Introduction to Data Management

By Bhavna Kaparaju (bgk378), Brett Nesfeder (bmn644), Callie Gilmore (cgg756), and Dawson Cook (dcc2436)

Final Project - The Home Depot

# **The Home Depot Background & Data Strategy**

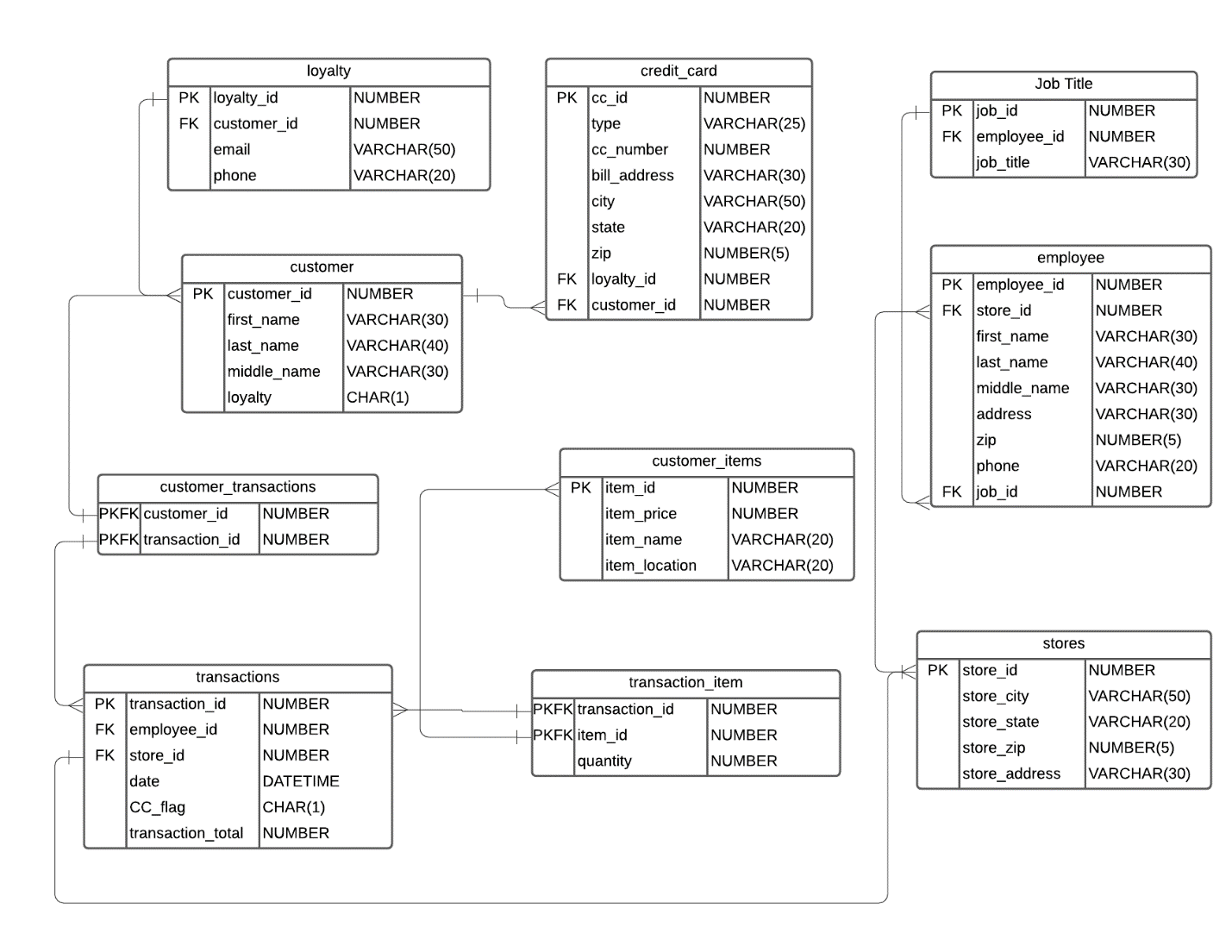
The Home Depot is the largest home improvement retailer in America. They supply tools, construction products, and home improvement services. It was founded in 1978 in Atlanta, Georgia and now has 2,278 stores across North America. With many stores and locations, The Home Depot deals with a huge amount of data on a daily basis.

We thought an offensive data strategy would be best for The Home Depot. An offensive data strategy would be optimal for The Home Depot since they are a retail store with big competitors such as Lowe’s, Walmart and Amazon. This also suits The Home Depot well because the data they would be using would be flexible and would allow them to create multiple versions of the truth. By implementing this offensive strategy, The Home Depot would be supporting revenue growth and generating customer insights in order to keep up with the competition. The strategy would also help with managerial decisions and overall make The Home Depot’s data management system much more effective.

# **Transactional Management Applications ERDs**

For our project, we chose The Home Depot as our organization. From this we decided on our 3 transactional management applications as Customer Point of Sale, Inventory and Wholesale purchasing. We chose these 3 as we believe that are important components in The Home Depot’s daily business operations.

## Customer Point of Sale ERD



## Inventory ERD

Diagram

Description automatically generated

## Wholesale Purchasing ERD

Diagram, schematic

Description automatically generated

# **Transactional Management Applications DDL**

## Customer DDL

--drop sequences

DROP SEQUENCE c\_customer\_id\_seq;

DROP SEQUENCE c\_loyalty\_id\_seq;

DROP SEQUENCE c\_cc\_id\_seq;

DROP SEQUENCE c\_transactions\_id\_seq;

DROP SEQUENCE c\_customer\_itemid\_seq;

DROP SEQUENCE c\_storeid\_seq;

DROP SEQUENCE c\_employee\_id\_seq;

DROP SEQUENCE c\_job\_id\_seq;

--drop views

DROP VIEW transaction\_item\_view;

DROP VIEW customer\_transactions\_view;

DROP VIEW credit\_card\_view;

DROP VIEW loyalty\_view;

DROP VIEW customer\_view;

DROP VIEW transactions\_view;

DROP VIEW customer\_items\_view;

DROP VIEW employee\_view;

DROP VIEW job\_title\_view;

DROP VIEW customer\_stores\_view;

--drop tables

DROP TABLE transaction\_item;

DROP TABLE customer\_transactions;

DROP TABLE credit\_card;

DROP TABLE loyalty;

DROP TABLE customer;

DROP TABLE transactions;

DROP TABLE customer\_items;

DROP TABLE employee;

DROP TABLE job\_title;

DROP TABLE customer\_stores;

--sequence for customer\_id

CREATE SEQUENCE c\_customer\_id\_seq

START WITH 1 INCREMENT BY 1;

--sequence for loyalty

CREATE SEQUENCE c\_loyalty\_id\_seq

START WITH 10 INCREMENT BY 1;

--sequence for cc\_id

CREATE SEQUENCE c\_cc\_id\_seq

START WITH 100 INCREMENT BY 1;

--sequence for transaction\_id

CREATE SEQUENCE c\_transactions\_id\_seq

START WITH 1000 INCREMENT BY 1;

--sequence for item\_id

CREATE SEQUENCE c\_customer\_itemid\_seq

START WITH 1000 INCREMENT BY 1;

--sequence for store\_id

CREATE SEQUENCE c\_storeid\_seq

START WITH 2000 INCREMENT BY 1;

--sequence for employee\_id

CREATE SEQUENCE c\_employee\_id\_seq

START WITH 1000 INCREMENT BY 1;

--sequence for job\_id

CREATE SEQUENCE c\_job\_id\_seq

START WITH 1000 INCREMENT BY 1;

--create tables

CREATE TABLE customer

(

customer\_id NUMBER DEFAULT c\_customer\_id\_seq.NEXTVAL PRIMARY KEY,

first\_name VARCHAR(30) NOT NULL,

middle\_name VARCHAR(30),

last\_name VARCHAR(40) NOT NULL,

loyalty CHAR(1) NOT NULL

);

CREATE TABLE loyalty

(

loyalty\_id NUMBER DEFAULT c\_loyalty\_id\_seq.NEXTVAL PRIMARY KEY,

customer\_id NUMBER REFERENCES customer(customer\_id),

email VARCHAR(50) UNIQUE,

phone VARCHAR(20) NOT NULL

);

CREATE TABLE credit\_card

(

cc\_id NUMBER DEFAULT c\_cc\_id\_seq.NEXTVAL PRIMARY KEY,

loyalty\_id NUMBER REFERENCES loyalty (loyalty\_id),

customer\_id NUMBER REFERENCES customer (customer\_id),

card\_type VARCHAR(25) NOT NULL,

card\_num VARCHAR(16) NOT NULL,

billing\_address VARCHAR(30) NOT NULL,

city VARCHAR(50) NOT NULL,

state VARCHAR(20) NOT NULL,

zip NUMBER(5) NOT NULL

);

CREATE TABLE transactions

(

transactions\_id NUMBER DEFAULT c\_transactions\_id\_seq.NEXTVAL PRIMARY KEY,

transaction\_date DATE NOT NULL,

credit\_flag CHAR(1) NOT NULL,

transaction\_total NUMBER NOT NULL

);

CREATE TABLE customer\_transactions

(

customer\_id NUMBER REFERENCES customer (customer\_id),

transactions\_id NUMBER REFERENCES transactions (transactions\_id),

CONSTRAINT c\_customer\_transaction\_cpk PRIMARY KEY (customer\_id, transactions\_id)

);

CREATE TABLE customer\_items

(

item\_id NUMBER DEFAULT c\_customer\_itemid\_seq.NEXTVAL PRIMARY KEY,

item\_price NUMBER NOT NULL,

item\_name VARCHAR(20) NOT NULL,

item\_description VARCHAR(100) NOT NULL,

item\_location VARCHAR(20) NOT NULL

);

CREATE TABLE transaction\_item

(

transactions\_id NUMBER REFERENCES transactions (transactions\_id),

item\_id NUMBER REFERENCES customer\_items (item\_id),

CONSTRAINT c\_transaction\_item\_cpk PRIMARY KEY (transactions\_id,item\_id),

quantity NUMBER NOT NULL

);

CREATE TABLE customer\_stores

(

store\_id NUMBER DEFAULT c\_storeid\_seq.NEXTVAL NOT NULL PRIMARY KEY,

store\_city VARCHAR(50) NOT NULL,

store\_state VARCHAR(20) NOT NULL,

store\_zip NUMBER(5) NOT NULL,

store\_address VARCHAR(30) NOT NULL

);

CREATE TABLE job\_title

(

job\_id NUMBER DEFAULT c\_job\_id\_seq.NEXTVAL PRIMARY KEY,

job\_title VARCHAR(30)

);

CREATE TABLE employee

(

employee\_id NUMBER DEFAULT c\_employee\_id\_seq.NEXTVAL PRIMARY KEY,

store\_id NUMBER REFERENCES customer\_stores (store\_id),

first\_name VARCHAR(30) NOT NULL,

last\_name VARCHAR(40) NOT NULL,

middle\_name VARCHAR(30),

address VARCHAR(30) NOT NULL,

zip NUMBER(5) NOT NULL,

phone VARCHAR(20) NOT NULL,

job\_id NUMBER REFERENCES job\_title (job\_id)

);

CREATE OR REPLACE VIEW transaction\_item\_view

AS SELECT \*

FROM transaction\_item;

CREATE OR REPLACE VIEW customer\_transactions\_view

AS SELECT \*

FROM customer\_transactions;

CREATE OR REPLACE VIEW credit\_card\_view

AS SELECT \*

FROM credit\_card;

CREATE OR REPLACE VIEW loyalty\_view

AS SELECT \*

FROM loyalty;

CREATE OR REPLACE VIEW customer\_view

AS SELECT \*

FROM customer;

CREATE OR REPLACE VIEW transactions\_view

AS SELECT \*

FROM transactions;

CREATE OR REPLACE VIEW customer\_items\_view

AS SELECT \*

FROM customer\_items;

CREATE OR REPLACE VIEW employee\_view

AS SELECT \*

FROM employee;

CREATE OR REPLACE VIEW job\_title\_view

AS SELECT \*

FROM job\_title;

CREATE OR REPLACE VIEW customer\_stores\_view

AS SELECT \*

FROM customer\_stores;

--insert statements

INSERT INTO customer

(first\_name, middle\_name, last\_name, loyalty)

VALUES('Jeffrey', 'John', 'Jacobs', 'Y');

INSERT INTO customer\_items

(item\_price, item\_name, item\_description, item\_location)

VALUES (20, 'Wood plank', 'plank of wood', 'Aisle 14');

INSERT INTO loyalty

(customer\_id, email, phone)

VALUES(1, 'jjj@gmail.com', '5126780965');

INSERT INTO credit\_card

(loyalty\_id, customer\_id, card\_type, card\_num, billing\_address, city, state, zip)

VALUES(10, 1, 'Visa', '0000111122223333', '100 House Ave', 'Los Angeles', 'CA', 78735);

INSERT INTO transactions

(transaction\_date, credit\_flag, transaction\_total)

VALUES('25-JUN-20', 'Y', 6000);

INSERT INTO customer\_transactions

(customer\_id, transactions\_id)

VALUES(1, 1000);

INSERT INTO job\_title

(job\_title)

VALUES('Manager');

INSERT INTO customer\_stores

(store\_address, store\_city, store\_state, store\_zip)

VALUES ('1200 Home Depot Blvd', 'Los Angeles', 'California', 90210);

INSERT INTO employee

(store\_id, first\_name, last\_name, middle\_name, address, zip, phone, job\_id)

VALUES(2000, 'Pete', 'Plunker', 'Patrick', '23 Champion Dr', 90210, '5126157824', 1000);

INSERT INTO transaction\_item

(transactions\_id, item\_id, quantity)

VALUES(1000, 1000, 2);

COMMIT;

## Inventory DDL

--drop sequences

DROP SEQUENCE i\_inventory\_itemid\_sequence;

DROP SEQUENCE i\_depid\_sequence;

DROP SEQUENCE i\_storeid\_sequence;

--drop views

DROP VIEW store\_items\_view;

DROP VIEW department\_items\_view;

DROP VIEW department\_view;

DROP VIEW inventory\_stores\_view;

DROP VIEW inventory\_items\_view;

--drop tables

DROP TABLE store\_items;

DROP TABLE department\_items;

DROP TABLE department;

DROP TABLE inventory\_stores;

DROP TABLE inventory\_items;

--sequence for items\_id

CREATE SEQUENCE i\_inventory\_itemid\_sequence

START WITH 1000

INCREMENT BY 1;

--sequence for department\_ids

CREATE SEQUENCE i\_depid\_sequence

START WITH 1000

INCREMENT BY 1;

--sequence for store\_ids

CREATE SEQUENCE i\_storeid\_sequence

START WITH 1000

INCREMENT BY 1;

--create tables

CREATE TABLE inventory\_items

(

item\_id NUMBER DEFAULT i\_inventory\_itemid\_sequence.NEXTVAL NOT NULL PRIMARY KEY,

item\_name VARCHAR(30) NOT NULL,

item\_description VARCHAR(100) NOT NULL,

item\_quantity NUMBER NOT NULL,

item\_restock CHAR(1) NOT NULL

);

CREATE TABLE inventory\_stores

(

store\_id NUMBER DEFAULT i\_storeid\_sequence.NEXTVAL NOT NULL PRIMARY KEY,

store\_address VARCHAR(30) NOT NULL,

store\_city VARCHAR(40) NOT NULL,

store\_state VARCHAR(20) NOT NULL,

store\_zip NUMBER(5) NOT NULL

);

CREATE TABLE department

(

department\_id NUMBER DEFAULT i\_depid\_sequence.NEXTVAL NOT NULL PRIMARY KEY,

department\_name VARCHAR(30) NOT NULL

);

CREATE TABLE store\_items

(

store\_id NUMBER NOT NULL,

item\_id NUMBER NOT NULL,

CONSTRAINT i\_store\_item\_pck PRIMARY KEY (store\_id, item\_id),

CONSTRAINT i\_store\_id\_fk FOREIGN KEY (store\_id) REFERENCES inventory\_stores(store\_id),

CONSTRAINT i\_item\_store\_id\_fk FOREIGN KEY (item\_id) REFERENCES inventory\_items(item\_id)

);

CREATE TABLE department\_items

(

department\_id NUMBER NOT NULL,

item\_id NUMBER NOT NULL,

CONSTRAINT i\_department\_item\_pck PRIMARY KEY (department\_id, item\_id),

CONSTRAINT i\_department\_id\_fk FOREIGN KEY (department\_id) REFERENCES department(department\_id),

CONSTRAINT i\_item\_dep\_id\_fk FOREIGN KEY (item\_id) REFERENCES inventory\_items(item\_id)

);

CREATE OR REPLACE VIEW store\_items\_view

AS SELECT \*

FROM store\_items;

CREATE OR REPLACE VIEW department\_items\_view

AS SELECT \*

FROM department\_items;

CREATE OR REPLACE VIEW department\_view

AS SELECT \*

FROM department;

CREATE OR REPLACE VIEW inventory\_stores\_view

AS SELECT \*

FROM inventory\_stores;

CREATE OR REPLACE VIEW inventory\_items\_view

AS SELECT \*

FROM inventory\_items;

--Inserts

INSERT INTO inventory\_items

(item\_name, item\_description, item\_quantity, item\_restock)

VALUES ('Nest Hello Video Doorbell', 'A wired video doorbell', '50', 'N');

INSERT INTO inventory\_stores

(store\_address, store\_city, store\_state, store\_zip)

VALUES ('1200 Home Depot Blvd', 'Los Angeles', 'California', 90210);

INSERT INTO department

(department\_name)

VALUES ('Doorbells');

INSERT INTO store\_items

(store\_id, item\_id)

VALUES ('1000', '1000');

INSERT INTO department\_items

(department\_id, item\_id)

VALUES ('1000', '1000');

COMMIT;

## Wholesale Purchasing DDL

--drop sequences

DROP SEQUENCE w\_wholesaleitemid\_sequence;

DROP SEQUENCE w\_orderid\_sequence;

DROP SEQUENCE w\_storeid\_sequence;

DROP SEQUENCE w\_vendorid\_sequence;

DROP SEQUENCE w\_managerid\_sequence;

--drop views

DROP VIEW item\_order\_view;

DROP VIEW orders\_view;

DROP VIEW vendors\_view;

DROP VIEW purchase\_manager\_view;

DROP VIEW wholesale\_stores\_view;

DROP VIEW wholesale\_items\_view;

--drop tables

DROP TABLE item\_order;

DROP TABLE orders;

DROP TABLE vendors;

DROP TABLE purchase\_manager;

DROP TABLE wholesale\_stores;

DROP TABLE wholesale\_items;

--sequence for items\_id

CREATE SEQUENCE w\_wholesaleitemid\_sequence

START WITH 1000

INCREMENT BY 1;

--sequence for order\_ids

CREATE SEQUENCE w\_orderid\_sequence

START WITH 1000

INCREMENT BY 1;

--sequence for store\_ids

CREATE SEQUENCE w\_storeid\_sequence

START WITH 3000

INCREMENT BY 1;

--sequence for vendor\_ids

CREATE SEQUENCE w\_vendorid\_sequence

START WITH 1000

INCREMENT BY 1;

--sequence for manager\_ids

CREATE SEQUENCE w\_managerid\_sequence

START WITH 1000

INCREMENT BY 1;

--create tables

CREATE TABLE wholesale\_items

(

item\_id NUMBER DEFAULT w\_wholesaleitemid\_sequence.NEXTVAL NOT NULL PRIMARY KEY,

item\_name VARCHAR(30) NOT NULL,

item\_description VARCHAR(100) NOT NULL,

item\_price NUMBER NOT NULL

);

CREATE TABLE wholesale\_stores

(

store\_id NUMBER DEFAULT w\_storeid\_sequence.NEXTVAL NOT NULL PRIMARY KEY,

store\_city VARCHAR(40) NOT NULL,

store\_state VARCHAR(20) NOT NULL,

store\_zip NUMBER(5) NOT NULL,

store\_address VARCHAR(30) NOT NULL

);

CREATE TABLE purchase\_manager

(

manager\_id NUMBER DEFAULT w\_managerid\_sequence.NEXTVAL NOT NULL PRIMARY KEY,

manager\_firstname VARCHAR(30) NOT NULL,

manager\_middlename VARCHAR(30) NOT NULL,

manager\_lastname VARCHAR(30) NOT NULL,

store\_id NUMBER NOT NULL,

CONSTRAINT w\_store\_id\_manager\_fk FOREIGN KEY (store\_id) REFERENCES wholesale\_stores(store\_id)

);

CREATE TABLE vendors

(

vendor\_id NUMBER DEFAULT w\_vendorid\_sequence.NEXTVAL NOT NULL PRIMARY KEY,

vendor\_name VARCHAR(30) NOT NULL,

vendor\_address VARCHAR(30) NOT NULL,

vendor\_city VARCHAR(40) NOT NULL,

vendor\_state VARCHAR(20) NOT NULL,

vendor\_zip NUMBER(5) NOT NULL,

vendor\_contact\_name VARCHAR(50) NOT NULL,

vendor\_contact\_phone VARCHAR(12) NOT NULL,

vendor\_contact\_email VARCHAR(50) NOT NULL UNIQUE

);

CREATE TABLE orders

(

order\_id NUMBER DEFAULT w\_orderid\_sequence.NEXTVAL NOT NULL PRIMARY KEY,

order\_date\_time DATE NOT NULL,

order\_total NUMBER NOT NULL,

manager\_id NUMBER NOT NULL,

store\_id NUMBER NOT NULL,

vendor\_id NUMBER NOT NULL,

CONSTRAINT w\_store\_id\_order\_fk FOREIGN KEY (store\_id) REFERENCES wholesale\_stores(store\_id),

CONSTRAINT w\_manager\_id\_order\_fk FOREIGN KEY (manager\_id) REFERENCES purchase\_manager(manager\_id),

CONSTRAINT w\_vendor\_id\_order\_fk FOREIGN KEY (vendor\_id) REFERENCES vendors(vendor\_id)

);

CREATE TABLE item\_order

(

order\_id NUMBER NOT NULL,

item\_id NUMBER NOT NULL,

quantity NUMBER NOT NULL,

CONSTRAINT w\_order\_id\_itemorder\_fk FOREIGN KEY (order\_id) REFERENCES orders(order\_id),

CONSTRAINT w\_item\_id\_itemorder\_fk FOREIGN KEY (item\_id) REFERENCES wholesale\_items(item\_id)

);

CREATE OR REPLACE VIEW item\_order\_view

AS SELECT \*

FROM item\_order;

CREATE OR REPLACE VIEW orders\_view

AS SELECT \*

FROM orders;

CREATE OR REPLACE VIEW vendors\_view

AS SELECT \*

FROM vendors;

CREATE OR REPLACE VIEW purchase\_manager\_view

AS SELECT \*

FROM purchase\_manager;

CREATE OR REPLACE VIEW wholesale\_stores\_view

AS SELECT \*

FROM wholesale\_stores;

CREATE OR REPLACE VIEW wholesale\_items\_view

AS SELECT \*

FROM wholesale\_items;

--Inserts

INSERT INTO wholesale\_items

(item\_name, item\_description, item\_price)

VALUES ('Lamp', 'Rainbow Colorful Lamp', 100);

INSERT INTO wholesale\_stores

(store\_address, store\_city, store\_state, store\_zip)

VALUES ('1300 Home Depot Ave', 'Los Angeles', 'California', 88746);

INSERT INTO purchase\_manager

(manager\_firstname, manager\_middlename, manager\_lastname, store\_id)

VALUES ('Jeff', 'Keith', 'Adams', 3000);

INSERT INTO vendors

(vendor\_name, vendor\_address, vendor\_city, vendor\_state, vendor\_zip, vendor\_contact\_name, vendor\_contact\_phone, vendor\_contact\_email)

VALUES ('Lamps Inc', '101 Bunny Run', 'Los Angeles', 'California', '90210', 'Bob Roberts', '7816549087', 'bob.roberts@gmail.com');

INSERT INTO orders

(order\_date\_time, order\_total, manager\_id, store\_id, vendor\_id)

VALUES ('25-SEP-14', 4000, 1000, 3000, 1000);

INSERT INTO item\_order

(order\_id, item\_id, quantity)

VALUES (1000, 1000, 200);

COMMIT;

# **Data Warehouse**

## Data Warehouse ERD

Diagram

Description automatically generated

## Data Warehouse DDL

--drop sequences

DROP SEQUENCE dw\_inventory\_itemid\_sequence;

DROP SEQUENCE dw\_depid\_sequence;

DROP SEQUENCE dw\_storeid\_sequence;

DROP SEQUENCE dw\_customer\_id\_seq;

DROP SEQUENCE dw\_loyalty\_id\_seq;

DROP SEQUENCE dw\_cc\_id\_seq;

DROP SEQUENCE dw\_transactions\_id\_seq;

DROP SEQUENCE dw\_customer\_itemid\_sequence;

DROP SEQUENCE dw\_employee\_id\_seq;

DROP SEQUENCE dw\_job\_id\_seq;

DROP SEQUENCE dw\_wholesaleitemid\_sequence;

DROP SEQUENCE dw\_orderid\_sequence;

--DROP SEQUENCE dw\_storeid\_sequence;

DROP SEQUENCE dw\_vendorid\_sequence;

DROP SEQUENCE dw\_managerid\_sequence;

--drop tables

DROP TABLE transaction\_item\_dw;

DROP TABLE customer\_transactions\_dw;

DROP TABLE item\_order\_dw;

DROP TABLE department\_items\_dw;

DROP TABLE orders\_dw;

DROP TABLE store\_items\_dw;

DROP TABLE department\_dw;

DROP TABLE inventory\_items\_dw;

DROP TABLE credit\_card\_dw;

DROP TABLE loyalty\_dw;

DROP TABLE customer\_dw;

DROP TABLE wholesale\_items\_dw;

DROP TABLE employee\_dw;

DROP TABLE job\_title\_dw;

DROP TABLE transactions\_dw;

DROP TABLE customer\_items\_dw;

DROP TABLE vendors\_dw;

DROP TABLE purchase\_manager\_dw;

DROP TABLE stores\_dw;

DROP PROCEDURE final\_etl\_proc;

--create sequences

CREATE SEQUENCE dw\_inventory\_itemid\_sequence

START WITH 1000

INCREMENT BY 1;

--sequence for department\_ids

CREATE SEQUENCE dw\_depid\_sequence

START WITH 1000

INCREMENT BY 1;

--sequence for store\_ids

CREATE SEQUENCE dw\_storeid\_sequence

START WITH 1002

INCREMENT BY 1;

CREATE SEQUENCE dw\_customer\_id\_seq

START WITH 1 INCREMENT BY 1;

--sequence for loyalty

CREATE SEQUENCE dw\_loyalty\_id\_seq

START WITH 10 INCREMENT BY 1;

--sequence for cc\_id

CREATE SEQUENCE dw\_cc\_id\_seq

START WITH 100 INCREMENT BY 1;

--sequence for transaction\_id

CREATE SEQUENCE dw\_transactions\_id\_seq

START WITH 1000 INCREMENT BY 1;

--sequence for item\_id

CREATE SEQUENCE dw\_customer\_itemid\_sequence

START WITH 1000 INCREMENT BY 1;

--sequence for employee\_id

CREATE SEQUENCE dw\_employee\_id\_seq

START WITH 1000 INCREMENT BY 1;

--sequence for job\_id

CREATE SEQUENCE dw\_job\_id\_seq

START WITH 1000 INCREMENT BY 1;

--sequence for items\_id

CREATE SEQUENCE dw\_wholesaleitemid\_sequence

START WITH 1000

INCREMENT BY 1;

--sequence for order\_ids

CREATE SEQUENCE dw\_orderid\_sequence

START WITH 1000

INCREMENT BY 1;

--sequence for vendor\_ids

CREATE SEQUENCE dw\_vendorid\_sequence

START WITH 1000

INCREMENT BY 1;

--sequence for manager\_ids

CREATE SEQUENCE dw\_managerid\_sequence

START WITH 1000

INCREMENT BY 1;

--create tables

CREATE TABLE inventory\_items\_dw

(

item\_id NUMBER DEFAULT dw\_inventory\_itemid\_sequence.NEXTVAL NOT NULL PRIMARY KEY,

item\_name VARCHAR(30) NOT NULL,

item\_description VARCHAR(100) NOT NULL,

item\_quantity NUMBER NOT NULL,

item\_restock CHAR(1) NOT NULL

);

CREATE TABLE stores\_dw

(

store\_id NUMBER DEFAULT dw\_storeid\_sequence.NEXTVAL NOT NULL PRIMARY KEY,

store\_city VARCHAR(40) NOT NULL,

store\_state VARCHAR(20) NOT NULL,

store\_zip NUMBER(5) NOT NULL,

store\_address VARCHAR(30) NOT NULL

);

CREATE TABLE department\_dw

(

department\_id NUMBER DEFAULT dw\_depid\_sequence.NEXTVAL NOT NULL PRIMARY KEY,

department\_name VARCHAR(30) NOT NULL

);

CREATE TABLE store\_items\_dw

(

store\_id NUMBER NOT NULL,

item\_id NUMBER NOT NULL,

CONSTRAINT dw\_store\_item\_pck PRIMARY KEY (store\_id, item\_id),

CONSTRAINT dw\_store\_id\_fk FOREIGN KEY (store\_id) REFERENCES stores\_dw(store\_id),

CONSTRAINT dw\_item\_store\_id\_fk FOREIGN KEY (item\_id) REFERENCES inventory\_items\_dw(item\_id)

);

CREATE TABLE department\_items\_dw

(

department\_id NUMBER NOT NULL,

item\_id NUMBER NOT NULL,

CONSTRAINT dw\_department\_item\_pck PRIMARY KEY (department\_id, item\_id),

CONSTRAINT dw\_department\_id\_fk FOREIGN KEY (department\_id) REFERENCES department\_dw(department\_id),

CONSTRAINT dw\_item\_dep\_id\_fk FOREIGN KEY (item\_id) REFERENCES inventory\_items\_dw(item\_id)

);

CREATE TABLE customer\_dw

(

customer\_id NUMBER DEFAULT dw\_customer\_id\_seq.NEXTVAL PRIMARY KEY,

first\_name VARCHAR(30) NOT NULL,

middle\_name VARCHAR(30),

last\_name VARCHAR(40) NOT NULL,

loyalty CHAR(1) NOT NULL

);

CREATE TABLE loyalty\_dw

(

loyalty\_id NUMBER DEFAULT dw\_loyalty\_id\_seq.NEXTVAL PRIMARY KEY,

customer\_id NUMBER REFERENCES customer\_dw(customer\_id),

email VARCHAR(50) UNIQUE,

phone VARCHAR(20) NOT NULL

);

CREATE TABLE credit\_card\_dw

(

cc\_id NUMBER DEFAULT dw\_cc\_id\_seq.NEXTVAL PRIMARY KEY,

loyalty\_id NUMBER REFERENCES loyalty\_dw (loyalty\_id),

customer\_id NUMBER REFERENCES customer\_dw (customer\_id),

card\_type VARCHAR(25) NOT NULL,

card\_num VARCHAR(16) NOT NULL,

billing\_Address VARCHAR(30) NOT NULL,

city VARCHAR(50) NOT NULL,

state VARCHAR(20) NOT NULL,

zip NUMBER(5) NOT NULL

);

CREATE TABLE transactions\_dw

(

transactions\_id NUMBER DEFAULT dw\_transactions\_id\_seq.NEXTVAL PRIMARY KEY,

transaction\_date TIMESTAMP NOT NULL,

credit\_flag CHAR(1) NOT NULL,

transaction\_total NUMBER NOT NULL

);

CREATE TABLE customer\_transactions\_dw

(

customer\_id NUMBER REFERENCES customer\_dw (customer\_id),

transactions\_id NUMBER REFERENCES transactions\_dw (transactions\_id),

CONSTRAINT dw\_customer\_transaction\_cpk PRIMARY KEY (customer\_id, transactions\_id)

);

CREATE TABLE customer\_items\_dw

(

item\_id NUMBER DEFAULT dw\_customer\_itemid\_sequence.NEXTVAL PRIMARY KEY,

item\_price NUMBER NOT NULL,

item\_name VARCHAR(20) NOT NULL,

item\_description VARCHAR(100) NOT NULL,

item\_location VARCHAR(20) NOT NULL

);

CREATE TABLE transaction\_item\_dw

(

transactions\_id NUMBER REFERENCES transactions\_dw (transactions\_id),

item\_id NUMBER REFERENCES customer\_items\_dw (item\_id),

CONSTRAINT dw\_transaction\_item\_cpk PRIMARY KEY (transactions\_id,item\_id),

quantity NUMBER NOT NULL

);

CREATE TABLE job\_title\_dw

(

job\_id NUMBER DEFAULT dw\_job\_id\_seq.NEXTVAL PRIMARY KEY,

job\_title VARCHAR(30)

);

CREATE TABLE employee\_dw

(

employee\_id NUMBER DEFAULT dw\_employee\_id\_seq.NEXTVAL PRIMARY KEY,

store\_id NUMBER REFERENCES stores\_dw (store\_id),

first\_name VARCHAR(30) NOT NULL,

last\_name VARCHAR(40) NOT NULL,

middle\_name VARCHAR(30),

address VARCHAR(30) NOT NULL,

zip NUMBER(5) NOT NULL,

phone VARCHAR(20) NOT NULL,

job\_id NUMBER REFERENCES job\_title\_dw (job\_id)

);

CREATE TABLE wholesale\_items\_dw

(

item\_id NUMBER DEFAULT dw\_wholesaleitemid\_sequence.NEXTVAL NOT NULL PRIMARY KEY,

item\_name VARCHAR(30) NOT NULL,

item\_description VARCHAR(100) NOT NULL,

item\_price NUMBER NOT NULL

);

CREATE TABLE purchase\_manager\_dw

(

manager\_id NUMBER DEFAULT dw\_managerid\_sequence.NEXTVAL NOT NULL PRIMARY KEY,

manager\_firstname VARCHAR(30) NOT NULL,

manager\_middlename VARCHAR(30) NOT NULL,

manager\_lastname VARCHAR(30) NOT NULL,

store\_id NUMBER NOT NULL,

CONSTRAINT dw\_store\_id\_purchase\_manager\_fk FOREIGN KEY (store\_id) REFERENCES stores\_dw(store\_id)

);

CREATE TABLE vendors\_dw

(

vendor\_id NUMBER DEFAULT dw\_vendorid\_sequence.NEXTVAL NOT NULL PRIMARY KEY,

vendor\_name VARCHAR(30) NOT NULL,

vendor\_address VARCHAR(30) NOT NULL,

vendor\_city VARCHAR(40) NOT NULL,

vendor\_state VARCHAR(20) NOT NULL,

vendor\_zip NUMBER(5) NOT NULL,

vendor\_contact\_name VARCHAR(50) NOT NULL,

vendor\_contact\_phone VARCHAR(12) NOT NULL,

vendor\_contact\_email VARCHAR(50) NOT NULL UNIQUE

);

CREATE TABLE orders\_dw

(

order\_id NUMBER DEFAULT dw\_orderid\_sequence.NEXTVAL NOT NULL PRIMARY KEY,

order\_date\_time DATE NOT NULL,

order\_total NUMBER NOT NULL,

manager\_id NUMBER NOT NULL,

store\_id NUMBER NOT NULL,

vendor\_id NUMBER NOT NULL,

CONSTRAINT dw\_store\_id\_orderdw\_fk FOREIGN KEY (store\_id) REFERENCES stores\_dw(store\_id),

CONSTRAINT dw\_manager\_id\_orderdw\_fk FOREIGN KEY (manager\_id) REFERENCES purchase\_manager\_dw(manager\_id),

CONSTRAINT dw\_vendor\_id\_orderdw\_fk FOREIGN KEY (vendor\_id) REFERENCES vendors\_dw(vendor\_id)

);

CREATE TABLE item\_order\_dw

(

order\_id NUMBER NOT NULL,

item\_id NUMBER NOT NULL,

quantity NUMBER NOT NULL,

CONSTRAINT dw\_order\_id\_itemorderdw\_fk FOREIGN KEY (order\_id) REFERENCES orders\_dw(order\_id),

CONSTRAINT dw\_item\_id\_itemorderdw\_fk FOREIGN KEY (item\_id) REFERENCES wholesale\_items\_dw(item\_id)

);

# **ETL Process**

DROP PROCEDURE final\_etl\_proc;

CREATE PROCEDURE final\_etl\_proc

AS

BEGIN

MERGE INTO inventory\_items\_dw i\_i\_dw

USING inventory\_items\_view i\_i

ON (i\_i\_dw.item\_id = i\_i.item\_id)

WHEN MATCHED THEN

UPDATE SET i\_i\_dw.item\_name = i\_i.item\_name, i\_i\_dw.item\_description = i\_i.item\_description, i\_i\_dw.item\_quantity = i\_i.item\_quantity, i\_i\_dw.item\_restock = i\_i.item\_restock;

INSERT INTO inventory\_items\_dw

SELECT \*

FROM inventory\_items\_view

WHERE NOT EXISTS (

SELECT \*

FROM inventory\_items\_dw

WHERE inventory\_items\_dw.item\_id = inventory\_items\_view.item\_id);

MERGE INTO stores\_dw s\_dw

USING customer\_stores\_view csv

ON (s\_dw.store\_id = csv.store\_id)

WHEN MATCHED THEN

UPDATE SET s\_dw.store\_city = csv.store\_city, s\_dw.store\_state = csv.store\_state, s\_dw.store\_zip = csv.store\_zip, s\_dw.store\_address = csv.store\_address;

MERGE INTO stores\_dw s\_dw

USING inventory\_stores\_view isv

ON (s\_dw.store\_id = isv.store\_id)

WHEN MATCHED THEN

UPDATE SET s\_dw.store\_city = isv.store\_city, s\_dw.store\_state = isv.store\_state, s\_dw.store\_zip = isv.store\_zip, s\_dw.store\_address = isv.store\_address;

MERGE INTO stores\_dw s\_dw

USING wholesale\_stores\_view wsv

ON (s\_dw.store\_id = wsv.store\_id)

WHEN MATCHED THEN

UPDATE SET s\_dw.store\_city = wsv.store\_city, s\_dw.store\_state = wsv.store\_state, s\_dw.store\_zip = wsv.store\_zip, s\_dw.store\_address = wsv.store\_address;

INSERT INTO stores\_dw

SELECT \*

FROM customer\_stores\_view

WHERE NOT EXISTS (

SELECT \*

FROM stores\_dw

WHERE stores\_dw.store\_id = customer\_stores\_view.store\_id);

INSERT INTO stores\_dw

SELECT \*

FROM inventory\_stores\_view

WHERE NOT EXISTS (

SELECT \*

FROM stores\_dw

WHERE stores\_dw.store\_id = inventory\_stores\_view.store\_id);

INSERT INTO stores\_dw

SELECT \*

FROM wholesale\_stores\_view

WHERE NOT EXISTS (

SELECT \*

FROM stores\_dw

WHERE stores\_dw.store\_id = wholesale\_stores\_view.store\_id);

MERGE INTO department\_dw d\_dw

USING department\_view d

ON (d\_dw.department\_id = d.department\_id)

WHEN MATCHED THEN

UPDATE SET d\_dw.department\_name = d.department\_name;

INSERT INTO department\_dw

SELECT \*

FROM department\_view

WHERE NOT EXISTS (

SELECT \*

FROM department\_dw

WHERE department\_dw.department\_id = department\_view.department\_id);

MERGE INTO store\_items\_dw si\_dw

USING store\_items\_view si

ON (si\_dw.store\_id = si.store\_id) --AND si\_dw.item\_id = si.item\_id)

WHEN MATCHED THEN

UPDATE SET si\_dw.item\_id = si.item\_id;

INSERT INTO store\_items\_dw

SELECT \*

FROM store\_items\_view

WHERE NOT EXISTS (

SELECT \*

FROM store\_items\_dw

WHERE store\_items\_dw.store\_id = store\_items\_view.store\_id);

MERGE INTO department\_items\_dw di\_dw

USING department\_items\_view di

ON (di\_dw.department\_id = di.department\_id) --AND si\_dw.item\_id = si.item\_id)

WHEN MATCHED THEN

UPDATE SET di\_dw.item\_id = di.item\_id;

INSERT INTO department\_items\_dw

SELECT \*

FROM department\_items\_view

WHERE NOT EXISTS (

SELECT \*

FROM department\_items\_dw

WHERE department\_items\_dw.department\_id = department\_items\_view.department\_id);

MERGE INTO customer\_dw c\_dw

USING customer\_view c

ON (c\_dw.customer\_id = c.customer\_id)

WHEN MATCHED THEN

UPDATE SET c\_dw.first\_name = c.first\_name, c\_dw.middle\_name = c.middle\_name, c\_dw.last\_name = c.last\_name, c\_dw.loyalty = c.loyalty;

INSERT INTO customer\_dw

SELECT \*

FROM customer\_view

WHERE NOT EXISTS (

SELECT \*

FROM customer\_dw

WHERE customer\_dw.customer\_id = customer\_view.customer\_id);

MERGE INTO loyalty\_dw l\_dw

USING loyalty\_view l

ON (l\_dw.loyalty\_id = l.loyalty\_id)

WHEN MATCHED THEN

UPDATE SET l\_dw.customer\_id = l.customer\_id, l\_dw.email = l.email, l\_dw.phone = l.phone;

INSERT INTO loyalty\_dw

SELECT \*

FROM loyalty\_view

WHERE NOT EXISTS (

SELECT \*

FROM loyalty\_dw

WHERE loyalty\_dw.loyalty\_id = loyalty\_view.loyalty\_id);

MERGE INTO credit\_card\_dw cc\_dw

USING credit\_card\_view cc

ON (cc\_dw.cc\_id = cc.cc\_id)

WHEN MATCHED THEN

UPDATE SET cc\_dw.loyalty\_id = cc.loyalty\_id, cc\_dw.customer\_id = cc.customer\_id, cc\_dw.card\_type = cc.card\_type, cc\_dw.card\_num = cc.card\_num, cc\_dw.billing\_Address = cc.billing\_Address, cc\_dw.city = cc.city, cc\_dw.state = cc.state, cc\_dw.zip = cc.zip;

INSERT INTO credit\_card\_dw

SELECT \*

FROM credit\_card\_view

WHERE NOT EXISTS (

SELECT \*

FROM credit\_card\_dw

WHERE credit\_card\_dw.cc\_id = credit\_card\_view.cc\_id);

MERGE INTO transactions\_dw t\_dw

USING transactions\_view t

ON (t\_dw.transactions\_id = t.transactions\_id)

WHEN MATCHED THEN

UPDATE SET t\_dw.transaction\_date = t.transaction\_date, t\_dw.credit\_flag = t.credit\_flag, t\_dw.transaction\_total = t.transaction\_total;

INSERT INTO transactions\_dw

SELECT \*

FROM transactions\_view

WHERE NOT EXISTS (

SELECT \*

FROM transactions\_dw

WHERE transactions\_dw.transactions\_id = transactions\_view.transactions\_id);

MERGE INTO customer\_transactions\_dw ct\_dw

USING customer\_transactions\_view ct

ON (ct\_dw.customer\_id = ct.customer\_id)

WHEN MATCHED THEN

UPDATE SET ct\_dw.transactions\_id = ct.transactions\_id;

INSERT INTO customer\_transactions\_dw

SELECT \*

FROM customer\_transactions\_view

WHERE NOT EXISTS (

SELECT \*

FROM customer\_transactions\_dw

WHERE customer\_transactions\_dw.customer\_id = customer\_transactions\_view.customer\_id);

MERGE INTO customer\_items\_dw ci\_dw

USING customer\_items\_view ci

ON (ci\_dw.item\_id = ci.item\_id)

WHEN MATCHED THEN

UPDATE SET ci\_dw.item\_price = ci.item\_price, ci\_dw.item\_name = ci.item\_name, ci\_dw.item\_description = ci.item\_description, ci\_dw.item\_location = ci.item\_location;

INSERT INTO customer\_items\_dw

SELECT \*

FROM customer\_items\_view

WHERE NOT EXISTS (

SELECT \*

FROM customer\_items\_dw

WHERE customer\_items\_dw.item\_id = customer\_items\_view.item\_id);

MERGE INTO transaction\_item\_dw ti\_dw

USING transaction\_item\_view tiv

ON (ti\_dw.transactions\_id = tiv.transactions\_id)

WHEN MATCHED THEN

UPDATE SET ti\_dw.item\_id = tiv.item\_id, ti\_dw.quantity = tiv.quantity;

INSERT INTO transaction\_item\_dw

SELECT \*

FROM transaction\_item\_view

WHERE NOT EXISTS (

SELECT \*

FROM transaction\_item\_dw

WHERE transaction\_item\_dw.transactions\_id = transaction\_item\_view.transactions\_id);

MERGE INTO job\_title\_dw jt\_dw

USING job\_title\_view jt

ON (jt\_dw.job\_id = jt.job\_id)

WHEN MATCHED THEN

UPDATE SET jt\_dw.job\_title = jt.job\_title;

INSERT INTO job\_title\_dw

SELECT \*

FROM job\_title\_view

WHERE NOT EXISTS (

SELECT \*

FROM job\_title\_dw

WHERE job\_title\_dw.job\_id = job\_title\_view.job\_id);

MERGE INTO employee\_dw e\_dw

USING employee\_view e

ON (e\_dw.employee\_id = e.employee\_id)

WHEN MATCHED THEN

UPDATE SET e\_dw.store\_id = e.store\_id, e\_dw.first\_name = e.first\_name, e\_dw.last\_name = e.last\_name, e\_dw.middle\_name = e.middle\_name, e\_dw.address = e.address, e\_dw.zip = e.zip, e\_dw.phone = e.phone, e\_dw.job\_id = e.job\_id;

INSERT INTO employee\_dw

SELECT \*

FROM employee\_view

WHERE NOT EXISTS (

SELECT \*

FROM employee\_dw

WHERE employee\_dw.employee\_id = employee\_view.employee\_id);

MERGE INTO wholesale\_items\_dw wi\_dw

USING wholesale\_items\_view wi

ON (wi\_dw.item\_id = wi.item\_id)

WHEN MATCHED THEN

UPDATE SET wi\_dw.item\_name = wi.item\_name, wi\_dw.item\_description = wi.item\_description, wi\_dw.item\_price = wi.item\_price;

INSERT INTO wholesale\_items\_dw

SELECT \*

FROM wholesale\_items\_view

WHERE NOT EXISTS (

SELECT \*

FROM wholesale\_items\_dw

WHERE wholesale\_items\_dw.item\_id = wholesale\_items\_view.item\_id);

MERGE INTO purchase\_manager\_dw pm\_dw

USING purchase\_manager\_view pm

ON (pm\_dw.manager\_id = pm.manager\_id)

WHEN MATCHED THEN

UPDATE SET pm\_dw.manager\_firstname = pm.manager\_firstname, pm\_dw.manager\_middlename = pm.manager\_middlename, pm\_dw.manager\_lastname = pm.manager\_lastname, pm\_dw.store\_id = pm.store\_id;

INSERT INTO purchase\_manager\_dw

SELECT \*

FROM purchase\_manager\_view

WHERE NOT EXISTS (

SELECT \*

FROM purchase\_manager\_dw

WHERE purchase\_manager\_dw.manager\_id = purchase\_manager\_view.manager\_id);

MERGE INTO vendors\_dw v\_dw

USING vendors\_view v

ON (v\_dw.vendor\_id = v.vendor\_id)

WHEN MATCHED THEN

UPDATE SET v\_dw.vendor\_name = v.vendor\_name, v\_dw.vendor\_address = v.vendor\_address, v\_dw.vendor\_city = v.vendor\_city, v\_dw.vendor\_state = v.vendor\_state, v\_dw.vendor\_zip = v.vendor\_zip, v\_dw.vendor\_contact\_name = v.vendor\_contact\_name, v\_dw.vendor\_contact\_phone = v.vendor\_contact\_phone, v\_dw.vendor\_contact\_email = v.vendor\_contact\_email;

INSERT INTO vendors\_dw

SELECT \*

FROM vendors\_view

WHERE NOT EXISTS (

SELECT \*

FROM vendors\_dw

WHERE vendors\_dw.vendor\_id = vendors\_view.vendor\_id);

MERGE INTO orders\_dw o\_dw

USING orders\_view o

ON (o\_dw.order\_id = o.order\_id)

WHEN MATCHED THEN

UPDATE SET o\_dw.order\_date\_time = o.order\_date\_time, o\_dw.order\_total = o.order\_total, o\_dw.manager\_id = o.manager\_id, o\_dw.store\_id = o.store\_id, o\_dw.vendor\_id = o.vendor\_id;

INSERT INTO orders\_dw

SELECT \*

FROM orders\_view

WHERE NOT EXISTS (

SELECT \*

FROM orders\_dw

WHERE orders\_dw.order\_id = orders\_view.order\_id);

MERGE INTO item\_order\_dw io\_dw

USING item\_order\_view io

ON (io\_dw.order\_id = io.order\_id)

WHEN MATCHED THEN

UPDATE SET io\_dw.item\_id = io.item\_id, io\_dw.quantity = io.quantity;

INSERT INTO item\_order\_dw

SELECT \*

FROM item\_order\_view

WHERE NOT EXISTS (

SELECT \*

FROM item\_order\_dw

WHERE item\_order\_dw.order\_id = item\_order\_view.order\_id);

END;

/

EXEC final\_etl\_proc;

SELECT \* FROM inventory\_items\_dw;

SELECT \* FROM stores\_dw;

SELECT \* FROM department\_dw;

SELECT \* FROM store\_items\_dw;

SELECT \* FROM department\_items\_dw;

SELECT \* FROM customer\_dw;

SELECT \* FROM loyalty\_dw;

SELECT \* FROM credit\_card\_dw;

SELECT \* FROM transactions\_dw;

SELECT \* FROM customer\_transactions\_dw;

SELECT \* FROM customer\_items\_dw;

SELECT \* FROM transaction\_item\_dw;

SELECT \* FROM job\_title\_dw;

SELECT \* FROM employee\_dw;

SELECT \* FROM wholesale\_items\_dw;

SELECT \* FROM purchase\_manager\_dw;

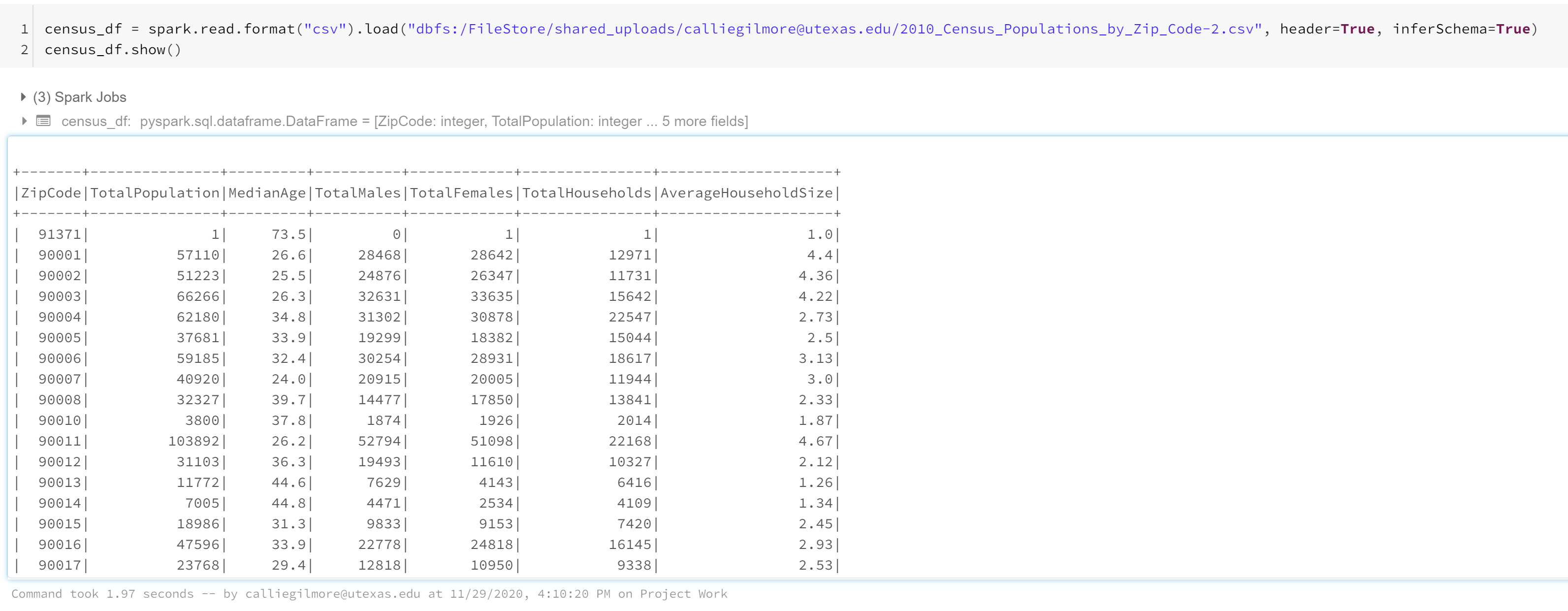
SELECT \* FROM vendors\_dw;

SELECT \* FROM orders\_dw;

SELECT \* FROM item\_order\_dw;

# **Data Lake**

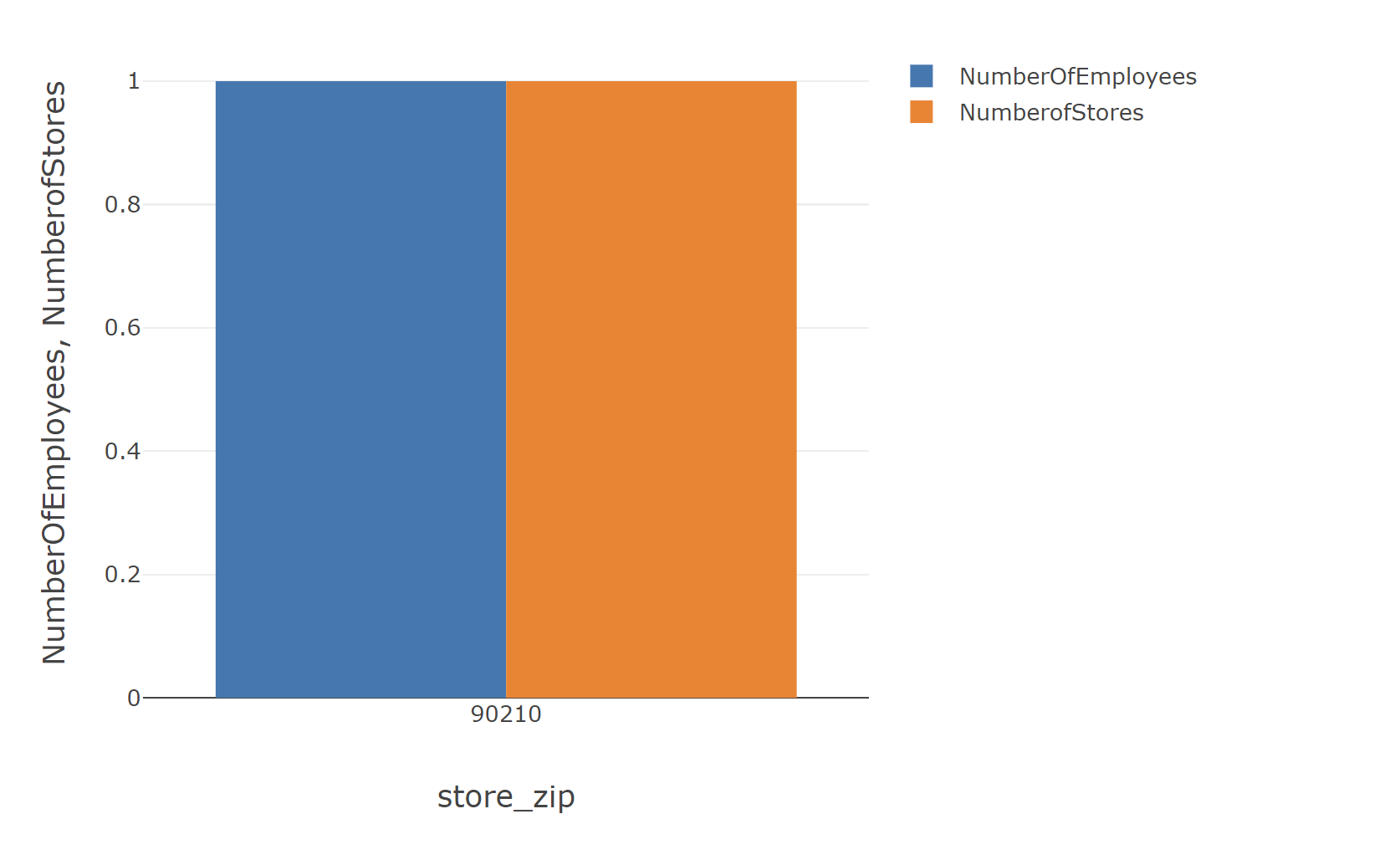
For our data lake, we decided to utilize the store data, employee data and vendor data that we created for our transactional process applications and utilized in our data warehouse through the ETL Process. We decided on this data because we thought it would be interesting to understand store, employee and vendor data by zip code and then compare that to zip code demographic data. Additionally, we found Census data that contained the Zip Code, Total Population, Median Age, Total Males, Total Females, Total Households and Average Household Size. We thought this demographic data would be interesting for home depot in relation to store data, vendor data and employee data. Below, you can see the census data that we utilized and the data lake we created where joined the census data with our data. With more data from Home Depot, this data lake would be very interesting to run analytics on store, employee and vendor data as it relates to store demographics. Additionally, we ran some aggregations to understand the data better. For example we ran a count function on employees, vendors and stores so that we could compare the number of stores, vendors and employees with the demographic data.



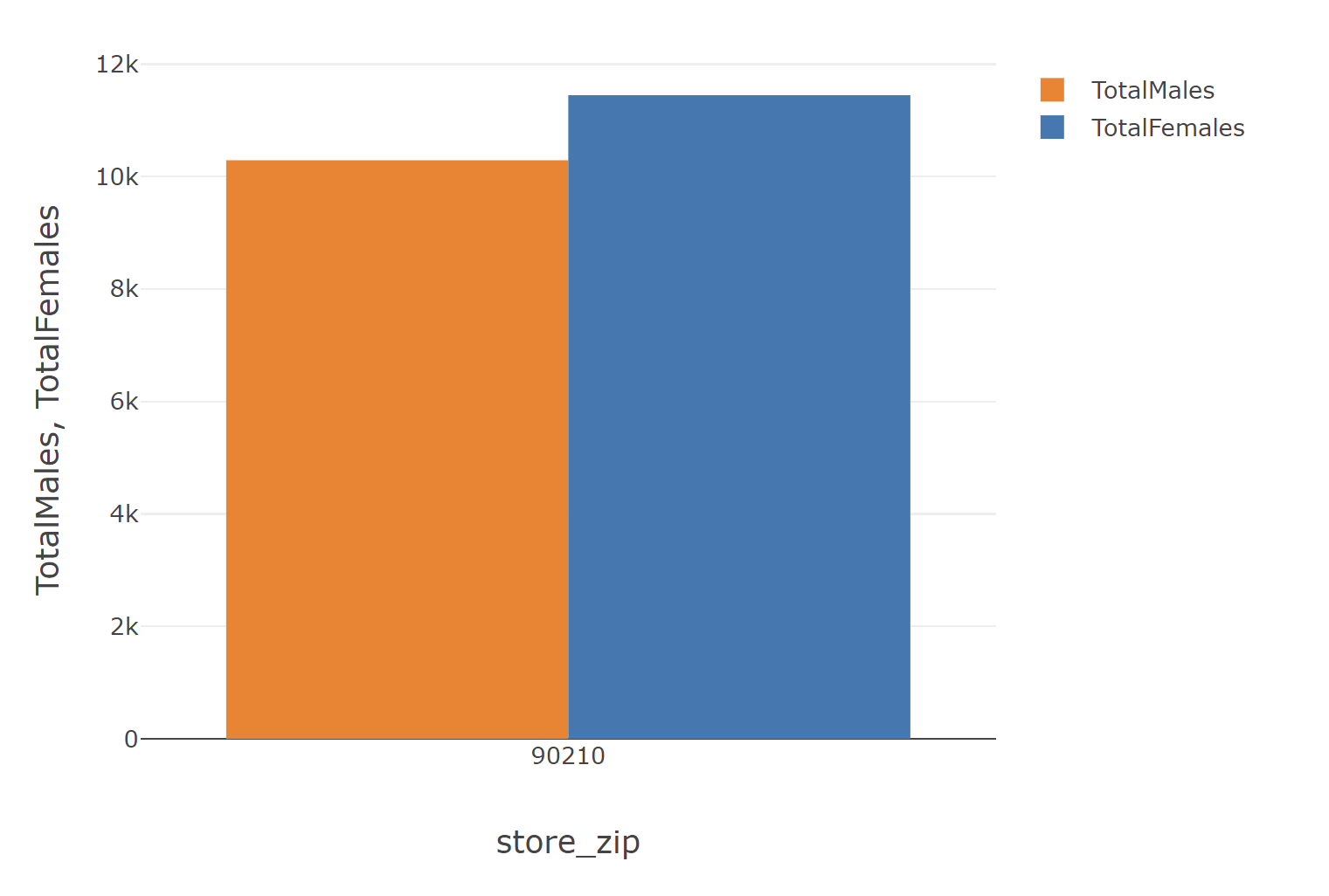
****

# **Analysis Patterns**

For our first analytical pattern, we decided to look at the number of stores and the number of employees by zip code. With only 1 data point, this is not very interesting. However, with multiple stores and employees in each zip code, this could help the Home Depot understand which zip codes are underserved either by number of stores or number of employees working at the stores. The proportion of employees to stores in each zip code is also important and could provide interesting information as well. Additionally, you could display this graph as a proportion of Total Zip Code Population by dividing these values by the Total Population. We chose not to display it this way because with only 1 data point, the proportion was very low and therefore the graph did not display anything.



For our second analytical pattern, we utilized our census data to understand the proportion to males and females in each zip code. This could benefit Home Depot because they could utilize their customer data to have a target demographic. Based on this target demographic, Home Depot could then utilize the proportion of males to females in each zip code as well as the median age to open more stores in zip codes that have a higher proportion of the target demographic.



# **Questions**

## What did you learn?

Overall, this project was a very valuable learning experience. One of our biggest takeaways was simply how complex a company’s database environment can be. We were only looking at 3 applications and the data warehouse and ETL processes were still especially complications. This made us realize how complex a company’s environment is when they have hundreds of applications all producing data that needs to be stored. This also made us realize how valuable corporate structure and organization is when it comes to data. The company must have very clear organization and structure to store this data in ways that it can be accessed. Additionally, data without the ability to access it, is utterly useless. So, the corporate structure is also essential to gain any sort of insights from the data. While this project seemed complicated with only 3 applications, it is astounding to think of what these massive corporation’s data infrastructure looks like.

## What was the most valuable?

Understanding just how complex these environments can be is truly valuable, but our biggest takeaway is not only the importance of organization but also the best practices for keeping these databases organized, such as naming conventions. We ran into one issue where two of our constraint’s names were duplicate names, but the error message was not clear that this was happening. This was very hard to find and de-bug to figure out that that was the issue. Now this could have been avoided with more organized naming conventions from the beginning. So, creating very clear naming conventions and organized structures and making sure all members of the organization/team maintain these is a very valuable lesson to ensure the data is accessible and there are no issues with the database.

## How can you use this learning going forward?

This lesson on consistency and organization is one that can be applied to many different areas and will apply to every singly job that we have leaving this program. So, this is a lesson that we all take with us. Ensuring that there is consistency between naming conventions, processes, systems etc. But ensuring consistency is only half the battle. The other half is communicating this effectively to other members of the team and ensuring they follow those practices as well. So, one way we want to practice this going forward would be with detailed documentation as go so that nothing is every left out.

## What are additional opportunities for learning that this project did not capture?

I think some additional opportunities would be more applications of conduction analytical patters on the data. We got a little bit pf practice doing this in the project; however, it was difficult with such few data. It would be interesting to be able to see how we can utilize data lakes and other items as well to perform data analytics. Our current view is that you simply query the database, export the data as a CSV and import it into your tool of choice. While this is a fine method, there are other ways to analyze the data from this format and it would be cool to learn that.

## How can we change this project to capture these opportunities?

One way that this could be done was that if the data were provided. I understand that this causes difficulty as we are creating the ERDs and DDLs. Maybe the project can we be in 2 parts: the first part as we did it where we create the ERDs, DDLs, Data Warehouse, ETL and Data Lake and the second part with given data and a given data lake and we could run analytical patterns and processes on the given data lake. Or simply have a class day where we could spectate someone performing these types of activities. Overall, this was still a very valuable learning experience.