsBought, shipping, jewelry, clothing, shoes, brands
TO employeeManager;
GRANT INSERT, UPDATE ON employees, people, customers, address, products, jewelry, shoes, clothing
To employeeManager;

CREATE ROLE employees;
GRANT SELECT ON products, shipping, brands, itemsBought, jewelry, clothing, shoes TO employees;

CREATE ROLE customers;
GRANT SELECT ON products, jewelry, clothing, shoes, brands
TO customers;

Implementation Notes:

Realistically, I would have to include many more tables besides just jewelry, clothing, and shoes, as Macy's has many more departments than just those 3. Additionally, I could have made many sub-entities off of types of clothes, jewelry, and shoes, but instead I just decided to generalize them for the sake of the project. There are millions of customers and employees around the country, so the snap shots of each table are highly misleading, as the numbers come out to be around 5 customers and 6 employees. Additionally, I could have added a table that lists all existing products in the world, so that different Macy's stores could sell different things. Instead, I just made it so the products table correlated with products that every Macy's sells.

Known Problems:

There is no way to fire an employee, delete products that are no longer sold, or discontinue a brand/supplier without deleting all of the other data; the model is able to just add onto, which isn't realistic. Also, the shipping table wastes a good amount of space, as the same dates are constantly repeated. For example, if a watch, a pair of sneakers, and a sweater were all shipped/arriving on the same day, then the dates would be repeated 3 times. I'm not sure how to fix that exactly at this moment in time, but it appears inefficient. Also, the products bought table isn't as efficient as it could be, as only one person can buy one product on a specific date. If a person buys more than 1 product, then the people id and the date sold get constantly repeated for each product bought.

Future Enhancements:

In the future, I want to incorporate more tables so that I can have more accurate and efficient data. By adding more tables, I'll be able to make the products more specific. Also, I could make a general products table and that way different Macy's stores could pull from that table instead of all selling the same thing. I also want to be able to be able to update all of the tables automatically according to the current status, such as new employees, new employee positions, new customers, new brands ect.