

DATABASE MANAGEMENT SYSTEMS - CS6106

PROJECT

ONLINE CLOTHING STORE MANAGEMENT SYSTEM

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

INDEX

SL NO	TOPIC	PAGE NUMBER
1	PROJECT ABSTRACT	1
2	FUNCTIONAL AND NON FUNCTIONAL REQUIREMENTS	2
3	ENTITY RELATIONSHIP DIAGRAM	3
4	RELATIONAL SCHEMA	4
5	USER INTERFACE DESIGN	5
6	CONNECTING TO THE DATABASE	7
7	SETTING UP RELATIONS	8
8	POPULATING THE TABLES	15
9	SETTING UP PROCEDURES	16
10	SETTING UP VIEWS	17
11	SETTING UP TRIGGERS	19
12	RESULTS	20
13	CONCLUSION	33
14	REFERENCES	34

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:

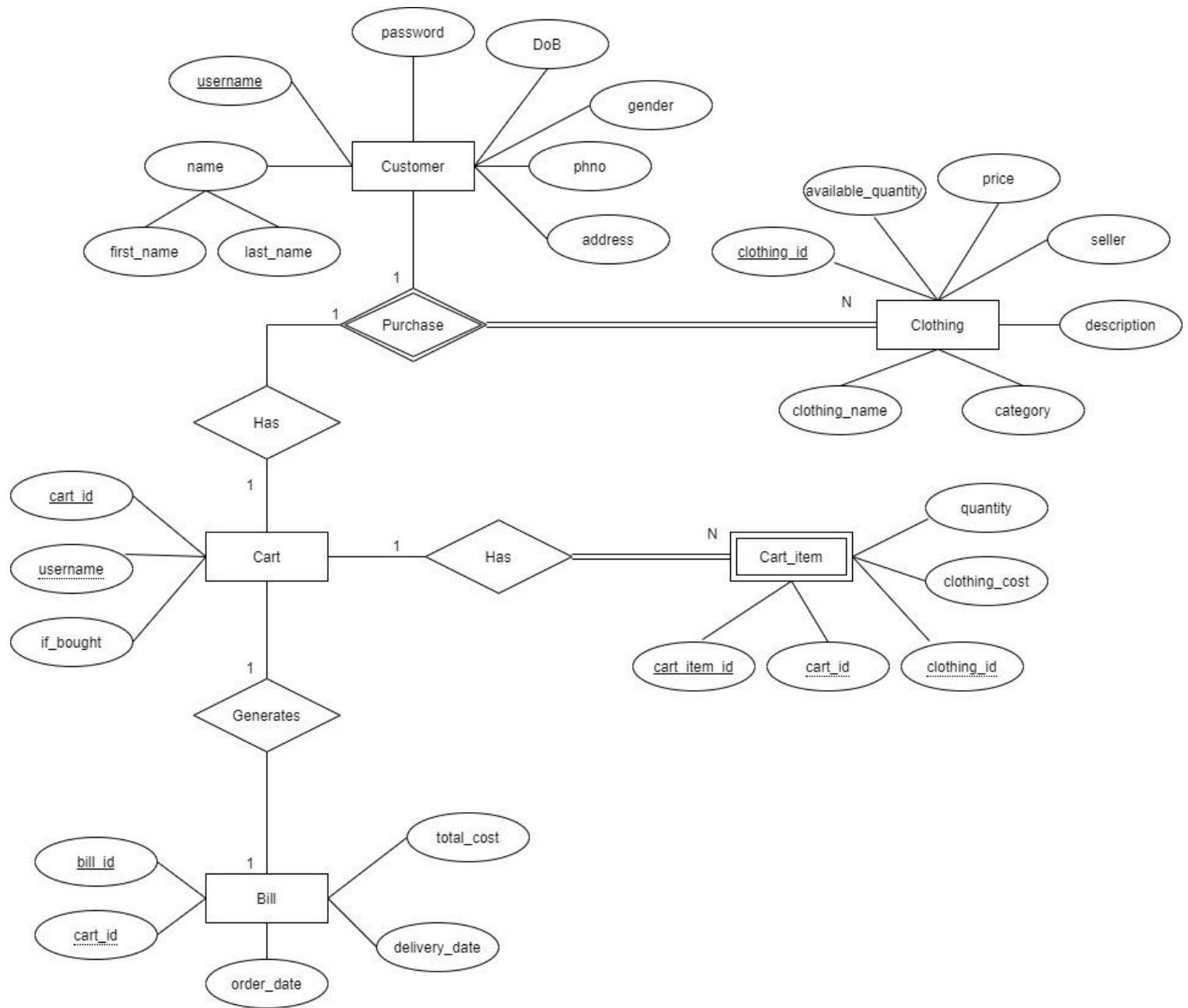
- Clothing details and available quantity
- User details
- Items currently in every user's cart
- Items ordered by every user
- Billing details of every user

Non Functional Requirements:

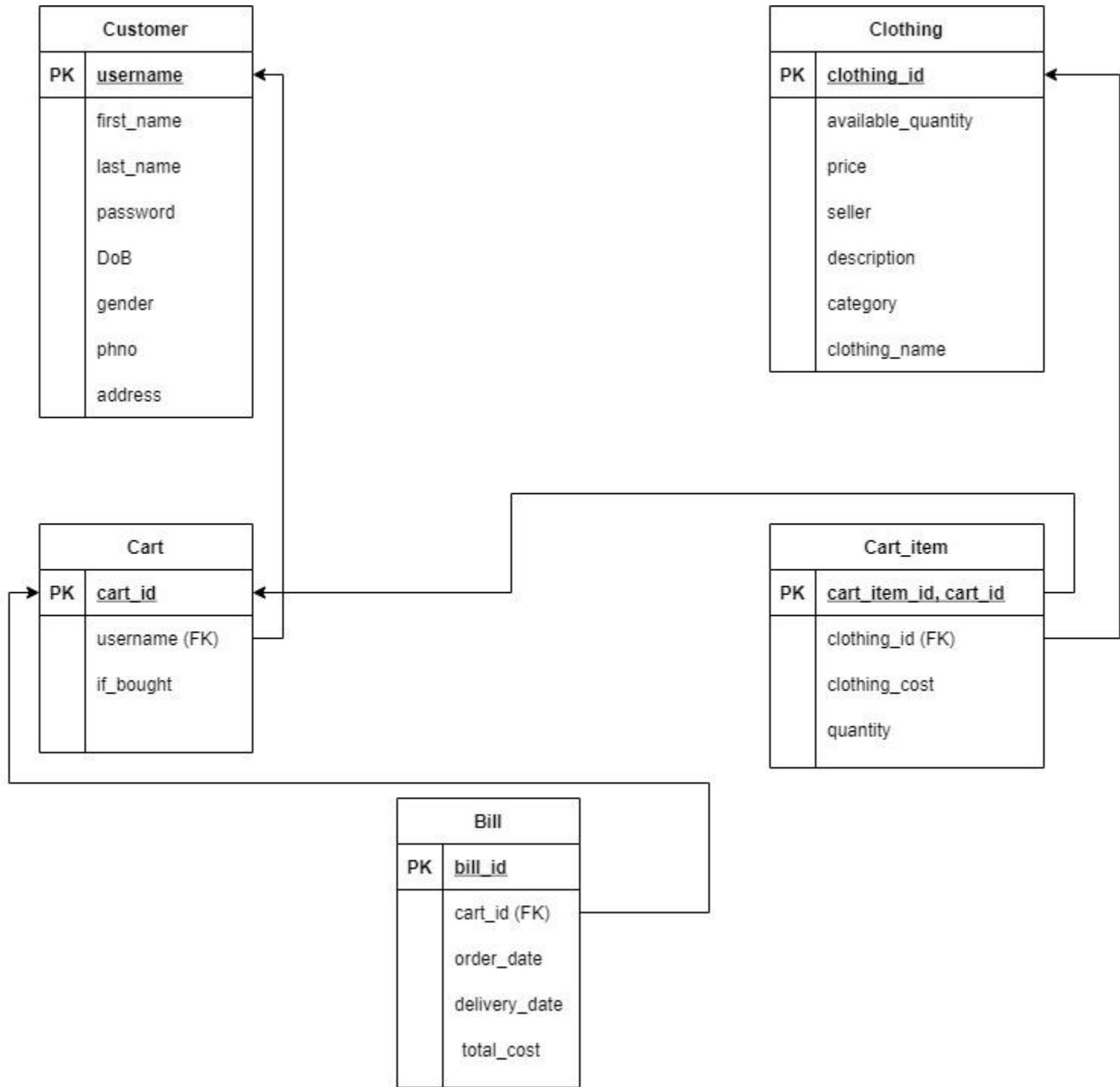
The system provides the following:

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

ENTITY RELATIONSHIP DIAGRAM



RELATIONAL SCHEMA



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     </header>
```

With the following end products:

The screenshot shows the homepage of the Spring Fashion website. On the left, there's a sidebar with a logo, navigation links for Home, About, Our Collection, Login, and Sign Up, and social media icons for Instagram, Facebook, and Twitter. The main content area features four large images: a rack of clothes, a stack of jeans, a row of shirts, and a pile of sweaters. Below these are two smaller images: a person's legs in jeans and a row of clothes hanging on a line.

The screenshot shows the 'About us' page of the Spring Fashion website. It features a historical black and white photograph of women working in a garment factory. To the right, there are three main sections: 'About us' (with a paragraph about the company's history and values), 'Creative Freedom' (with a paragraph about the sixteen métiers creating unique objects), and 'Responsible and sustainable development' (with a paragraph about the company's commitment to preserving resources). At the bottom, there's a section titled 'Springs around the World' with a paragraph about the company's global presence. The footer includes a copyright notice, social media links, and a system tray with battery status, language, and date.

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.

```
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".
online_store#
```

In NodeJS:

```
const connectionString = 'postgresql://postgres:060669@localhost:5432/online_store';
const { Client } = require('pg');

const client = new Client({
  connectionString
})
client.connect();
```

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 is a "Query Editor" tab, showing 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", and "Notifications". The "Messages" tab is selected, showing the message "ALTER TABLE". At the bottom, it says "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) |           |          |          | extended |          |
description | character varying(200) |           |          |          | extended |          |
seller | character varying(20) |           |          |          | extended |          |
price | integer |           |          | not null | plain   |          |
available_quantity | integer |           |          |          | plain   |          |
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 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. Under "Data Output", the message "ALTER TABLE" is shown. At the bottom, it says "Query returned successfully in 142 msec."

```
online_store=# \d+ cart
                                         Table "public.cart"
   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 "Data Output" tab is currently selected, showing the message "ALTER TABLE". At the bottom, it says "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

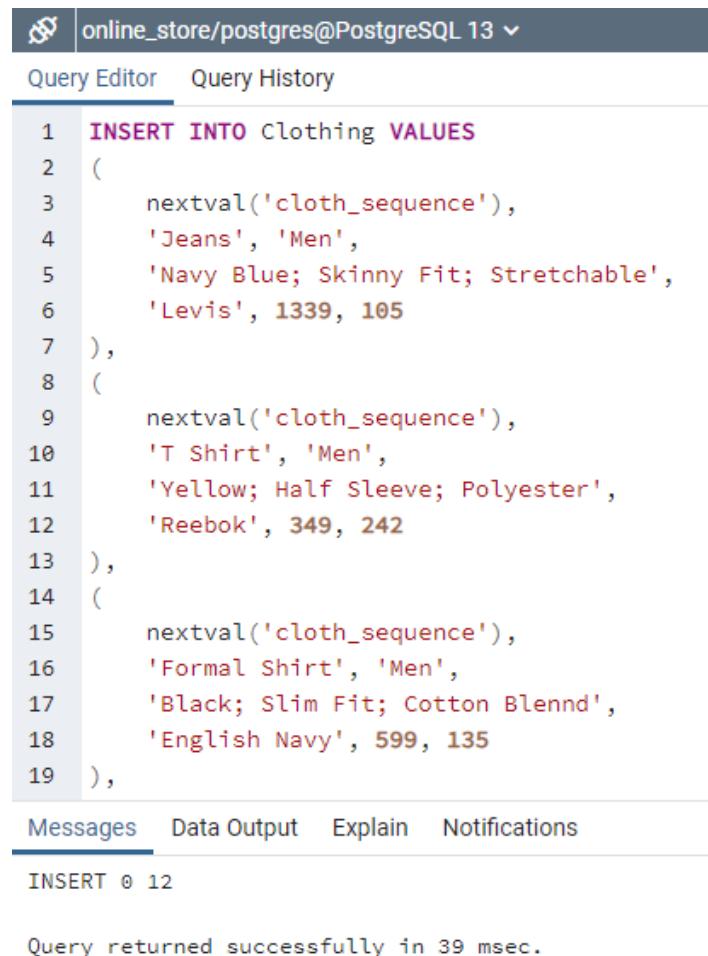
Query returned successfully in 52 msec.

```
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
```

All relations in the database are Boyce-Codd Normalized.

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 output:

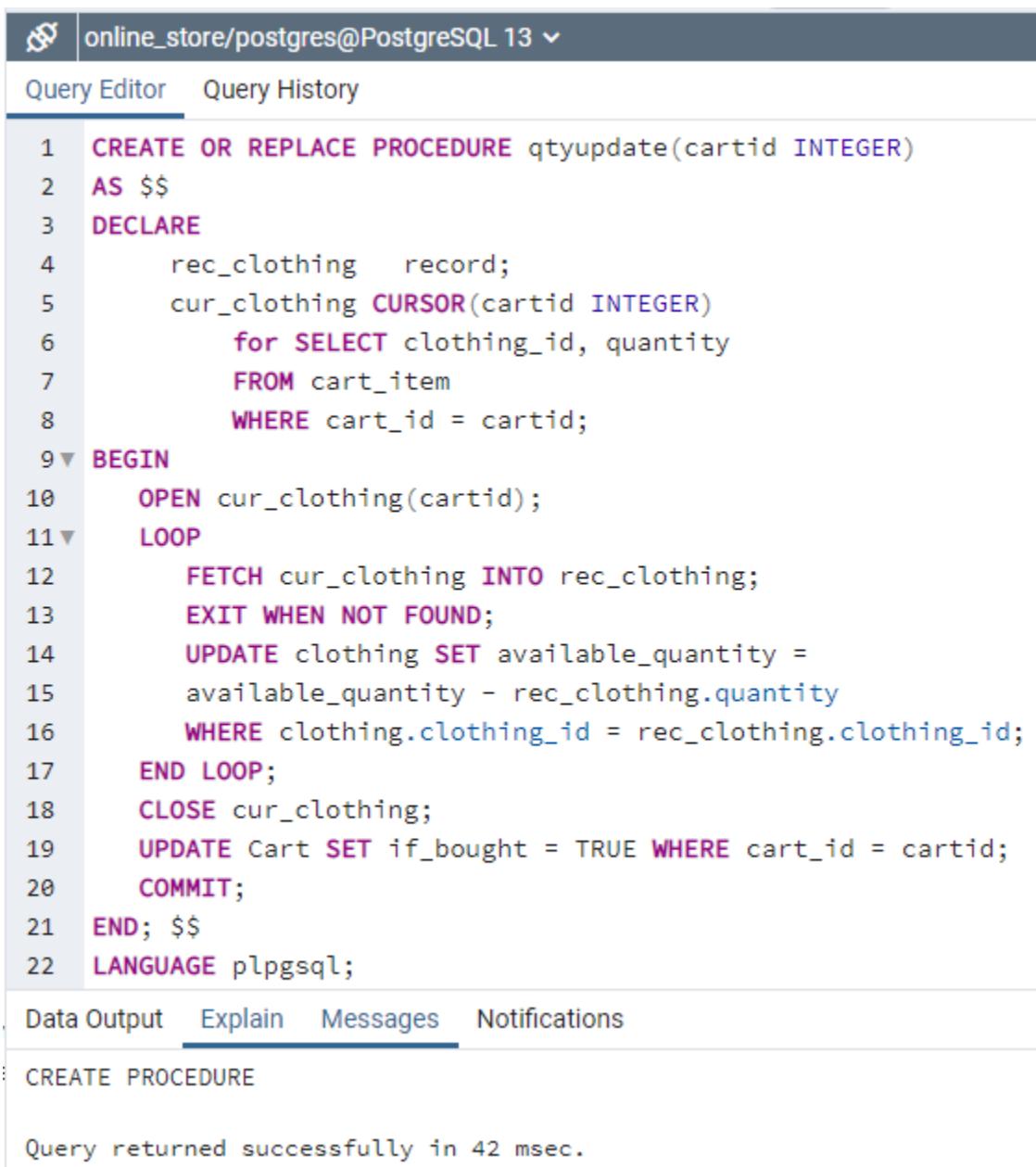
```
INSERT 0 12
```

At the bottom, it says 'Query returned successfully in 39 msec.'

SETTING UP PROCEDURES

A procedure update_qty() which takes in cartid 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 is a "Query Editor" tab, showing the following PL/pgSQL code:

```
1 CREATE OR REPLACE PROCEDURE qtyupdate(cartid INTEGER)
2 AS $$ 
3 DECLARE
4     rec_clothing record;
5     cur_clothing CURSOR(cartid INTEGER)
6         FOR SELECT clothing_id, quantity
7             FROM cart_item
8             WHERE cart_id = cartid;
9 BEGIN
10    OPEN cur_clothing(cartid);
11 LOOP
12    FETCH cur_clothing INTO rec_clothing;
13    EXIT WHEN NOT FOUND;
14    UPDATE clothing SET available_quantity =
15        available_quantity - rec_clothing.quantity
16        WHERE clothing.clothing_id = rec_clothing.clothing_id;
17 END LOOP;
18 CLOSE cur_clothing;
19 UPDATE Cart SET if_bought = TRUE WHERE cart_id = cartid;
20 COMMIT;
21 END; $$ 
22 LANGUAGE plpgsql;
```

Below the code, there are tabs for "Data Output", "Explain", "Messages", and "Notifications". The "Messages" tab is currently selected. The message area shows:

CREATE PROCEDURE

Query returned successfully in 42 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 script:

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

Below the script, 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 PL/pgSQL code:

```
1 CREATE OR REPLACE FUNCTION restock()
2 RETURNS TRIGGER
3 LANGUAGE PLPGSQL
4 AS
5 $$
6 BEGIN
7     UPDATE Clothing SET
8         available_quantity = available_quantity + 30
9     WHERE available_quantity < 20;
10    RETURN NEW;
11 END;
12 $$
```

Below the code, there are tabs for "Data Output", "Explain", "Messages", and "Notifications". The "Messages" tab is selected, showing the message "CREATE FUNCTION". At the bottom, it says "Query returned successfully in 206 msec."

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 TRIGGER clothing_restock
2 AFTER INSERT
3 ON Bill
4 FOR EACH STATEMENT
5 EXECUTE PROCEDURE restock();
```

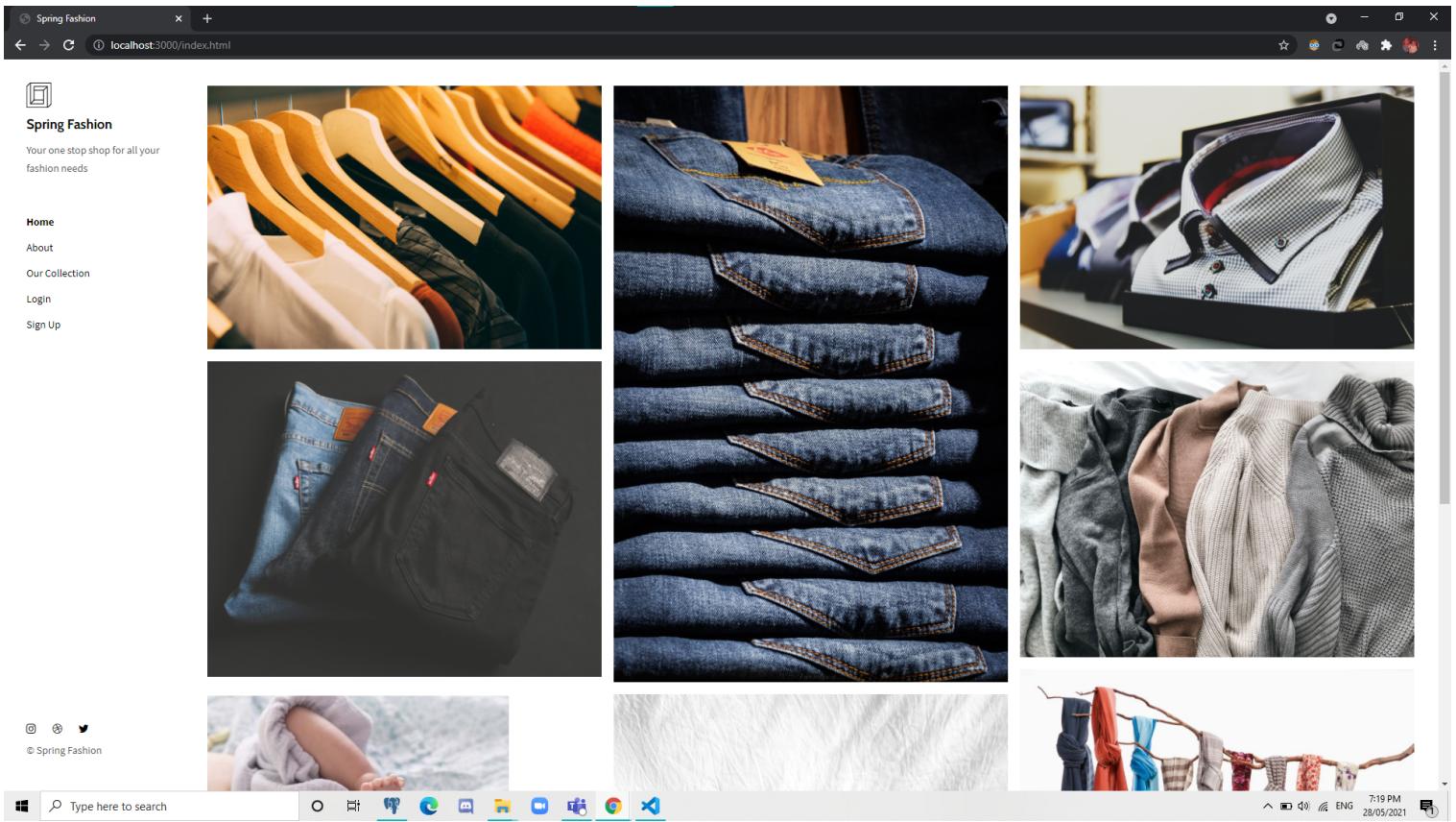
Below the code, there are tabs for "Data Output", "Explain", "Messages", and "Notifications". The "Messages" tab is selected, showing the message "CREATE TRIGGER". At the bottom, it says "Query returned successfully in 44 msec."

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

Spring Fashion X +
localhost:3000/signup.html



Spring Fashion
Your one stop shop for all your fashion needs

Home First Name
About Sarah
Our Collection Last Name
Login Anderson
Sign Up 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

On the server side, the following query is written to add the user information to the Customer table, where the password is hashed, after checking whether the username or email already exists:

```
app.post('/signup', (req, res) => {
  const {
    firstname,
    lastname,
    dob,
    phno,
    address,
    gender,
    email,
    username,
    password
  } = req.body;

  client
    .query('SELECT * FROM Customer WHERE email = $1', [email],
    (err, result) => {
      if(result.rows.length !== 0)
      {
        res.status(400).sendFile(path.join(__dirname, 'public', 'emailexists.html'));
        return;
      }
      // res.end();
    });
}

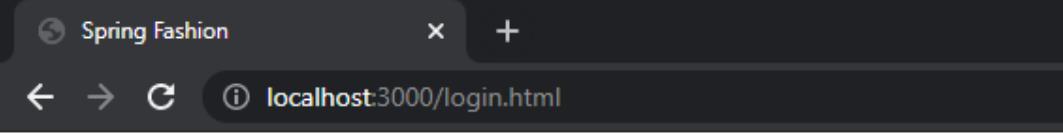
client
  .query('SELECT * FROM Customer WHERE username = $1', [username],
  (err, result) => {
    if(result.rows.length !== 0)
    {
      res.status(400).sendFile(path.join(__dirname, 'public', 'usernameexists.html'));
      return;
    }
    // res.end();
  });
}
```

```

bcrypt.hash(password, saltRounds, (err, hash) => {
  client
    .query('INSERT INTO Customer VALUES($1, $2, $3, $4, $5, $6, $7, $8, $9)',
    [username, hash, firstname, lastname, dob, gender, phno, email, address],
    (err, result) => {
      if(err) {
        console.log(err);
        res.sendStatus(500);
        return;
      }
      res.status(201).sendFile(path.join(__dirname, 'public', 'created.html'));
    });
});

```

4. Login with the newly created profile



The screenshot shows a web browser window with the following details:

- Title Bar:** Spring Fashion
- Address Bar:** localhost:3000/login.html
- Page Content:**
 - Logo:** A stylized square icon.
 - Page Title:** Spring Fashion
 - Tagline:** Your one stop shop for all your fashion needs
 - Login Form:**
 - Username: sarah_11
 - Password: (Redacted)
 - Text:** Don't have an account? [Sign Up](#)
 - Navigation Links:** Home, About, Our Collection, Login, Sign Up
 - Submit Button:** A large teal button labeled "Login".

The entered username and password is checked with the existing users from the Customer table using the following query:

```
app.post('/login', (req, res) => {
  const { username, password } = req.body;
  if(username && password) {

    client
      .query('SELECT * FROM Customer WHERE username = $1', [username],
      (err, result) => {
        if(err) {
          console.log(err);
          res.sendStatus(500);
          return;
        }
        if(result.rows.length !== 0) {
          bcrypt.compare(password, result.rows[0]['pass'], (error, result1) => {
            if(result1 === true)
              { req.session.loggedin = true;
                req.session.username = username;
                res.redirect('/welcome');
              }
            else
              res.redirect('/loginerror');
          });
        } else res.sendFile(path.join(__dirname, 'public', 'invaliduser.html'));
      });
  } else {
    res.send('<script>alert("INVALID")</script>');
  }
});
```

5. The following page is displayed

The screenshot shows a web browser window titled "Spring Fashion" with the URL "localhost:3000/welcome". The page content includes:

- A logo icon.
- The text "Welcome back! sarah_11".
- A sub-header "Spring Fashion".
- A tagline "Your one stop shop for all your fashion needs".
- A teal-colored button labeled "Continue Shopping".
- A sidebar with links: Home, About, Our Collection, Logout, Your Cart, and Past Orders.

6. Select Continue Shopping, the collection is displayed.

The screenshot shows a web browser window titled "Spring Fashion" with the URL "localhost:3000/collection". The page content includes:

- A logo icon.
- The text "SHOP WITH US FROM THE BEST BRANDS AROUND THE WORLD!".
- A sub-header "Spring Fashion".
- A tagline "Your one stop shop for all your fashion needs".
- A "Filter By" dropdown menu.
- A table displaying product information:

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

The collection is displayed from the Clothing table by executing the following query:

```
//To display clothing table
app.get('/collection', (req, res) => {
  client
    .query('SELECT * FROM CLOTHING',
  (err, result) => {
    if(err) {
      console.log(err);
      res.sendStatus(500);
      return;
    }
    if(result.rows.length > 0) {
      if(req.session.loggedin === true)
        res.render('collection_user.ejs', {result: result});
      else
        res.render('collection_guest.ejs', {result: result});
    }
  })
})
```

7. Add the desired items to the cart.

The screenshot shows a web browser window with the title "Spring Fashion". The URL in the address bar is "localhost:3000/usercart". The page content is as follows:

YOUR CART @sarah_11

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

Spring Fashion
Your one stop shop for all your fashion needs

[Home](#) [About](#) [Our Collection](#) [Logout](#) [Your Cart](#) [Past Orders](#)

Continue Shopping

Checkout

The user's cart is displayed from the Cart_item table by executing the following query:

```
var qry = "SELECT cart_item.clothing_id, clothing_name, category, description, seller, price, quantity, price*quantity AS total \
FROM Clothing, Cart_item \
WHERE Clothing.clothing_id = Cart_item.clothing_id \
AND Cart_item.cart_id = (SELECT cart_id FROM Cart \
WHERE if_bought = FALSE AND username = $1)";
client
  .query(qry,
  [req.session.username],
  (err, result) => {
    if(err) {
      console.log(err);
      res.sendStatus(500);
      return;
    }
    if(result.rows.length === 0)
      res.render('emptycart.ejs', {res: req.session.username});
    else {
      let qry1 = "SELECT SUM(price*quantity) AS sum FROM Clothing, Cart_item WHERE Clothing.clothing_id = Cart_item.clothing_id " +
                 "AND Cart_item.cart_id = (SELECT cart_id FROM Cart WHERE if_bought = FALSE AND username = $1)"
      client
        .query(qry1,
        [req.session.username],
        (err, total) => {
          if(err) {
            console.log(err);
            res.sendStatus(500);
            return;
          }
          res.render('usercart.ejs', {result: result, res: req.session.username, total: total.rows[0]['sum']});
        })
    }
  })
})
```

8. Click Checkout to get the receipt.

The screenshot shows a web browser window titled "Spring Fashion" with the URL "localhost:3000/bill". The page content includes a header with the store logo and name, a message to the user (@sarah_11), and a receipt summary. The receipt details the purchase of two items: Shorts and a Sweater, with a total price of 1023. It also provides delivery information and a note about delivery dates. A "Shop More" button is at the bottom.

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
Home	1	Shorts	Kids	Red; Regular Fit; Cotton	Jockey	499	1	499
About	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

The bill is calculated by executing the following queries:

```
client
  .query('SELECT cart_id FROM Cart WHERE if_bought = FALSE and username = $1',
  [req.session.username],
  (err, result) => {
    if(err) {
      console.log(err);
      res.sendStatus(500);
      return;
    }
    cartid = result.rows[0]['cart_id'];
    client
      .query('SELECT SUM(price*quantity) AS total FROM clothing, cart_item WHERE clothing.clothing_id = cart_item.clothing_id and cart_item.cart_id = $1',
      [cartid],
      (err, result1) => {
        if(err) {
          console.log(err);
          res.sendStatus(500);
          return;
        }
        client
          .query('INSERT INTO Bill(cart_id, order_date, delivery_date, total_cost) VALUES($1, $2, $3, $4)',
          [cartid, moment().format('DD-MM-YYYY'), moment().add(10, 'days').format('DD-MM-YYYY'), result1.rows[0]['total']],
          (err, result) => {
            if(err) {
              console.log(err);
              res.sendStatus(500);
              return;
            }
            client
              .query('CALL qtyupdate($1)',
              [cartid],
              (err, result) => {
                if(err) {
                  console.log(err);
                  res.sendStatus(500);
                  return;
                }
              })
            })
          res.redirect('/bill');
        })
      })
    })
  })
})
```

```
app.get('/bill', (req, res) => {
  if(req.session.loggedin !== true)
    res.redirect('/logintocontinue');
  let qry = "SELECT bill_id, TO_CHAR(order_date, 'DD-MM-YYYY') AS order_date, TO_CHAR(delivery_date, 'DD-MM-YYYY') AS delivery_date, " +
    "cart_item.clothing_id, clothing_name, category, description, ' +
    'seller, price, quantity, price*quantity AS total, total_cost ' +
    'FROM Bill, Clothing, Cart_item ' +
    'WHERE Clothing.clothing_id = Cart_item.clothing_id ' +
    'AND Cart_item.cart_id = $1 AND Bill.cart_id = $1';
  client
    .query(qry,
    [cartid],
    (err, results) => {
      if(err) {
        console.log(err);
        res.sendStatus(500);
        return;
      }
      res.render('bill.ejs', {result: results, res: req.session.username});
    })
})
```

Additional functionalities:

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

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

The past orders are displayed by executing the following query:

```
//To get past orders of user
app.get('/pastorders', (req, res) => {
  if(req.session.loggedin !== true)
    res.redirect('/logintocontinue');
  let qry = "SELECT bill_id, TO_CHAR(order_date, 'DD-MM-YYYY') AS order_date, TO_CHAR(delivery_date, 'DD-MM-YYYY') AS delivery_date, " +
            "total_cost, cart_id FROM Bill WHERE cart_id IN (SELECT cart_id FROM Cart WHERE if_bought = TRUE AND username = $1)";
  client
    .query(qry,
    [req.session.username],
    (err, result) => {
      if(err) {
        console.log(err);
        res.sendStatus(500);
        return;
      }
      res.render('pastorders.ejs', {result: result, res: req.session.username});
    })
})
```

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 title is '@sarah_11, RECEIPT FOR ORDER ON 28-05-2021'. It displays a receipt with the receipt number 10, a table of items ordered, and delivery details.

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

The particular receipt is displayed by executing the following query:

```
app.get('/receipt', (req, res) => {
  if(req.session.loggedin !== true)
    res.redirect('/logintocontinue');
  let qry = "SELECT bill_id, TO_CHAR(order_date, 'DD-MM-YYYY') AS order_date, TO_CHAR(delivery_date, 'DD-MM-YYYY') AS delivery_date, " +
            'cart_item.clothing_id, clothing_name, category, description, ' +
            'seller, price, quantity, price*quantity AS total, total_cost ' +
            'FROM Bill, Clothing, Cart_item ' +
            'WHERE Clothing.clothing_id = Cart_item.clothing_id ' +
            'AND Cart_item.cart_id = $1 AND Bill.cart_id = $1';
  client
    .query(qry,
    [cartid],
    (err, results) => {
      if(err) {
        console.log(err);
        res.sendStatus(500);
        return;
      }
      res.render('receipt.ejs', {result: results, res: req.session.username});
    });
})
```

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

The screenshot shows a web browser window with the URL localhost:3000/collection. The page title is "SHOP WITH US FROM THE BEST BRANDS AROUND THE WORLD!". On the left, there is a sidebar with links: Home, About, Our Collection, Logout, Your Cart, and Past Orders. The main content area displays a table of clothing items. A dropdown menu titled "Filter By" is open, showing options: MEN, WOMEN, and KIDS. The table has columns: SL NO, Clothing name, Category, Description, Seller, Price, Available Quantity, and Buy Now. Two items are listed:

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	<button>Add to Cart</button>
2	Formal Shirt	Men	Black; Slim Fit; Cotton Blennd	English Navy	599	132	<button>Add to Cart</button>

The screenshot shows a web browser window with the URL localhost:3000/men. The page title is "MEN'S CLOTHING". The sidebar and table structure are identical to the first screenshot. The dropdown menu "Filter By" is closed. The table lists four men's clothing items:

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	<button>Add to Cart</button>
2	T Shirt	Men	Yellow; Half Sleeve; Polyester	Reebok	349	240	<button>Add to Cart</button>
3	Jeans	Men	Navy Blue; Skinny Fit; Stretchable	Levis	1339	102	<button>Add to Cart</button>
4	Sweater	Men	Red Pullover; Long Sleeve; Cotton	HERE&NOW	524	354	<button>Add to Cart</button>

The filters are implemented by using the following query:

```
//to display men's clothing - view
app.get('/men', (req, res) => {
    client
        .query('SELECT * FROM men_clothing',
        (err, result) => {
            if(err) {
                console.log(err);
                res.sendStatus(500);
                return;
            }
            if(req.session.loggedin !== true)
                res.render('men_guest.ejs', {result: result});
            else
                res.render('men_user.ejs', {result: result});
        })
    });
}

//to display women's clothing - view
app.get('/women', (req, res) => {
    client
        .query('SELECT * FROM women_clothing',
        (err, result) => {
            if(err) {
                console.log(err);
                res.sendStatus(500);
                return;
            }
            if(req.session.loggedin !== true)
                res.render('women_guest.ejs', {result: result});
            else
                res.render('women_user.ejs', {result: result});
        })
    );
}

//to display kids' clothing - view
app.get('/kids', (req, res) => {
    client
        .query('SELECT * FROM kids_clothing',
        (err, result) => {
            if(err) {
                console.log(err);
                res.sendStatus(500);
                return;
            }
            if(req.session.loggedin !== true)
                res.render('kids_guest.ejs', {result: result});
            else
                res.render('kids_user.ejs', {result: result});
        })
    );
});
```

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](#)