

DATABASE MANAGEMENT SYSTEMS - CS6106

PROJECT

ONLINE CLOTHING STORE MANAGEMENT SYSTEM

Prepared and submitted by
Safiya Fathima - 2019103052
Nithin K - 2019103550

INDEX

SL NO	TOPIC	PAGE NUMBER
1	PROJECT ABSTRACT	1
2	FUNCTIONAL AND NON FUNCTIONAL REQUIREMENTS	2
3	ER DIAGRAM AND SCHEMA DESIGN	3
4	USER INTERFACE DESIGN	4
5	CONNECTING TO THE DATABASE	6
6	SETTING UP RELATIONS	7
7	POPULATING THE TABLES	14
8	SETTING UP PROCEDURES	15
9	SETTING UP VIEWS	16
10	SETTING UP TRIGGERS	18
11	RESULTS	19
12	CONCLUSION	26
13	REFERENCES	27

PROJECT ABSTRACT

The project ‘Online Clothing Store Management System’ is developed to maintain a record of clothes, users, carts and bills of a particular Ecommerce clothing website.

Languages, frameworks and tools used:

- Front-end: HTML, CSS, JavaScript, Bootstrap
- Back-end: NodeJS
- Database Server: PostgreSQL Server 13.2
- Database Tools: psql (PostgreSQL) Shell 13.2, pgAdmin 4

FUNCTIONAL AND NON FUNCTIONAL REQUIREMENTS

Functional Requirements:

The system maintains the data of:

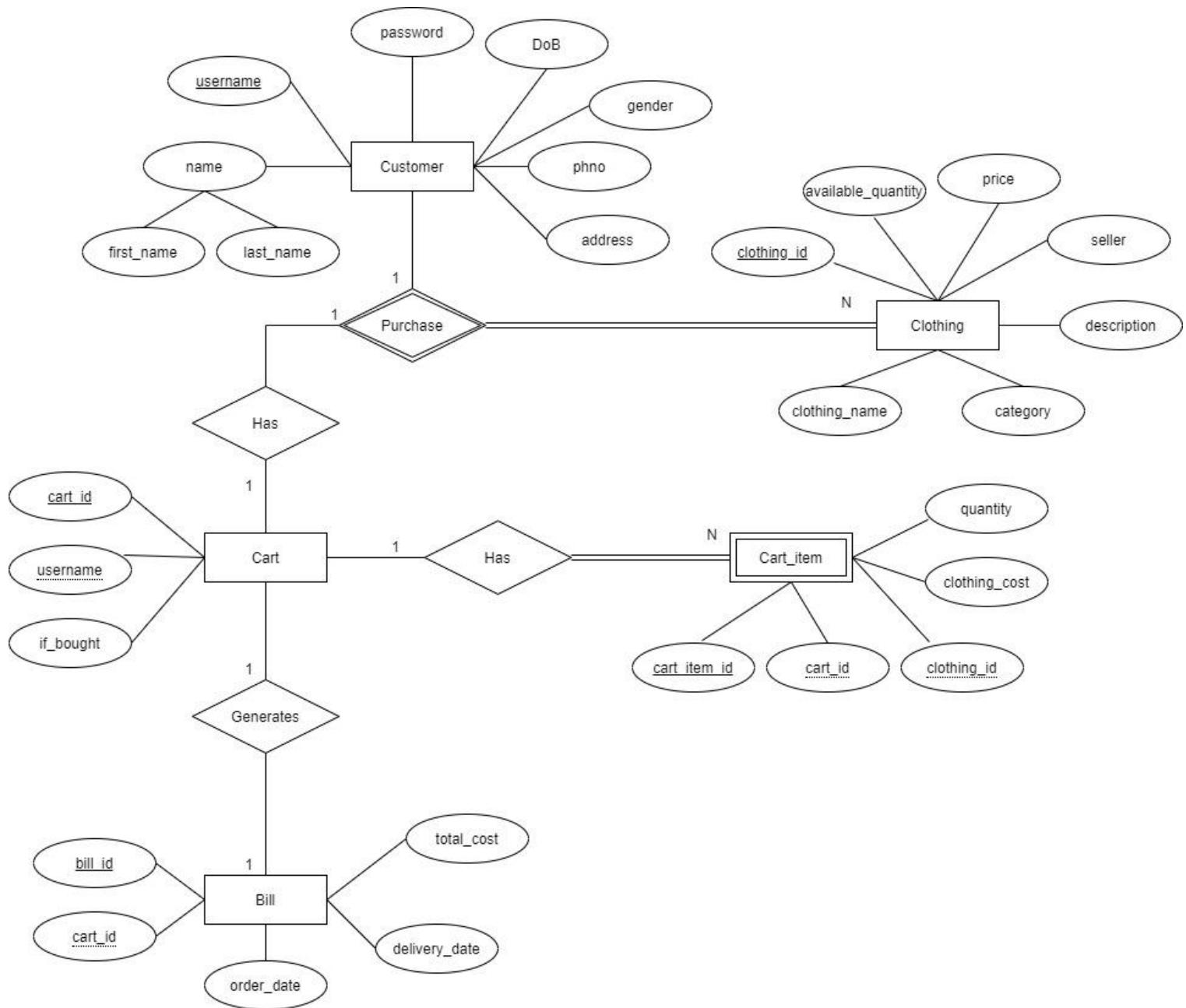
- Clothing details and available quantity
- User details
- Items currently in every user's cart
- Items ordered by every user
- Billing details of every user

Non Functional Requirements:

The system provides the following:

- Security
 - Hashing of passwords
- User Interface
 - A friendly and intuitive user interface has been developed for better user experience.

ENTITY RELATIONSHIP DIAGRAM



USER INTERFACE DESIGN

The user interface has been constructed using HTML, CSS and JavaScript, with help from frameworks like Bootstrap to get a pleasant user interface.

Here is a sample code snippet for the landing page of the application:

```
↳ index.html ×

main-app > public > ↳ index.html > ⏺ html > ⏺ head
1   <!DOCTYPE html>
2   <html lang="en">
3
4   <head>
5     <meta charset="UTF-8">
6     <meta content="IE=edge" http-equiv="X-UA-Compatible">
7     <meta content="width=device-width,initial-scale=1" name="viewport">
8     <meta content="Page description" name="description">
9     <meta name="google" content="notranslate" />
10
11    <!-- Disable tap highlight on IE -->
12    <meta name="msapplication-tap-highlight" content="no">
13
14    <link href="../assets/apple-icon-180x180.png" rel="apple-touch-icon">
15    <link href="../assets/favicon.ico" rel="icon">
16
17
18
19    <title>Spring Fashion</title>
20
21    <link href="../main.82cf66e.css" rel="stylesheet"></head>
22
23  <body>
24
25    <!-- Add your content of header -->
26    <header class="">
27      <div class="navbar navbar-default visible-xs">
28        <button type="button" class="navbar-toggle collapsed">
29          <span class="sr-only">More</span>
30          <span class="icon-bar"></span>
31          <span class="icon-bar"></span>
32          <span class="icon-bar"></span>
33        </button>
34        <a href="../index.html" class="navbar-brand">Spring Fashion</a>
35      </div>
36
```

With the following end products:

The screenshot shows a website for "Spring Fashion" with a navigation bar including Home, About, Our Collection, Login, and Sign Up. The main content area displays several images of clothing: a rack of shirts, a stack of jeans, a shirt with a red and blue striped cuff, a stack of folded jeans, a row of folded shirts, a close-up of a person's legs in jeans, and a row of colorful garments on a clothesline.

The screenshot shows a website for "Spring Fashion" with a navigation bar including Home, About, Our Collection, Login, and Sign Up. The main content area features a large historical black-and-white photograph of women working in a garment factory, hanging laundry, and a small child standing nearby. Social media sharing icons and a copyright notice are at the bottom.

About us

Since 1837, Spring has remained faithful to its artisanal model and its humanist values. The freedom to create, the constant quest for beautiful materials, and the transmission of exceptional know-how – which enable the creation of useful, and elegant objects which stand the test of time – forge the uniqueness of Spring. Family-run, independent and socially responsible, the company is committed to maintaining the majority of its production in France, through its 43 production sites, while developing its international distribution network of 306 stores in 45 countries.

Creative Freedom

The sixteen métiers of the house create collections that combine freedom with inventiveness and know-how. The objects are designed to be durable and to adapt to changing lifestyles.

Responsible and sustainable development

Humanist values, based in the artisanal world, guide the company in its development: Springs is committed to preserving resources, supporting the regions in which its manufacturers operate and ensuring the transmission of exceptional know-how.

Springs around the World

Each of the 306 stores across 45 countries is a welcoming and unique "house of objects" that combines the identity of Spring with the local culture, offering visitors a unique experience.

CONNECTING TO THE DATABASE

PostgreSQL database has been used for the implementation of the system.

An online_store database is created using the psql Shell which is accessed through the Windows command prompt.

```
PS C:\Users\safiy> Command Prompt - psql -U postgres
Microsoft Windows [Version 10.0.19041.928]
(c) Microsoft Corporation. All rights reserved.

C:\Users\safiy>psql -U postgres
Password for user postgres:
psql (13.2)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=# CREATE DATABASE online_store;
CREATE DATABASE
postgres=# \c online_store
You are now connected to database "online_store" as user "postgres".
```

SETTING UP RELATIONS

The following relations have been created using the pgAdmin GUI interface:

1. Clothing

Attributes:

- Clothing_id (primary key, uniquely identifies a clothing)
- Clothing_name
- Category
- Description
- Seller
- Price
- Available_quantity

All attributes are non-transitively determined by the primary key - clothing_id. Hence the relation is in **Boyce-Codd Normal Form**.

The screenshot shows the pgAdmin Query Editor interface. The title bar says 'online_store/postgres@PostgreSQL 13'. The main area contains the following SQL code:

```
1 CREATE TABLE Clothing (
2     clothing_id VARCHAR(5) PRIMARY KEY,
3     clothing_name VARCHAR(15) NOT NULL,
4     category VARCHAR(15),
5     description VARCHAR(45),
6     seller VARCHAR(20),
7     price INTEGER NOT NULL,
8     available_quantity INTEGER
9 );
10
11 ALTER TABLE Clothing ADD CONSTRAINT
12 chk_price CHECK(price >= 0);
13
```

Below the code, there are tabs for 'Data Output', 'Explain', 'Messages' (which is selected), and 'Notifications'. At the bottom, it says 'ALTER TABLE' and 'Query returned successfully in 122 msec.'

```

online_store=# \d+ clothing
                                         Table "public.clothing"
   Column      |      Type      | Collation | Nullable | Default | Storage | Stats target | Description
---+---+---+---+---+---+---+---+---+---+
clothing_id | integer |          | not null |          | plain   |          |          |
clothing_name | character varying(15) |          | not null |          | extended |          |          |
category | character varying(15) |          |          |          |          |          |
description | character varying(200) |          |          |          |          |          |
seller | character varying(20) |          |          |          |          |          |
price | integer |          |          | not null |          |          |          |
available_quantity | integer |          |          |          |          |          |          |
Indexes:
  "clothing_pkey" PRIMARY KEY, btree (clothing_id)
Check constraints:
  "chk_price" CHECK (price >= 0)
Referenced by:
  TABLE "cart_item" CONSTRAINT "clothingid_fk" FOREIGN KEY (clothing_id) REFERENCES clothing(clothing_id)
Access method: heap

```

2. Customer

Attributes:

- **Username** (primary key, uniquely identifies a customer)
- Password
- Name - First_name and Last_name
- DoB
- Gender
- Phno
- Email
- address

All attributes are non-transitively determined by the primary key - username. Hence the relation is in **Boyce-Codd Normal Form**.

online_store/postgres@PostgreSQL 13 ▾

Query Editor Query History

```

1 CREATE TABLE Customer (
2     username VARCHAR(10) PRIMARY KEY NOT NULL,
3     pass VARCHAR(15) NOT NULL,
4     first_name VARCHAR(20),
5     last_name VARCHAR(20),
6     DoB DATE,
7     gender CHAR,
8     phno INTEGER,
9     address VARCHAR(50)
10 );
11
12 ALTER TABLE Customer ADD CONSTRAINT
13 chk_gender CHECK (gender IN('M', 'F', 'O'));
14

```

Data Output Explain Messages **Messages** Notifications

ALTER TABLE

Query returned successfully in 166 msec.

```

online_store=# \d+ customer
                                         Table "public.customer"
 Column |          Type          | Collation | Nullable | Default | Storage | Stats target | Description
-----+---------------------+-----+-----+-----+-----+-----+-----+
username | character varying(20) |           | not null |           | extended |             |
pass      | character varying(65) |           | not null |           | extended |             |
first_name | character varying(100) |           |           |           | extended |             |
last_name | character varying(100) |           |           |           | extended |             |
dob       | date                |           |           |           | plain    |             |
gender    | character(1)         |           |           |           | extended |             |
phno      | bigint               |           |           |           | plain    |             |
email     | text                 |           |           | not null | extended |             |
address   | character varying(500) |           |           |           | extended |             |
Indexes:
  "customer_pkey" PRIMARY KEY, btree (username)
  "customer_email_key" UNIQUE CONSTRAINT, btree (email)
Check constraints:
  "chk_gender" CHECK (gender = ANY (ARRAY['M'::bpchar, 'F'::bpchar, 'O'::bpchar]))
Referenced by:
  TABLE "cart" CONSTRAINT "user_fk" FOREIGN KEY (username) REFERENCES customer(username)
Access method: heap

```

3. Cart

Attributes:

- **Cart id** (primary key, uniquely identifies a cart)
- Username - Foreign Key derived from customer table
- If_bought - A variable to indicate if a cart is open or closed, default value is FALSE

All attributes are non-transitively determined by the primary key - cart_id. Hence the relation is in **Boyce-Codd Normal Form**.

The screenshot shows a PostgreSQL query editor interface. At the top, it says "online_store/postgres@PostgreSQL 13". Below that is a toolbar with "Query Editor" and "Query History". The main area contains the following SQL code:

```
1 CREATE TABLE Cart (
2     cart_id SERIAL PRIMARY KEY NOT NULL,
3     username VARCHAR(10),
4     if_bought BOOL DEFAULT FALSE
5 );
6
7 ALTER TABLE Cart ADD CONSTRAINT
8 user_fk FOREIGN KEY(username) REFERENCES Customer(username);
9
```

Below the code, there are tabs for "Messages", "Data Output", "Explain", and "Notifications". The "Messages" tab is selected. It shows the message "ALTER TABLE". Underneath, it says "Query returned successfully in 142 msec."

The screenshot shows a PostgreSQL terminal window with the command "\d+ cart" entered. The output provides detailed information about the "public.cart" table:

Column	Type	Collation	Nullable	Default	Storage	Stats target	Description
cart_id	integer		not null	nextval('cart_cart_id_seq'::regclass)	plain		
username	character varying(10)				extended		
if_bought	boolean			false	plain		

Indexes:
"cart_pkey" PRIMARY KEY, btree (cart_id)

Foreign-key constraints:
"user_fk" FOREIGN KEY (username) REFERENCES customer(username)

Referenced by:
TABLE "bill" CONSTRAINT "cartid_fk" FOREIGN KEY (cart_id) REFERENCES cart(cart_id)
TABLE "cart_item" CONSTRAINT "cid_fk" FOREIGN KEY (cart_id) REFERENCES cart(cart_id)

Access method: heap

4. Cart_item

Attributes:

- Cart_item_id
- Cart_id - Foreign Key derived from Cart table
- Clothing_id - Foreign Key derived from Clothing table
- Clothing_cost
- Quantity

Here, a composite primary key of cart_item_id and cart_id is implemented to ensure no duplication.

All attributes are non-transitively determined by the primary key - cart_item_id. Hence the relation is in **Boyce-Codd Normal Form**.

The screenshot shows a PostgreSQL query editor window titled "online_store/postgres@PostgreSQL 13". The main area displays the following SQL code:

```
1 CREATE TABLE Cart_Item (
2     cart_item_id SERIAL NOT NULL,
3     cart_id SERIAL NOT NULL,
4     clothing_id INT NOT NULL,
5     clothing_cost DECIMAL,
6     quantity INTEGER
7 );
8
9 ALTER TABLE Cart_Item ADD CONSTRAINT
10 cid_pk PRIMARY KEY(cart_item_id, cart_id);
11
12 ALTER TABLE Cart_Item ADD CONSTRAINT
13 cid_fk FOREIGN KEY(cart_id)
14 REFERENCES Cart(cart_id);
15
16 ALTER TABLE Cart_Item ADD CONSTRAINT
17 clothingid_fk FOREIGN KEY(clothing_id)
18 REFERENCES Clothing(clothing_id);
```

Below the code, there are tabs for "Messages", "Data Output", "Explain", and "Notifications". The status bar at the bottom shows "ALTER TABLE" and "Query returned successfully in 164 msec."

```

online_store=# \d+ cart_item
                                         Table "public.cart_item"
   Column |  Type   | Collation | Nullable | Default           | Storage | Stats target | Description
---+-----+-----+-----+-----+-----+-----+-----+
cart_item_id | integer |          | not null | nextval('cart_item_cart_item_id_seq'::regclass) | plain   |          |
cart_id       | integer |          | not null | nextval('cart_item_cart_id_seq'::regclass)      | plain   |          |
clothing_id   | integer |          | not null |                               | plain   |          |
clothing_cost | numeric |          |          |                               | main    |          |
quantity      | integer |          |          |                               | plain   |          |
Indexes:
"cid_pk" PRIMARY KEY, btree (cart_item_id, cart_id)
Foreign-key constraints:
"cid_fk" FOREIGN KEY (cart_id) REFERENCES cart(cart_id)
"clothingid_fk" FOREIGN KEY (clothing_id) REFERENCES clothing(clothing_id)
Access method: heap

```

5. Bill

Attributes:

- **Bill_id** (primary key, uniquely identifies a bill)
- Cart_id - Foreign Key derived from Cart table
- Order_date
- Delivery_date
- Total_cost

All attributes are non-transitively determined by the primary key - cart_item_id. Hence the relation is in **Boyce-Codd Normal Form**.

online_store/postgres@PostgreSQL 13

Query Editor Query History

```
1 CREATE TABLE Bill (
2     bill_id SERIAL PRIMARY KEY NOT NULL,
3     cart_id BIGINT UNIQUE NOT NULL,
4     order_date DATE,
5     delivery_date DATE,
6     total_cost BIGINT
7 );
8
9 ALTER TABLE Bill ADD CONSTRAINT
10 cartid_fk FOREIGN KEY(cart_id)
11 REFERENCES Cart(cart_id);
```

Messages Data Output Explain Notifications

ALTER TABLE

Query returned successfully in 52 msec.

All relations in the database are Boyce-Codd Normalized.

```
online_store=# \d+ bill
                                         Table "public.bill"
   Column |  Type   | Collation | Nullable |          Default          | Storage | Stats target | Description
-----+-----+-----+-----+-----+-----+-----+-----+
bill_id | integer |           | not null | nextval('bill_bill_id_seq'::regclass) | plain   |             |
cart_id | bigint  |           | not null |                         | plain   |             |
order_date | date   |           |            |                         | plain   |             |
delivery_date | date   |           |            |                         | plain   |             |
total_cost | bigint  |           |            |                         | plain   |             |
Indexes:
 "bill_pkey" PRIMARY KEY, btree (bill_id)
 "bill_cart_id_key" UNIQUE CONSTRAINT, btree (cart_id)
Foreign-key constraints:
 "cartid_fk" FOREIGN KEY (cart_id) REFERENCES cart(cart_id)
Access method: heap
```

POPULATING THE TABLES

The Clothing table is populated with different clothing items. The remaining tables are populated in real time- as and when users interact with the web app.



A screenshot of a PostgreSQL Query Editor window. The title bar says 'online_store/postgres@PostgreSQL 13'. The main area shows a multi-line SQL query:

```
1  INSERT INTO Clothing VALUES
2  (
3      nextval('cloth_sequence'),
4      'Jeans', 'Men',
5      'Navy Blue; Skinny Fit; Stretchable',
6      'Levis', 1339, 105
7  ),
8  (
9      nextval('cloth_sequence'),
10     'T Shirt', 'Men',
11     'Yellow; Half Sleeve; Polyester',
12     'Reebok', 349, 242
13 ),
14 (
15     nextval('cloth_sequence'),
16     'Formal Shirt', 'Men',
17     'Black; Slim Fit; Cotton Blennd',
18     'English Navy', 599, 135
19 ),
```

Below the query, there are tabs for 'Messages', 'Data Output', 'Explain', and 'Notifications'. The 'Messages' tab is selected, showing the message 'INSERT 0 12'. At the bottom, it says 'Query returned successfully in 39 msec.'

SETTING UP PROCEDURES

A procedure update_qty() which takes in cart_id as argument, has been implemented. This procedure is executed after the bill is generated so as to reduce the amount of clothing items ordered from the clothing inventory.

The procedure also closes the particular open cart of the user, after the bill is generated.

The screenshot shows a PostgreSQL Query Editor interface. The title bar says "online_store/postgres@PostgreSQL 13". The main area contains the SQL code for creating the procedure:

```
1 CREATE OR REPLACE PROCEDURE update_qty
2     (cartid IN Cart.cart_id%TYPE)
3 LANGUAGE plpgsql
4 AS $$*
5 BEGIN
6     UPDATE Clothing SET available_quantity =
7         available_quantity - (SELECT quantity FROM
8             Cart_item WHERE Clothing.clothing_id = Cart_item.clothing_id
9                 AND cart_id = cartid)
10    WHERE Clothing.clothing_id = (SELECT clothing_id
11        FROM Cart_item WHERE Cart_item.clothing_id =
12            Clothing.clothing_id AND cart_id = cartid);
13
14    UPDATE Cart SET if_bought = TRUE WHERE
15        cart_id = cartid;
16
17    COMMIT;
18 END $$;
```

Below the code, there are tabs for "Messages", "Data Output", "Explain", and "Notifications". The "Messages" tab is active. It displays two lines of output:

```
NOTICE: type reference cart.cart_id%TYPE converted to integer
CREATE PROCEDURE
```

At the bottom, it says "Query returned successfully in 121 msec."

SETTING UP VIEWS

Views have been implemented in order to display the clothing collection as per the user choice - Men, Women or Kids

1. Men_clothing:

The screenshot shows a PostgreSQL Query Editor interface. The title bar says "online_store/postgres@PostgreSQL 13". The main area contains the following SQL code:

```
1 CREATE OR REPLACE VIEW men_clothing AS
2 SELECT * FROM Clothing
3 WHERE category = 'Men';
```

Below the code, the status message "CREATE VIEW" is displayed, followed by "Query returned successfully in 135 msec."

2. Women_clothing:

The screenshot shows a PostgreSQL Query Editor interface. The title bar says "online_store/postgres@PostgreSQL 13". The main area contains the following SQL code:

```
1 CREATE OR REPLACE VIEW women_clothing AS
2 SELECT * FROM Clothing
3 WHERE category = 'Women';
4
```

Below the code, the status message "CREATE VIEW" is displayed, followed by "Query returned successfully in 146 msec."

3. Kids_clothing:

The screenshot shows a PostgreSQL query editor interface. At the top, there is a connection bar with a gear icon and the text "online_store/postgres@PostgreSQL 13". Below the connection bar, there are two tabs: "Query Editor" (which is selected) and "Query History". The main area contains a numbered SQL query:

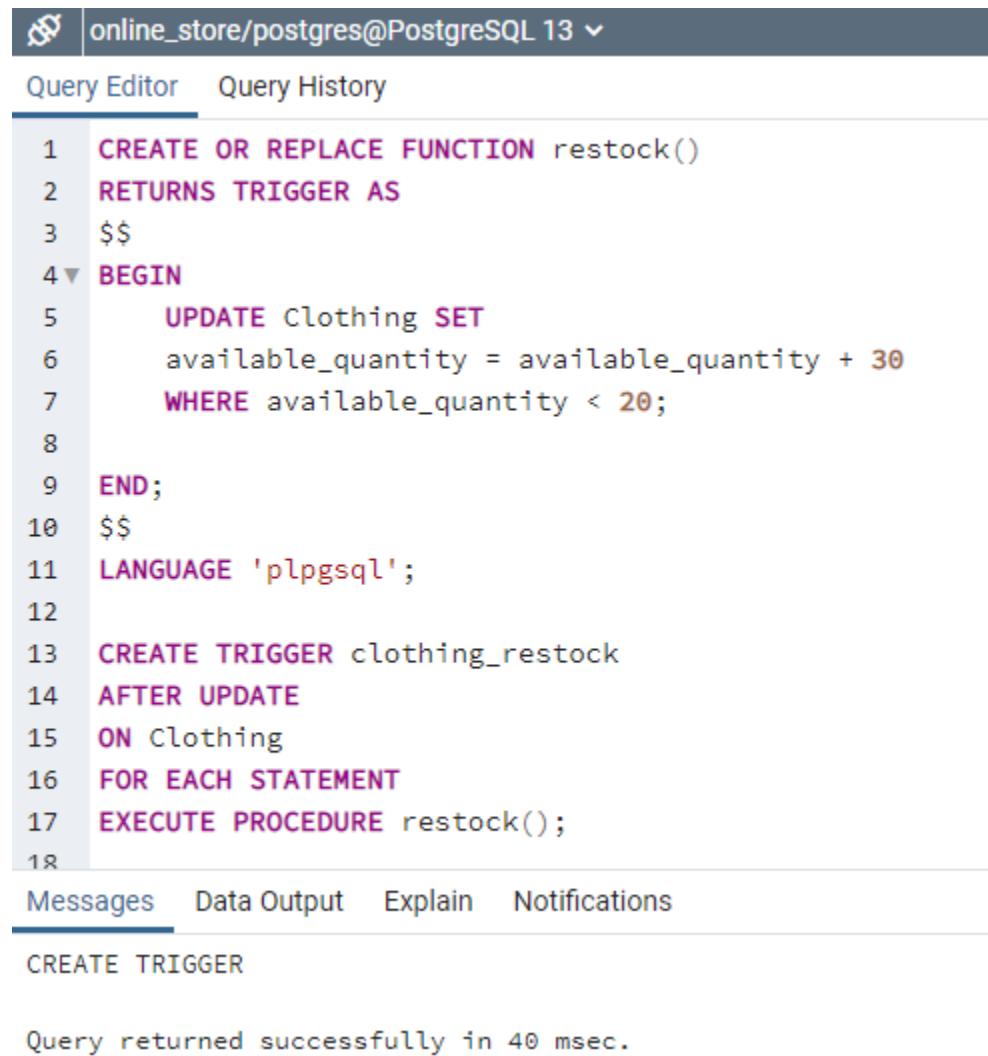
```
1 CREATE OR REPLACE VIEW kids_clothing AS
2 SELECT * FROM Clothing
3 WHERE category = 'Kids';
4
```

Below the query, there are four tabs: "Messages" (selected), "Data Output", "Explain", and "Notifications". A status message "CREATE VIEW" is displayed above the message log. The message log shows the successful execution of the query:

Query returned successfully in 120 msec.

SETTING UP TRIGGERS

A trigger has been set up so as to automatically restock the available quantity, once it goes below the minimum threshold.



The screenshot shows a PostgreSQL query editor interface. The title bar says 'online_store/postgres@PostgreSQL 13'. The main area contains the following SQL code:

```
1 CREATE OR REPLACE FUNCTION restock()
2 RETURNS TRIGGER AS
3 $$
4 BEGIN
5     UPDATE Clothing SET
6         available_quantity = available_quantity + 30
7     WHERE available_quantity < 20;
8
9 END;
10 $$;
11 LANGUAGE 'plpgsql';
12
13 CREATE TRIGGER clothing_restock
14 AFTER UPDATE
15 ON Clothing
16 FOR EACH STATEMENT
17 EXECUTE PROCEDURE restock();
18
```

Below the code, there are tabs for 'Messages' (which is selected), 'Data Output', 'Explain', and 'Notifications'. The message area shows:

CREATE TRIGGER

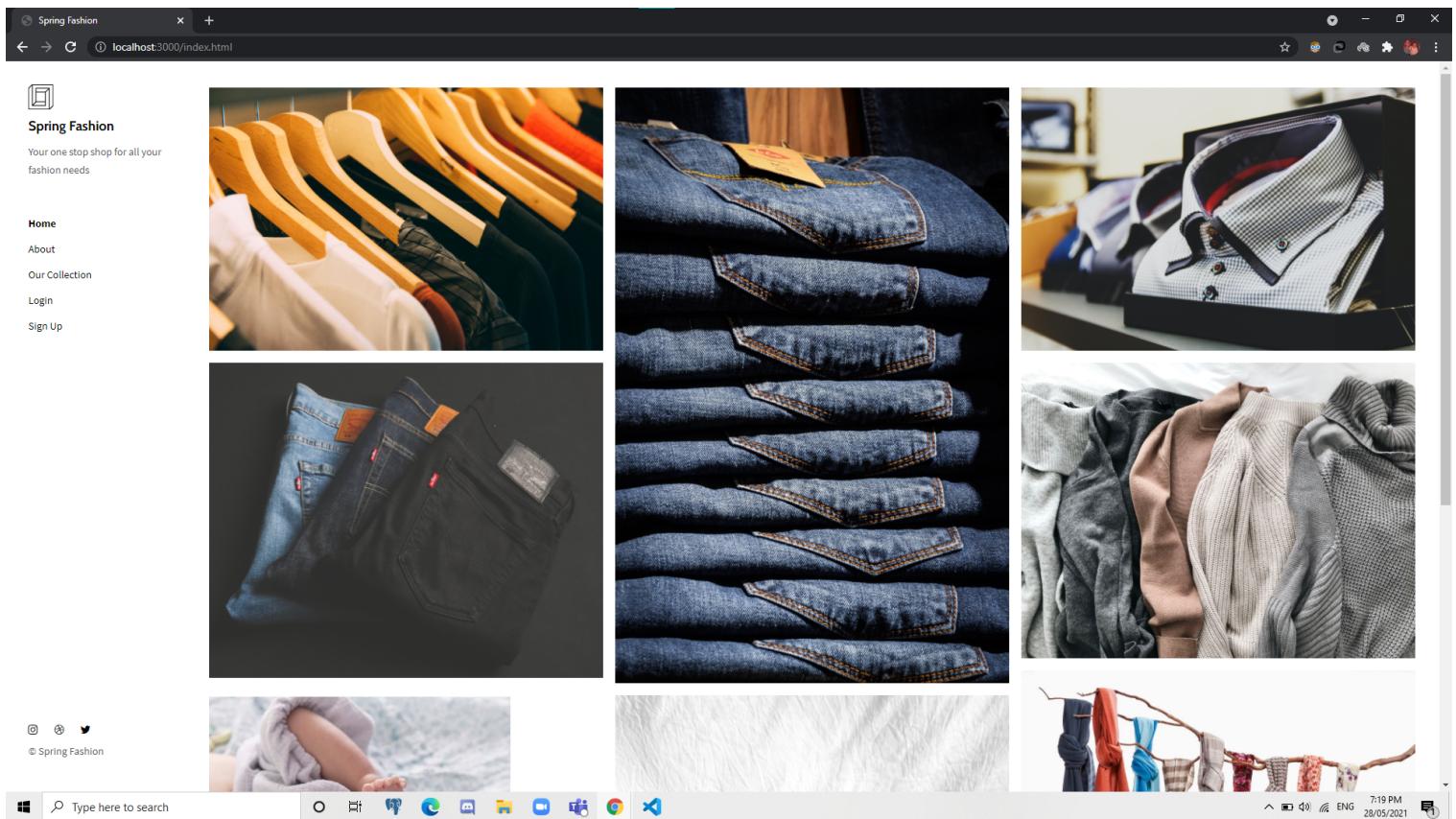
Query returned successfully in 40 msec.

RESULTS

1. First, we start the server so as to run the application.

```
safiy@LAPTOP-AQ9CTGNG MINGW64 ~/Documents/Safiya/textbooks/YEAR 2/SEM4/DBMS/PROJECT/DBMS-Project/main-app (main)
$ node server.js
App is running on port 3000
[]
```

2. Open <http://localhost:3000> to access the application



3.Create a new user

The screenshot shows a web browser window with the title "Spring Fashion". The address bar indicates the URL is "localhost:3000/signup.html". The page content is a sign-up form for a fashion website.

Spring Fashion
Your one stop shop for all your fashion needs

[Home](#) [About](#) [Our Collection](#) [Login](#) [Sign Up](#)

First Name: Sarah

Last Name: Anderson

Email: sarah@gmail.com

Date Of Birth: 11/08/1999

Phone Number: 8943211903

Address: Chennai, Tamil Nadu

Gender:

M

F

o

Username: sarah_11

Password:

I agree to the Privacy Policy

Already have an account? [Login](#)

[Sign Up](#)



© Spring Fashion

4. Login with the newly created profile

The screenshot shows a web browser window titled "Spring Fashion". The address bar displays "localhost:3000/login.html". The page content includes a logo with a square icon containing a stylized letter 'S', followed by the text "Spring Fashion" and a tagline "Your one stop shop for all your fashion needs". To the right, there are two input fields: "Username" containing "sarah_11" and "Password" containing "*****". Below these fields is a link "Don't have an account? [Sign Up](#)". On the left side of the page, there is a vertical menu with links: "Home", "About", "Our Collection", "Login" (which is highlighted in blue), and "Sign Up". A large teal "Login" button is centered below the password field.

5. The following page is displayed

The screenshot shows a web browser window titled "Spring Fashion". The address bar displays "localhost:3000/welcome". The page content includes a logo with a square icon containing a stylized letter 'S', followed by the text "Spring Fashion" and a tagline "Your one stop shop for all your fashion needs". On the right, a large teal button says "Welcome back! sarah_11". On the left, there is a vertical menu with links: "Home", "About", "Our Collection", "Logout" (which is highlighted in blue), "Your Cart", and "Past Orders". A teal "Continue Shopping" button is located near the bottom of the page.

6. Select Continue Shopping, the collection is displayed.

The screenshot shows a web browser window for 'Spring Fashion' at localhost:3000/collection. The page title is 'SHOP WITH US FROM THE BEST BRANDS AROUND THE WORLD!'. On the left, there's a sidebar with links: Home, About, Our Collection (selected), Logout, Your Cart, and Past Orders. The main content area displays a table of products:

SL NO	Clothing name	Category	Description	Seller	Price	Available Quantity	Buy Now
1	Shorts	Kids	Red; Regular Fit; Cotton	Jockey	499	231	Add to Cart
2	Jeans	Men	Navy Blue; Skinny Fit; Stretchable	Levis	1339	104	Add to Cart
3	Sweater	Men	Red Pullover; Long Sleeve; Cotton	HERE&NOW	524	357	Add to Cart
4	Dupatta	Women	White; Block Print; Pure Cotton	Taavi	439	204	Add to Cart
5	Kurti	Women	Black and Gold; Straight Kurti; Round Neck	Sangria	563	359	Add to Cart

7. Add the desired items to the cart.

The screenshot shows a web browser window for 'Spring Fashion' at localhost:3000/usercart. The page title is 'YOUR CART @sarah_11'. On the left, there's a sidebar with links: Home, About, Our Collection (selected), Logout, Your Cart, and Past Orders. The main content area displays a table of items in the cart:

SL NO	Clothing name	Category	Description	Seller	Price	Quantity	Delete
1	Shorts	Kids	Red; Regular Fit; Cotton	Jockey	499	1	Delete from Cart
2	Sweater	Men	Red Pullover; Long Sleeve; Cotton	HERE&NOW	524	1	Delete from Cart

At the bottom of the cart table, there are two buttons: 'Continue Shopping' (in a teal box) and 'Checkout' (in a red box).

8.Click Checkout to get the receipt.

Spring Fashion

Your one stop shop for all your fashion needs

@sarah_11, THANK YOU FOR SHOPPING WITH SPRING FASHION!

Receipt Number: 9
(please keep receipt number for future references)

SL NO	Clothing name	Category	Description	Seller	Price	Quantity	Total Price
1	Shorts	Kids	Red; Regular Fit; Cotton	Jockey	499	1	499
2	Sweater	Men	Red Pullover; Long Sleeve; Cotton	HERE&NOW	524	1	524

TOTAL = 1023

Order Date: 28-05-2021
Delivery Date: 07-06-2021*

*Delivery dates might vary due to the ongoing pandemic situation

[Shop More](#)

Additional functionalities:

1. View past orders by selecting the past orders option in the navigation bar:

Spring Fashion

Your one stop shop for all your fashion needs

@sarah_11, YOUR ORDER HISTORY

SL NO	Receipt Number	Order Date	Delivery Date	Total Cost
1	9	28-05-2021	07-06-2021	1023
2	10	28-05-2021	07-06-2021	14736

[View Receipt](#)

[View Receipt](#)

a) Select View Receipt to get the Receipt of the desired order

The screenshot shows a web browser window for 'Spring Fashion' at 'localhost:3000/receipt'. The page displays a receipt for order number 10, dated 28-05-2021, for user '@sarah_11'. The receipt includes a table of items, a total amount, and delivery information.

Spring Fashion
Your one stop shop for all your fashion needs

@sarah_11, RECEIPT FOR ORDER ON 28-05-2021

Receipt Number: 10

SL NO	Clothing name	Category	Description	Seller	Price	Quantity	Total Price
1	Saree	Women	Yellow Woven; Zari Border; Viscose Rayon	Pothys	3224	2	6448
2	Dupatta	Women	White; Block Print; Pure Cotton	Taavi	439	1	439
3	Sweater	Men	Red Pullover; Long Sleeve; Cotton	HERE&NOW	524	2	1048
4	Jeans	Women	Black; Wide Leg; Cotton	Kotty	899	1	899
5	Jeans	Men	Navy Blue; Skinny Fit; Stretchable	Levis	1339	2	2678
6	Saree	Women	Green Silk; Taping Border; Chiffon	Pothys	3224	1	3224

TOTAL = 14736

Order Date: 28-05-2021
Delivery Date: 07-06-2021*

*Delivery dates might vary due to the ongoing pandemic situation

Shop More

Back

2. Filter the clothing items displayed as per user choice:

Spring Fashion localhost:3000/collection

SHOP WITH US FROM THE BEST BRANDS AROUND THE WORLD!

Your one stop shop for all your fashion needs

Home About Our Collection Logout Your Cart Past Orders

SL NO	Clothing name	Category	Description	Seller	Price	Available Quantity	Buy Now
1	Kurti	Women	Black and Gold; Straight Kurti; Round Neck	Sangria	563	359	Add to Cart
2	Formal Shirt	Men	Black; Slim Fit; Cotton Blennd	English Navy	599	132	Add to Cart

Filter By

- Filter By
- MEN
- WOMEN
- KIDS

Spring Fashion localhost:3000/men

MEN'S CLOTHING

Your one stop shop for all your fashion needs

Home About Our Collection Logout Your Cart Past Orders

SL NO	Clothing name	Category	Description	Seller	Price	Available Quantity	Buy Now
1	Formal Shirt	Men	Black; Slim Fit; Cotton Blennd	English Navy	599	132	Add to Cart
2	T Shirt	Men	Yellow; Half Sleeve; Polyester	Reebok	349	240	Add to Cart
3	Jeans	Men	Navy Blue; Skinny Fit; Stretchable	Levis	1339	102	Add to Cart
4	Sweater	Men	Red Pullover; Long Sleeve; Cotton	HERE&NOW	524	354	Add to Cart

CONCLUSION

The ‘Online Clothing Store Management System’ application has been created and implemented successfully.

Source code for the same can be found at:

<https://github.com/sxfiy-a/DBMS-Project>

REFERENCES

- [Introduction · Bootstrap v5.0](#)
- [What is a good reason to use SQL views?](#)
- [PostgreSQL with Nodejs. Most of the applications need at some... | by Laurent Renard | DailyJS](#)
- [Postgres Stored Procedures with Input and Output Parameters SQL | ObjectRocket](#)
- [10 Examples of PostgreSQL Stored Procedures | EDB](#)
- [PostgreSQL Create View with Example](#)
- [Date in mmm yyyy format in postgresql](#)
- [Next pg.Client](#)
- [Geshan's Blog Node.js Postgresql tutorial: Build a simple REST API with Express step-by-step](#)
- [bcrypt](#)
- [Documentation: 9.5: Trigger Functions](#)
- [PostgreSQL Trigger: Create, Drop Example](#)