

PES UNIVERSITY

100 feet Ring Road, BSK 3rd Stage, Bengaluru 560085

Department of Computer Science and Engineering
Jan – May 2020

UE18CS252
Database Management Systems

Project Report

SUPERMARKET DATABASE MANAGEMENT SYSTEM

Submitted by -

Manah Shetty
PES1201801733
4B

Project Summary

A supermarket needs a database management system to keep record of details of the customers, employees, items and their inventory, financial accounts. The data model has the relations: customer(identified by the customer id), employee(identified by the employee id), dependents(for extra information on the employees), store(identified by the store id), checkout(identified by the checkout id, it's the equivalent of a bill), items(identified by the item id, stores details like the brand, name, price and weight), inventory(identified by the store id and the item id, keeps stock of the items) and the action of checking out. The database keeps records on a chain of stores all falling under the same supermarket. Queries can be performed on the transactions of individual customers or the amount made by any single store or an aggregation of stores, sales of any particular product, stock of items in a particular store or across all stores. A trigger is needed to update the inventory on the quantity of items every time a purchase is made. The database is capable of setting up efficient record keeping. It makes it easier to query and chart the sales and thus plan business effectively. It keeps the running smooth.

TABLE OF CONTENTS

Project Report	1
Introduction	3
Data Model	6
Functional Dependencies	7
Normalisation	9
DDL	10
Triggers	16
Output	19
SQL Queries	21
Conclusion	23

INTRODUCTION

The Supermarket Database Management System focuses on the small aspect of a customer buying items across stores, all under the same supermarket brand and the employees and inventory across these stores. It includes the following entities:

1. Customer:

This entity type represents all the people that shop at the grocery store. A customer performs a checkout. the CUSTOMER entity relates to the CHECKOUT entity via the BUY ITEM relationship. The CUST_ID primary key is a foreign key in the checkout table.

2. Employee:

This entity type represents all the people that work for the supermarket chain. It is a superclass because both managers and salaried people are in this entity type. It related to the STORE entity with the WORKS FOR relationship. It has a recursive relationship with SUPERVISING as employees manage themselves. Employees can have dependents through the DEPENDENTS relationship.

3. Dependents:

This is a weak entity type representing anyone who is a dependent of an EMPLOYEE via the DEPENDENTS relationship.

4. Checkout:

This entity type represents an atomic transaction of a customer purchasing items in the store. It relates to STORE via the CHECKOUT LOCATION relationship. It relates to CUSTOMERS via the BUY ITEMS relationship. It relates to the employee who performed the checkout with the EMPLOYEE CHECKOUT ACTION relationship. It is connected to the ITEMS entity through the CHECKOUT ACTION relationship. This relationship will become a table using the primary key from CHECKOUT and ITEMS to join every item on each individual checkout transaction.

5. Items:

This entity type represents the individual store items someone would purchase like milk, cheese, meat, etc. ITEMS is related to CHECKOUT as described above.

6. Store:

This entity type represents the actual brick and mortar buildings where food is placed and customers go to and buy. It has two relationships with EMPLOYEES. One is WORKS FOR and that relationship describes which EMPLOYEES are at which STORE. The second relationship is MANAGES FOR. This connects which manager supervises which store. The Store_ID primary key is useful as foreign key in multiple tables.

The Relationship sets are as follows:

1. WORKS FOR:

It describes the interaction between the EMPLOYEE and STORE entity types. It is a binary relationship type. Many employees can work for one store. Every employee must work for a store, but only one store. Therefore there is total participation for each side of the WORKS FOR relationship. The mapping cardinality of EMPLOYEE:STORE is M:1. The multiplicity of EMPLOYEE in WORKS FOR is 1..M as each employee must be at a store and more than one could be. The multiplicity of STORE in WORKS FOR is 0..M as a STORE could be new and have no employees initially assigned.

2. MANAGES FOR:

The MANAGES FOR relationship describes the interaction between the EMPLOYEES entity and the STORE entity. It is a binary relationship. The employees table has an identification for which employees are also managers. These managers are assigned stores to manage. A store can be new and not have any managers assigned, therefore it is a partial participation. The mapping cardinality of EMPLOYEES:STORE within this relationship is M:1. The multiplicity of EMPLOYEE in MANAGES FOR is 1 as one manager can manage a store. . The multiplicity of STORE in MANAGES FOR is 0..1 as a STORE could be new and have no managers initially assigned.

3. CHECKOUT ACTION:

The CHECKOUT ACTION relationship is a binary relationship between the CHECKOUT entity and the ITEMS entity. A checkout transaction can contain many separate items but each item is only contained once in a specific checkout. This relationship is constructed by the combination of the primary key of CHECKOUT entity (checkout_ID) and the PK of ITEMS entity (items_ID). This primary key merger is represented as a new CHECKOUT_ITEMS entity. The multiplicity between the CHECKOUT entity and the CHECKOUT_ITEMS entity is 1..M with full participation. The multiplicity between the ITEMS entity and the CHECKOUT_ITEMS entity is 1..M also. The ER multiplicity between the entities CHECKOUT and ITEMS is M:M as a checkout can contain many items and many checkouts can contain the same item.

4. INVENTORY:

The INVENTORY relationship is a binary relationship. It connects the STORE entity with the ITEMS entity. Many items are contained in a store and each item will be in multiple stores. Therefore, the ER multiplicity between the entities STORE and ITEMS is M:M. This relationship is constructed by the combination of the PK of STORE entity (checkout_ID) and the PK of ITEMS entity (items_ID). The multiplicity between the STORE entity and the INVENTORY entity is 1..M with full participation. The multiplicity between the ITEMS entity and the INVENTORY entity is 1..M also.

5. CHECKOUT_LOCATION:

This relationship connects the CHECKOUT entity and the STORE entity. It tells the manager where the checkout transaction happened. Many checkouts can happen at each store and a checkout must have a store as part of it so there is a total participation between the CHECKOUT entity and the STORE entity. The mapping cardinality of CHECKOUT:STORE is M:1. The multiplicity between the CHECKOUT entity and the CHECKOUT LOCATION relationship entity is M..1 as many checkouts happen per

location. The multiplicity between the STORE entity and the CHECKOUT LOCATION relationship entity is 1..0.

6. BUY ITEM:

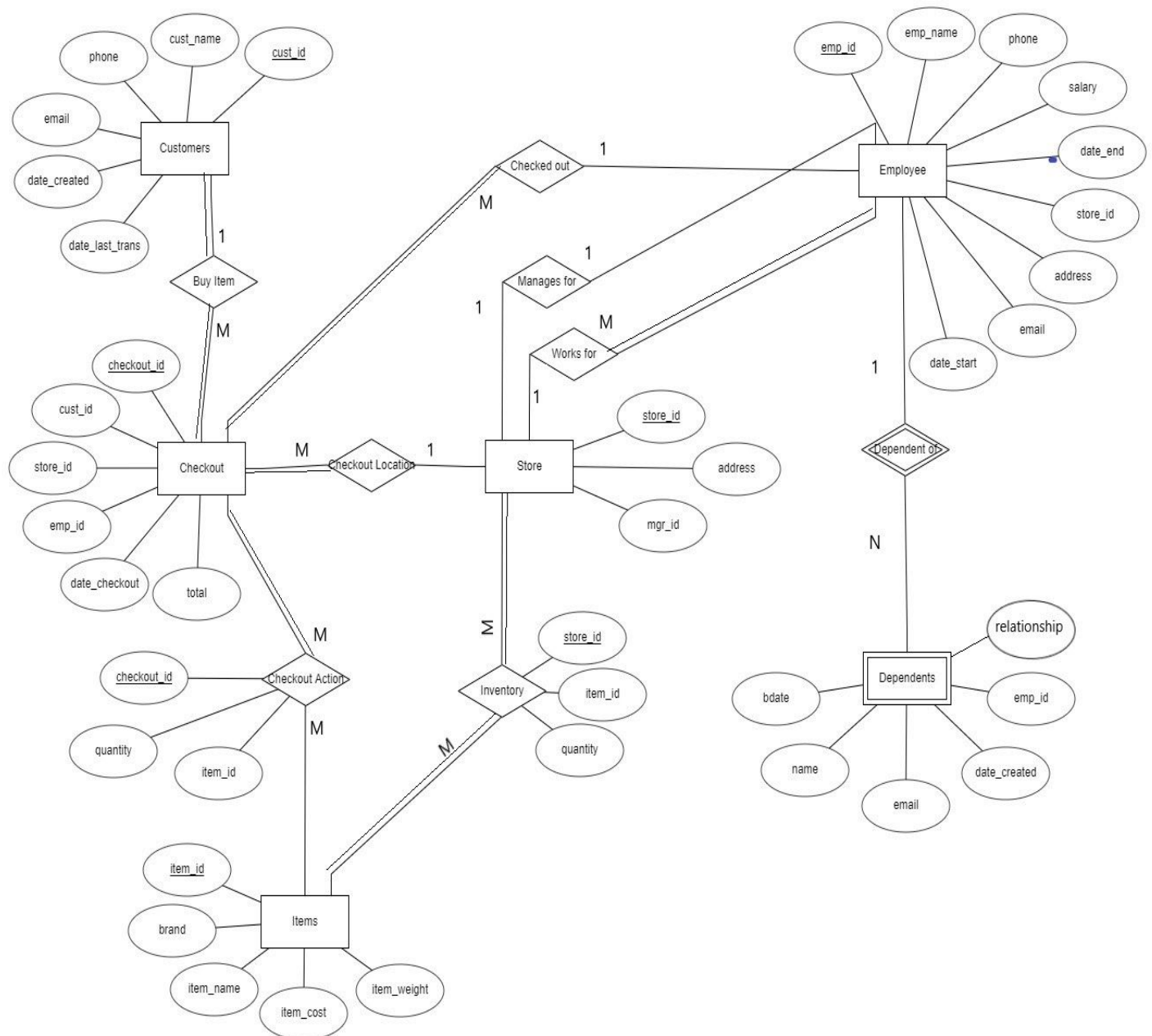
This relationship connects the CUSTOMERS entity to the CHECKOUT entity and is a binary relationship. Many customers can do many checkouts but each checkout has one customer assigned. Therefore there is a 1:M mapping cardinality between CUSTOMERS:CHECKOUT. There is full participation between the CUSTOMERS entity and the BUY ITEM relationship and between the CHECKOUT entity and the BUY ITEM relationship. The multiplicity of CUSTOMERS in BUY ITEM is 1..M. The multiplicity of CHECKOUT in BUY ITEM is 1..M.

7. CHECKED OUT:

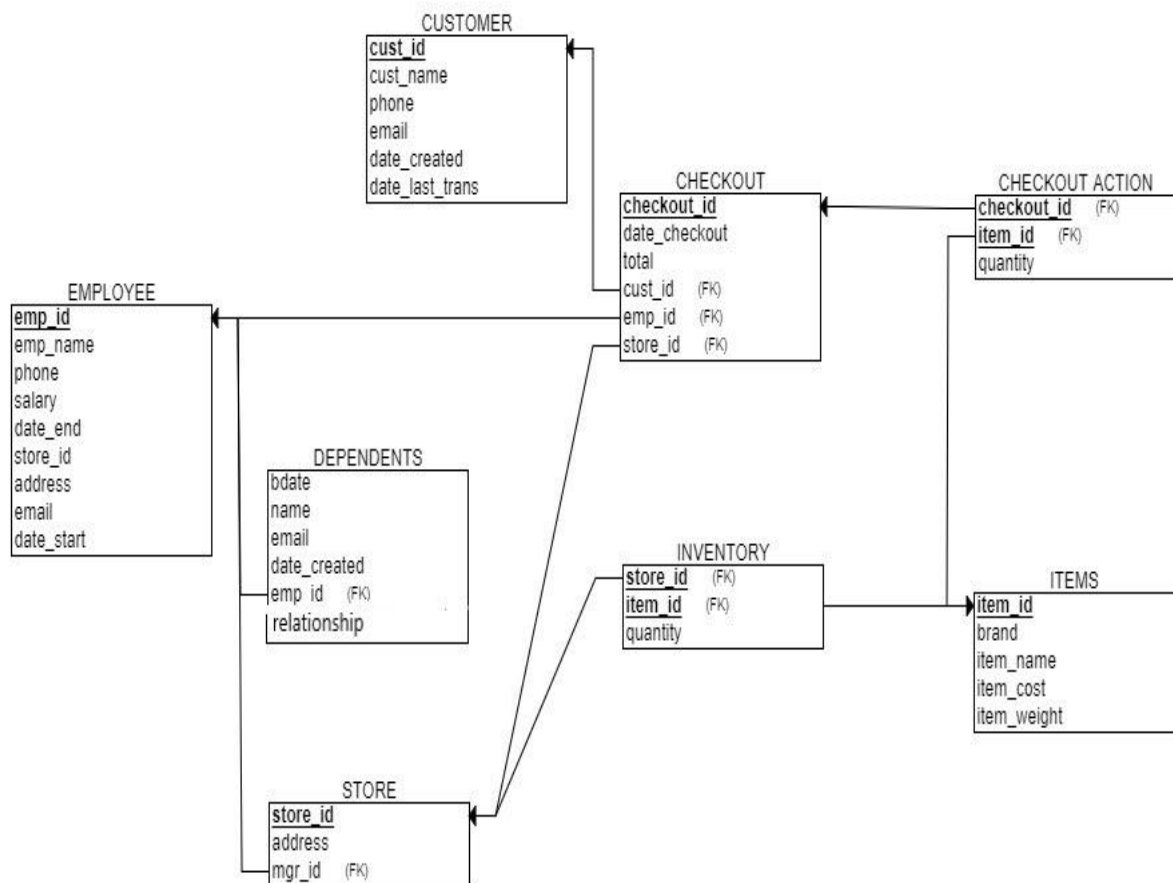
This relationship relates the EMPLOYEE to CHECKOUT. CHECKOUT has total participation, since every checkout has to be made by an employee. However, if an employee is new, he won't have any checkouts to his association. Hence the employee entity is partially dependent. This relationship (CHECKOUT : EMPLOYEE) is of M:1 cardinality, since a checkout is made by 1 employee but an employee can make many checkouts. The multiplicity of EMPLOYEE in CHECKED OUT is 0..M since an employee can make multiple checkouts. The multiplicity of CHECKOUT in CHECKED OUT is 1, since a checkout is made by a single employee.

DATA MODEL:

ER Model :



Schema:



FUNCTIONAL DEPENDENCIES

1. CUSTOMER:

- $\text{cust_id} \rightarrow \{\text{cust_id}, \text{cust_name}, \text{phone}, \text{email}, \text{date_created}, \text{date_last_trans}\}$
- $\text{phone} \rightarrow \{\text{cust_id}, \text{cust_name}, \text{phone}, \text{email}, \text{date_created}, \text{date_last_trans}\}$
- $\text{email} \rightarrow \{\text{cust_id}, \text{cust_name}, \text{phone}, \text{email}, \text{date_created}, \text{date_last_trans}\}$

PRIMARY KEY: cust_id

CANDIDATE KEY: cust_id, phone, email

2. EMPLOYEE:

➤ emp_id → {emp_id, emp_name, phone, salary, date_end, store_id, address, email, date_start}

➤ phone → {emp_id, emp_name, phone, salary, date_end, store_id, address, email, date_start}

➤ email → {emp_id, emp_name, phone, salary, date_end, store_id, address, email, date_start}

PRIMARY KEY: emp_id

CANDIDATE KEY: emp_id, phone, email

3. DEPENDENTS:

➤ {emp_id, name} → {emp_id, bdate, name, email, date_created, relationship}

PARTIAL KEY : {emp_id, name}

4. STORE:

➤ store_id → { store_id, address, mgr_id }

➤ address → { store_id, address, mgr_id }

➤ mgr_id → { store_id, address, mgr_id }

PRIMARY KEY: store_id

CANDIDATE KEY: store_id, address, mgr_id

5. CHECKOUT:

➤ checkout_id → {checkout_id, cust_id, store_id, emp_id, total}

PRIMARY KEY: checkout_id

CANDIDATE KEY: checkout_id

6. ITEMS:

➤ item_id → {item_id, brand, item_name, cost, weight}

➤ {brand, item_name} → {item_id, brand, item_name, cost, weight}

PRIMARY KEY: item_id

CANDIDATE KEY: checkout_id, {brand, item_name}

7. INVENTORY:

➤ {store_id, item_id} → {store_id, item_id, quantity}

PRIMARY KEY: {store_id, item_id}

CANDIDATE KEY: {store_id, item_id}

8. CHECKOUT_ACTION:

➤ checkout_id → {checkout_id, quantity, item_id}

PRIMARY KEY: checkout_id

CANDIDATE KEY: checkout_id

NORMALISATIONS

1. First Normal Form:

A relation is in first normal form if every attribute in that relation is a single valued attribute. The relational model obtained above is thus in the first normal form as all attributes in all relations are single valued.

If a customer/employee were allowed to have two phone numbers, then this would be a violation of the first normal form.

2. Second Normal Form:

A relation is in second normal form if:

- it's in the first normal form.
- no non primary attribute is functionally dependent on a proper subset of the candidate key.

All the relations of the model above are in the second normal form since such a functional dependency is not observed.

The second normal form would be violated if, for example, the address column were split and the table had attributes - state, country, pincode.

Functional Dependency:

pincode → {state, country}

where state, country are non-prime attributes. Pincode is a subset of candidate key.

3. Third Normal Form:

A relation is in third normal form if:

- it's in the third normal form.
- no non-primary attribute is functionally dependent on another non-primary attribute.

All the relations of the model above are in the third normal form since such a functional dependency is not observed.

Third normal form would be violated if, as mentioned above, attributes state, country were added.

Then,

state \rightarrow country

where both state and country are non-prime attributes.

LOSSLESS JOIN

If a relation is not in normal form, the relation is split such that on performing join on the obtained relations, the original relation can be obtained. This is ensured by ensuring that the primary key of the original relation forms the foreign key/ common attribute between the two new relations.

Since our relations are already normalised, lossless decomposition needn't be formed.

DDL

```
DROP DATABASE supermarket;  
CREATE DATABASE supermarket;  
  
\c supermarket  
  
CREATE TABLE customer
```

```

(
    cust_id int,
    cust_name varchar(50) NOT NULL,
    phone bigint NOT NULL,
    email varchar(50),
    date_created date,
    date_last_trans date,
    PRIMARY KEY(cust_id),
    CONSTRAINT chk_email check (email like '%@%.____')
);

CREATE TABLE employee
(
    emp_id int,
    emp_name varchar(50) NOT NULL,
    phone bigint NOT NULL,
    salary int NOT NULL,
    date_end date,
    store_id int NOT NULL,
    address varchar(128),
    email varchar(50),
    date_start date NOT NULL,
    PRIMARY KEY(emp_id),
    CONSTRAINT chk_phone_salary_email check(email like '%@%.____' AND
phone<=9999999999 AND salary>0 )
);

CREATE TABLE dependents
(
    bdate date,
    name varchar(50) NOT NULL,
    email varchar(50),
    date_created date NOT NULL,
    emp_id int,
    relationship varchar(50),
    FOREIGN KEY(emp_id) REFERENCES employee(emp_id)
);

CREATE TABLE store
(
    store_id int,

```

```

    address varchar(128),
    mgr_id int NOT NULL,
    PRIMARY KEY(store_id),
    FOREIGN KEY(mgr_id) REFERENCES employee(emp_id)
);

CREATE TABLE checkout
(
    checkout_id int,
    cust_id int NOT NULL,
    store_id int NOT NULL,
    emp_id int NOT NULL,
    date_checkout date NOT NULL,
    total int,
    PRIMARY KEY(checkout_id),
    FOREIGN KEY(cust_id) REFERENCES customer(cust_id),
    FOREIGN KEY(store_id) REFERENCES store(store_id),
    FOREIGN KEY(emp_id) REFERENCES employee(emp_id),
    CONSTRAINT chk_total check(total > 0)
);

CREATE TABLE items
(
    item_id int,
    brand varchar(32) NOT NULL,
    item_name varchar(50) NOT NULL,
    item_cost int NOT NULL,
    item_weight int,
    PRIMARY KEY(item_id)
);

CREATE TABLE inventory
(
    store_id int NOT NULL,
    item_id int NOT NULL,
    quantity int NOT NULL,
    PRIMARY KEY(store_id, item_id),
    FOREIGN KEY(store_id) REFERENCES store(store_id),
    FOREIGN KEY(item_id) REFERENCES items(item_id),
    CONSTRAINT chk_qty check(quantity>0)
);

```

```

CREATE TABLE checkout_action
(
    checkout_id int NOT NULL,
    quantity int NOT NULL,
    item_id int NOT NULL,
    PRIMARY KEY (checkout_id, item_id),
    FOREIGN KEY(checkout_id) REFERENCES checkout(checkout_id),
    FOREIGN KEY(item_id) REFERENCES items(item_id),
    CONSTRAINT chk_qty check(quantity>0)
);

INSERT INTO customer(cust_id, cust_name, phone, email, date_created,
date_last_trans)
VALUES
    (50, 'Bob Hope', 6615552485, 'bobhope@gmail.com', '2001-1-1', '2001-5-7'),
    (51, 'Renee Hicks', 4589854588, 'Dragonthing@aol.com', '2005-5-5',
'2009-4-25'),
    (52, 'Scott Sheer', 4176521425, 'Scotts@hotmail.com', '2011-12-12',
'2012-3-4'),
    (53, 'Colleen Mctyre', 9008752320, 'CMcT@ct.com', '2008-8-12',
'2009-5-9'),
    (58, 'Bart Simpson', 9886652035, 'bart@gmail.com', '2001-6-6',
'2007-8-25'),
    (67, 'Lisa Girl', 6619755896, 'lisa@gmail.com', '1999-4-9', '2000-4-6'),
    (99, 'Jeremy Scott', 4586895847, 'TheBigMan@gmail.com', '2000-1-9',
'2001-10-10'),
    (105, 'Master Shake', 5555555555, 'MixMaster@crimefighter.org',
'2000-8-25', '2001-8-18'),
    (178, 'Bruce Wayne', 6619872145, 'IamBatman@crimefighter.org', '2000-1-9',
'2001-12-5'),
    (179, 'Seymoure Butes', 4789582145, 'SButes@education.edu', '2001-5-20',
'2001-8-18') ;

INSERT INTO employee(emp_id, emp_name, phone, salary, date_end, store_id,
address, email, date_start)
VALUES
    (1, 'Darrel Philbin', 5489659874, 20, '2011-2-2', 854, '286 Clown St.',
'baldman@gmailcom', '1985-4-5'),
    (2, 'Ricky Tanner', 6988532587, 10000, '1999-6-10', 354, '1587 H St',
'omegaman@gmail.com', '1990-6-8'),

```

```

(3, 'Susan Phillips', 9856984523, 1500, NULL, 696, '695 LMNOP St.',
'streetsmart@gmail.com', '1972-6-9'),
(4, 'George Scott', 2586521452, 42000, NULL, 159, '4521 Gold St.', NULL,
'1999-7-25'),
(5, 'Erin Abernathy', 5896583541, 30000, NULL, 674, '635 Numero Inn',
'drinkerster@gmail.com', '1998-12-20'),
(6, 'Ted Smith', 4736593569, 50000, NULL, 369, '12 S st', NULL,
'1989-6-8'),
(7, 'Harry Buts', 2586584763, 12, NULL, 778, '1 wonder st', NULL,
'1970-10-20'),
(8, 'Maynar Teener', 2596573257, 9250, NULL, 989, '24 Nice Inn',
'meme69@gmail.com', '2005-6-4'),
(9, 'Matt Longfellow', 5249868525, 60000, NULL, 247, '6144 Computer Way',
'thisisshirt@gmail.com', '2000-9-21'),
(10, 'Jerry Garcia', 6521458569, 52000, NULL, 348, '214 q st.',
'govlot@gmail.com', '1990-9-24'),
(11, 'Lawarnc Tom', 9658745632, 15000, '2011-9-1', 348, '2154 Beech st',
NULL, '1989-1-20'),
(12, 'Dexter Robert', 1111111111, 12.25, NULL, 854, '365 Moon dr', NULL,
'1990-5-6'),
(13, 'Mark Nick', 2225478512, 8250, NULL, 854, '65412 b St.', NULL,
'1998-2-5'),
(14, 'Jeremy David', 2356895654, 16000, NULL, 159, '2 Molly Way', NULL,
'2000-6-3'),
(15, 'Luke Ted', 2144544123, 20000, NULL, 778, 'Southland Avy', NULL,
'2004-9-9');

```

```
INSERT INTO dependents(bdate, name, email, date_created, emp_id)
```

```
VALUES
```

```

('1965-3-21', 'Darrel Philbin', 'baldman@gmail.com', '1985-4-5', 1),
('1979-4-26', 'Ricky Tanner', 'omegaman@gmail.com', '1990-6-8', 2),
('1968-7-25', 'Susan Philips', 'streetsmart@gmail.com', '1972-6-9', 3),
('1980-5-4', 'George Scott', NULL, '1999-7-25', 4),
('1980-3-4', 'Erin Abernathy', 'drinkerstre@gmail.com', '1998-12-20', 5),
('1975-2-27', 'Ted Smith', NULL, '1989-6-8', 6),
('1950-3-30', 'Harry Buts', NULL, '1970-10-20', 7),
('1996-11-18', ' Maynar Teener', 'meme69@gmail.com', '2005-6-4', 8),
('1973-10-14', 'Matt Longfellow', 'thisisshirt@gmail.com', '2000-9-21',
9),
('1981-12-31', 'Jerry Garcia', 'govlot@gmail.com', '1990-9-24', 10),
('1969-12-13', 'Lawrance Tom', NULL, '1989-1-20', 11),

```

```
( '1968-6-6', 'Dexter Robert', NULL, '1990-5-6', 12),
( '1973-7-14', 'Mark Nick', NULL, '1998-2-5', 13),
( '1872-7-9', 'Jeremy David', NULL, '2000-6-3', 14),
( '1990-5-12', 'Luke Ted', NULL, '2004-9-9', 15);
```

```
INSERT INTO store(store_id, address, mgr_id)
VALUES
```

```
(854, '22556 Elm St', 13),
(354, '820 Birch Rd', 2),
(696, '710 Edison Dr', 3),
(159, '13636 Fir St', 4),
(674, '14496 Maple Way', 5),
(369, '940 Green St', 6),
(778, '341 Main St', 15),
(989, '25459 Aspen Blvd', 8),
(247, '13695 Alder St', 9),
(348, '650 Beech St', 11);
```

```
INSERT INTO checkout (checkout_id, cust_id, store_id, emp_id, date_checkout,
total)
```

```
VALUES
```

```
(201, 50, 854, 2, '2011-6-10', 65.25),
(32, 51, 354, 2, '2011-6-9', 115.25),
(6589, 52, 696, 3, '2010-8-12', 66.52),
(2147, 99, 159, 4, '2010-6-5', 500.25),
(210, 179, 854, 5, '2009-11-5', 41.35),
(2141, 105, 369, 6, '2007-4-5', 64.25),
(3652, 178, 778, 6, '2011-12-12', 14.25),
(125, 58, 989, 7, '2005-12-24', 80.85);
```

```
INSERT INTO items(item_id, brand, item_name, item_cost, item_weight)
```

```
VALUES
```

```
(12, 'Nabisco', 'Cookies', 2.25, 22.4),
(658, 'PhillpMorris', 'Cigarettes', 5.00, 89),
(4587, 'Kraft', 'Cheese', 6.00, .11),
(2365, 'Kellogg', 'Cereal', 1.99, 18),
(84854, 'Quaker', 'Oatmeal', 2.50, 1),
(3521, 'Nabisco', 'Crackers', 4.00, 2),
(355, 'HomeBrand', 'Spaghehetti', .99, 3),
(1566, 'DelMonte', 'Canned Fruit', .50, 5.2),
(256, 'Hersey', 'Cookies', 3.99, 52.8),
```

```
(145, 'Kleenex', 'Tissues', 2.99, 34);
```

```
INSERT INTO inventory(store_id, item_id, quantity)
```

```
VALUES
```

```
(854, 12, 10),
(854, 658, 10),
(354, 1566, 4),
(696, 12, 23),
(696, 658, 38),
(159, 355, 27),
(159, 1566, 31),
(674, 4587, 23),
(674, 2365, 28),
(854, 84854, 10),
(696, 256, 100),
(354, 256, 50),
(778, 145, 200);
```

```
INSERT INTO checkout_action(checkout_id, quantity, item_id)
```

```
VALUES
```

```
(210, 2, 12),
(210, 2, 658),
(32, 2, 84854),
(32, 2, 3521),
(6589, 2, 4587),
(6589, 2, 2365),
(2147, 2, 4587),
(201, 2, 2365),
(201, 2, 256);
```

TRIGGERS

A trigger is created to update the inventory every time an item is purchased.i.e, every time a row is inserted into the CHECKOUT_ACTION relation, the QUANTITY of the item is decreased in the INVENTORY of that particular item in that particular store.

'NEW' is a record that holds the recently updated values. With ITEM_ID as the Foreign Key from INVENTORY to CHECKOUT_ACTION, the update is made with the trigger update_stock.

A function update() is created to perform the change in the value / update in the quantity cell of the INVENTORY relation.


Attached below is the code for the trigger:

```
CREATE OR REPLACE FUNCTION update()
    RETURNS trigger AS
$BODY$
BEGIN
    UPDATE inventory
    SET quantity = quantity - new.quantity
    WHERE exists (SELECT * FROM inventory,
                  (SELECT item_id, store_id FROM checkout, checkout_action
                   WHERE (new.checkout_id = checkout_action.checkout_id))
                  as interm
                  WHERE (inventory.item_id = interm.item_id AND
inventory.store_id = interm.store_id)
                  );
    RETURN NEW;
END;
$BODY$
language plpgsql;

CREATE TRIGGER update_stock
AFTER INSERT
ON checkout_action
FOR EACH ROW
EXECUTE PROCEDURE update();
```

Before Updating CHECKOUT_ACTION:

```
supermarket1=# SELECT * FROM INVENTORY;
store_id | item_id | quantity
-----+-----+-----
      854 |      12 |        10
      854 |     658 |        10
      854 |    4587 |        23
      854 |    2365 |        10
      354 |    3521 |         4
      354 |    1566 |         4
      696 |      12 |        23
      696 |     658 |        38
      159 |     355 |        27
      159 |    1566 |        31
      674 |    4587 |        23
      674 |    2365 |        28
      369 |      12 |        50
      989 |     658 |         3
      247 |     355 |        12
      348 |     145 |        45
      348 |     256 |        17
      854 |   84854 |        10
      696 |     256 |       100
      354 |     256 |        50
      778 |     145 |       200
(21 rows)
```



After updating CHECKOUT_ACTION:

```
supermarket1=# INSERT INTO CHECKOUT VALUES(150, 52, 778, 2, '2012-5-5', 100);
INSERT 0 1
supermarket1=# INSERT INTO CHECKOUT_ACTION VALUES(150, 2, 145);
INSERT 0 1
supermarket1=# SELECT * FROM INVENTORY;
store_id | item_id | quantity
-----+-----+-----
      854 |      12 |         8
      854 |     658 |         8
      854 |    4587 |        21
      854 |    2365 |         8
      354 |    3521 |         2
      354 |    1566 |         2
      696 |      12 |        21
      696 |     658 |        36
      159 |     355 |        25
      159 |    1566 |        29
      674 |    4587 |        21
      674 |    2365 |        26
      369 |      12 |        48
      989 |     658 |         1
      247 |     355 |        10
      348 |     145 |        43
      348 |     256 |        15
      854 |   84854 |         8
      696 |     256 |        98
      354 |     256 |        48
      778 |     145 |       198
(21 rows)
```

The last row now has the updated value for QUANTITY : 200.

OUTPUT

```
supermarket1=# \d
```

List of relations			
Schema	Name	Type	Owner
public	checkout	table	postgres
public	checkout_action	table	postgres
public	customer	table	postgres
public	dependents	table	postgres
public	employee	table	postgres
public	inventory	table	postgres
public	items	table	postgres
public	store	table	postgres

```
(8 rows)
```

```
supermarket1=# SELECT * FROM CUSTOMER;
```

cust_id	cust_name	phone	email	date_created	date_last_trans
50	Bob Hope	6615552485	bobhope@gmail.com	2001-01-01	2001-05-07
51	Renee Hicks	4589854588	Dragonthing@aol.com	2005-05-05	2009-04-25
52	Scott Sheer	4176521425	Scotts@hotmail.com	2011-12-12	2012-03-04
53	Colleen Mctyre	9008752320	CMcT@ct.com	2008-08-12	2009-05-09
58	Bart Simpson	9886652035	bart@gmail.com	2001-06-06	2007-08-25
67	Lisa Girl	6619755896	lisa@gmail.com	1999-04-09	2000-04-06
99	Jeremy Scott	4586895847	TheBigMan@gmail.com	2000-01-09	2001-10-10
105	Master Shake	5555555555	MixMaster@crimefighter.org	2000-08-25	2001-08-18
178	Bruce Wayne	6619872145	IamBatman@crimefighter.org	2000-01-09	2001-12-05
179	Seymour Butes	4789582145	SButes@education.edu	2001-05-20	2001-08-18
180	Daniel Radcliffe	476889012	harrypotter@gmail.com	2005-07-08	2018-11-12
182	George Raffe	476889089	giraffe@gmail.com	2005-09-08	2018-11-12
190	Toby Elliot	4768895432	telliotr@gmail.com	2005-11-08	2018-12-12
185	Donald Trump	543289012	trump@gmail.com	2005-12-08	2018-12-21
207	Mary Poppins	543289057	poppins@gmail.com	2005-12-09	2018-12-27
260	Drew Barrymore	543283241	dmore@gmail.com	2005-12-10	2018-12-19
215	Amelia Earheart	543289999	amelia@gmail.com	2005-12-11	2018-12-15
295	George Weasley	543287777	weasley@gmail.com	2005-12-12	2018-12-13
301	Fred Weasley	543288888	fred@gmail.com	2005-12-12	2018-12-30
302	Hermione Granger	543289989	hermione@gmail.com	2005-12-13	2018-12-31

```
(20 rows)
```

```
supermarket1=# SELECT * FROM EMPLOYEE;
```

emp_id	emp_name	phone	salary	date_end	store_id	address	email	date_start
1	Darrel Philbin	5489659874	20	2011-02-02	854	286 Clown St.	baldman@gmail.com	1985-04-05
2	Ricky Tanner	6988532587	10000	1999-06-10	354	1587 H St	omegaman@gmail.com	1990-06-08
3	Susan Phillips	9856984523	1500		696	695 LMNOP St.	streetsmart@gmail.com	1972-06-09
4	George Scott	2586521452	42000		159	4521 Gold St.		1999-07-25
5	Erin Abernathy	5896583541	30000		674	635 Numero Inn	drinkerster@gmail.com	1998-12-20
6	Ted Smith	4736593569	50000		369	12 S st		1989-06-08
7	Harry Buts	2586584763	12		778	1 wonder st		1970-10-20
8	Maynar Teener	2596573257	9250		989	24 Nice Inn	mem69@gmail.com	2005-06-04
9	Matt Longfellow	5249868525	60000		247	6144 Computer Way	thisisshirt@gmail.com	2000-09-21
10	Jerry Garcia	6521458569	52000		348	214 q st.	govlot@gmail.com	1990-09-24
11	Lawarnc Tom	9658745632	15000	2011-09-01	348	2154 Beech st		1989-01-20
12	Dexter Robert	1111111111	12		854	365 Moon dr		1990-05-06
13	Mark Nick	2225478512	8250		854	65412 b St.		1998-02-05
14	Jeremy David	2356895654	16000		159	2 Molly Way		2000-06-03
15	Luke Ted	2144544123	20000		778	Southland Avy		2004-09-09

```
(15 rows)
```

```
supermarket1=# SELECT * FROM DEPENDENTS;
```

bdate	name	email	date_created	emp_id
1965-03-21	Darrel Philbin	baldman@gmail.com	1985-04-05	1
1979-04-26	Ricky Tanner	omegaman@gmail.com	1990-06-08	2
1968-07-25	Susan Philips	streetsmart@gmail.com	1972-06-09	3
1980-05-04	George Scott		1999-07-25	4
1980-03-04	Erin Abernathy	drinkerstre@gmail.com	1998-12-20	5
1975-02-27	Ted Smith		1989-06-08	6
1950-03-30	Harry Buts		1970-10-20	7
1996-11-18	Maynar Teener	meme69@gmail.com	2005-06-04	8
1973-10-14	Matt Longfellow	thisisshirt@gmail.com	2000-09-21	9
1981-12-31	Jerry Garcia	govlot@gmail.com	1990-09-24	10
1969-12-13	Lawrance Tom		1989-01-20	11
1968-06-06	Dexter Robert		1990-05-06	12
1973-07-14	Mark Nick		1998-02-05	13
1872-07-09	Jeremy David		2000-06-03	14
1990-05-12	Luke Ted		2004-09-09	15

(15 rows)

```
supermarket1=# SELECT * FROM STORE;
```

store_id	address	mgr_id
854	22556 Elm St	13
354	820 Birch Rd	2
696	710 Edison Dr	3
159	13636 Fir St	4
674	14496 Maple Way	5
369	940 Green St	6
778	341 Main St	15
989	25459 Aspen Blvd	8
247	13695 Alder St	9
348	650 Beech St	11

(10 rows)

```
supermarket1=# SELECT * FROM ITEMS;
```

item_id	brand	item_name	item_cost	item_weight
12	Nabisco	Cookies	2	22
658	PhillpMorris	Cigarettes	5	89
4587	Kraft	Cheese	6	0
2365	Kellogg	Cereal	2	18
84854	Quaker	Oatmeal	3	1
3521	Nabisco	Crackers	4	2
355	HomeBrand	Spagheetti	1	3
1566	DelMonte	Canned Fruit	1	5
256	Hersey	Cookies	4	53
145	Kleenex	Tissues	3	34

(10 rows)

```
supermarket1=# SELECT * FROM INVENTORY;
```

store_id	item_id	quantity
854	12	10
854	658	10
854	4587	23
854	2365	10
354	3521	4
354	1566	4
696	12	23
696	658	38
159	355	27
159	1566	31
674	4587	23
674	2365	28
369	12	50
989	658	3
247	355	12
348	145	45
348	256	17
854	84854	10
696	256	100
354	256	50
778	145	200

(21 rows)


```
supermarket1=# SELECT * FROM CHECKOUT_ACTION;
checkout_id | quantity | item_id
-----+-----+-----
          210 |         2 |      12
          210 |         2 |     658
           32 |         2 |   84854
           32 |         2 |   3521
         6589 |         2 |   4587
         6589 |         2 |   2365
        2147 |         2 |   4587
          201 |         2 |   2365
          201 |         2 |    256
          123 |         5 |    145
          124 |         2 |    256
(11 rows)
```

SQL QUERIES

1. Retrieve the name and phone number of all the customers who made a single transaction > 100.

```
SELECT cust_name, phone FROM customer as c, checkout as co
      WHERE (co.total > 100 AND co.cust_id = c.cust_id);
```

```
supermarket1=# SELECT cust_name, phone FROM customer as c, checkout as co
supermarket1=# WHERE (co.total > 100 AND co.cust_id = c.cust_id);
cust_name | phone
-----+-----
Renee Hicks | 4589854588
Jeremy Scott | 4586895847
(2 rows)
```

2. Find the second highest salary among the employees.

```
SELECT min(sal.salary) FROM
      (SELECT DISTINCT salary FROM employee ORDER BY salary desc LIMIT 2) AS sal;
```

```
supermarket1=# SELECT min(sal.salary) FROM
supermarket1=# (SELECT DISTINCT salary FROM employee ORDER BY salary desc
supermarket1=# LIMIT 2) AS sal;
min
-----
52000
(1 row)
```

3. Retrieve the contact details of all the managers who manage a store with more than 2 employees.

```
SELECT e.emp_name, e.phone, e.email FROM employee as e,
      (SELECT mgr_id as m_i FROM store as s,
        (SELECT store_id as s_i, COUNT(*) as emp_count FROM employee GROUP
BY(store_id))
        as emp_per_store
      WHERE s.store_id = emp_per_store.s_i AND emp_per_store.emp_count>1 ) as
mgr_list
WHERE e.emp_id = mgr_list.m_i;
```

```
supermarket1=# SELECT e.emp_name, e.phone, e.email FROM employee as e,
supermarket1-# (SELECT mgr_id as m_i FROM store as s,
supermarket1-# (SELECT store_id as s_i, COUNT(*) as emp_count FROM employee GROUP BY(store_id))
supermarket1-# as emp_per_store
supermarket1-# WHERE s.store_id = emp_per_store.s_i AND emp_per_store.emp_count>1 ) as mgr_list
supermarket1-# WHERE e.emp_id = mgr_list.m_i;
 emp_name  |   phone   | email
-----+-----+-----
Mark Nick  | 2225478512 |
George Scott | 2586521452 |
Luke Ted   | 2144544123 |
Lawarnce Tom | 9658745632 |
(4 rows)
```

4 . Retrieve the stock of 'Cookies' across all stores.

```
SELECT SUM(quantity) FROM inventory as i
      INNER JOIN (SELECT * FROM items WHERE item_name = 'Cookies')
      as cookie ON i.item_id = cookie.item_id;
```

```
supermarket1=# SELECT SUM(quantity) FROM inventory as i
supermarket1-# INNER JOIN (SELECT * FROM items WHERE item_name = 'Cookies')
supermarket1-# as cookie ON i.item_id = cookie.item_id;
 sum
-----
 250
(1 row)
```

5. Retrieve the stock of all items across all the stores.

```
SELECT i.item_id, i.brand, i.item_name, stock
  FROM items as i
    INNER JOIN (SELECT item_id, SUM(quantity) as stock FROM inventory GROUP
BY(item_id))
    AS stock_summary
    ON i.item_id=stock_summary.item_id;
```

```
supermarket1=# SELECT i.item_id, i.brand, i.item_name, stock
supermarket1=# FROM items as i
supermarket1=# INNER JOIN (SELECT item_id, SUM(quantity) as stock FROM inventory GROUP BY(item_id))
supermarket1=# AS stock_summary
supermarket1=# ON i.item_id=stock_summary.item_id;
 item_id |   brand   | item_name | stock
-----+-----+-----+-----
      12 | Nabisco   | Cookies   |    83
     658 | PhillpMorris | Cigarettes |    51
    4587 | Kraft     | Cheese    |    46
    2365 | Kellogg   | Cereal     |    38
   84854 | Quaker    | Oatmeal    |    10
    3521 | Nabisco   | Crackers   |     4
     355 | HomeBrand | Spaghetti  |    39
    1566 | DelMonte  | Canned Fruit |    35
     256 | Hersey    | Cookies    |   167
     145 | Kleenex   | Tissues    |   245
(10 rows)
```

CONCLUSION

A supermarket database management system is thus created to keep records that are dynamically updated with the help of triggers. It could also be improvised to allow updates on EMPLOYEE info, CUSTOMER info, MANAGER (on promotions and resignations) and other relations using triggers. Triggers could be used to increase sales with rewards and coupons by keeping track of a customer's transactions.

The database handles all the relationships existing in the real world and hence is a good implementation for a supermarket database management system.