MINI PROJECT REPORT

1. SHORT DESCRIPTION AND SCOPE OF THE PROJECT

Description :

This is a mini project on textile store management system, where a complete database consists of the stores(branches), items in each store, customer records, employees present and their respective managers too. In addition, we also have a large number of items with varieties of categories and collections. Also, we have a mapping of customers buying the respective products and stores with the stock of particular items. We also have the record of how the items come to the store from the suppliers, the mode of travelling and tracking the order with the date. A couple of triggers, functions, procedures and cursors are implemented in order to make the working of database in a smooth and sophisticated way. Views are also implemented to show the dynamic changes that occurs in the database as a whole. This database handles for 4 branches of the Textile Store and this essence are completely felt, when we perform some of join, aggregate and set operations. Overall, we get to see a miniature of real – time textile store management.

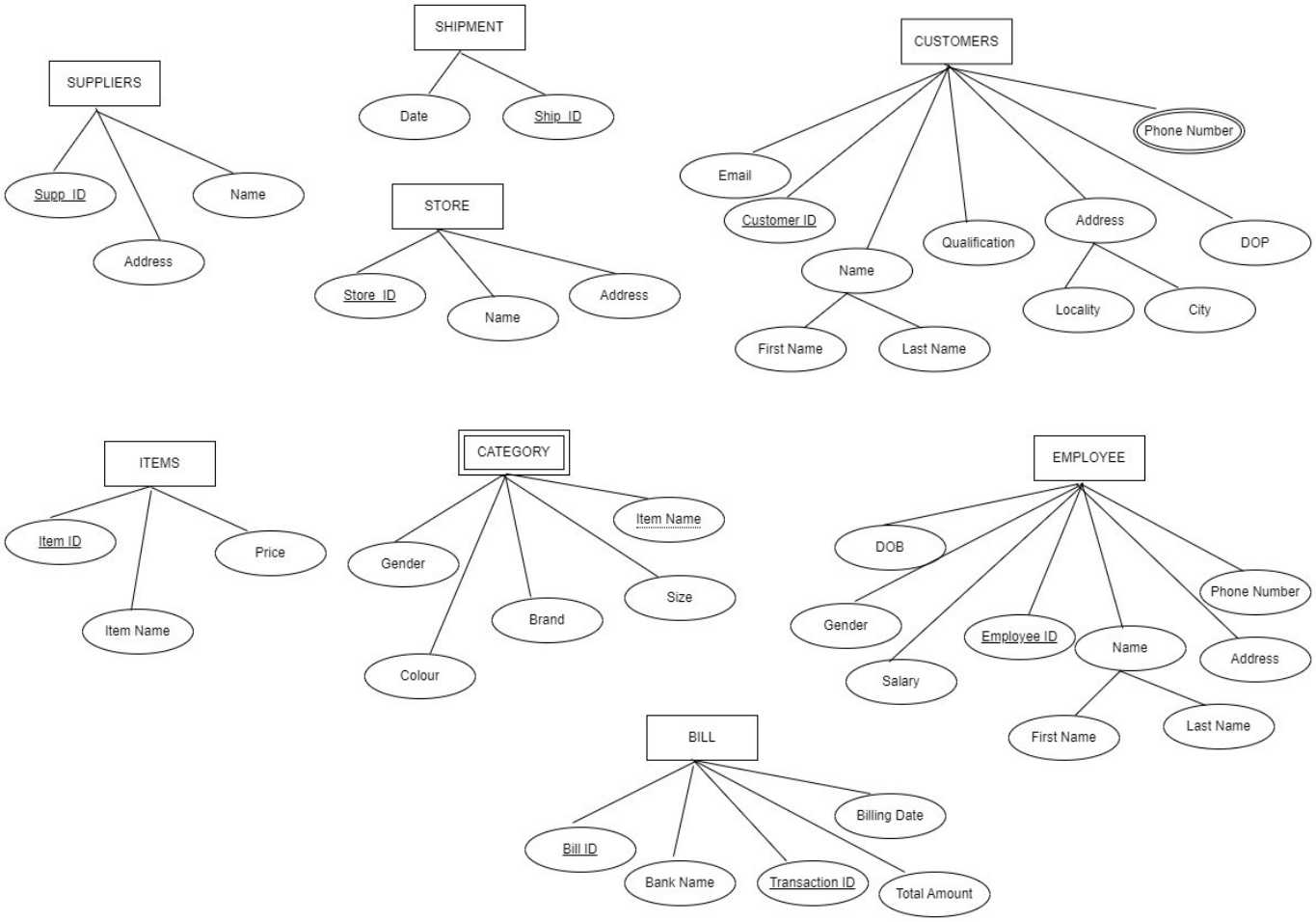
Scope :

This mini project aims to deliver the real – time essence of textile management, which is prevalent and widely developed by many software enthusiasts all over the world. The relational database is quiet deeply modelled and revealed in a very schematic way. The output of this DBMS is put out in a technical sense to bring out the effectiveness and usefulness of such a model and system. Also, a front end for this project using streamlit is made in order to access and perform CRUD operations in simple way. This also creates a high reachability of the software. Hence these are few of the scopes of this project. Though the challenges to build the queries are high, MySQL

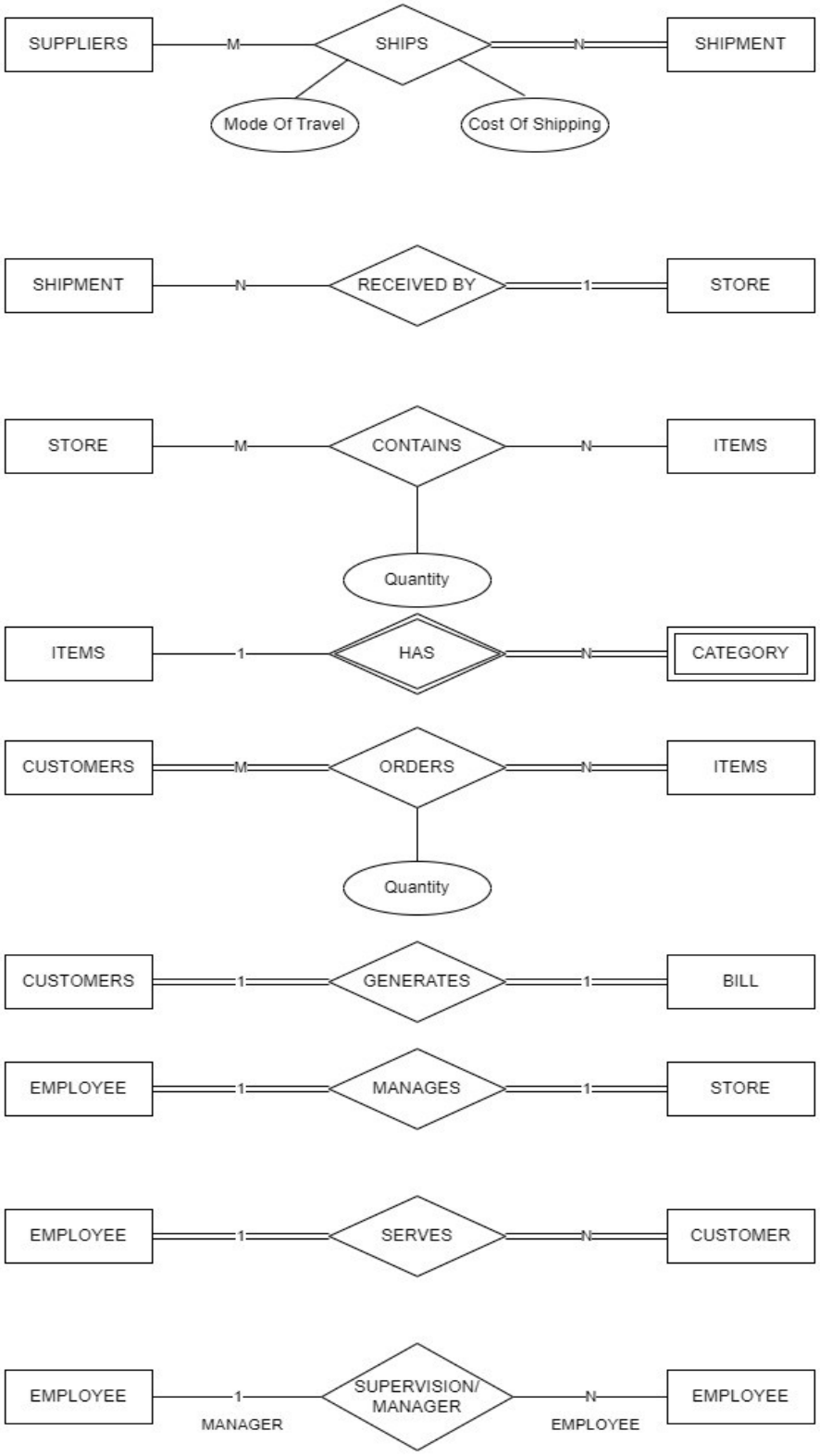
rather supports all these in a sophistic way with the help of the reserved keywords and clauses provided by them.

2. E-R DIAGRAM

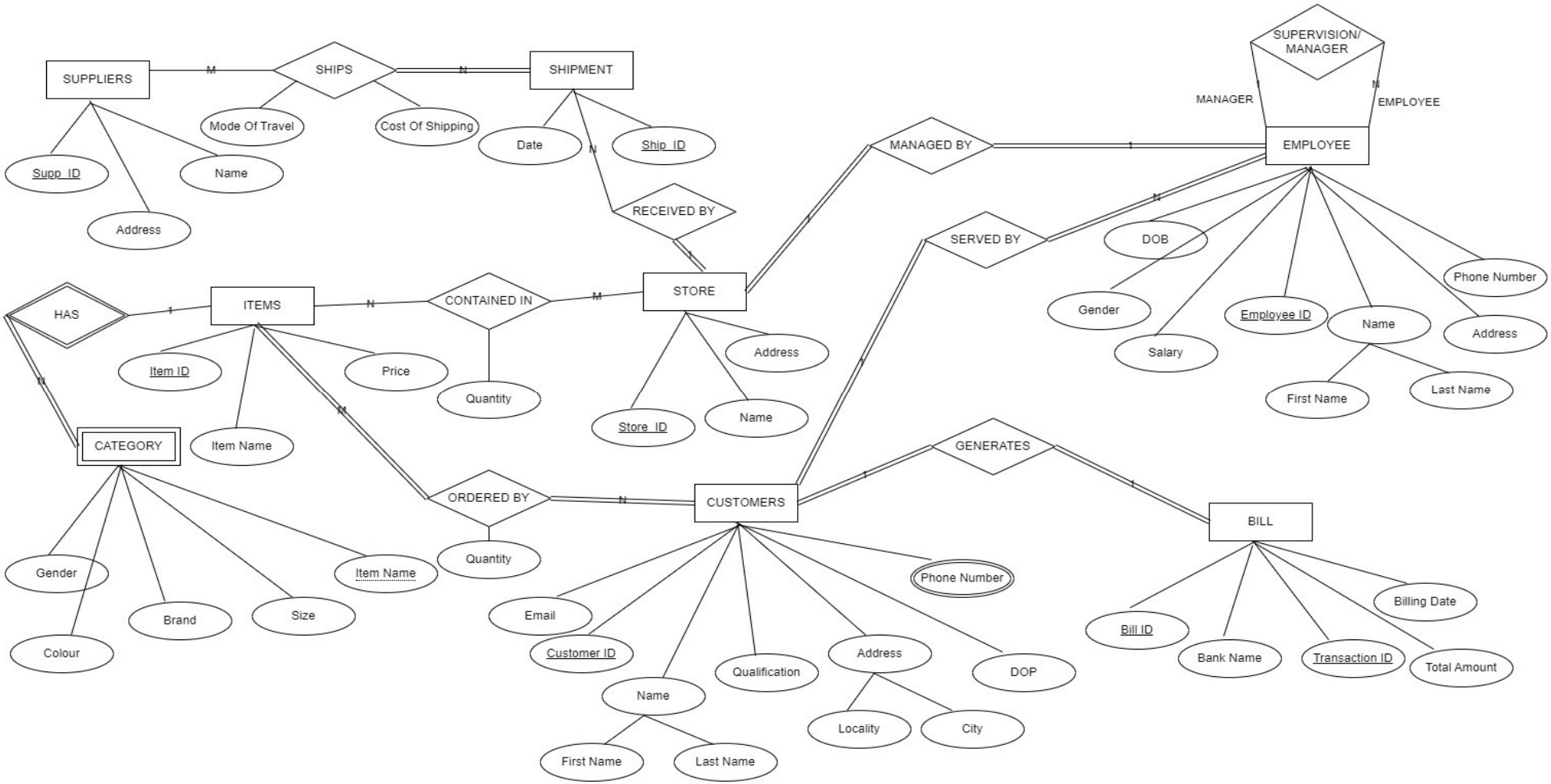
a. Entity – Attribute Diagram :



b. Entity – Relations Diagram :

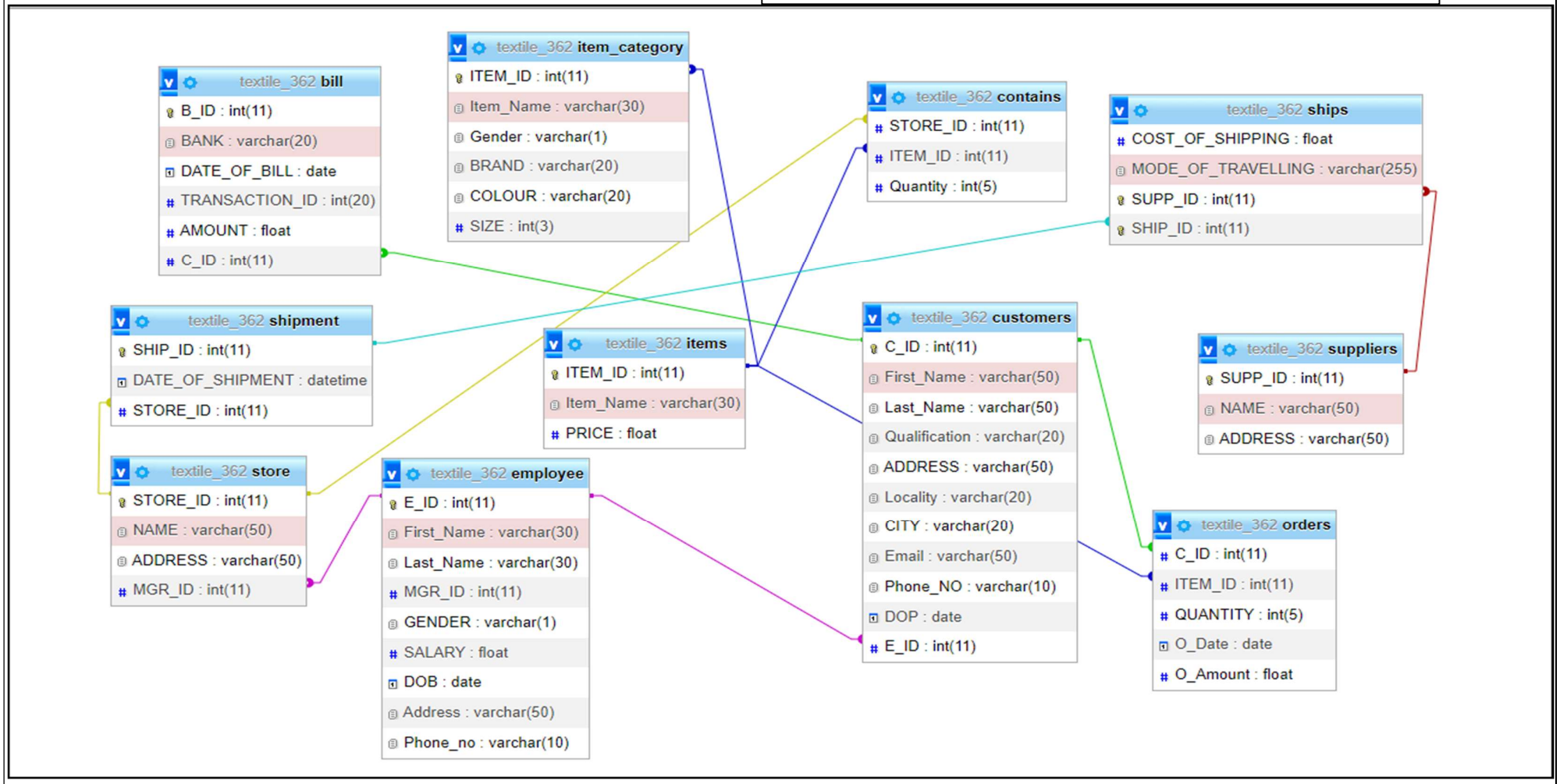


c. The complete E-R Diagram :



3. RELATIONAL – SCHEMA

Bigger Dot (Foreign Key) references (→) **Smaller Dot** (Primary Key)



4. a. DDL statements - Building the database

```
-- Table structure for table `bill`
--

CREATE TABLE `bill` (
  `B_ID` int(11) NOT NULL,
  `BANK` varchar(20) DEFAULT NULL,
  `DATE_OF_BILL` date DEFAULT NULL,
  `TRANSACTION_ID` varchar(20) DEFAULT NULL,
  `AMOUNT` float NOT NULL,
  `C_ID` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
-- Table structure for table `contains`
--

CREATE TABLE `contains` (
  `STORE_ID` int(11) NOT NULL,
  `ITEM_ID` int(11) NOT NULL,
  `Quantity` int(5) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
-- Table structure for table `customers`
--

CREATE TABLE `customers` (
  `C_ID` int(11) NOT NULL,
  `First_Name` varchar(50) DEFAULT NULL,
  `Last_Name` varchar(50) NOT NULL,
  `Qualification` varchar(20) DEFAULT NULL,
  `ADDRESS` varchar(50) DEFAULT NULL,
  `Locality` varchar(20) DEFAULT NULL,
  `CITY` varchar(20) DEFAULT NULL,
  `Email` varchar(50) DEFAULT NULL,
```

```
  `Phone_NO` varchar(10) DEFAULT NULL,
  `DOP` date NOT NULL,
  `E_ID` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
-- Table structure for table `employee`
--

CREATE TABLE `employee` (
  `E_ID` int(11) NOT NULL,
  `First_Name` varchar(30) DEFAULT NULL,
  `Last_Name` varchar(30) NOT NULL,
  `MGR_ID` int(11) DEFAULT NULL,
  `GENDER` varchar(1) NOT NULL,
  `SALARY` float NOT NULL,
  `DOB` date NOT NULL,
  `Address` varchar(50) DEFAULT NULL,
  `Phone_no` varchar(10) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
-- Table structure for table `items`
--

CREATE TABLE `items` (
  `ITEM_ID` int(11) NOT NULL,
  `Item_Name` varchar(30) DEFAULT NULL,
  `PRICE` float DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
-- Table structure for table `item_category`
--

CREATE TABLE `item_category` (
```

```
  `ITEM_ID` int(11) NOT NULL,
  `Item_Name` varchar(30) NOT NULL,
  `Gender` varchar(1) NOT NULL,
  `BRAND` varchar(20) DEFAULT 'SMS Textiles',
  `COLOUR` varchar(20) DEFAULT NULL,
  `SIZE` varchar(10) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
-- Table structure for table `orders`
--

CREATE TABLE `orders` (
  `C_ID` int(11) NOT NULL,
  `ITEM_ID` int(11) NOT NULL,
  `Price` float DEFAULT NULL,
  `QUANTITY` int(5) NOT NULL,
  `O_Date` date DEFAULT NULL,
  `O_Amount` float DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
-- Table structure for table `shipment`
--

CREATE TABLE `shipment` (
  `SHIP_ID` int(11) NOT NULL,
  `DATE_OF_SHIPMENT` date DEFAULT NULL,
  `STORE_ID` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
-- Table structure for table `ships`
--

CREATE TABLE `ships` (
```

```
  `COST_OF_SHIPPING` float DEFAULT NULL,
  `MODE_OF_TRAVELLING` varchar(255) DEFAULT NULL,
  `SUPP_ID` int(11) NOT NULL,
  `SHIP_ID` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
-- Table structure for table `store`
--

CREATE TABLE `store` (
  `STORE_ID` int(11) NOT NULL,
  `NAME` varchar(50) DEFAULT NULL,
  `ADDRESS` varchar(50) DEFAULT NULL,
  `MGR_ID` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
-- Table structure for table `suppliers`
--

CREATE TABLE `suppliers` (
  `SUPP_ID` int(11) NOT NULL,
  `NAME` varchar(50) DEFAULT NULL,
  `ADDRESS` varchar(50) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

4. b. CONSTRAINTS OF PK AND FK

```
--
-- Indexes for table `bill`
--
ALTER TABLE `bill`
  ADD PRIMARY KEY (`B_ID`),
  ADD KEY `c_fid` (`C_ID`);
```

```
--
-- Indexes for table `contains`
--
ALTER TABLE `contains`
  ADD PRIMARY KEY (`STORE_ID`,`ITEM_ID`),
  ADD KEY `i_fid1` (`ITEM_ID`);

--
-- Indexes for table `customers`
--
ALTER TABLE `customers`
  ADD PRIMARY KEY (`C_ID`),
  ADD KEY `e_fid` (`E_ID`);

--
-- Indexes for table `employee`
--
ALTER TABLE `employee`
  ADD PRIMARY KEY (`E_ID`);

--
-- Indexes for table `items`
--
ALTER TABLE `items`
  ADD PRIMARY KEY (`ITEM_ID`);

--
-- Indexes for table `item_category`
--
ALTER TABLE `item_category`
  ADD PRIMARY KEY (`ITEM_ID`);

--
-- Indexes for table `orders`
--
ALTER TABLE `orders`
  ADD PRIMARY KEY (`C_ID`,`ITEM_ID`),
```

```
  ADD KEY `item_fid2` (`ITEM_ID`);

--
-- Indexes for table `shipment`
--
ALTER TABLE `shipment`
  ADD PRIMARY KEY (`SHIP_ID`),
  ADD KEY `store_fid` (`STORE_ID`);

--
-- Indexes for table `ships`
--
ALTER TABLE `ships`
  ADD PRIMARY KEY (`SUPP_ID`,`SHIP_ID`),
  ADD KEY `ship_fid` (`SHIP_ID`);

--
-- Indexes for table `store`
--
ALTER TABLE `store`
  ADD PRIMARY KEY (`STORE_ID`),
  ADD KEY `mgr_fid` (`MGR_ID`);

--
-- Indexes for table `suppliers`
--
ALTER TABLE `suppliers`
  ADD PRIMARY KEY (`SUPP_ID`);

--
-- Constraints for dumped tables
--

--
-- Constraints for table `bill`
--
ALTER TABLE `bill`
```

```
  ADD CONSTRAINT `c_fid` FOREIGN KEY (`C_ID`) REFERENCES
`customers` (`C_ID`) ON DELETE CASCADE;

--
-- Constraints for table `contains`
--
ALTER TABLE `contains`
  ADD CONSTRAINT `i_fid1` FOREIGN KEY (`ITEM_ID`) REFERENCES
`items` (`ITEM_ID`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `store_fid1` FOREIGN KEY (`STORE_ID`)
REFERENCES `store` (`STORE_ID`) ON DELETE CASCADE ON UPDATE
CASCADE;

--
-- Constraints for table `customers`
--
ALTER TABLE `customers`
  ADD CONSTRAINT `e_fid` FOREIGN KEY (`E_ID`) REFERENCES
`employee` (`E_ID`) ON DELETE CASCADE ON UPDATE CASCADE;

--
-- Constraints for table `item_category`
--
ALTER TABLE `item_category`
  ADD CONSTRAINT `item_fid` FOREIGN KEY (`ITEM_ID`)
REFERENCES `items` (`ITEM_ID`) ON DELETE CASCADE ON UPDATE
CASCADE;

--
-- Constraints for table `orders`
--
ALTER TABLE `orders`
  ADD CONSTRAINT `c_fid2` FOREIGN KEY (`C_ID`) REFERENCES
`customers` (`C_ID`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `item_fid2` FOREIGN KEY (`ITEM_ID`)
REFERENCES `items` (`ITEM_ID`) ON DELETE CASCADE ON UPDATE
CASCADE;
```

```
--
-- Constraints for table `shipment`
--
ALTER TABLE `shipment`
ALTER TABLE `shipment`
  ADD CONSTRAINT `shipment_fid` FOREIGN KEY (`SHIP_ID`)
REFERENCES `ships` (`SHIP_ID`) ON DELETE CASCADE ON UPDATE
CASCADE,
  ADD CONSTRAINT `store_fid` FOREIGN KEY (`STORE_ID`)
REFERENCES `store` (`STORE_ID`) ON DELETE CASCADE ON UPDATE
CASCADE;

--
-- Constraints for table `ships`
--
ALTER TABLE `ships`
  ADD CONSTRAINT `supp_fid` FOREIGN KEY (`SUPP_ID`)
REFERENCES `suppliers` (`SUPP_ID`) ON DELETE CASCADE ON
UPDATE CASCADE;

--
-- Constraints for table `store`
--
ALTER TABLE `store`
  ADD CONSTRAINT `mgr_fid` FOREIGN KEY (`MGR_ID`) REFERENCES
`employee` (`E_ID`) ON DELETE CASCADE ON UPDATE CASCADE;
COMMIT;
```

5. POPULATING THE DATABASE

```
-- Dumping data for table `bill`
--

INSERT INTO `bill` (`B_ID`, `BANK`, `DATE_OF_BILL`,
`TRANSACTION_ID`, `AMOUNT`, `C_ID`) VALUES
(9001, 'ICICI', '2022-09-12', '12345678901', 31000, 2001),
(9002, 'HDFC', '2022-10-16', '98765432102', 8348, 2002),
(9003, 'SBI', '2022-10-25', '34676876876', 20800, 2003),
(9004, 'HDFC', '2022-08-22', '46548767588', 9500, 2004),
(9005, 'SBI', '2022-10-09', '89876543876', 4250, 2005),
(9006, 'HDFC', '2022-10-07', '42637267463', 22592, 2006),
(9007, 'ICICI', '2022-08-21', '12348765454', 30797, 2007),
(9008, 'HDFC', '2022-07-19', '43644783600', 28200, 2008),
(9009, 'SBI', '2022-08-31', '52343477809', 6500, 2009),
(9010, 'ICICI', '2022-08-02', '32356654896', 7350, 2010),
(9011, 'SBI', '2022-10-07', '32456487876', 15588, 2011),
(9012, 'HDFC', '2022-10-17', '34678871313', 14995, 2012),
(9013, 'ICICI', '2022-09-18', '54467945789', 10250, 2013),
(9014, 'ICICI', '2022-08-29', '32869248768', 11200, 2014),
(9015, 'SBI', '2022-10-30', '34557897968', 20500, 2015),
(9016, 'HDFC', '2022-08-07', '45467884334', 29300, 2016),
(9017, 'HDFC', '2022-09-23', '34879565998', 32250, 2017),
(9018, 'SBI', '2022-10-27', '12350767426', 14396, 2018),
(9019, 'SBI', '2022-09-26', '53594235687', 19600, 2019),
(9020, 'ICICI', '2022-08-14', '34365587090', 24900, 2020);
```

```
-- Dumping data for table `contains`
--

INSERT INTO `contains` (`STORE_ID`, `ITEM_ID`, `Quantity`)
VALUES
(6001, 4002, 32),
(6001, 4011, 32),
(6001, 4041, 30),
```

```
(6001, 4050, 29),
(6001, 4081, 29),
(6001, 4082, 34),
(6001, 4090, 26),
(6001, 4100, 37),
(6001, 4101, 32),
(6001, 4111, 33),
(6001, 4130, 22),
(6001, 4161, 25),
(6001, 4162, 26),
(6001, 4170, 27),
(6001, 4181, 36),
(6001, 4201, 24),
(6002, 4010, 46),
(6002, 4030, 43),
(6002, 4042, 43),
(6002, 4061, 44),
(6002, 4083, 41),
(6002, 4090, 46),
(6002, 4092, 44),
(6002, 4100, 36),
(6002, 4110, 39),
(6002, 4120, 47),
(6002, 4122, 41),
(6002, 4140, 46),
(6002, 4163, 40),
(6002, 4180, 38),
(6002, 4200, 48),
(6002, 4201, 38),
(6003, 4002, 25),
(6003, 4010, 27),
(6003, 4060, 21),
(6003, 4061, 38),
(6003, 4070, 25),
(6003, 4071, 35),
(6003, 4080, 40),
(6003, 4091, 28),
(6003, 4092, 35),
```

```
(6003, 4100, 32),
(6003, 4102, 31),
(6003, 4121, 36),
(6003, 4122, 40),
(6003, 4130, 28),
(6004, 4000, 27),
(6004, 4092, 22),
(6004, 4112, 34),
(6004, 4131, 31),
(6004, 4150, 33),
(6004, 4151, 23),
(6004, 4160, 34),
(6004, 4163, 28),
(6004, 4171, 31),
(6004, 4190, 35),
(6004, 4300, 50);
```

```
-- Dumping data for table `customers`
--

INSERT INTO `customers` (`C_ID`, `First_Name`, `Last_Name`,
`Qualification`, `ADDRESS`, `Locality`, `CITY`, `Email`,
`Phone_NO`, `DOP`, `E_ID`) VALUES
(2001, 'Sujatha', 'Mohan', 'Doctor', 'Vijay Apartment',
'Bilekahalli', 'Bangalore', 'Sujatha2022  @gmail.com',
'1234567890', '2022-09-12', 1002),
(2002, 'Jhanavhi', '', 'Teacher', 'Jahnavi Enclave',
'Begur', 'Bangalore', 'Jhanavhi2022  @gmail.com',
'9876543210', '2022-10-16', 1013),
(2003, 'Mohan ', 'Raj', 'Engineer', 'Vishwas Apartments',
'Hongasandra', 'Bangalore', 'Mohan 2022  @gmail.com',
'8654432318', '2022-10-25', 1001),
(2004, 'Adarsh', 'Liju', 'Software Engineer', 'Vashist
Apartments', 'Gottigere', 'Bangalore',
'Adarsh2022  @gmail.com', '8763313499', '2022-08-22', 1003),
(2005, 'Vignesh', 'Sheshadri', 'Teacher', 'Shuddha
Shelters', 'Arikere', 'Bangalore',
```

```
'Vignesh2022  @gmail.com', '2454676898', '2022-10-09',
1005),
(2006, 'Harsh ', 'Chowdhary', 'Doctor', 'Vishwas
Apartments', 'Hulimavu', 'Bangalore', 'Harsh
2022  @gmail.com', '7364837638', '2022-10-07', 1006),
(2007, 'Rohith', 'Jain', 'Software Engineer', 'Hareetas',
'Hongasandra', 'Bangalore', 'Rohith2022  @gmail.com',
'5365384899', '2022-08-21', 1007),
(2008, 'Himanshu', '', 'Engineer', 'Nandana Greens',
'Bilekahalli', 'Bangalore', 'Himanshu2022  @gmail.com',
'7259966769', '2022-07-19', 1013),
(2009, 'Sutharsan', 'Raj', 'Student', 'Vijay apartment',
'Begur', 'Bangalore', 'Sutharsan2022  @gmail.com',
'8396364290', '2022-08-31', 1004),
(2010, 'Kavi', 'Priya', 'Teacher', 'Vishwas Apartments',
'Hongasandra', 'Bangalore', 'Kavi2022  @gmail.com',
'7678347343', '2022-08-02', 1013),
(2011, 'Varuna', 'Shree', 'Student', 'Hasmitha Nandana',
'Gottigere', 'Bangalore', 'Varuna2022  @gmail.com',
'2646747346', '2022-10-07', 1013),
(2012, 'Gopinath', 'Gokul', 'Doctor', 'Prestige Song of
South', 'Arekere', 'Bangalore', 'Gopinath2022  @gmail.com',
'8464987736', '2022-10-17', 1009),
(2013, 'Krishna', 'Kumar', 'Student', 'Uday Apartments',
'Arekere', 'Bangalore', 'Krishna2022  @gmail.com',
'4567893210', '2022-09-18', 1011),
(2014, 'Divya', 'Shree', 'Doctor', 'Prestige Song of South',
'Hulimavu', 'Bangalore', 'Divya2022  @gmail.com',
'8664313546', '2022-08-29', 1012),
(2015, 'Siddharth', 'Seetharaman', 'Engineer', 'Pride
Apartments', 'Hulimavu', 'Bangalore',
'Siddharth2022  @gmail.com', '8765432345', '2022-10-30',
1005),
(2016, 'Gokul', 'Sreenath', 'Student', 'Hasmitha Nandana',
'Begur', 'Bangalore', 'Gokul2022  @gmail.com', '6432469890',
'2022-08-07', 1013),
```

```
(2017, 'Ramesh', 'Agarwal', 'Activist', 'Anugraha',
'Hongasandra', 'Bangalore', 'Ramesh2022  @gmail.com',
'8632145805', '2022-09-23', 1004),
(2018, 'Suresh', 'Sathish', 'Teacher', 'Phoenix One',
'Bilekahalli', 'Bangalore', 'Suresh2022  @gmail.com',
'7332668789', '2022-10-27', 1001),
(2019, 'Om', 'Katkam', 'Driver', 'Brigade Millenium',
'Gottigere', 'Bangalore', 'Om2022  @gmail.com',
'1298765235', '2022-09-26', 1003),
(2020, 'Shashank', 'Singh', 'Driver', 'Pride Apartments',
'Arekere', 'Bangalore', 'Shashank2022  @gmail.com',
'9876542345', '2022-08-14', 1006),
(2021, 'ABC', 'XYZ', 'Activist', 'Banashankari',
'Bannerghatta', 'Bangalore', 'abc@gmail.com', '9876543219',
'2022-11-02', 1005);
```

```
-- Dumping data for table `employee`
--

INSERT INTO `employee` (`E_ID`, `First_Name`, `Last_Name`,
`MGR_ID`, `GENDER`, `SALARY`, `DOB`, `Address`, `Phone_no`)
VALUES
(1001, 'Akshay', 'Kumar', 1010, 'M', 69709, '1980-01-23',
'MG Road', '8596092928'),
(1002, 'Akarsh', 'Poojari', 0, 'M', 112526, '1970-06-12',
'Basavangudi', '7152011668'),
(1003, 'Bhanu', 'Preetham', 1002, 'M', 77901, '1990-03-15',
'Banashankari', '7479776634'),
(1004, 'Bhavya ', 'Shree', 1007, 'F', 67891, '1984-10-17',
'RR Nagar', '3391172715'),
(1005, 'Milan', '', 1002, 'M', 79103, '1987-09-27',
'Basavangudi', '5605646750'),
(1006, 'Meenakshi', 'Saravanan', 1007, 'F', 59795, '1986-10-
08', 'Banashankari', '2015362539'),
(1007, 'Ramya', 'Pandian', 0, 'F', 115297, '1971-02-19',
'Banashankari', '9525847821'),
```

```
(1008, 'Nisha ', 'Advani', 1007, 'F', 55859, '1989-02-28',
'Basavangudi', '5884750721'),
(1009, 'Surendra', 'Jain', 1013, 'M', 63712, '1992-03-14',
'RR Nagar', '5002946135'),
(1010, 'Shina', 'Sudhir', 0, 'F', 109979, '1974-11-23', 'RR
Nagar', '8954874497'),
(1011, 'Tissa', 'Varghese', 1013, 'F', 75312, '1988-12-04',
'MG Road', '7654873190'),
(1012, 'Navaneeth', 'Purohit', 1010, 'M', 60593, '1993-12-
07', 'MG Road', '1480171904'),
(1013, 'Yuvraj', 'Singh', 0, 'M', 118701, '1975-05-23',
'Banashankari', '3791768076');
```

```
-- Dumping data for table `items`
--

INSERT INTO `items` (`ITEM_ID`, `Item_Name`, `PRICE`) VALUES
(4000, 'Shirt', 900),
(4001, 'Shirt', 1499),
(4002, 'Shirt', 999),
(4010, 'Pant', 2999),
(4011, 'Pant', 4000),
(4030, 'Coat', 15000),
(4041, 'Gown', 7999),
(4042, 'Gown', 4999),
(4050, 'Cap/Hat', 499),
(4060, 'Sweater', 999),
(4061, 'Sweater', 1799),
(4070, 'Jacket', 900),
(4071, 'Jacket', 600),
(4080, 'Legging', 400),
(4081, 'Legging', 349),
(4082, 'Legging', 499),
(4083, 'Legging', 550),
(4090, 'Jeggings', 700),
(4091, 'Jeggings', 599),
(4092, 'Jeggings', 445),
```

```
(4100, 'Tops', 250),
(4101, 'Tops', 399),
(4102, 'Tops', 549),
(4110, 'Saree', 1749),
(4111, 'Saree', 2499),
(4112, 'Saree', 3999),
(4120, 'Chudidhar', 799),
(4121, 'Chudidhar', 999),
(4122, 'Chudidhar', 1599),
(4130, 'Frock', 1199),
(4131, 'Frock', 1499),
(4140, 'Lehenga', 3000),
(4150, 'Dhoti', 400),
(4151, 'Dhoti', 500),
(4160, 'Tshirt', 799),
(4161, 'Tshirt', 699),
(4162, 'Tshirt', 1000),
(4163, 'Tshirt', 900),
(4170, 'Shorts', 500),
(4171, 'Shorts', 599),
(4180, 'Skirt', 699),
(4181, 'Skirt', 700),
(4190, 'Pyjama', 650),
(4200, 'Kurta', 999),
(4201, 'Kurta', 799),
(4300, 'Palazo', 649);
```

```
-- Dumping data for table `item_category`
--

INSERT INTO `item_category` (`ITEM_ID`, `Item_Name`,
`Gender`, `BRAND`, `COLOUR`, `SIZE`) VALUES
(4000, 'Shirt', 'M', 'Ramraj', 'Red', '38'),
(4001, 'Shirt', 'M', 'Allen Solly', 'Blue', '40'),
(4002, 'Shirt', 'M', 'Peterson', 'Green', '42'),
(4010, 'Pant', 'M', 'Buffallo', 'Black', '38'),
(4011, 'Pant', 'M', 'Polo', 'Brown', '40'),
```

```
(4030, 'Coat', 'F', 'Raymond', 'Grey', 'XXL'),
(4041, 'Gown', 'F', 'Chennai Silks', 'Pink', 'XXL'),
(4042, 'Gown', 'M', 'ARRS Silks', 'Green', 'XL'),
(4050, 'Cap/Hat', 'M', 'MAX', 'Cyan', ''),
(4060, 'Sweater', 'F', 'Deccathlon', 'Navyblue', 'XL'),
(4061, 'Sweater', 'M', 'Deccathlon', 'Light_Brown', 'L'),
(4070, 'Jacket', 'M', 'Kumaran Tex', 'Red', 'L'),
(4071, 'Jacket', 'M', 'Vittal Dresses', 'Brown', 'XXL'),
(4080, 'Legging', 'F', 'Chennai Silks', 'Maroon', 'XL'),
(4081, 'Legging', 'F', 'ARRS Silks', 'White', 'M'),
(4082, 'Legging', 'F', 'Kumaran Tex', 'Black', 'XXL'),
(4083, 'Legging', 'F', 'Pothys', 'Dark_Blue', 'XXXL'),
(4090, 'Jeggings', 'F', 'Chennai Silks', 'Purple', 'XXL'),
(4091, 'Jeggings', 'F', 'ARRS Silks', 'Grey', 'XL'),
(4092, 'Jeggings', 'F', 'Kumaran Tex', 'Yellow', 'L'),
(4100, 'Tops', 'F', 'Pothys', 'Black', 'XL'),
(4101, 'Tops', 'F', 'Trends', 'Red', 'XL'),
(4102, 'Tops', 'F', 'Trends', 'Blue', 'XXL'),
(4110, 'Saree', 'F', 'Chennai Silks', 'Pink', ''),
(4111, 'Saree', 'F', 'Kanchipuram Textiles', 'Purple', ''),
(4112, 'Saree', 'F', 'Kanchipuram Textiles', 'Blue', ''),
(4120, 'Chudidhar', 'F', 'Chennai Silks', 'Yellow', 'L'),
(4121, 'Chudidhar', 'F', 'ARRS Silks', 'Orange', 'XL'),
(4122, 'Chudidhar', 'F', 'Pothys', 'Green', 'XXL'),
(4130, 'Frock', 'F', 'Ramraj', 'Rainbow', '38'),
(4131, 'Frock', 'F', 'MAX', 'Violet', '42'),
(4140, 'Lehenga', 'F', 'Raymond', 'Baby_Pink', 'XL'),
(4150, 'Dhoti', 'M', 'Ramraj', 'White', '40'),
(4151, 'Dhoti', 'M', 'Pothys', 'Light_Brown', '50'),
(4160, 'Tshirt', 'M', 'Van Hussen', 'Red', '42'),
(4161, 'Tshirt', 'M', 'Van Hussen', 'Black', '44'),
(4162, 'Tshirt', 'M', 'MAX', 'Brown', '40'),
(4163, 'Tshirt', 'M', 'Polo', 'Grey', '38'),
(4170, 'Shorts', 'M', 'Polo', 'Green', '36'),
(4171, 'Shorts', 'M', 'Buffallo', 'Blue', '38'),
(4180, 'Skirt', 'F', 'Kanchipuram Tex', 'Light_Green',
'42'),
(4181, 'Skirt', 'F', 'Kumaran Tex', 'Pink', '38'),
```

```
(4190, 'Pyjama', 'M', 'Chennai Silks', 'Grey', '40'),
(4200, 'Kurta', 'M', 'Raymond', 'White', '42'),
(4201, 'Kurta', 'M', 'Trends', 'Red', '36'),
(4300, 'Palazo', 'F', 'Prisma', 'Pink', 'XL');
```

```
-- Dumping data for table `orders`
--

INSERT INTO `orders` (`C_ID`, `ITEM_ID`, `Price`,
`QUANTITY`, `O_Date`, `O_Amount`) VALUES
(2001, 4011, 4000, 4, '2022-12-09', 16000),
(2001, 4090, 700, 10, '2022-12-09', 7000),
(2001, 4162, 1000, 8, '2022-09-12', 8000),
(2002, 4002, 999, 2, '2022-10-16', 2000),
(2002, 4080, 400, 4, '2022-08-16', 1600),
(2002, 4102, 549, 5, '2022-10-16', 2750),
(2002, 4121, 999, 2, '2022-10-16', 1998),
(2003, 4000, 900, 10, '2022-10-25', 8000),
(2003, 4090, 700, 7, '2022-10-25', 4900),
(2003, 4151, 500, 5, '2022-10-25', 2500),
(2003, 4171, 599, 9, '2022-10-25', 5400),
(2004, 4082, 499, 12, '2022-08-22', 6000),
(2004, 4170, 500, 7, '2022-08-22', 3500),
(2005, 4002, 999, 1, '2022-09-10', 1000),
(2005, 4081, 349, 5, '2022-09-10', 1750),
(2005, 4082, 499, 3, '2022-09-10', 1500),
(2006, 4120, 799, 7, '2022-07-10', 5600),
(2006, 4140, 3000, 3, '2022-07-10', 9000),
(2006, 4201, 799, 8, '2022-01-08', 7992),
(2007, 4042, 4999, 1, '2022-08-21', 5000),
(2007, 4122, 1599, 3, '2022-08-21', 4797),
(2007, 4140, 3000, 7, '2022-08-21', 21000),
(2008, 4002, 999, 3, '2022-07-19', 3000),
(2008, 4010, 2999, 6, '2022-07-19', 18000),
(2008, 4061, 1799, 4, '2022-07-19', 7200),
(2009, 4083, 550, 6, '2022-08-31', 3300),
(2009, 4200, 999, 4, '2022-08-31', 3200),
```

```
(2010, 4071, 600, 4, '2022-02-08', 2400),
(2010, 4092, 445, 7, '2022-02-08', 3150),
(2011, 4122, 1599, 3, '2022-07-10', 4797),
(2011, 4130, 1199, 7, '2022-07-10', 8393),
(2012, 4060, 999, 1, '2022-10-17', 1000),
(2012, 4061, 1799, 3, '2022-10-17', 5400),
(2012, 4091, 599, 6, '2022-10-17', 3600),
(2012, 4121, 999, 5, '2022-10-17', 4995),
(2013, 4070, 900, 1, '2022-09-18', 900),
(2013, 4102, 549, 3, '2022-09-18', 1650),
(2014, 4160, 799, 8, '2022-08-29', 6400),
(2015, 4002, 999, 5, '2022-10-30', 5000),
(2015, 4011, 4000, 2, '2022-10-30', 8000),
(2015, 4111, 2499, 3, '2022-10-30', 7500),
(2016, 4030, 15000, 1, '2022-07-08', 15000),
(2016, 4061, 1799, 3, '2022-07-08', 3600),
(2016, 4180, 699, 5, '2022-07-08', 3500),
(2016, 4200, 999, 9, '2022-07-08', 7200),
(2017, 4030, 15000, 2, '2022-09-23', 30000),
(2017, 4100, 250, 9, '2022-09-23', 2250),
(2018, 4131, 1499, 4, '2022-10-27', 5996),
(2018, 4160, 799, 5, '2022-10-27', 4000),
(2018, 4190, 650, 11, '2022-10-27', 4400),
(2019, 4041, 7999, 2, '2022-09-26', 16000),
(2019, 4050, 499, 3, '2022-09-26', 1500),
(2019, 4090, 700, 3, '2022-09-26', 2100),
(2020, 4100, 250, 15, '2022-08-14', 3750),
(2020, 4110, 1749, 9, '2022-08-14', 15750),
(2020, 4163, 900, 6, '2022-08-14', 5400),
(2021, 4041, 7999, 2, '2022-11-02', 15998);
```

```
-- Dumping data for table `shipment`
--

INSERT INTO `shipment` (`SHIP_ID`, `DATE_OF_SHIPMENT`,
`STORE_ID`) VALUES
(8001, '2022-05-22', 6001),
```

```
(8002, '2022-06-30', 6001),
(8003, '2022-09-05', 6001),
(8004, '2022-12-05', 6002),
(8005, '2022-04-21', 6004),
(8006, '2022-06-06', 6003),
(8007, '2022-06-25', 6004),
(8008, '2022-06-16', 6002),
(8009, '2022-05-31', 6003),
(8010, '2022-04-27', 6004);
```

```
-- Dumping data for table `ships`
--

INSERT INTO `ships` (`COST_OF_SHIPPING`,
`MODE_OF_TRAVELLING`, `SUPP_ID`, `SHIP_ID`) VALUES
(5489, 'Roadways', 7001, 8005),
(4000, 'Airways', 7001, 8006),
(6597, 'Railways', 7002, 8001),
(870, 'Railways', 7002, 8009),
(5500, 'Railways', 7003, 8004),
(5500, 'Railways', 7003, 8008),
(7000, 'Airways', 7004, 8003),
(3500, 'Roadways', 7004, 8007),
(6290, 'Roadways', 7005, 8002),
(6290, 'Waterways', 7005, 8010);
```

```
-- Dumping data for table `store`
--

INSERT INTO `store` (`STORE_ID`, `NAME`, `ADDRESS`,
`MGR_ID`) VALUES
(6001, 'Bannerghatta', 'BG Road', 1002),
(6002, 'Jayanagar', 'Near Cool Joint', 1007),
(6003, 'Rajajinagar', 'Opposite to Rajarajeshwari Medical
College', 1013),
```

```
(6004, 'Malleshwaram', 'Near Railway Station', 1010);
```

```
-- Dumping data for table `suppliers`
--

INSERT INTO `suppliers` (`SUPP_ID`, `NAME`, `ADDRESS`)
VALUES
(7001, 'Sukeerthan', 'Haritasa Apartments'),
(7002, 'Monisha', 'Mahaveer Marvel'),
(7003, 'Kavana', 'Tvs Emarold jordin'),
(7004, 'Roshan', 'Thayappa Garden'),
(7005, 'Satvik', 'Bhavani Apartments');
```

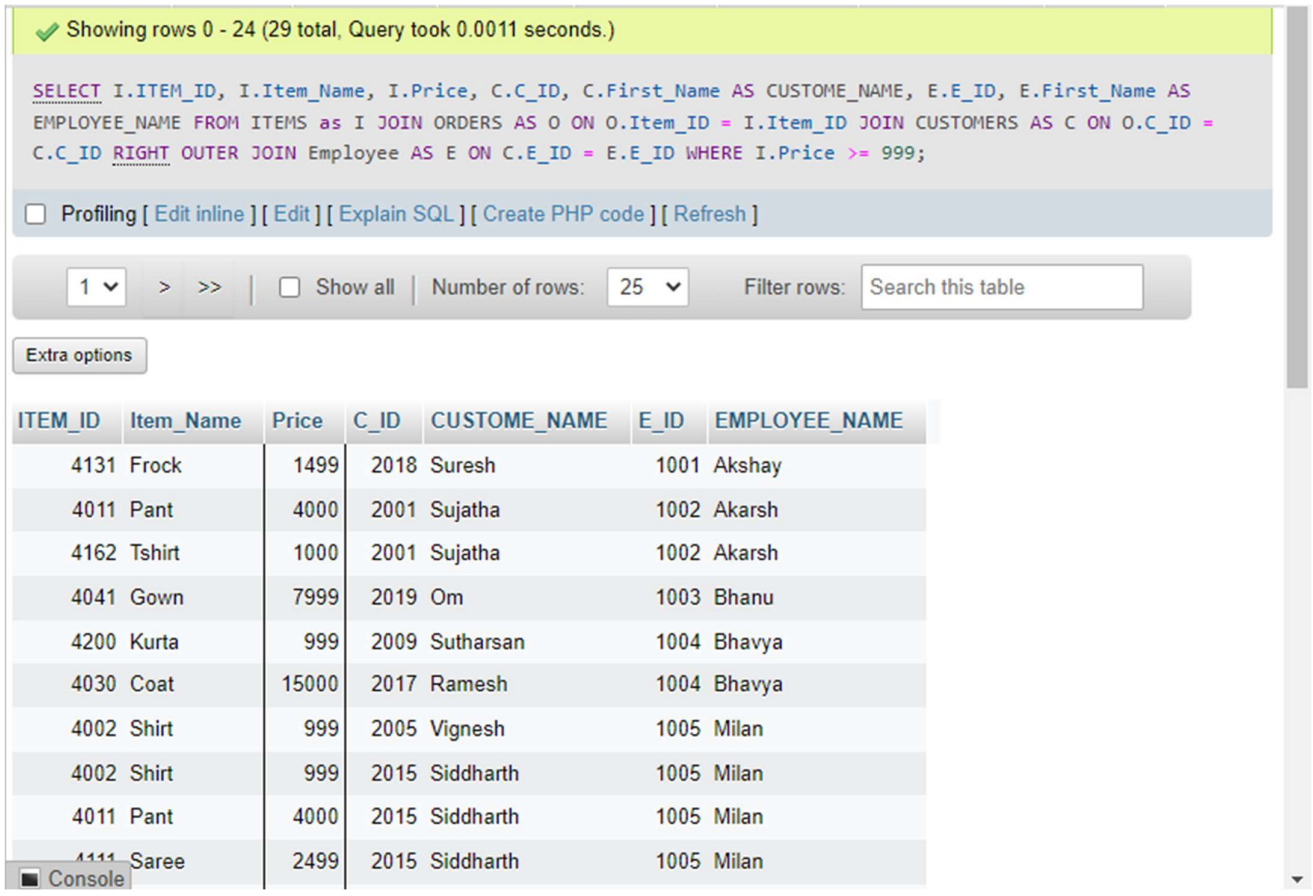
6. JOIN QUERIES

a. Retrieve the first name , last name and item names sold by employees to the customers where item price > 1000

QUERY :

```
SELECT I.ITEM_ID, I.Item_Name, I.Price, C.C_ID, C.First_Name
AS CUSTOME_NAME, E.E_ID, E.First_Name AS EMPLOYEE_NAME
FROM ITEMS as I JOIN ORDERS AS O ON O.Item_ID = I.Item_ID
JOIN CUSTOMERS AS C ON O.C_ID = C.C_ID
RIGHT OUTER JOIN Employee AS E
ON C.E_ID = E.E_ID
WHERE I.Price >= 999;
```

SCREENSHOT :



ITEM_ID	Item_Name	Price	C_ID	CUSTOME_NAME	E_ID	EMPLOYEE_NAME
4131	Frock	1499	2018	Suresh	1001	Akshay
4011	Pant	4000	2001	Sujatha	1002	Akarsh
4162	Tshirt	1000	2001	Sujatha	1002	Akarsh
4041	Gown	7999	2019	Om	1003	Bhanu
4200	Kurta	999	2009	Sutharsan	1004	Bhavya
4030	Coat	15000	2017	Ramesh	1004	Bhavya
4002	Shirt	999	2005	Vignesh	1005	Milan
4002	Shirt	999	2015	Siddharth	1005	Milan
4011	Pant	4000	2015	Siddharth	1005	Milan
4111	Saree	2499	2015	Siddharth	1005	Milan

ITEM_ID	Item_Name	Price	C_ID	CUSTOME_NAME	E_ID	EMPLOYEE_NAME
4111	Saree	2499	2015	Siddharth	1005	Milan
4041	Gown	7999	2021	ABC	1005	Milan
4140	Lehenga	3000	2006	Harsh	1006	Meenakshi
4110	Saree	1749	2020	Shashank	1006	Meenakshi
4042	Gown	4999	2007	Rohith	1007	Ramya
4122	Chudidhar	1599	2007	Rohith	1007	Ramya
4140	Lehenga	3000	2007	Rohith	1007	Ramya
4060	Sweater	999	2012	Gopinath	1009	Surendra
4061	Sweater	1799	2012	Gopinath	1009	Surendra
4121	Chudidhar	999	2012	Gopinath	1009	Surendra
4002	Shirt	999	2002	Jhanavhi	1013	Yuvraj
4121	Chudidhar	999	2002	Jhanavhi	1013	Yuvraj
4002	Shirt	999	2008	Himanshu	1013	Yuvraj
4010	Pant	2999	2008	Himanshu	1013	Yuvraj
4061	Sweater	1799	2008	Himanshu	1013	Yuvraj
4122	Chudidhar	1599	2011	Varuna	1013	Yuvraj

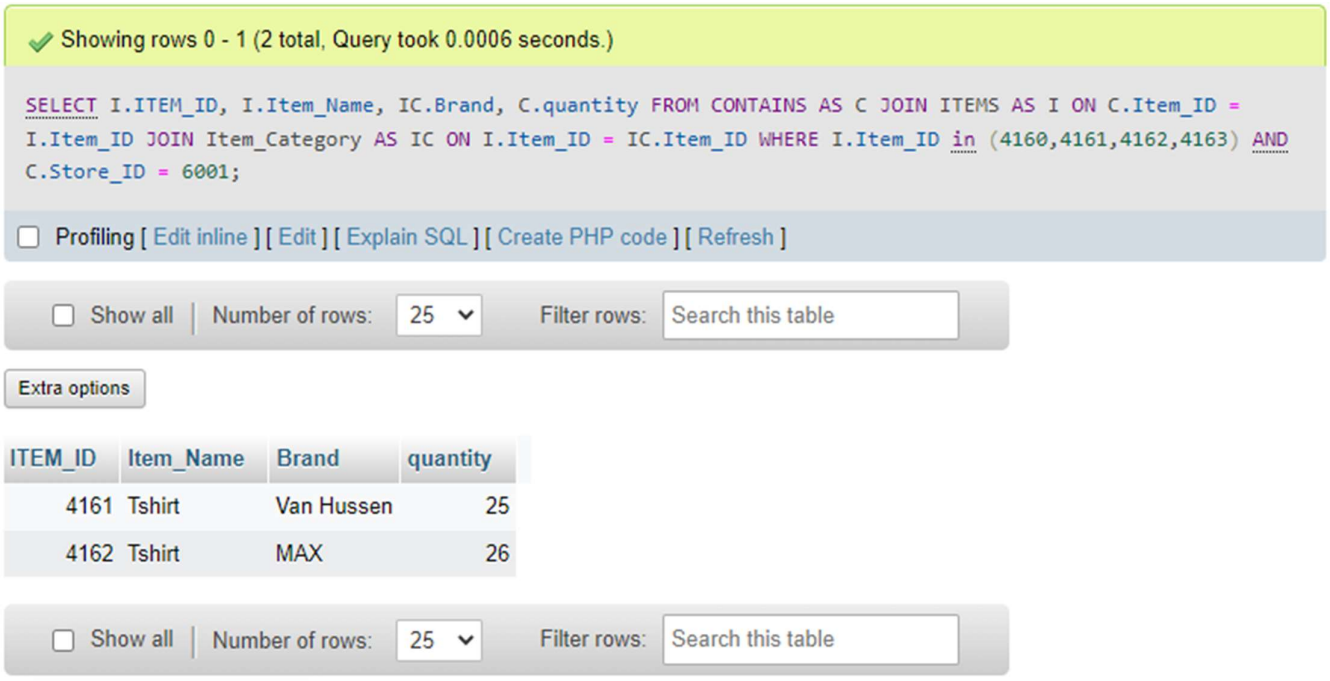
1 > >> Show all Number of rows: 25 Filter rows: Search this table

b. List all the brands of the T-Shirt available in STORE : 6001 along with their quantities.

QUERY :

```
SELECT I.ITEM_ID, I.Item_Name, IC.Brand, C.quantity
FROM CONTAINS AS C JOIN ITEMS AS I ON C.Item_ID = I.Item_ID
JOIN Item_Category AS IC ON I.Item_ID = IC.Item_ID
WHERE I.Item_ID in (4160,4161,4162,4163) AND C.Store_ID =
6001;
```

SCREENSHOT :



c. Retrieve the mode of travelling and names of suppliers, whose products were shipped during June.

QUERY :

```
d. SELECT SU.SUPP_ID, SH.Mode_of_travelling, SH.SHIP_ID,
   SHM.Date_of_Shipment
e. FROM Suppliers AS SU NATURAL JOIN Ships AS SH NATURAL
   JOIN SHIPMENT AS SHM
f. WHERE month(SHM.Date_of_Shipment) = 6;
g.
```

SCREENSHOT :

Showing rows 0 - 3 (4 total, Query took 0.0018 seconds.)

```
SELECT SU.SUPP_ID, SH.Mode_of_travelling, SH.SHIP_ID, SHM.Date_of_Shipment FROM Suppliers AS SU NATURAL
JOIN Ships AS SH NATURAL JOIN SHIPMENT AS SHM WHERE month(SHM.Date_of_Shipment) = 6;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

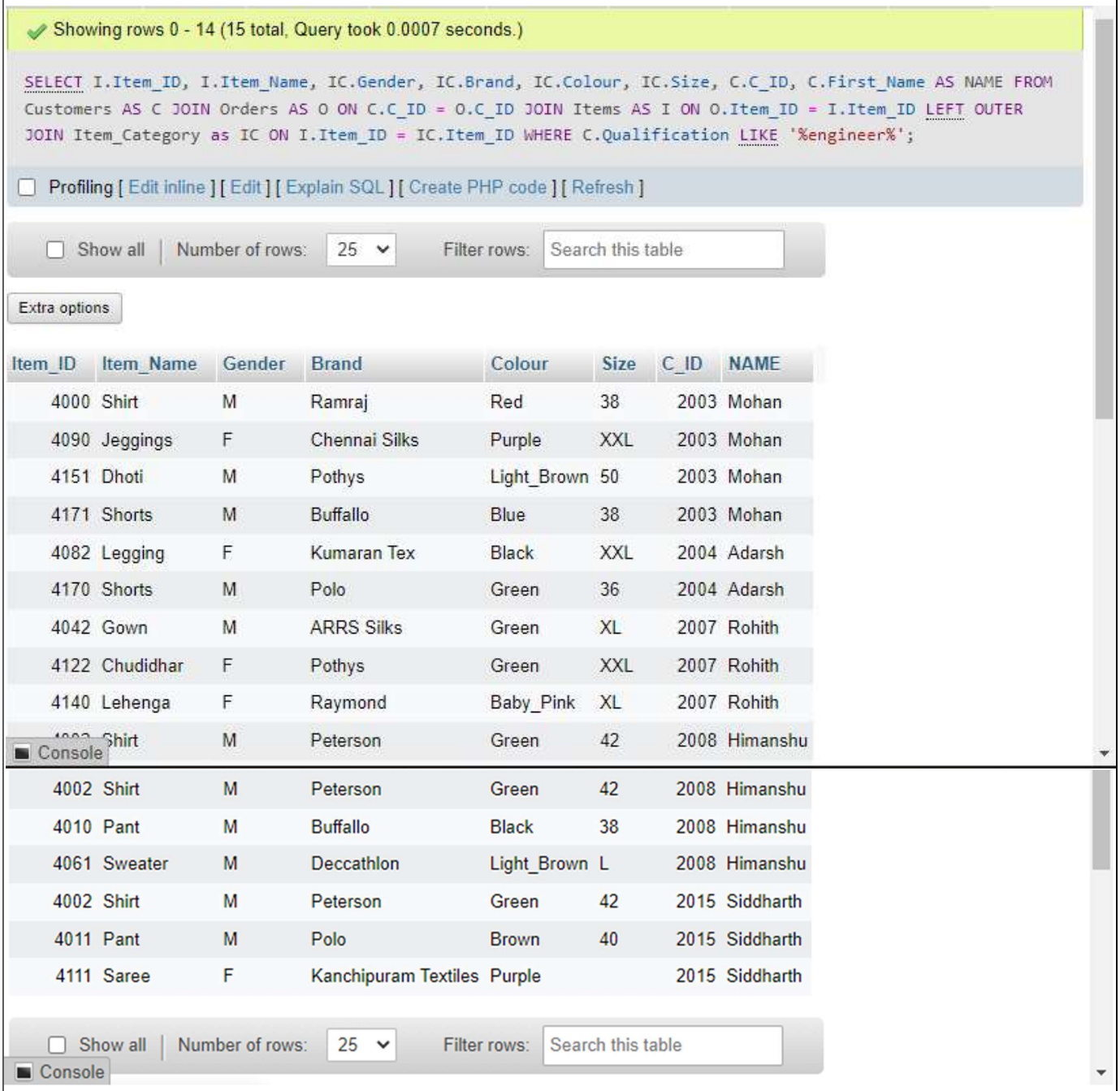
SUPP_ID	Mode_of_travelling	SHIP_ID	Date_of_Shipment
7001	Airways	8006	2022-06-06
7003	Railways	8008	2022-06-16
7004	Roadways	8007	2022-06-25
7005	Roadways	8002	2022-06-30

d. List the details of items bought by customers who are engineers

QUERY :

```
SELECT I.Item_ID, I.Item_Name, IC.Gender, IC.Brand,
IC.Colour, IC.Size, C.C_ID, C.First_Name AS NAME
FROM Customers AS C JOIN Orders AS O ON C.C_ID = O.C_ID JOIN
Items AS I ON O.Item_ID = I.Item_ID
LEFT OUTER JOIN
Item_Category as IC ON I.Item_ID = IC.Item_ID
WHERE C.Qualification LIKE '%engineer%';
```

SCREENSHOT :



Showing rows 0 - 14 (15 total, Query took 0.0007 seconds.)

```
SELECT I.Item_ID, I.Item_Name, IC.Gender, IC.Brand, IC.Colour, IC.Size, C.C_ID, C.First_Name AS NAME FROM
Customers AS C JOIN Orders AS O ON C.C_ID = O.C_ID JOIN Items AS I ON O.Item_ID = I.Item_ID LEFT OUTER
JOIN Item_Category as IC ON I.Item_ID = IC.Item_ID WHERE C.Qualification LIKE '%engineer%';
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

Item_ID	Item_Name	Gender	Brand	Colour	Size	C_ID	NAME
4000	Shirt	M	Ramraj	Red	38	2003	Mohan
4090	Jeggings	F	Chennai Silks	Purple	XXL	2003	Mohan
4151	Dhoti	M	Pothys	Light_Brown	50	2003	Mohan
4171	Shorts	M	Buffallo	Blue	38	2003	Mohan
4082	Legging	F	Kumaran Tex	Black	XXL	2004	Adarsh
4170	Shorts	M	Polo	Green	36	2004	Adarsh
4042	Gown	M	ARRS Silks	Green	XL	2007	Rohith
4122	Chudidhar	F	Pothys	Green	XXL	2007	Rohith
4140	Lehenga	F	Raymond	Baby_Pink	XL	2007	Rohith
[illegible]hirt		M	Peterson	Green	42	2008	Himanshu

Console

4002	Shirt	M	Peterson	Green	42	2008	Himanshu
4010	Pant	M	Buffallo	Black	38	2008	Himanshu
4061	Sweater	M	Deccathlon	Light_Brown	L	2008	Himanshu
4002	Shirt	M	Peterson	Green	42	2015	Siddharth
4011	Pant	M	Polo	Brown	40	2015	Siddharth
4111	Saree	F	Kanchipuram Textiles	Purple		2015	Siddharth

Show all | Number of rows: 25 Filter rows: Search this table

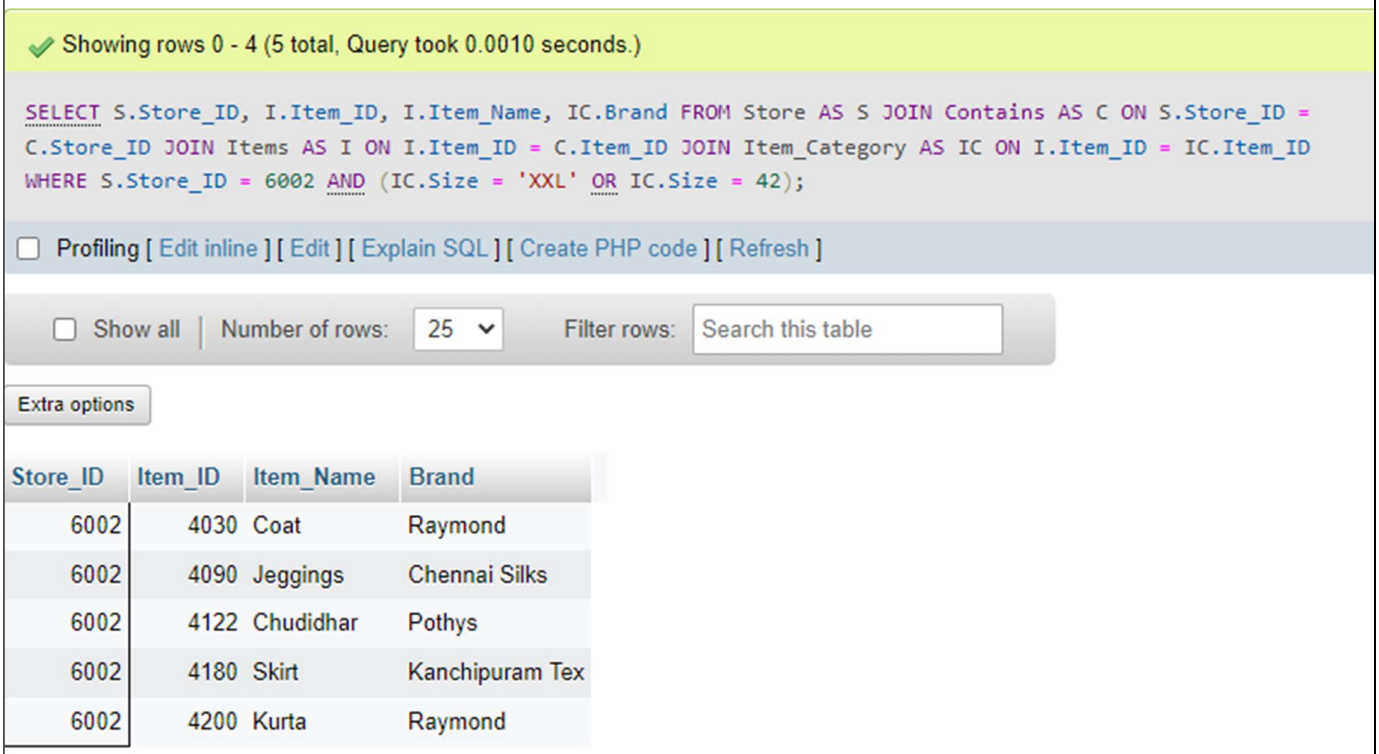
Console

e. Show the item name and brand in STORE = 6002 which has dress with size XXL or 42

QUERY :

```
SELECT S.Store_ID, I.Item_ID, I.Item_Name, IC.Brand
FROM Store AS S JOIN Contains AS C ON S.Store_ID =
C.Store_ID JOIN Items AS I ON I.Item_ID = C.Item_ID
JOIN Item_Category
AS IC ON I.Item_ID = IC.Item_ID
WHERE S.Store_ID = 6002 AND (IC.Size = 'XXL' OR IC.Size =
42);
```

SCREENSHOT :



Showing rows 0 - 4 (5 total, Query took 0.0010 seconds.)

```
SELECT S.Store_ID, I.Item_ID, I.Item_Name, IC.Brand FROM Store AS S JOIN Contains AS C ON S.Store_ID =
C.Store_ID JOIN Items AS I ON I.Item_ID = C.Item_ID JOIN Item_Category AS IC ON I.Item_ID = IC.Item_ID
WHERE S.Store_ID = 6002 AND (IC.Size = 'XXL' OR IC.Size = 42);
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

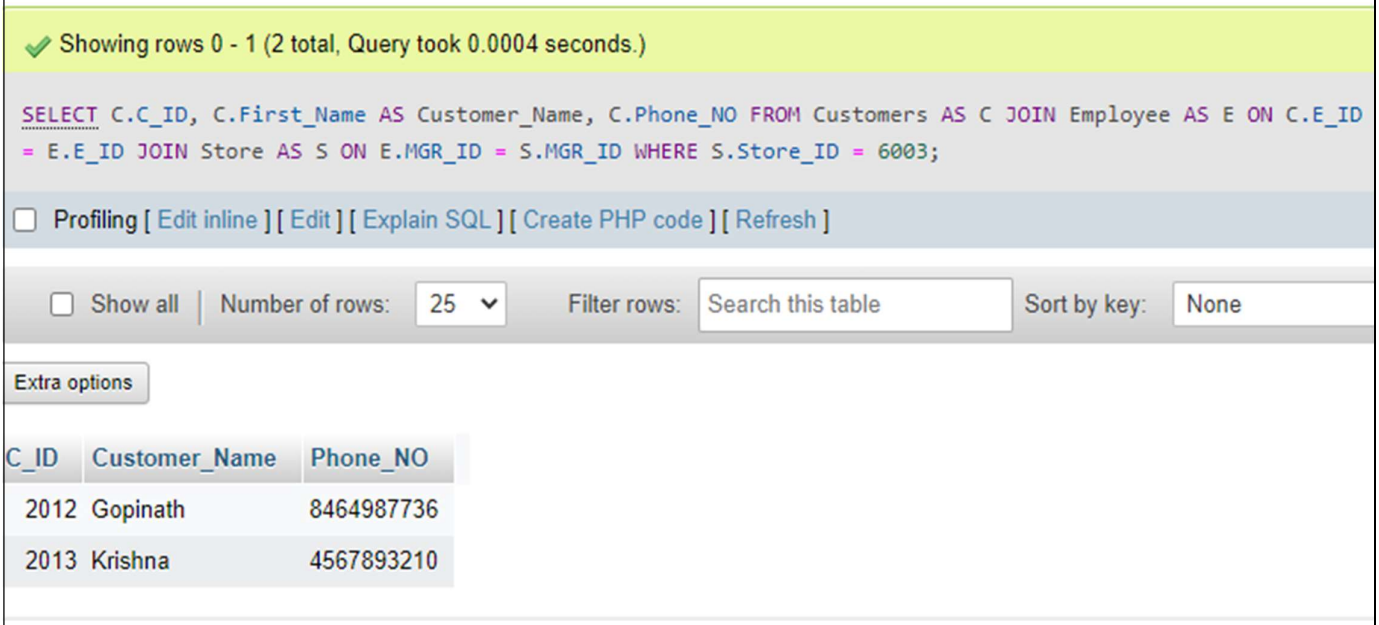
Store_ID	Item_ID	Item_Name	Brand
6002	4030	Coat	Raymond
6002	4090	Jeggings	Chennai Silks
6002	4122	Chudidhar	Pothys
6002	4180	Skirt	Kanchipuram Tex
6002	4200	Kurta	Raymond

f. List the customers visiting Store : 6003

QUERY :

```
SELECT C.C_ID, C.First_Name AS Customer_Name, C.Phone_NO
FROM Customers AS C JOIN Employee AS E ON C.E_ID = E.E_ID
JOIN Store AS S ON E.MGR_ID = S.MGR_ID
WHERE S.Store_ID = 6003;
```

SCREENSHOT :



Showing rows 0 - 1 (2 total, Query took 0.0004 seconds.)

```
SELECT C.C_ID, C.First_Name AS Customer_Name, C.Phone_NO FROM Customers AS C JOIN Employee AS E ON C.E_ID
= E.E_ID JOIN Store AS S ON E.MGR_ID = S.MGR_ID WHERE S.Store_ID = 6003;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

C_ID	Customer_Name	Phone_NO
2012	Gopinath	8464987736
2013	Krishna	4567893210

7. AGGREGATE FUNCTIONS QUERIES

a. Average salary of all the employees in their respective stores

QUERY :

```
SELECT S.Store_ID, S.Name, S.MGR_ID, AVG(E.Salary) AS
Average_Salary
FROM Store AS S JOIN Employee AS E ON S.MGR_ID = E.MGR_ID
GROUP BY E.MGR_ID
ORDER BY AVG(E.Salary) ASC;
```

SCREENSHOT :

Showing rows 0 - 3 (4 total, Query took 0.0025 seconds.)

```
SELECT S.Store_ID, S.Name, S.MGR_ID, AVG(E.Salary) AS Average_Salary FROM Store AS S JOIN Employee AS E ON
S.MGR_ID = E.MGR_ID GROUP BY E.MGR_ID ORDER BY AVG(E.Salary) ASC;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

Store_ID	Name	MGR_ID	Average_Salary
6002	Jayanagar	1007	61181.666666666664
6004	Malleshwaram	1010	65151
6003	Rajajinagar	1013	69512
6001	Bannerghatta	1002	78502

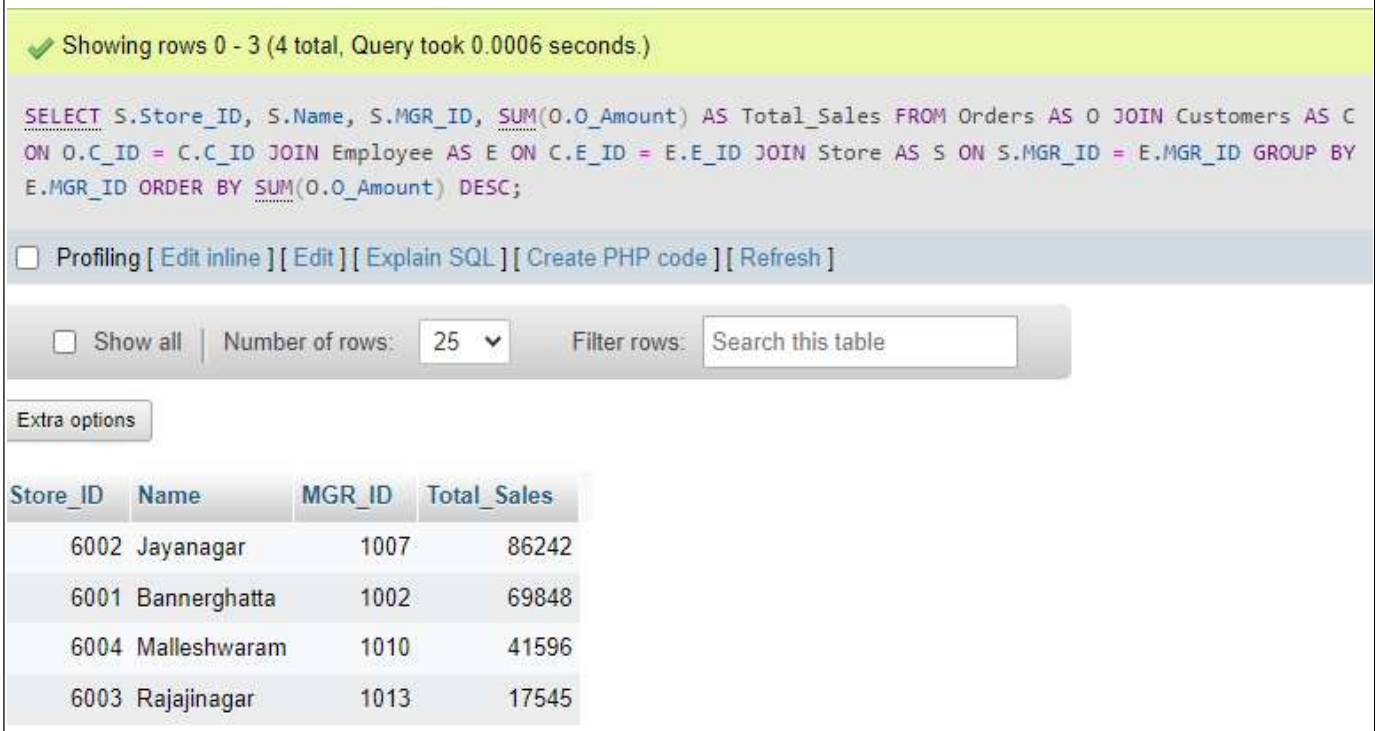
b. Find out total sales across all the stores during July to December

QUERY :

```
SELECT  S.Store_ID, S.Name, S.MGR_ID, SUM(O.O_Amount) AS
Total_Sales
FROM Orders AS O JOIN Customers AS C ON O.C_ID = C.C_ID JOIN
Employee AS E ON C.E_ID = E.E_ID
JOIN Store AS S ON S.MGR_ID = E.MGR_ID
```

```
GROUP BY E.MGR_ID
ORDER BY SUM(O.O_Amount) DESC;
```

SCREENSHOT :



Showing rows 0 - 3 (4 total, Query took 0.0006 seconds.)

```
SELECT S.Store_ID, S.Name, S.MGR_ID, SUM(O.O_Amount) AS Total_Sales FROM Orders AS O JOIN Customers AS C
ON O.C_ID = C.C_ID JOIN Employee AS E ON C.E_ID = E.E_ID JOIN Store AS S ON S.MGR_ID = E.MGR_ID GROUP BY
E.MGR_ID ORDER BY SUM(O.O_Amount) DESC;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

Store_ID	Name	MGR_ID	Total_Sales
6002	Jayanagar	1007	86242
6001	Bannerghatta	1002	69848
6004	Malleshwaram	1010	41596
6003	Rajajinagar	1013	17545

c. Count the total number of customers according to their qualifications

QUERY :

```
SELECT Qualification, COUNT(qualification)
FROM CUSTOMERS
GROUP BY Qualification;
```

SCREENSHOT :

Showing rows 0 - 6 (7 total, Query took 0.0004 seconds.)

```
SELECT Qualification, COUNT(qualification) FROM CUSTOMERS GROUP BY Qualification;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

Qualification	COUNT(qualification)
Activist	2
Doctor	4
Driver	2
Engineer	3
Software Engineer	2
Student	4
Teacher	4

d. Find the customers with highest and lowest number of order amount

QUERY :

```
SELECT O.C_ID, sum(O.O_Amount) AS MAXIMUM
FROM ORDERS AS O
GROUP BY O.C_ID
ORDER BY sum(O.O_Amount) DESC
LIMIT 1;

SELECT O.C_ID, sum(O.O_Amount) AS MINIMUM
FROM ORDERS AS O
GROUP BY O.C_ID
ORDER BY sum(O.O_Amount) ASC
LIMIT 1;
```

SCREENSHOT :

e. Retrieve the maximum and minimum orders by each and every customer.

QUERY :

```
SELECT O.C_ID, MAX(O.O_Amount) AS MAXIMUM,  MIN(O.O_Amount)
FROM ORDERS AS O
GROUP BY O.C_ID
ORDER BY (O.O_Amount) ASC
```

SCREENSHOT :

Showing rows 0 - 20 (21 total, Query took 0.0008 seconds.)

```
SELECT O.C_ID, MAX(O.O_Amount) AS MAXIMUM, MIN(O.O_Amount) FROM ORDERS AS O GROUP BY O.C_ID ORDER BY (O.O_Amount) ASC;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

	C_ID	MAXIMUM	MIN(O.O_Amount)
Edit Copy Delete	2013	1650	900
Edit Copy Delete	2012	5400	1000
Edit Copy Delete	2005	1750	1000
Edit Copy Delete	2002	2750	1600
Edit Copy Delete	2010	3150	2400
Edit Copy Delete	2008	18000	3000
Edit Copy Delete	2009	3300	3200
Edit Copy Delete	2020	15750	3750

	C_ID	MAXIMUM	MIN(O.O_Amount)
Edit Copy Delete	2011	8393	4797
Edit Copy Delete	2015	8000	5000
Edit Copy Delete	2007	21000	4797
Edit Copy Delete	2006	9000	5600
Edit Copy Delete	2018	5996	4000
Edit Copy Delete	2004	6000	3500
Edit Copy Delete	2014	6400	6400
Edit Copy Delete	2003	8000	2500
Edit Copy Delete	2016	15000	3500
Edit Copy Delete	2021	15998	15998
Edit Copy Delete	2019	16000	1500
Edit Copy Delete	2001	16000	7000
Edit Copy Delete	2017	30000	2250

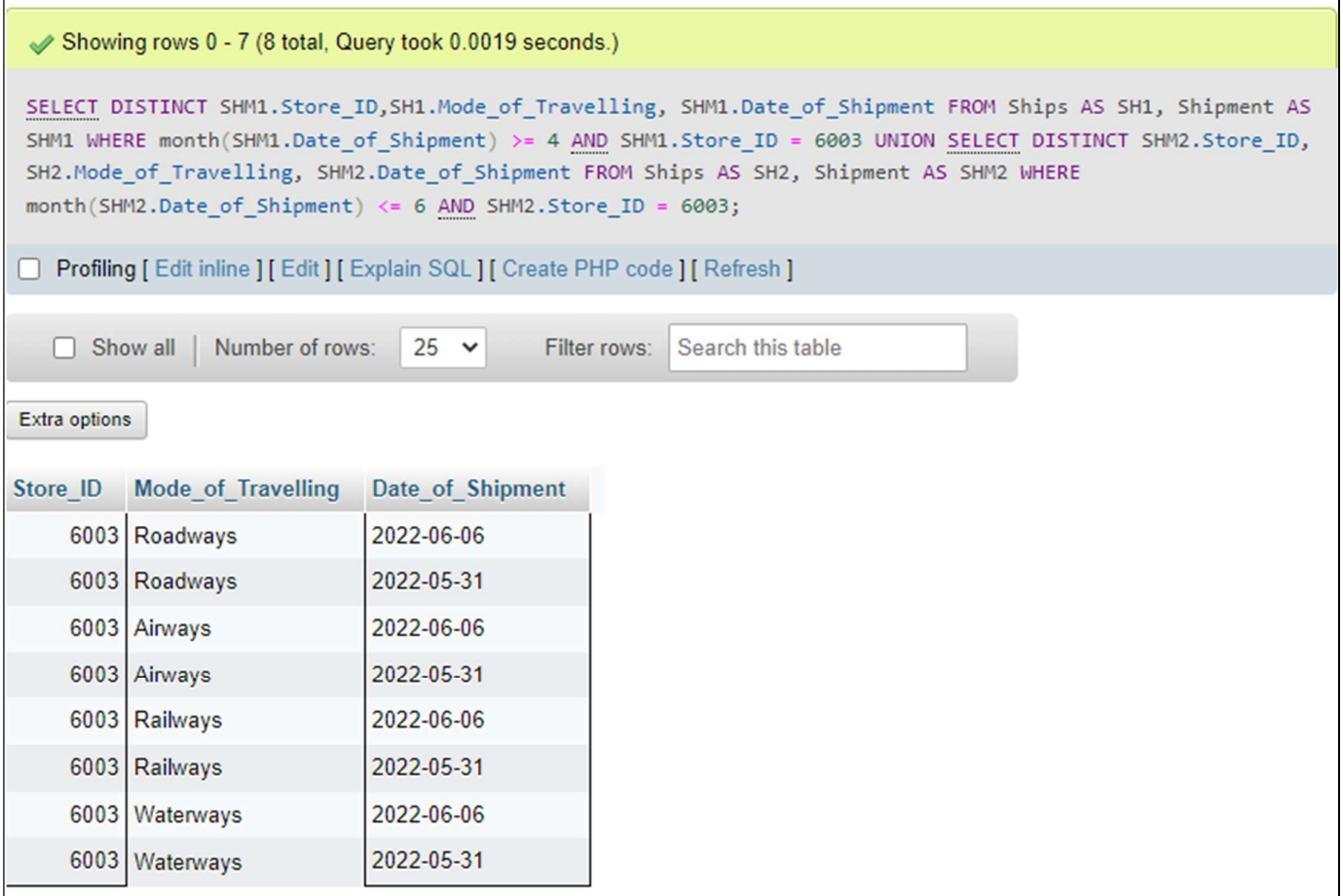
8. SET OPERATIONS

a. List the mode of travelling to store 6003 between April and June.

QUERY :

```
SELECT DISTINCT SHM1.Store_ID,SH1.Mode_of_Travelling,
SHM1.Date_of_Shipment
FROM Ships AS SH1, Shipment AS SHM1
WHERE month(SHM1.Date_of_Shipment) >= 4 AND SHM1.Store_ID =
6003
UNION
SELECT DISTINCT SHM2.Store_ID, SH2.Mode_of_Travelling,
SHM2.Date_of_Shipment
FROM Ships AS SH2, Shipment AS SHM2
WHERE month(SHM2.Date_of_Shipment) <= 6 AND SHM2.Store_ID =
6003;
```

SCREENSHOT :



Showing rows 0 - 7 (8 total, Query took 0.0019 seconds.)

```
SELECT DISTINCT SHM1.Store_ID,SH1.Mode_of_Travelling, SHM1.Date_of_Shipment FROM Ships AS SH1, Shipment AS
SHM1 WHERE month(SHM1.Date_of_Shipment) >= 4 AND SHM1.Store_ID = 6003 UNION SELECT DISTINCT SHM2.Store_ID,
SH2.Mode_of_Travelling, SHM2.Date_of_Shipment FROM Ships AS SH2, Shipment AS SHM2 WHERE
month(SHM2.Date_of_Shipment) <= 6 AND SHM2.Store_ID = 6003;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

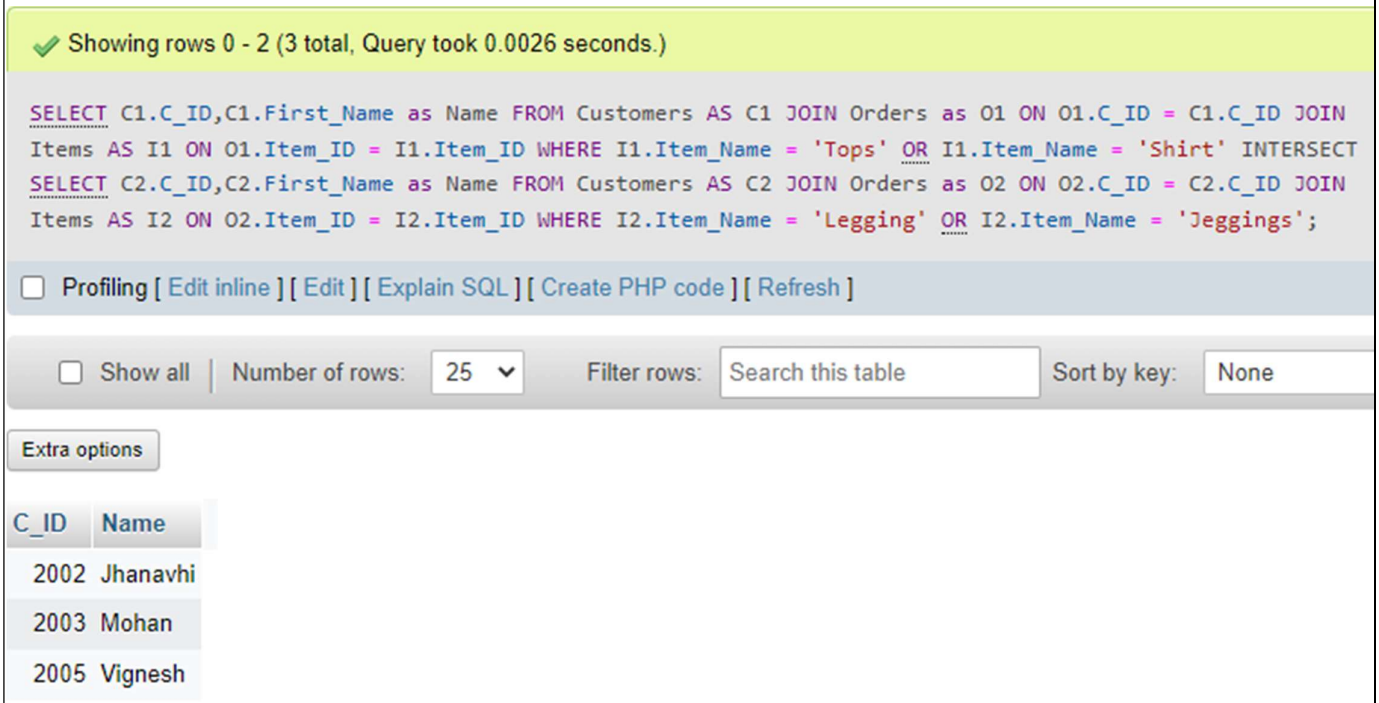
Store_ID	Mode_of_Travelling	Date_of_Shipment
6003	Roadways	2022-06-06
6003	Roadways	2022-05-31
6003	Airways	2022-06-06
6003	Airways	2022-05-31
6003	Railways	2022-06-06
6003	Railways	2022-05-31
6003	Waterways	2022-06-06
6003	Waterways	2022-05-31

b. List the customers who bought both Tops/Shirt and Leggings/Jeggings

QUERY :

```
SELECT C1.C_ID,C1.First_Name as Name
FROM Customers AS C1 JOIN Orders as O1 ON O1.C_ID = C1.C_ID
JOIN Items AS I1 ON O1.Item_ID = I1.Item_ID
WHERE I1.Item_Name = 'Tops' OR I1.Item_Name = 'Shirt'
INTERSECT
SELECT C2.C_ID,C2.First_Name as Name
FROM Customers AS C2 JOIN Orders as O2 ON O2.C_ID = C2.C_ID
JOIN Items AS I2 ON O2.Item_ID = I2.Item_ID
WHERE I2.Item_Name = 'Legging' OR I2.Item_Name = 'Jeggings';
```

SCREENSHOT :

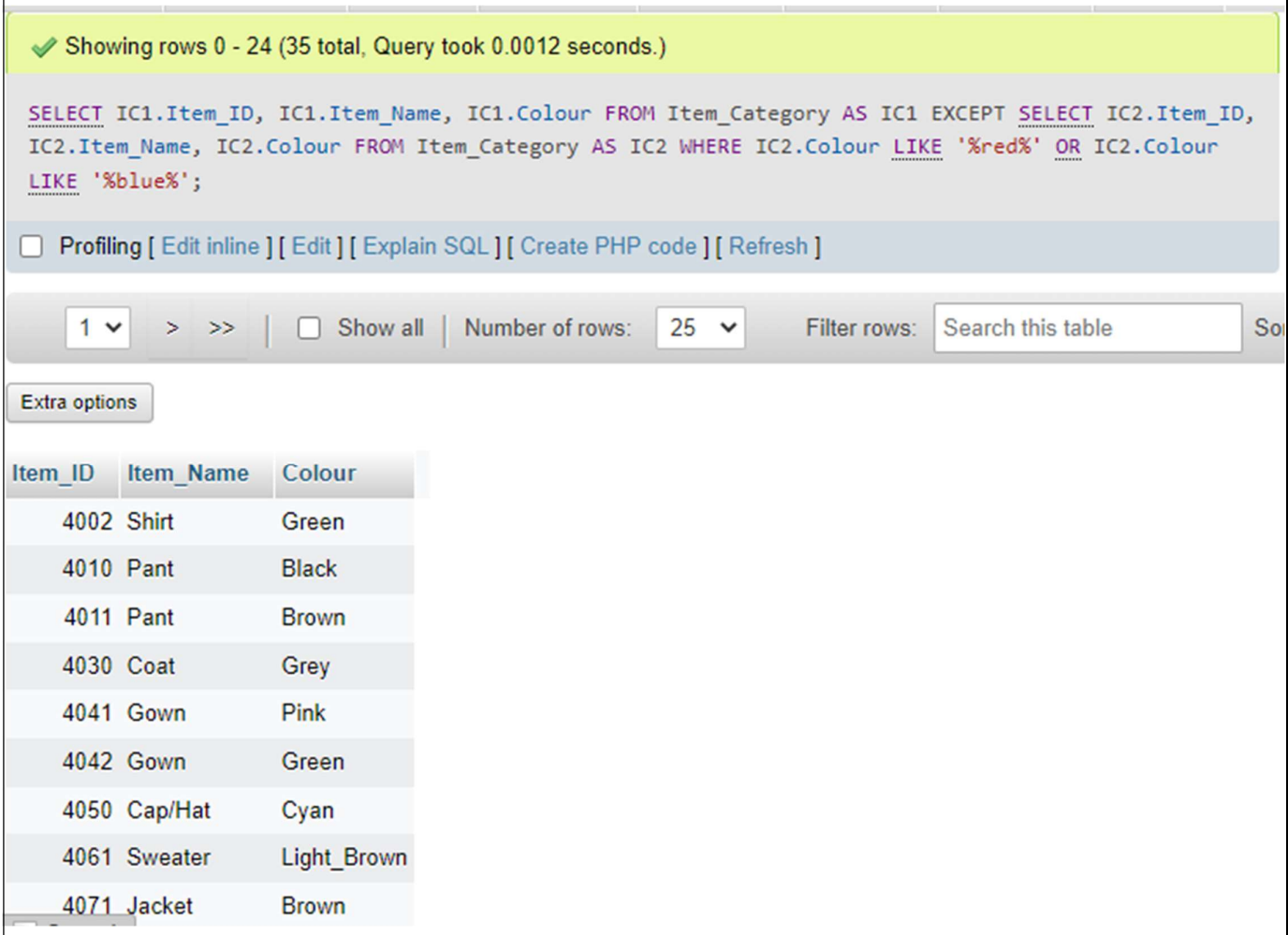


c. Details of dresses of all colours except 'red' and 'blue'

QUERY :

```
SELECT IC1.Item_ID, IC1.Item_Name, IC1.Colour
FROM Item_Category AS IC1
EXCEPT
SELECT IC2.Item_ID, IC2.Item_Name, IC2.Colour
FROM Item_Category AS IC2
WHERE IC2.Colour LIKE '%red%' OR IC2.Colour LIKE '%blue%'
```

SCREENSHOT :



Item_ID	Item_Name	Colour
4071	Jacket	Brown
4080	Legging	Maroon
4081	Legging	White
4082	Legging	Black
4090	Jeggings	Purple
4091	Jeggings	Grey
4092	Jeggings	Yellow
4100	Tops	Black
4110	Saree	Pink
4111	Saree	Purple

Item_ID	Item_Name	Colour
4111	Saree	Purple
4120	Chudidhar	Yellow
4121	Chudidhar	Orange
4122	Chudidhar	Green
4130	Frock	Rainbow
4131	Frock	Violet
4140	Lehenga	Baby_Pink
4150	Dhoti	White
4151	Dhoti	Light_Brown
4161	Tshirt	Black
4162	Tshirt	Brown
4163	Tshirt	Grey
4170	Shorts	Green
4180	Skirt	Light_Green
4181	Skirt	Pink
4190	Pyjama	Grey
4200	Kurta	White
[illegible]Palazo		Pink

Console

d. Find all items which are bought in September or October-2022

QUERY :

```
SELECT I1.Item_ID, I1.Item_Name, O1.O_Date
FROM Orders AS O1 JOIN Items as I1 ON O1.Item_ID =
I1.Item_ID
WHERE month(O1.O_Date) = 9
UNION ALL
SELECT I2.Item_ID, I2.Item_Name, O2.O_Date
FROM Orders AS O2 JOIN Items as I2 ON O2.Item_ID =
I2.Item_ID
WHERE month(O2.O_Date) = 10;
```

SCREENSHOT :

Showing rows 0 - 24 (28 total, Query took 0.0005 seconds.)

```
SELECT I1.Item_ID, I1.Item_Name, O1.O_Date FROM Orders AS O1 JOIN Items as I1 ON O1.Item_ID =
I1.Item_ID WHERE month(O1.O_Date) = 9 UNION ALL SELECT I2.Item_ID, I2.Item_Name, O2.O_Date FROM
Orders AS O2 JOIN Items as I2 ON O2.Item_ID = I2.Item_ID WHERE month(O2.O_Date) = 10;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

1 > >> | Show all | Number of rows: 25 Filter rows: Search this table So

Extra options

Item_ID	Item_Name	O_Date
4162	Tshirt	2022-09-12
4002	Shirt	2022-09-10
4081	Legging	2022-09-10
4082	Legging	2022-09-10

Item_ID	Item_Name	O_Date
4082	Legging	2022-09-10
4070	Jacket	2022-09-18
4102	Tops	2022-09-18
4030	Coat	2022-09-23
4100	Tops	2022-09-23
4041	Gown	2022-09-26
4050	Cap/Hat	2022-09-26
4090	Jeggings	2022-09-26
4002	Shirt	2022-10-16
4102	Tops	2022-10-16
4121	Chudidhar	2022-10-16
4000	Shirt	2022-10-25

Item_ID	Item_Name	O_Date
4000	Shirt	2022-10-25
4090	Jeggings	2022-10-25
4151	Dhoti	2022-10-25
4171	Shorts	2022-10-25
4060	Sweater	2022-10-17
4061	Sweater	2022-10-17
4091	Jeggings	2022-10-17
4121	Chudidhar	2022-10-17
4002	Shirt	2022-10-30
4011	Pant	2022-10-30
4111	Saree	2022-10-30

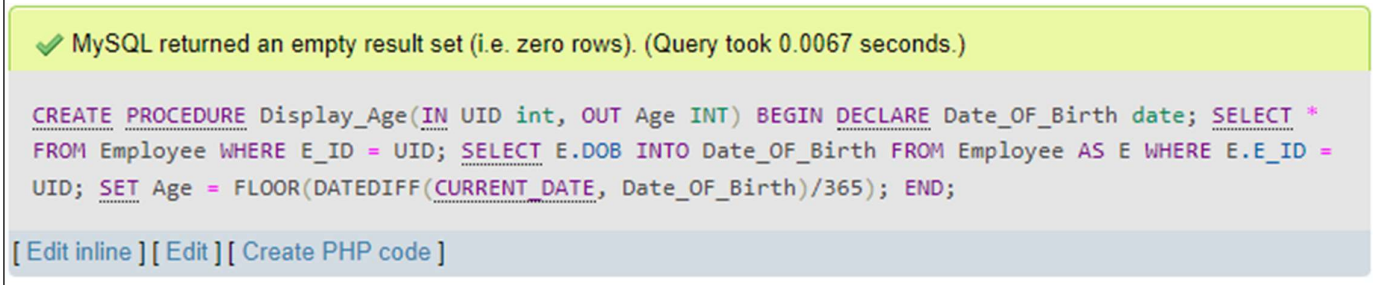
9. PROCEDURES AND FUNCTIONS

a. PROCEDURE to display the age of an employee provided the Employee ID and this procedure returns age as the output

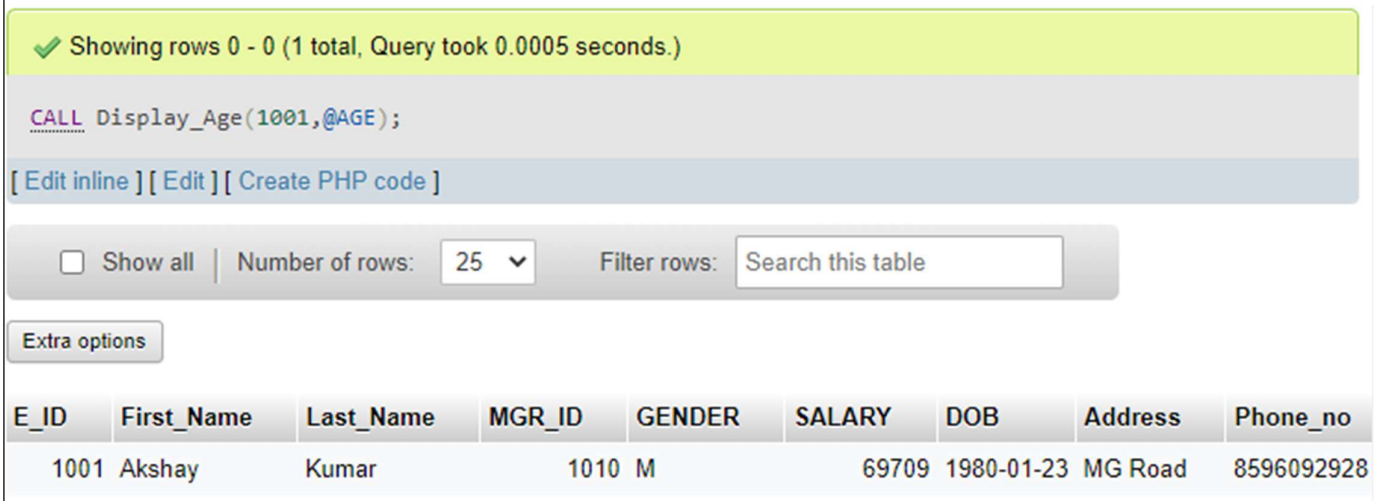
QUERY :

```
DELIMITER &&
CREATE PROCEDURE Display_Age(IN UID int, OUT Age INT)
BEGIN
    DECLARE Date_OF_Birth date;
    SELECT * FROM Employee WHERE E_ID = UID;
    SELECT E.DOB INTO Date_OF_Birth
    FROM Employee AS E
    WHERE E.E_ID = UID;
    SET Age = FLOOR(DATEDIFF(CURRENT_DATE,
Date_OF_Birth)/365);
END &&
DELIMITER ;
```

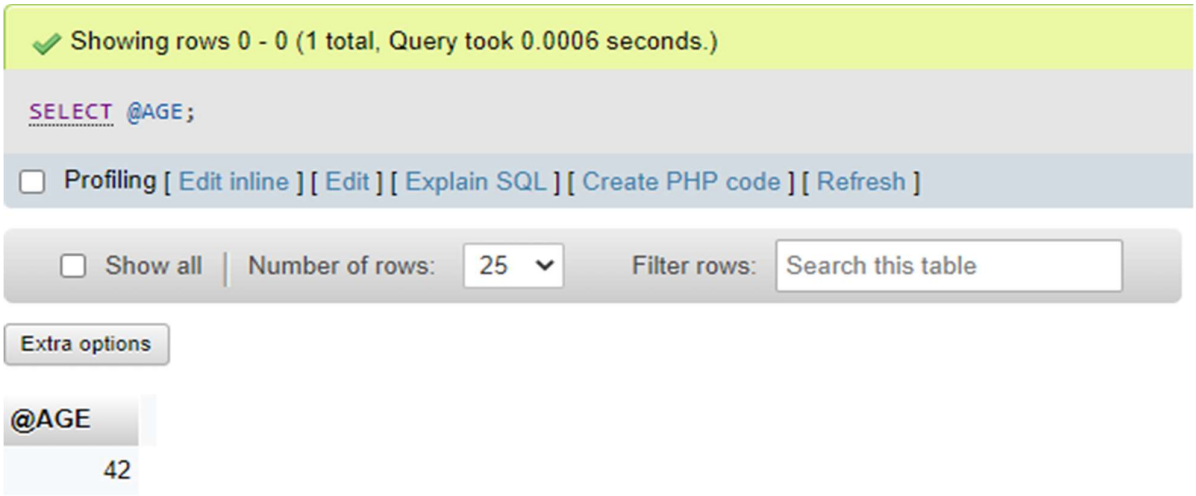
SCREENSHOT :



CALLING THE PROCEDURE :-



E_ID	First_Name	Last_Name	MGR_ID	GENDER	SALARY	DOB	Address	Phone_no
1001	Akshay	Kumar	1010	M	69709	1980-01-23	MG Road	8596092928



b. PROCEDURE to backup the table after deleting the customer

NOTE : Solution done in the Triggers part as problem statement is more relevant there

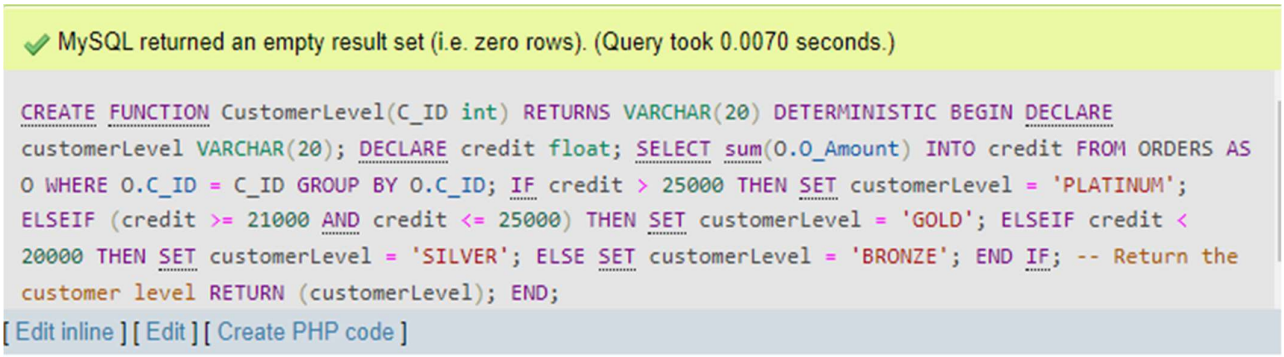
c. FUNCTION to display the level of a customer as PLATINUM, GOLD, SILVER, BRONZE based on his/her total purchase

QUERY :

```
DELIMITER $$

CREATE FUNCTION CustomerLevel(C_ID int)
RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
    DECLARE customerLevel VARCHAR(20);
    DECLARE credit float;
    SELECT sum(O.O_Amount) INTO credit
    FROM ORDERS AS O
    WHERE O.C_ID = C_ID
    GROUP BY O.C_ID;


    IF credit > 25000 THEN
```

```
        SET customerLevel = 'PLATINUM';
    ELSEIF (credit >= 21000 AND
            credit <= 25000) THEN
        SET customerLevel = 'GOLD';
    ELSEIF credit < 20000 THEN
        SET customerLevel = 'SILVER';
    ELSE
        SET customerLevel = 'BRONZE';
    END IF;
    -- Return the customer level
    RETURN (customerLevel);
END$$
DELIMITER ;
```

SCREENSHOT :



CALLING THE FUNCTION :-

Showing rows 0 - 20 (21 total, Query took 0.0016 seconds.)

```
SELECT `C_ID`,`First_Name`,`Last_Name`, CustomerLevel(`C_ID`) FROM `Customers`;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort

Extra options

	C_ID	First_Name	Last_Name	CustomerLevel(`C_ID`)
Edit Copy Delete	2001	Sujatha	Mohan	PLATINUM
Edit Copy Delete	2002	Jhanavhi		SILVER
Edit Copy Delete	2003	Mohan	Raj	BRONZE
Edit Copy Delete	2004	Adarsh	Liju	SILVER
Edit Copy Delete	2005	Vignesh	Sheshadri	SILVER
Edit Copy Delete	2006	Harsh	Chowdhary	GOLD
Edit Copy Delete	2007	Rohith	Jain	PLATINUM
Edit Copy Delete	2008	Himanshu		PLATINUM

	C_ID	First_Name	Last_Name	CustomerLevel(`C_ID`)
Edit Copy Delete	2009	Sutharsan	Raj	SILVER
Edit Copy Delete	2010	Kavi	Priya	SILVER
Edit Copy Delete	2011	Varuna	Shree	SILVER
Edit Copy Delete	2012	Gopinath	Gokul	SILVER
Edit Copy Delete	2013	Krishna	Kumar	SILVER
Edit Copy Delete	2014	Divya	Shree	SILVER
Edit Copy Delete	2015	Siddharth	Seetharaman	BRONZE
Edit Copy Delete	2016	Gokul	Sreenath	PLATINUM
Edit Copy Delete	2017	Ramesh	Agarwal	PLATINUM
Edit Copy Delete	2018	Suresh	Sathish	SILVER
Edit Copy Delete	2019	Om	Katkam	SILVER
Edit Copy Delete	2020	Shashank	Singh	GOLD
Edit Copy Delete	2021	ABC	XYZ	SILVER

d. FUNCTION to give discount based on the total amount of the bill

QUERY :

```
DELIMITER $$
CREATE FUNCTION Discount(C_ID int)
RETURNS float
DETERMINISTIC
BEGIN
    DECLARE credit float;
    DECLARE discount float;
    DECLARE final_amount float;
    SELECT Amount INTO credit
    FROM Bill AS B
    WHERE B.C_ID = C_ID;


    IF credit > 25000 THEN
        SET final_amount = credit - (credit * 0.2);
        SET discount = credit * 0.2;
    ELSEIF (credit > 10000 AND credit <= 25000) THEN
        SET final_amount = credit - (credit * 0.1);
        SET discount = credit * 0.1;
    ELSEIF (credit > 5000 AND credit <= 10000) THEN
        SET final_amount = credit - (credit * 0.08);
        SET discount = credit * 0.08;
    ELSE
        SET final_amount = credit - (credit * 0.05);
        SET discount = credit * 0.05;
    END IF;
    RETURN (discount);
END$$
DELIMITER ;
SELECT B_ID,Amount,C_ID, Discount(C_ID) FROM Bill;
```

SCREENSHOT :

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0048 seconds.)

```
CREATE FUNCTION Discount(C_ID int) RETURNS float DETERMINISTIC BEGIN DECLARE credit float; DECLARE discount
float; DECLARE final_amount float; SELECT Amount INTO credit FROM Bill AS B WHERE B.C_ID = C_ID; IF credit >
25000 THEN SET final_amount = credit - (credit * 0.2); SET discount = credit * 0.2; ELSEIF (credit > 10000 AND
credit <= 25000) THEN SET final_amount = credit - (credit * 0.1); SET discount = credit * 0.1; ELSEIF (credit >
5000 AND credit <= 10000) THEN SET final_amount = credit - (credit * 0.08); SET discount = credit * 0.08; ELSE
SET final_amount = credit - (credit * 0.05); SET discount = credit * 0.05; END IF; RETURN (discount); END;
```

[Edit inline] [Edit] [Create PHP code]

CALLING THE FUNCTION :

Showing rows 0 - 20 (21 total, Query took 0.0023 seconds.)

```
SELECT B_ID,Amount,C_ID, Discount(C_ID) FROM Bill;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

	B_ID	Amount	C_ID	Discount(C_ID)
Edit Copy Delete	9001	25997	2001	5199.4
Edit Copy Delete	9002	8348	2002	667.84
Edit Copy Delete	9003	20800	2003	2080
Edit Copy Delete	9004	9500	2004	760
Edit Copy Delete	9005	4250	2005	212.5
Edit Copy Delete	9006	22592	2006	2259.2
Edit Copy Delete	9007	30797	2007	6159.4
Edit Copy Delete	9008	28200	2008	5640
Edit Copy Delete	9009	6500	2009	520

	B_ID	Amount	C_ID	Discount(C_ID)
Edit Copy Delete	9009	6500	2009	520
Edit Copy Delete	9010	7350	2010	588
Edit Copy Delete	9011	15588	2011	1558.8
Edit Copy Delete	9012	14995	2012	1499.5
Edit Copy Delete	9013	10250	2013	1025
Edit Copy Delete	9014	11200	2014	1120
Edit Copy Delete	9015	20500	2015	2050
Edit Copy Delete	9016	29300	2016	5860
Edit Copy Delete	9017	32250	2017	6450
Edit Copy Delete	9018	14396	2018	1439.6
Edit Copy Delete	9019	19600	2019	1960
Edit Copy Delete	9020	24900	2020	2490
Edit Copy Delete	9022	10597	2022	1059.7

Check all *With selected:* Edit Copy Delete Export

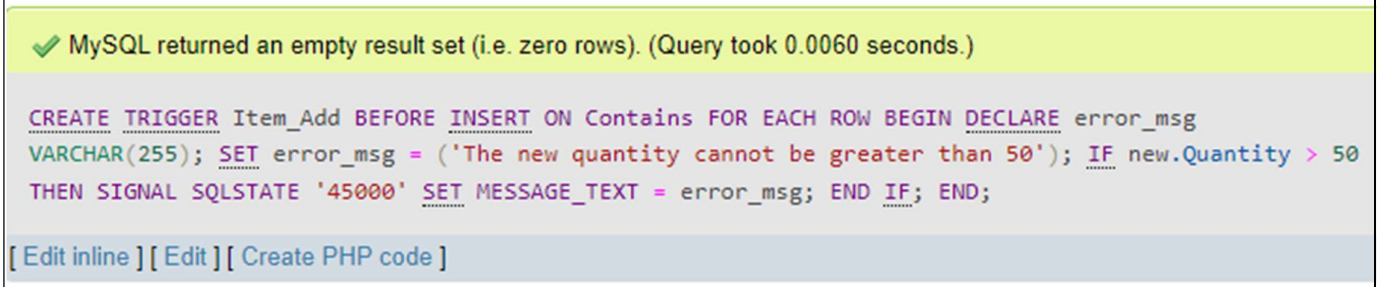
10. TRIGGERS AND CURSORS

a. Set a TRIGGER to check if the quantity of an item exceeds the size 50 in a store. If so, return an error message.

QUERY :

```
DELIMITER $$
CREATE TRIGGER Item_Add
BEFORE INSERT
ON Contains FOR EACH ROW
BEGIN
    DECLARE error_msg VARCHAR(255);
    SET error_msg = ('The new quantity cannot be greater
than 50');
    IF new.Quantity > 50 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = error_msg;
    END IF;
END $$
DELIMITER ;
```

SCREENSHOT :



WRONG CONDITION :

Error

SQL query: Copy

```
INSERT INTO `Contains`(`Store_ID`,`Item_ID`,`Quantity`)
VALUES (6004,4300,60);
```

MySQL said:

```
#1644 - The new quantity cannot be greater than 50
```

CORRECT CONDITION :

1 row inserted. (Query took 0.0004 seconds.)

```
INSERT INTO `Contains`(`Store_ID`,`Item_ID`,`Quantity`) VALUES (6004,4300,40);
```

[Edit inline] [Edit] [Create PHP code]

b. Set a TRIGGER AND CURSOR by calling a stored procedure to create a backup of customer and his associated details after removing a customer from the database.

QUERY :

```
-- CREATE A BACKUP TABLE
CREATE TABLE `Customer_Backuplog` (
  `C_ID` int(11) NOT NULL,
  `First_Name` varchar(50) DEFAULT NULL,
  `Last_Name` varchar(50) NOT NULL,
  `Qualification` varchar(20) DEFAULT NULL,
  `ADDRESS` varchar(50) DEFAULT NULL,
  `Locality` varchar(20) DEFAULT NULL,
  `CITY` varchar(20) DEFAULT NULL,
  `Email` varchar(50) DEFAULT NULL,
  `Phone_NO` varchar(10) DEFAULT NULL,
  `DOP` date DEFAULT NULL,
```

```
  `E_ID` int(11) DEFAULT NULL,
  `Store_ID` int DEFAULT NULL,
  `Item_ID` int DEFAULT NULL,
  `Quantity` int DEFAULT NULL,
  `O_Amount` float DEFAULT NULL);

-- STORED PROCEDURE TO FORM A CURSOR AND DELETE THE CUSTOMER

DELIMITER $$
CREATE procedure Delete_Customer(C_ID_val int)
BEGIN
DECLARE done INT DEFAULT 0;
DECLARE val int;
DECLARE `C_ID` int;
DECLARE `First_Name` varchar(50);
DECLARE `Last_Name` varchar(50);
DECLARE `Qualification` varchar(20);
DECLARE `ADDRESS` varchar(50);
DECLARE `Locality` varchar(20);
DECLARE `CITY` varchar(20);
DECLARE `Email` varchar(50);
DECLARE `Phone_NO` varchar(10);
DECLARE `DOP` date;
DECLARE `E_ID` int(11);
DECLARE `Item_ID` int;
DECLARE `Price` float;
DECLARE `Quantity` int;
DECLARE `O_Amount` float;

DECLARE cur CURSOR FOR SELECT
C.C_ID, C.First_Name, C.Last_Name, C.Qualification,
C.Address, C.Locality, C.City, C.Email, C.Phone_NO, C.DOP,
C.E_ID,
O.Item_ID, O.Price, O.Quantity, O.O_Amount
FROM Customers AS C, Orders as O
WHERE O.C_ID = C_ID_val AND C.C_ID = C_ID_val;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
```

```
OPEN cur;
LABEL:loop

FETCH cur INTO  C_ID, First_Name, Last_Name, Qualification,
Address,
Locality, City, Email, Phone_NO, DOP, E_ID,
Item_ID, Price, Quantity, O_Amount;

INSERT INTO Customer_Backuplog VALUES(C_ID, First_Name,
Last_Name, Qualification, Address,
Locality, City, Email, Phone_NO, DOP, E_ID,
Item_ID, Price, Quantity, O_Amount);

IF done = 1 THEN LEAVE LABEL;
END IF;
END loop;
CLOSE cur;
SET val = (select count(*) from Customer_Backuplog where
Customer_Backuplog.C_ID = C_ID_val);
IF val > 0 THEN
    DELETE FROM Orders
    WHERE Orders.C_ID = C_ID_val;
END IF;
END $$
DELIMITER ;

-- TRIGGER TO DELETE THE CUSTOMER

DELIMITER $$
CREATE TRIGGER Before_Delete_CustomerInfo
BEFORE DELETE
ON Customers FOR EACH ROW
BEGIN
CALL Delete_Customer(OLD.C_ID);
END$$
DELIMITER ;
```

SCREENSHOT :

Creating a backup table

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0004 seconds.)

```
CREATE TABLE `Customer_Backuplog` ( `C_ID` int(11) NOT NULL, `First_Name` varchar(50) DEFAULT NULL, `Last_Name` varchar(50) NOT NULL, `Qualification` varchar(20) DEFAULT NULL, `ADDRESS` varchar(50) DEFAULT NULL, `Locality` varchar(20) DEFAULT NULL, `CITY` varchar(20) DEFAULT NULL, `Email` varchar(50) DEFAULT NULL, `Phone_NO` varchar(10) DEFAULT NULL, `DOP` date DEFAULT NULL, `E_ID` int(11) DEFAULT NULL, `Store_ID` int DEFAULT NULL, `Item_ID` int DEFAULT NULL, `Quantity` int DEFAULT NULL, `O_Amount` float DEFAULT NULL);
```

[Edit inline] [Edit] [Create PHP code]

Procedure and Cursor

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0066 seconds.)

```
CREATE procedure Delete_Customer(C_ID_val int) BEGIN DECLARE done INT DEFAULT 0; DECLARE val int; DECLARE `C_ID` int; DECLARE `First_Name` varchar(50); DECLARE `Last_Name` varchar(50); DECLARE `Qualification` varchar(20); DECLARE `ADDRESS` varchar(50); DECLARE `Locality` varchar(20); DECLARE `CITY` varchar(20); DECLARE `Email` varchar(50); DECLARE `Phone_NO` varchar(10); DECLARE `DOP` date; DECLARE `E_ID` int(11); DECLARE `Item_ID` int; DECLARE `Quantity` int; DECLARE `O_Amount` float; DECLARE cur CURSOR FOR SELECT C.C_ID, C.First_Name, C.Last_Name, C.Qualification, C.Address, C.Locality, C.City, C.Email, C.Phone_NO, C.DOP, C.E_ID, O.Item_ID, O.Quantity, O.O_Amount FROM Customers AS C, Orders as O WHERE O.C_ID = C_ID_val AND C.C_ID = C_ID_val; DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1; OPEN cur; LABEL:loop FETCH cur INTO C_ID, First_Name,
```

[Edit]

Trigger

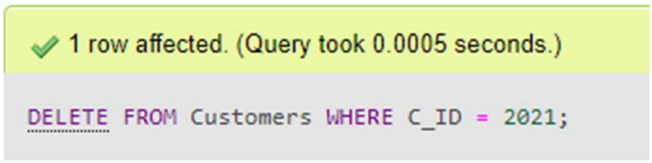
MySQL returned an empty result set (i.e. zero rows). (Query took 0.0076 seconds.)

```
CREATE TRIGGER Before_Delete_CustomerInfo BEFORE DELETE ON Customers FOR EACH ROW BEGIN CALL Delete_Customer(OLD.C_ID); END;
```

Table before deleting : C_ID = 2021

C_ID	First_Name	Last_Name	Qualification	ADDRESS	Locality	CITY	Email	Phone_NO	DOP	E_ID
2019	Om	Katkam	Driver	Brigade Millenium	Gottigere	Bangalore	Om2022 @gmail.com	1298765235	2022-09-26	1003
2020	Shashank	Singh	Driver	Pride Apartments	Arekere	Bangalore	Shashank2022 @gmail.com	9876542345	2022-08-14	1006
2021	ABC	XYZ	Activist	Banashankari	Bannerghatta	Bangalore	abc@gmail.com	9876543219	2022-11-02	1005

Deleting the Customer : C_ID = 2021



BackUp with the relevant details of the customer in the backuplog table :-

Showing rows 0 - 1 (2 total, Query took 0.0003 seconds.)

```
SELECT * FROM `customer_backuplog`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

C_ID	First_Name	Last_Name	Qualification	ADDRESS	Locality	CITY	Email	Phone_NO	DOP	E_ID	Store_ID	Item_ID	Quantity
2021	ABC	XYZ	Activist	Banashankari	Bannerghatta	Bangalore	abc@gmail.com	9876543219	2022-11-02	1005	4041	7999	2
2021	ABC	XYZ	Activist	Banashankari	Bannerghatta	Bangalore	abc@gmail.com	9876543219	2022-11-02	1005	4041	7999	2

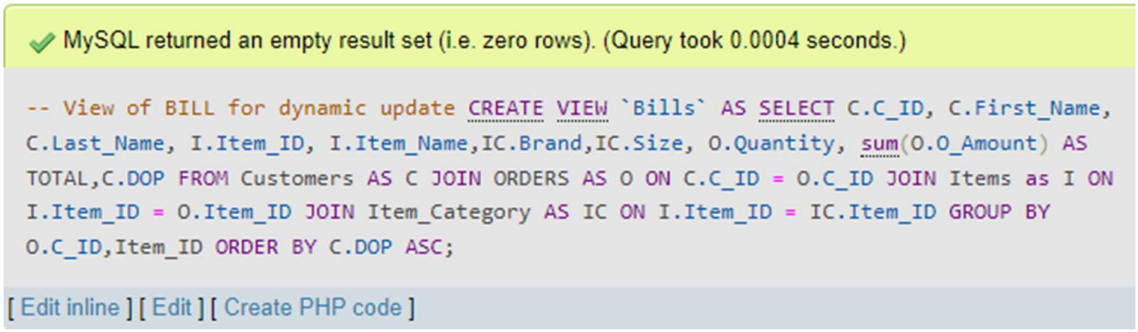
11. VIEWS

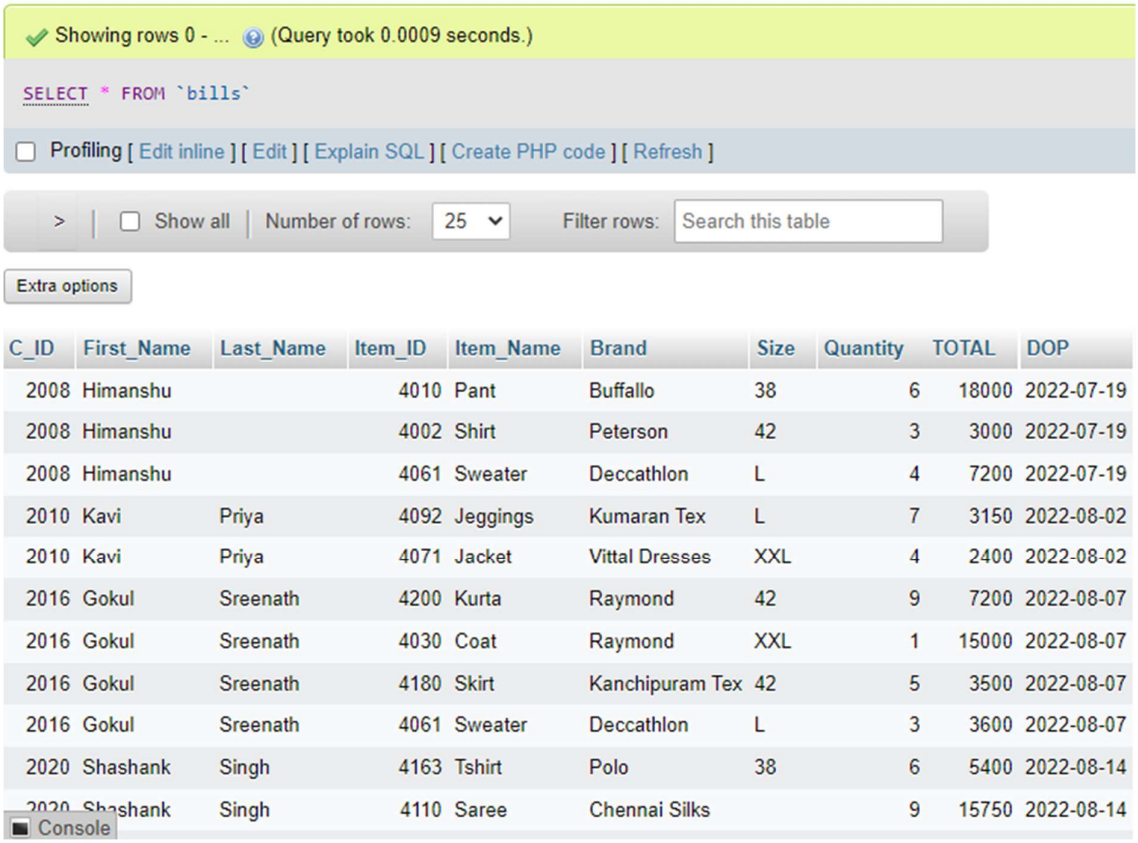
a. VIEW of BILL with dynamic update with each item and quantity mapped to each customer with total amount

QUERY:

```
-- View of BILL for dynamic update
CREATE VIEW `Bills` AS
SELECT C.C_ID, C.First_Name, C.Last_Name, I.Item_ID,
I.Item_Name,IC.Brand,IC.Size, O.Quantity, sum(O.O_Amount) AS
TOTAL,C.DOP
FROM Customers AS C JOIN ORDERS AS O ON C.C_ID = O.C_ID JOIN
Items as I ON I.Item_ID = O.Item_ID
JOIN Item_Category AS IC ON I.Item_ID = IC.Item_ID
GROUP BY O.C_ID,Item_ID
ORDER BY C.DOP ASC;
```

SCREENSHOT :





b. VIEW of BILL with dynamic update with each item and quantity mapped to each

QUERY :

```
CREATE VIEW `Total_Orders` AS
SELECT O.Item_ID,O.C_ID,I.Price,O.Quantity,(O.quantity *
I.price) AS O_Amount
FROM Orders AS O JOIN Items AS I
on O.Item_ID = I.Item_ID;
```

SCREENSHOT :

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0004 seconds.)

```
CREATE VIEW `Total_Orders` AS SELECT O.Item_ID,O.C_ID,I.Price,O.Quantity,(O.quantity *
I.price) AS O_Amount FROM Orders AS O JOIN Items AS I on O.Item_ID = I.Item_ID;
```

[Edit inline] [Edit] [Create PHP code]

Showing rows 0 - ... (Query took 0.0004 seconds.)

```
SELECT * FROM `total_orders`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

> | Show all | Number of rows: 25 | Filter rows: Search this table

Extra options

	Item_ID	C_ID	Price	Quantity	O_Amount
Edit Copy Delete	4011	2001	4000	4	16000
Edit Copy Delete	4090	2001	700	10	7000
Edit Copy Delete	4162	2001	1000	8	8000
Edit Copy Delete	4002	2002	999	2	1998
Edit Copy Delete	4080	2002	400	4	1600
Edit Copy Delete	4102	2002	549	5	2745
Edit Copy Delete	4121	2002	999	2	1998
Edit Copy Delete	4000	2003	900	10	9000
Edit Copy Delete	4090	2003	700	7	4900

12. DEVELOPING A FRONTEND

APP.PY : The main interface of the web app

```
# Importing pakages
import streamlit as st
import mysql.connector

from create import *
from database import *
from delete import *
from read import *
from update import *

# mydb = mysql.connector.connect(
#     host = "localhost",
#     user = "root",
#     password = ""
# )
# c = mydb.cursor()
# c.execute("USE Textile_362")


def main():
    st.set_page_config(
    page_title = "Textile 362",
    # page_icon = "📦",
    layout = "wide",
    initial_sidebar_state = "expanded",
    )
    # st.snow()
    # st.balloons()
    st.image('./Header.png')
    st.sidebar.image("Fashion.png")
    st.title("SMS TEXTILE STORES - 362")
    #st.write("-----------------------------------------
---------")
    menu = ["ADD", "VIEW", "EDIT", "REMOVE"]
    st.sidebar.header("MENU")
```

```
    ch =
st.sidebar.selectbox("OPTIONS",["CUSTOMER","EMPLOYEE","ITEMS
-STOCK","SUPPLIER"])
    option = st.sidebar.selectbox("ACTION", menu)
    ch2 = st.sidebar.selectbox("RUN ANY QUERY",["Click
here/Scroll Down \u2193"])
    if ch == "CUSTOMER":
        cu_create_table()
        if option == "ADD":
            st.subheader("Enter CUSTOMER Details :")
            cu_create()

        elif option == "VIEW":
            st.subheader("View added CUSTOMER :")
            cu_read()

        elif option == "EDIT":
            st.subheader("Update CUSTOMER Details :")
            cu_update()

        elif option == "REMOVE":
            st.subheader("Delete CUSTOMER Details :")
            cu_delete()

        else:
            st.subheader("About tasks")

##############################################################
################################################

    if ch == "EMPLOYEE":
        em_create_table()
        if option == "ADD":
            st.subheader("Enter EMPLOYEE Details :")
            em_create()

        elif option == "VIEW":
```

```
            st.subheader("View added EMPLOYEE :")
            em_read()

        elif option == "EDIT":
            st.subheader("Update EMPLOYEE Details :")
            em_update()

        elif option == "REMOVE":
            st.subheader("Delete EMPLOYEE Details :")
            em_delete()

        else:
            st.subheader("About tasks")

##################################################################
################################################

    if ch == "ITEMS-STOCK":
        it_create_table()
        if option == "ADD":
            st.subheader("Enter ITEM Details :")
            it_create()

        elif option == "VIEW":
            st.subheader("View added ITEMS :")
            it_read()

        elif option == "EDIT":
            st.subheader("Update ITEM Details :")
            it_update()

        elif option == "REMOVE":
            st.subheader("Delete ITEM Details :")
            it_delete()

        else:
            st.subheader("About tasks")
```

```
#############################################################
#################################################

    if ch == "SUPPLIER":
        su_create_table()
        if option == "ADD":
            st.subheader("Enter SUPPLIER AND SHIPPING
Details :")
            su_create()

        elif option == "VIEW":
            st.subheader("View added SUPPLIER :")
            su_read()

        elif option == "EDIT":
            st.subheader("Update SUPPLIER AND SHIPPING
Details :")
            su_update()

        elif option == "REMOVE":
            st.subheader("Delete SUPPLIER Details :")
            su_delete()

        else:
            st.subheader("About tasks")

#############################################################
#################################################
    def any_query_data(query):
        conn = mysql.connector.connect(
            host = "localhost",
            user = "root",
            password = "",
            database = "textile_362"
            )
        cur = conn.cursor()
        cur.execute(query)
        result = cur.fetchall()
```

```
        conn.commit()
        conn.close()
        return result
    def any_query():
        st.subheader("RUN ANY QUERY")
        query = st.text_area("Enter your query here",value =
"SELECT * FROM BILL LEFT OUTER JOIN CUSTOMERS ON BILL.C_ID =
CUSTOMERS.C_ID")
        if st.button("Run Query"):
            result = any_query_data(query)
            df = pd.DataFrame(result)
            st.dataframe(df)
    st.write("-----------------------------------------------
--------")
    if ch2 == "Click here/Scroll Down \u2193":
        any_query()


if __name__ == '__main__':
    main()
```

CREATE.PY : Contains create operation for all the tables

```
import streamlit as st
from database import *


def cu_create():
    col1, col2, col3 = st.columns(3)
    with col1:
        c_id = st.number_input("Customer ID: ",min_value =
2000, max_value = 2999)
        cf_name = st.text_input("First Name: ")
        cl_name = st.text_input("Last Name: ")
        c_qual = st.text_input("Qualification: ")
        c_phone = st.text_input("Phone: ")
        b_bank = st.text_input("Bank: ")

    with col2:
```

```
        c_address = st.text_input("Address: ")
        c_locate = st.text_input("Locality: ")
        c_city = st.text_input("City: ")
        c_email = st.text_input("Email: ")
        c.execute("SELECT E_ID FROM Employee")
        emp_data = c.fetchall()
        e_data = []
        for i in emp_data:
            for j in i:
                e_data.append(int(j))
        c_emp = st.selectbox("Serving Employee", e_data)
        b_tid = st.text_input("Transaction ID: ")
        b_id = c_id + 7000

    with col3:
        c.execute("SELECT Item_ID FROM Items")
        data = c.fetchall()
        item_data = []
        for i in data:
            for j in i:
                item_data.append(int(j))
        c_item_id = st.selectbox("Item", item_data)
        c.execute("SELECT Item_Name,Brand,Size FROM
Item_Category WHERE Item_ID ={}".format(c_item_id))
        it_n = c.fetchall()
        in_data = []
        for i in it_n:
            for j in i:
                in_data.append((j))
        in_data = in_data[0] + " :: "+in_data[1] + " :: " +
in_data[2]
        st.write("Item Name :: Item Brand :: Item Size")
        st.markdown('`{}`'.format(in_data))

        c_qty = st.number_input("Quantity: ",min_value=1,
max_value=50)
        c.execute("SELECT Price FROM Items WHERE Item_ID =
{}".format(c_item_id))
```

```
        it_da = c.fetchall()
        price_data = []
        for i in it_da:
            for j in i:
                price_data.append(float(j))
        c_price = price_data[0]
        st.markdown("Price: ")
        st.write(c_price)
        c_dop = st.date_input("Purchase Date:")
        st.write("Total item order value")
        c_total = st.write(c_price * c_qty)
        c_total = c_price * c_qty

    c.execute("SELECT sum(O.O_Amount) AS Total_Bill FROM
ORDERS AS O WHERE O.C_ID = {} ORDER BY sum(O.O_Amount)
DESC".format(c_id))
    b_amt = c.fetchall()
    price_data = []
    for i in b_amt:
            for j in i:
                price_data.append((j))
    b_amount = price_data[0]
    with col3:
        st.markdown("Total Bill Amount: ")
        st.write(b_amount)

    with col1:
        if st.button("Add CUSTOMER"):
                cu_add_data(c_id,cf_name,cl_name,c_qual,c_ad
dress,c_locate,c_city,c_email,c_phone,c_dop,c_emp)
                st.success("Successfully added CUSTOMER:
{}".format(cf_name))
    with col2:
        if st.button("Add ORDERS"):
                cu_orders_add_data(c_id,c_item_id,c_price,c_
qty,c_dop,c_total)
                st.success("Successfully added ORDERS:
{}".format(c_item_id))
```

```
    if st.button("GENERATE BILL"):
        cu_bill(b_id,b_bank,c_dop,b_tid,b_amount,c_id)
        st.success("Successfully generated BILL ::
{}".format(b_id))
        view_bill(c_id)
        st.success("Successfully printed BILL ::
{}".format(b_id))

##################################################################
#################################################

def em_create():
    col1, col2 = st.columns(2)
    with col1:
        e_id = st.number_input("Employee ID: ",min_value =
1000,max_value = 1999)
        e_fn = st.text_input("First Name: ")
        e_ln = st.text_input("Last Name: ")
        e_address = st.text_input("Address: ")
        e_phone = st.text_input("Phone: ")
    with col2:
        c.execute("SELECT DISTINCT MGR_ID FROM EMPLOYEE
WHERE MGR_ID <> 0")
        data = c.fetchall()
        mgr_data = []
        for i in data:
            for j in i:
                mgr_data.append(int(j))
        emgr_id = st.selectbox("Manager", mgr_data)
        e_gender = st.radio("Gender",('M', 'F'))
        e_salary = st.number_input("Salary: ")
        e_dob = st.date_input("Date of Birth:")

    if st.button("Add Employee"):
        em_add_data(e_id,e_fn,e_ln,emgr_id,e_gender,e_salary
,e_dob,e_address,e_phone)
```

```
        st.success("Successfully added EMPLOYEE:
{}".format(e_fn))

##################################################################
#################################################

def it_create():
    col1, col2, col3 = st.columns(3)
    with col1:
        i_id = st.number_input("ITEM ID: ",min_value =
4000,max_value = 4999)
        i_name = st.text_input("Name: ")
        i_price = st.number_input("Price: ")

    with col2:
        i_brand = st.text_input("Brand: ")
        i_colour = st.text_input("Colour: ")
        i_size = st.selectbox("Size",
["NA","28","30","32","34","36","38","40","42","44","46","48"
,"50","52","54","56","58","60","M","L","XL","XXL","XXXL"])
    with col3:
        i_gender = st.radio("Gender",('M', 'F'))
        i_quantity = st.number_input("Quantity: ",min_value
= 1, max_value = 50)
        c.execute("SELECT STORE_ID FROM STORE")
        data = c.fetchall()
        store_data = []
        for i in data:
            for j in i:
                store_data.append(int(j))
        st_id = st.selectbox("STORE-ID",store_data)


    if st.button("Add ITEM-STOCK"):
        it_add_data(i_id,i_name,i_price,i_gender,i_brand,i_c
olour,i_size,i_quantity,st_id)
        st.success("Successfully added ITEM TO STORE:
{}".format(i_id))
```

```
#############################################################
#################################################

def su_create():
    col1, col2 = st.columns(2)
    with col1:
        su_id = st.number_input("Supplier ID: ",min_value =
7000,max_value = 7999)
        su_name = st.text_input("Name: ")
        su_address = st.text_input("Address: ")
        c.execute("SELECT STORE_ID FROM STORE")
        data = c.fetchall()
        store_data = []
        for i in data:
            for j in i:
                store_data.append(int(j))

        su_st_id = st.selectbox("STORE-ID",store_data)
    with col2:
        sh_id = st.number_input("Ship ID: ",min_value =
8000,max_value = 8999)
        sh_cost = st.number_input("Shipping Cost: ")
        sh_date = st.date_input("Date of Shipment: ")
        sh_mode = st.selectbox("Mode of Travel",
["Roadways","Railways","Airways","Waterways"])



    if st.button("Add SUPPLIER - SHIP"):
        su_add_data(su_id,su_name,su_address,sh_id,sh_cost,s
h_mode,sh_date,su_st_id)
        st.success("Successfully added SUPPLIER-SHIP:
{}".format(su_name))

#############################################################
#################################################
```

READ.PY : Contains methods for reading the values for all the tables

```
import pandas as pd
import streamlit as st
import plotly.express as px
from database import *


def cu_read():
    result = cu_view_all_data()
    # st.write(result)
    df = pd.DataFrame(result, columns = ['C_ID',
'First_Name','Last
Name','Qualification','Address','Locality','City','Email','P
hone_NO','DOP','E_ID','Order_C_ID','Item_ID','Price','Quanti
ty','O_Date','O_Amount'])
    with st.expander("View all Customers"):
        st.dataframe(df)
    with st.expander("Name"):
        task_df = df['First_Name'].value_counts().to_frame()
        task_df = task_df.reset_index()
        task_df.columns = ['First_Name', 'Count']
        st.dataframe(task_df)
        p1 = px.pie(task_df, names = 'First_Name', values =
'Count')
        st.plotly_chart(p1)

##################################################################
################################################

def em_read():
    result = em_view_all_data()
    # st.write(result)
    df = pd.DataFrame(result, columns = ['E_ID',
'First_Name','Last_Name','MGR_ID','GENDER','SALARY','DOB','A
DDRESS','Phone_NO'])
    with st.expander("View all Employees"):
        st.dataframe(df)
    with st.expander("First_Name"):
```

```
        task_df = df['First_Name'].value_counts().to_frame()
        task_df = task_df.reset_index()
        task_df.columns = ['First_Name', 'Count']
        st.dataframe(task_df)
        p1 = px.pie(task_df, names = 'First_Name', values =
'Count')
        st.plotly_chart(p1)

############################################################
##################################################

def it_read():
    result = it_view_all_data()
    # st.write(result)
    df = pd.DataFrame(result, columns = ['Item_ID',
'Item_Name','Gender','Brand','Colour','Size','Quantity','Sto
re_ID','Store_Name'])
    with st.expander("View all Items with their stores"):
        st.dataframe(df)
    with st.expander("Item_ID"):
        task_df =
df[['Item_ID','Quantity']].value_counts().to_frame()
        task_df = task_df.reset_index()
        task_df.columns = ['Item_ID', 'Quantity', 'Index']
        st.dataframe(task_df)
        p1 = px.pie(task_df, names = 'Item_ID', values =
'Quantity')
        st.plotly_chart(p1)

############################################################
##################################################

def su_read():
    result = su_view_all_data()
    # st.write(result)
    df = pd.DataFrame(result, columns =
['Supp_ID','Name','Address','Ship_ID','Cost_of_shipping','Mo
de_of_Travelling','Date_Of_Shipment','Store_ID'])
```

```
    with st.expander("View all Suppliers and Ships"):
        st.dataframe(df)
    with st.expander("Name"):
        task_df = df['Name'].value_counts().to_frame()
        task_df = task_df.reset_index()
        task_df.columns = ['Supplier_Name', 'Count']
        st.dataframe(task_df)
        p1 = px.pie(task_df, names = 'Supplier_Name', values
= 'Count')
        st.plotly_chart(p1)

##############################################################
#################################################
```

UPDATE.PY : Contains methods for doing update operations on all the tables

```
import datetime
import pandas as pd
import streamlit as st
from database import *


def cu_update():
    result = cu_view_cust_data()
    # st.write(result)
    df = pd.DataFrame(result, columns=['C_ID',
'First_Name','Last
Name','Qualification','Address','Locality','City','Email','P
hone_NO'])
    with st.expander("Current Customers"):
        st.dataframe(df)
    list_of_customers = [i[0] for i in cu_view_only()]
    selected_cus = st.selectbox("Customer to Edit",
list_of_customers)
    selected_result = cu_get(selected_cus)
    # st.write(selected_result)
```

```
    if selected_result:
        c_id = selected_result[0][0]
        c_fn = selected_result[0][1]
        c_ln = selected_result[0][2]
        c_qual = selected_result[0][3]
        c_address = selected_result[0][4]
        c_locate = selected_result[0][5]
        c_city = selected_result[0][6]
        c_email = selected_result[0][7]
        c_phone = selected_result[0][8]


        # Layout of Create

    col1, col2 = st.columns(2)
    with col1:
        new_c_id = st.number_input("Customer ID: ",max_value
= 2999,value = c_id)
        new_c_fn = st.text_input("First Name: ",value =
c_fn)
        new_c_ln = st.text_input("Last Name: ",value = c_ln)
        new_c_qual = st.text_input("Qualification: ",value =
c_qual)
        new_c_phone = st.text_input("Phone: ",value =
c_phone)

    with col2:
        new_c_address = st.text_input("Address: ",value =
c_address)
        new_c_locate = st.text_input("Locality: ",value =
c_locate)
        new_c_city = st.text_input("City: ",value = c_city)
        new_c_email = st.text_input("Email: ",value =
c_email)
        st.markdown("\n")
        st.markdown("\n")

        if st.button("Update Customer Details"):
```

```
            cu_edit(new_c_id,new_c_fn,new_c_ln,new_c_qual,
new_c_address, new_c_locate, new_c_city,
new_c_email,new_c_phone,c_id,c_fn,c_ln, c_qual, c_address,
c_locate, c_city, c_email,c_phone)
            st.success("Successfully updated:: {} to :: {}
and other details".format(c_fn, new_c_fn))

    result2 = cu_view_all_data()
    df2 = pd.DataFrame(result2, columns = ['C_ID',
'First_Name','Last
Name','Qualification','Address','Locality','City','Email','P
hone_NO','DOP','E_ID','Order_C_ID','Item_ID','Price','Quanti
ty','O_Date','O_Amount'])
    with st.expander("Updated Data"):
        st.dataframe(df2)

##################################################################
#################################################

def em_update():
    result = em_view_all_data()
    # st.write(result)
    df = pd.DataFrame(result, columns = ['E_ID',
'First_Name','Last_Name','MGR_ID','GENDER','SALARY','DOB','A
DDRESS','Phone_NO'])
    with st.expander("Current Employees"):
        st.dataframe(df)
    list_of_trains = [i[0] for i in em_view_only()]
    selected_train = st.selectbox("Employee name to Edit",
list_of_trains)
    selected_result = em_get(selected_train)
    # st.write(selected_result)
    if selected_result:
        e_id = selected_result[0][0]
        e_fn = selected_result[0][1]
        e_ln = selected_result[0][2]
        emgr_id = selected_result[0][3]
        e_gender = selected_result[0][4]
```

```
        e_salary = selected_result[0][5]
        e_dob = selected_result[0][6]
        e_address = selected_result[0][7]
        e_phone = selected_result[0][8]

        # Layout of Create

        col1, col2 = st.columns(2)
    with col1:
        new_e_id = st.number_input("Employee ID: ",min_value
= 1000, max_value = 1999,value = e_id)
        new_e_fn = st.text_input("First Name: ",value =
e_fn)
        new_e_ln = st.text_input("Last Name: ",value = e_ln)
        new_e_address = st.text_input("Address: ",value =
e_address)
        new_e_phone = st.text_input("Phone: ",value =
e_phone)
    with col2:
        c.execute("SELECT DISTINCT MGR_ID FROM EMPLOYEE
WHERE MGR_ID <> 0")
        data = c.fetchall()
        mgr_data = []
        for i in data:
            for j in i:
                mgr_data.append(int(j))
        new_emgr_id = st.selectbox("Manager", mgr_data)
        new_e_gender = st.radio("Gender",('M', 'F'))
        new_e_salary = st.number_input("Salary: ",value =
e_salary)
        new_e_dob = st.date_input("Date of Birth:",value =
e_dob)
        st.markdown("\n")

        if st.button("Update Employee"):
            em_edit(new_e_id,new_e_fn,new_e_ln,new_emgr_id,n
ew_e_gender,new_e_salary,new_e_dob,new_e_address,new_e_phone
```

```
,e_id,e_fn,e_ln,emgr_id,e_gender,e_salary,e_dob,e_address,e_
phone)
            st.success("Successfully updated :: {} to :: {}
and other details".format(e_fn, new_e_fn))

    result2 = em_view_all_data()
    df2 = pd.DataFrame(result2, columns = ['E_ID',
'First_Name','Last_Name','MGR_ID','GENDER','SALARY','DOB','A
DDRESS','Phone_NO'])
    with st.expander("Updated Data"):
        st.dataframe(df2)

##################################################################
##################################################

def it_update():
    result = it_view_only()
    # st.write(result)
    df = pd.DataFrame(result, columns = ['Item_ID',
'Item_Name','Price','Gender','Brand','Colour','Size','Quanti
ty','Store_ID'])
    with st.expander("Current Items in Stores"):
        st.dataframe(df)
    list_of_trains = [i[0] for i in it_view_only()]
    selected_train = st.selectbox("Items in Store to Edit",
list_of_trains)
    selected_result = it_get(selected_train)
    # st.write(selected_result)
    if selected_result:
        i_id = selected_result[0][0]
        i_name = selected_result[0][1]
        i_price = selected_result[0][2]
        i_gender = selected_result[0][3]
        i_brand = selected_result[0][4]
        i_colour = selected_result[0][5]
        i_size = selected_result[0][6]
        i_quantity = selected_result[0][7]
        st_id = selected_result[0][8]
```

```
        # Layout of Create

        col1, col2, col3 = st.columns(3)
    with col1:
        new_i_id = st.number_input("ITEM ID: ",min_value =
4000,max_value = 4999,value = i_id)
        new_i_name = st.text_input("Name: ",value = i_name)
        new_i_price = st.number_input("Price: ",value =
i_price)

    with col2:
        new_i_brand = st.text_input("Brand: ",value =
i_brand)
        new_i_colour = st.text_input("Colour: ",value =
i_colour)
        new_i_size = st.selectbox("Size",
["NA","28","30","32","34","36","38","40","42","44","46","48"
,"50","52","54","56","58","60","M","L","XL","XXL","XXXL"])

    with col3:
        new_i_gender = st.radio("Gender",('M', 'F'))
        new_i_quantity = st.number_input("Quantity:
",min_value = 1, max_value = 50,value = i_quantity)
        new_st_id = st.selectbox("STORE-
ID",[6001,6002,6003,6004])

    with col1:
        if st.button("Update Item in Store"):
            it_edit(new_i_id,new_i_name,new_i_price,new_i_br
and,new_i_colour,new_i_size,new_i_gender,new_i_quantity,new_
st_id,i_id,i_name,i_price,i_brand,i_colour,i_size,i_gender,i
_quantity,st_id)
            st.success("Successfully updated :: {} to :: {}
and other details".format(i_name, new_i_name))

    result2 = it_view_all_data()
```

```
    df2 = pd.DataFrame(result2, columns = ['Item_ID',
'Item_Name','Price','Gender','Brand','Colour','Size','Quanti
ty','Store_ID'])
    with st.expander("Updated Data"):
        st.dataframe(df2)

##################################################################
#################################################

def su_update():
    result = su_view_all_data()
    # st.write(result)
    df = pd.DataFrame(result, columns =
['Supp_ID','Name','Address','Ship_ID','Cost_of_shipping','Mo
de_of_Travelling','Date_Of_Shipment','Store_ID'])
    with st.expander("Current Suppliers and Shipping
Details"):
        st.dataframe(df)
    list_of_sup = [i[0] for i in su_view_only()]
    selected_sup = st.selectbox("Supplier/Ship to Edit",
list_of_sup)
    selected_result = su_get(selected_sup)
    # st.write(selected_result)
    if selected_result:
        su_id = selected_result[0][0]
        su_name = selected_result[0][1]
        su_address = selected_result[0][2]
        sh_id = selected_result[0][3]
        sh_cost = selected_result[0][4]
        sh_mode = selected_result[0][5]
        sh_date = selected_result[0][6]
        su_st_id = selected_result[0][7]

        # Layout of Create

        col1, col2 = st.columns(2)
    with col1:
```

```
        new_su_id = st.number_input("Supplier ID:
",min_value = 7000,max_value = 7999,value = su_id)
        new_su_name = st.text_input("Name: ",value =
su_name)
        new_su_address = st.text_input("Address: ",value =
su_address)
        c.execute("SELECT STORE_ID FROM STORE")
        data = c.fetchall()
        store_data = []
        for i in data:
            for j in i:
                #j = j.replace(",","")
                store_data.append(int(j))

        new_su_st_id = st.selectbox("STORE-ID",store_data)

    with col2:
        new_sh_id = st.number_input("Ship ID: ",min_value =
8000,max_value = 8999,value = sh_id)
        new_sh_cost = st.number_input("Shipping Cost:
",value = sh_cost)
        new_sh_date = st.date_input("Date of Shipment:
",value = sh_date)
        new_sh_mode = st.selectbox("Mode of Travel",
["Roadways","Railways","Airways","Waterways"])

    with col1:
        if st.button("Update SUPPLIER - SHIP"):
            su_edit(new_su_id,new_su_name,new_su_address,new
_sh_id,new_sh_cost,new_sh_mode,new_sh_date,new_su_st_id,su_i
d,su_name,su_address,sh_id,sh_cost,sh_mode,sh_date,su_st_id)
            st.success("Successfully updated :: {} to :: {}
and other details".format(su_name, new_su_name))

    result2 = su_view_all_data()
    df2 = pd.DataFrame(result2, columns =
['Supp_ID','Name','Address','Ship_ID','Cost_of_shipping','Mo
de_of_Travelling','Date_Of_Shipment','Store_ID'])
```

```
    with st.expander("Updated Data"):
        st.dataframe(df2)

#################################################################
#################################################
```

DELETE.PY : Contains methods for deleting values in all of the tables

```
import pandas as pd
import streamlit as st
from database import *


def cu_delete():
    result = cu_view_cust_data()
    df = pd.DataFrame(result, columns = ['C_ID',
'First_Name','Last
Name','Qualification','Address','Locality','City','Email','P
hone_NO'])
    with st.expander("Current Data present"):
        st.dataframe(df)

    list_of_customers = [i[0] for i in cu_view_only()]
    selected_customers = st.selectbox("Customer to Delete",
list_of_customers)
    st.warning("Do you want to delete ::
{}".format(selected_customers))
    if st.button("Delete Customer"):
        cu_delete_data(selected_customers)
        st.success("Customer has been deleted successfully")
    new_result = cu_view_all_data()
    df2 = pd.DataFrame(new_result, columns = ['C_ID',
'First_Name','Last
Name','Qualification','Address','Locality','City','Email','P
hone_NO','DOP','E_ID','Order_C_ID','Item_ID','Price','Quanti
ty','O_Date','O_Amount'])
    with st.expander("Updated data"):
```

```
        st.dataframe(df2)

    result2 = cu_read_delete()
    df = pd.DataFrame(result2, columns = ['C_ID',
'First_Name','Last
Name','Qualification','Address','Locality','City','Email','P
hone_NO','DOP','E_ID','Store_ID','Item_ID','Quantity','O_Amo
unt'])
    with st.expander("History of Deleted Customers"):
        st.dataframe(df)

##################################################################
#################################################

def em_delete():
    result = em_view_all_data()
    df = pd.DataFrame(result, columns = ['E_ID',
'First_Name','Last_Name','MGR_ID','GENDER','SALARY','DOB','A
DDRESS','Phone_NO'])
    with st.expander("Current Data present"):
        st.dataframe(df)

    list_of_emp = [i[0] for i in em_view_only()]
    selected_emp = st.selectbox("Employee to Delete",
list_of_emp)
    st.warning("Do you want to delete ::
{}".format(selected_emp))
    if st.button("Delete Employee"):
        em_delete_data(selected_emp)
        st.success("Employee has been deleted successfully")
    new_result = em_view_all_data()
    df2 = pd.DataFrame(new_result, columns = ['E_ID',
'First_Name','Last_Name','MGR_ID','GENDER','SALARY','DOB','A
DDRESS','Phone_NO'])
    with st.expander("Updated data"):
        st.dataframe(df2)
```

```
#############################################################
#################################################

def it_delete():
    result = it_view_only()
    df = pd.DataFrame(result, columns = ['Item_ID',
'Item_Name','Price','Gender','Brand','Colour','Size','Quanti
ty','Store_ID'])
    with st.expander("Current Data present"):
        st.dataframe(df)

    list_of_items = [i[0] for i in it_view_only()]
    selected_items = st.selectbox("Item to Delete",
list_of_items)
    st.warning("Do you want to delete ::
{}".format(selected_items))
    if st.button("Delete Item from Store"):
        it_delete_data(selected_items)
        st.success("Item in store has been deleted
successfully")
    new_result = it_view_only()
    df2 = pd.DataFrame(new_result, columns = ['Item_ID',
'Item_Name','Price','Gender','Brand','Colour','Size','Quanti
ty','Store_ID'])
    with st.expander("Updated data"):
        st.dataframe(df2)

#############################################################
#################################################

def su_delete():
    result = su_view_all_data()
    df = pd.DataFrame(result, columns =
['Supp_ID','Name','Address','Ship_ID','Cost_of_shipping','Mo
de_of_Travelling','Date_Of_Shipment','Store_ID'])
    with st.expander("Current Data present"):
        st.dataframe(df)
```

```
    list_of_sup = [i[0] for i in su_view_only()]
    selected_sup = st.selectbox("Supplier to Delete",
list_of_sup)
    st.warning("Do you want to delete ::
{}".format(selected_sup))
    if st.button("Delete Supplier"):
        su_delete_data(selected_sup)
        st.success("Supplier has been deleted successfully")
    new_result = su_view_all_data()
    df2 = pd.DataFrame(new_result, columns =
['Supp_ID','Name','Address','Ship_ID','Cost_of_shipping','Mo
de_of_Travelling','Date_Of_Shipment','Store_ID'])
    with st.expander("Updated data"):
        st.dataframe(df2)

############################################################
#################################################
```

DATABASE.PY : Contains all the helper functions for the above files

```
# pip install mysql-connector-python
import mysql.connector
import streamlit as st
import pandas as pd

mydb = mysql.connector.connect(
    host = "localhost",
    user = "root",
    password = "",
    database = "textile_362"
)
c = mydb.cursor()

############################################################
#################################################

def cu_create_table():
```

```
    c.execute('CREATE TABLE IF NOT EXISTS Customers(C_ID
INT(11), First_Name VARCHAR(50),Last_Name
VARCHAR(50),Qualification VARCHAR(20),ADDRESS
VARCHAR(50),Locality VARCHAR(20),CITY VARCHAR(20),Email
VARCHAR(50),Phone_NO VARCHAR(10),DOP date,E_ID INT(11))')
    c.execute('CREATE TABLE IF NOT EXISTS Orders(C_ID
INT(11), ITEM_ID INT(11), Price FLOAT, Quantity
INT(11),O_Date date,O_Amount FLOAT)')

def
cu_add_data(c_id,cf_name,cl_name,c_qual,c_address,c_locate,c
_city,c_email,c_phone,c_dop,c_emp):
    c.execute('INSERT INTO Customers(C_ID,
First_Name,Last_Name,Qualification,Address,Locality,City,Ema
il,Phone_NO,DOP,E_ID) VALUES
(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)',(c_id,cf_name,cl_name,c_
qual,c_address,c_locate,c_city,c_email,c_phone,c_dop,c_emp))

    mydb.commit()

def cu_orders_add_data(c_id, c_item_id, c_price, c_qty,
c_dop, c_total):
    c.execute('INSERT INTO Orders(C_ID, Item_ID, Price,
Quantity,O_Date,O_Amount) VALUES
(%s,%s,%s,%s,%s,%s)',(c_id,c_item_id,c_price,c_qty,c_dop,c_t
otal))
    mydb.commit()



def cu_view_all_data():
    c.execute('SELECT * FROM Customers AS C JOIN Orders AS O
ON C.C_ID = O.C_ID GROUP BY O.C_ID, O.Item_ID ORDER BY
O.C_ID;')
    data = c.fetchall()
    return data

def cu_view_cust_data():
```

```
    c.execute('SELECT C_ID,
First_Name,Last_Name,Qualification,Address,Locality,City,Ema
il,Phone_NO FROM Customers')
    data = c.fetchall()
    return data


def cu_view_only():
    c.execute('SELECT C_ID,First_Name,Last_Name FROM
Customers')
    data = c.fetchall()
    return data


def cu_get(c_id):
    c.execute('SELECT * FROM CUSTOMERS WHERE C_ID
="{}"'.format(c_id))
    data = c.fetchall()
    return data


def cu_edit(new_c_id,new_cf,new_cl,new_c_qual,
new_c_address, new_c_locate, new_c_city,
new_c_email,new_c_phone, c_id, c_fn, c_ln, c_qual,
c_address, c_locate, c_city, c_email,c_phone):
    c.execute("UPDATE Customers SET C_ID = %s, First_Name =
%s, Last_Name = %s, Qualification = %s, Address = %s,
Locality = %s, City = %s, Email = %s, Phone_NO = %s  WHERE
C_ID = %s and First_Name = %s and Last_Name = %s and
Qualification = %s and Address = %s and Locality = %s and
City = %s and Email = %s and Phone_NO =
%s",(new_c_id,new_cf,new_cl,new_c_qual, new_c_address,
new_c_locate, new_c_city,
new_c_email,new_c_phone,c_id,c_fn,c_ln,c_qual, c_address,
c_locate, c_city, c_email,c_phone))
    mydb.commit()
    data = c.fetchmany()
    return data
```

```
def cu_delete_data(c_id):
    c.execute('DELETE FROM CUSTOMERS WHERE C_ID =
"{}"'.format(c_id))
    mydb.commit()

def cu_read_delete():
    c.execute("SELECT * FROM customer_backuplog")
    data = c.fetchall()
    return data
##################################################################
#######################

def cu_bill(b_id, b_bank, c_dop,b_tid, b_amount,c_id):
    c.execute('CREATE TABLE IF NOT EXISTS Bill(B_ID INT(11),
BANK VARCHAR(50),DATE_OF_BILL date,Transaction_ID
varchar(20), Amount float,C_ID INT(11))')
    c.execute('DELETE FROM Bill WHERE C_ID =
"{}"'.format(c_id))
    c.execute('INSERT INTO Bill(B_ID,
BANK,DATE_OF_BILL,Transaction_ID, Amount,C_ID) VALUES
(%s,%s,%s,%s,%s,%s)',(b_id, b_bank, c_dop,b_tid,
b_amount,c_id))
    mydb.commit()

def bill_view_all_data(c_id):
    c.execute('SELECT
B_ID,Bank,Date_Of_Bill,Transaction_ID,Amount FROM Bill WHERE
C_ID = "{}"'.format(c_id))
    data = c.fetchall()
    return data

def bill_view_all_data2(c_id):
    c.execute('SELECT
Item_ID,Item_Name,Brand,Size,Quantity,Total FROM Bills WHERE
C_ID = "{}"'.format(c_id))
    data = c.fetchall()
    return data
```

```
def bill_view_all_data3(c_id):
    c.execute('SELECT Discount(C_ID) FROM Bill WHERE C_ID =
"{}"'.format(c_id))
    data = c.fetchall()
    price_data = []
    for i in data:
        for j in i:
            price_data.append(float(j))
    r1 = price_data[0]
    c.execute('SELECT Amount FROM Bill WHERE C_ID =
"{}"'.format(c_id))
    data2 = c.fetchall()
    price_data = []
    for i in data2:
        for j in i:
            price_data.append(float(j))
    r2 = price_data[0]
    res = r2 - r1
    return data,res

def view_bill(c_id):
    result = bill_view_all_data(c_id)
    result2 = bill_view_all_data2(c_id)
    result3,res4 = bill_view_all_data3(c_id)
    # st.write(result)
    st.markdown(' `BILL SUMMARY:` ')
    df = pd.DataFrame(result2, columns =
['Item_ID','Item_Name','Brand','Size','Quantity','Total'])
    st.dataframe(df)
    st.markdown(' `TOTAL AMOUNT:` ')
    df = pd.DataFrame(result, columns =
['Bill_ID','Bank','Date_Of_Bill','Transaction_ID','Amount'])
    st.dataframe(df)
    st.markdown(' `DISCOUNT:` ')
    df = pd.DataFrame(result3, columns = ['Discount'])
    st.dataframe(df)
    st.markdown('TOTAL AMOUNT AFTER DISCOUNT: `INR
{}`'.format(res4))
```

```
#################################################################
#################################################

def em_create_table():
    c.execute('CREATE TABLE IF NOT EXISTS Employee(E_ID
INT(11), First_Name VARCHAR(30),Last_Name VARCHAR(30),MGR_ID
INT(11), GENDER VARCHAR(1),SALARY float, DOB date, ADDRESS
VARCHAR(50),Phone_NO VARCHAR(10))')

def
em_add_data(e_id,e_fn,e_ln,emgr_id,e_gender,e_salary,e_dob,e
_address,e_phone):
    c.execute('INSERT INTO EMPLOYEE(E_ID,
First_Name,Last_Name,MGR_ID,GENDER,SALARY,DOB,ADDRESS,Phone_
NO) VALUES
(%s,%s,%s,%s,%s,%s,%s,%s,%s)',(e_id,e_fn,e_ln,emgr_id,e_gend
er,e_salary,e_dob,e_address,e_phone))
    mydb.commit()


def em_view_all_data():
    c.execute('SELECT * FROM EMPLOYEE')
    data = c.fetchall()
    return data


def em_view_only():
    c.execute('SELECT First_name FROM Employee')
    data = c.fetchall()
    return data


def em_get(e_fn):
    c.execute('SELECT * FROM Employee WHERE
First_Name="{}"'.format(e_fn))
    data = c.fetchall()
    return data
```

```
def
em_edit(new_e_id,new_e_fn,new_e_ln,new_emgr_id,new_e_gender,
new_e_salary,new_e_dob,new_e_address,new_e_phone,e_id,e_fn,e
_ln,emgr_id,e_gender,e_salary,e_dob,e_address,e_phone):
    c.execute("UPDATE Employee SET E_ID = %s, First_Name =
%s,Last_Name = %s,MGR_ID = %s,GENDER = %s,SALARY = %s,DOB =
%s,ADDRESS = %s,Phone_NO = %s  WHERE E_ID = %s AND
First_Name = %s AND Last_Name = %s AND MGR_ID = %s AND
GENDER = %s AND SALARY = %s AND DOB = %s AND ADDRESS = %s
AND Phone_NO = %s",
(new_e_id,new_e_fn,new_e_ln,new_emgr_id,new_e_gender,new_e_s
alary,new_e_dob,new_e_address,new_e_phone,e_id,e_fn,e_ln,emg
r_id,e_gender,e_salary,e_dob,e_address,e_phone))
    mydb.commit()
    data = c.fetchmany()
    return data


def em_delete_data(e_fn):
    c.execute('DELETE FROM Employee WHERE First_Name =
"{}"'.format(e_fn))
    mydb.commit()


##########################################################
##################################################

def it_create_table():
    c.execute('CREATE TABLE IF NOT EXISTS ITEMS(ITEM_ID
INT(11),Item_Name VARCHAR(30),Price float)')
    c.execute('CREATE TABLE IF NOT EXISTS
ITEM_CATEGORY(ITEM_ID INT(11),Item_Name VARCHAR(30),Gender
VARCHAR(1),BRAND VARCHAR(30),CATEGORY VARCHAR(20),COLOUR
VARCHAR(20),SIZE VARCHAR(10))')
    c.execute('CREATE TABLE IF NOT EXISTS CONTAINS(STORE_ID
INT(11),ITEM_ID INT(11),QUANTITY INT(11))')
```

```
def
it_add_data(i_id,i_name,i_price,i_gender,i_brand,i_colour,i_
size,i_quantity,st_id):
    c.execute('INSERT INTO ITEMS(Item_ID,Item_Name,Price)
VALUES (%s,%s,%s)',(i_id,i_name,i_price))
    c.execute('INSERT INTO
ITEM_CATEGORY(Item_ID,Item_Name,Gender,BRAND,COLOUR,SIZE)
VALUES
(%s,%s,%s,%s,%s,%s)',(i_id,i_name,i_gender,i_brand,i_colour,
i_size))
    c.execute('INSERT INTO
CONTAINS(STORE_ID,ITEM_ID,QUANTITY) VALUES
(%s,%s,%s)',(st_id,i_id,i_quantity))
    mydb.commit()


def it_view_all_data():
    c.execute('SELECT I.Item_ID,
I.Item_Name,IC.Gender,IC.Brand,IC.Colour,IC.Size,
C.Quantity,C.Store_ID,S.Name FROM ITEM_CATEGORY AS IC JOIN
ITEMS AS I ON IC.ITEM_ID=I.ITEM_ID JOIN CONTAINS AS C ON
I.ITEM_ID=C.ITEM_ID JOIN STORE AS S ON
C.STORE_ID=S.STORE_ID')
    data = c.fetchall()
    return data


def it_view_only():
    c.execute('SELECT I.Item_ID,
I.Item_Name,I.Price,IC.Gender,IC.Brand,IC.Colour,IC.Size,
C.Quantity,C.Store_ID FROM ITEM_CATEGORY AS IC JOIN ITEMS AS
I ON IC.ITEM_ID=I.ITEM_ID JOIN CONTAINS AS C ON
I.ITEM_ID=C.ITEM_ID')
    data = c.fetchall()
    return data


def it_get(i_id):
```

```
    c.execute('SELECT I.Item_ID,
I.Item_Name,I.Price,IC.Gender,IC.Brand,IC.Colour,IC.Size,
C.Quantity,C.Store_ID FROM ITEM_CATEGORY AS IC JOIN ITEMS AS
I ON IC.ITEM_ID=I.ITEM_ID JOIN CONTAINS AS C ON
I.ITEM_ID=C.ITEM_ID WHERE I.Item_ID="{}"'.format(i_id))
    data = c.fetchall()
    return data


def
it_edit(new_i_id,new_i_name,new_i_price,new_i_gender,new_i_b
rand,new_i_colour,new_i_size,new_i_quantity,new_st_id,i_id,i
_name,i_price,i_gender,i_brand,i_colour,i_size,i_quantity,st
_id):
    c.execute("UPDATE Items SET Item_ID = %s, Item_Name =
%s, Price = %s  WHERE Item_ID = %s and Item_Name = %s and
Price = %s",
(new_i_id,new_i_name,new_i_price,i_id,i_name,i_price))
    c.execute("UPDATE Item_Category SET Item_ID = %s,
Item_Name = %s,GENDER = %s,BRAND = %s,COLOUR = %s,SIZE =
%s  WHERE Item_ID = %s and Item_Name = %s and Gender = %s
and BRAND = %s and COLOUR = %s and SIZE = %s",
(new_i_id,new_i_name,new_i_gender,new_i_brand,new_i_colour,n
ew_i_size,i_id,i_name,i_gender,i_brand,i_colour,i_size))
    c.execute("UPDATE CONTAINS SET STORE_ID = %s, ITEM_ID =
%s, QUANTITY = %s  WHERE STORE_ID=%s and ITEM_ID=%s and
QUANTITY=%s",
(new_st_id,new_i_id,new_i_quantity,st_id,i_id,i_quantity))
    mydb.commit()
    data = c.fetchmany()
    return data


def it_delete_data(i_id):
    c.execute('DELETE FROM Items WHERE Item_ID =
"{}"'.format(i_id))
    mydb.commit()
```

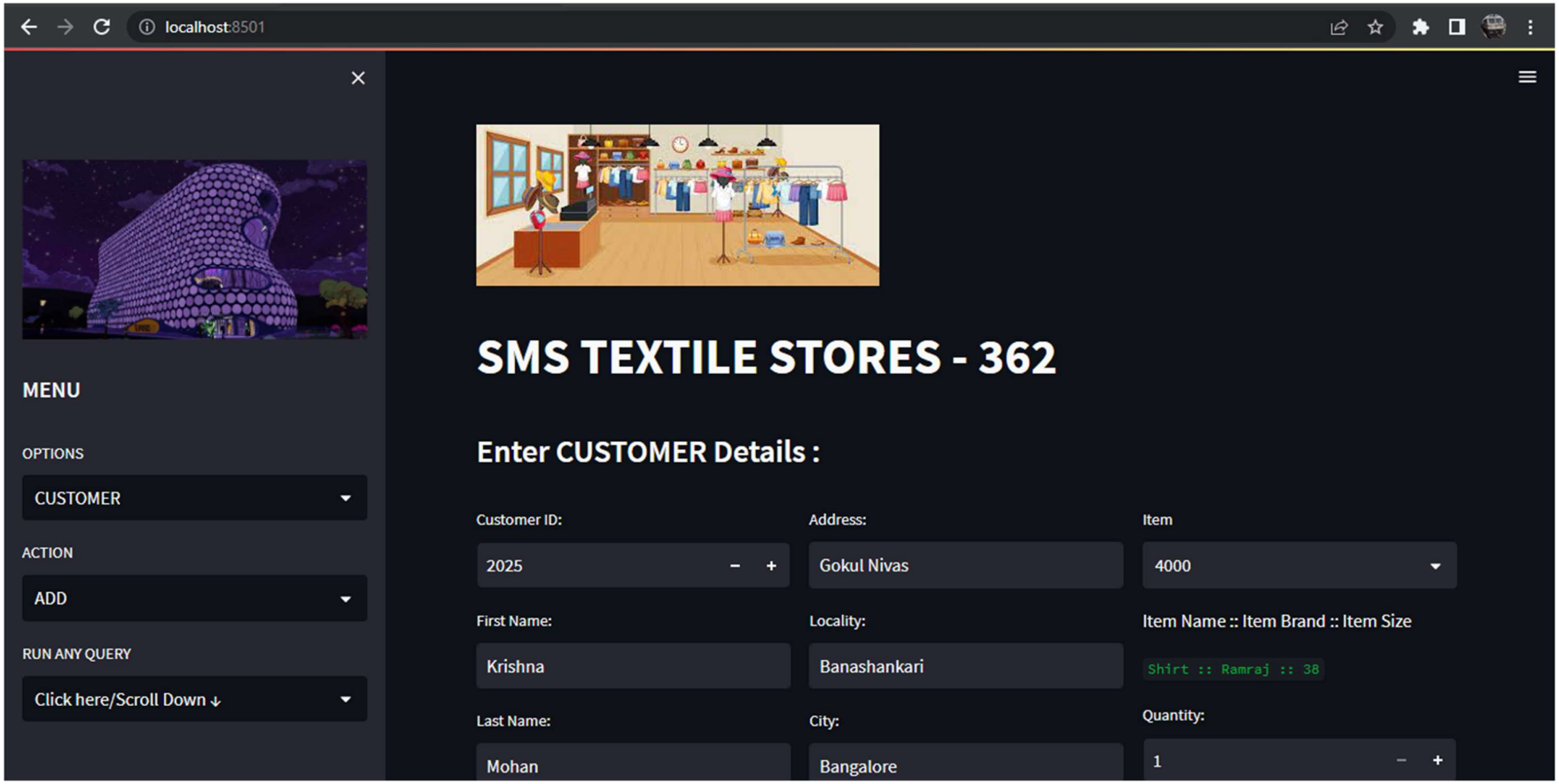
```
#################################################################
#################################################

def su_create_table():
    c.execute('CREATE TABLE IF NOT EXISTS Suppliers(SUPP_ID
int(11), NAME varchar(50), ADDRESS varchar(50))')
    c.execute('CREATE TABLE IF NOT EXISTS
SHIPS(COST_OF_SHIPPING float,MODE_OF_TRAVELLING
varchar(50),SUPP_ID int(11), SHIP_ID int(11))')
    c.execute('CREATE TABLE IF NOT EXISTS Shipment(Ship_ID
int(11),Date_Of_Shipment date,Store_ID INT(11))')

def
su_add_data(su_id,su_name,su_address,sh_id,sh_cost,sh_mode,s
h_date,su_st_id):
    c.execute('INSERT INTO Suppliers(SUPP_ID,NAME,ADDRESS)
VALUES (%s,%s,%s)',(su_id,su_name,su_address))
    c.execute('INSERT INTO
SHIPS(COST_OF_SHIPPING,MODE_OF_TRAVELLING,SUPP_ID,SHIP_ID)
VALUES (%s,%s,%s,%s)',(sh_cost,sh_mode,su_id,sh_id))
    c.execute('INSERT INTO
Shipment(Ship_ID,Date_Of_Shipment,Store_ID) VALUES
(%s,%s,%s)',(sh_id,sh_date,su_st_id))
    mydb.commit()

def su_view_all_data():
    c.execute('SELECT
SU.Supp_ID,SU.Name,SU.Address,SH.Ship_ID,SH.Cost_of_shipping
,SH.mode_of_Travelling,S.Date_Of_Shipment,S.Store_ID FROM
Suppliers AS SU JOIN SHIPS AS SH ON SU.SUPP_ID=SH.SUPP_ID
JOIN Shipment AS S ON SH.SHIP_ID=S.SHIP_ID')
    data = c.fetchall()
    return data

def su_view_only():
    c.execute('SELECT Name FROM Suppliers')
    data = c.fetchall()
    return data
```

```
def su_get(su_name):
    c.execute('SELECT
SU.Supp_ID,SU.Name,SU.Address,SH.Ship_ID,SH.Cost_of_shipping
,SH.mode_of_Travelling,S.Date_Of_Shipment,S.Store_ID FROM
Suppliers AS SU JOIN SHIPS AS SH ON SU.SUPP_ID=SH.SUPP_ID
JOIN Shipment AS S ON SH.SHIP_ID=S.SHIP_ID WHERE
SU.Name="{}"'.format(su_name))
    data = c.fetchall()
    return data


def
su_edit(new_su_id,new_su_name,new_su_address,new_sh_id,new_s
h_cost,new_sh_mode,new_sh_date,new_su_st_id,su_id,su_name,su
_address,sh_id,sh_cost,sh_mode,sh_date,su_st_id):
    c.execute("UPDATE Suppliers SET SUPP_ID = %s, NAME = %s,
ADDRESS = %s  WHERE SUPP_ID = %s AND NAME = %s AND ADDRESS =
%s",
(new_su_id,new_su_name,new_su_address,su_id,su_name,su_addre
ss))
    c.execute("UPDATE SHIPS SET COST_OF_SHIPPING =
%s,MODE_OF_TRAVELLING = %s,SUPP_ID = %s, SHIP_ID = %s  WHERE
COST_OF_SHIPPING = %s and MODE_OF_TRAVELLING = %s and
SUPP_ID = %s and SHIP_ID = %s",
(new_sh_cost,new_sh_mode,new_su_id,new_sh_id,sh_cost,sh_mode
,su_id,sh_id))
    c.execute("UPDATE Shipment SET Ship_ID =
%s,Date_Of_Shipment = %s,Store_ID = %s  WHERE Ship_ID = %s
and Date_Of_Shipment = %s and Store_ID = %s",
(new_sh_id,new_sh_date,new_su_st_id,sh_id,sh_date,su_st_id))
    mydb.commit()
    data = c.fetchmany()
    return data


def su_delete_data(su_name):
```

```
    c.execute('DELETE FROM Suppliers WHERE Name =
"{}"'.format(su_name))
    mydb.commit()

##################################################################
#####################################################
```

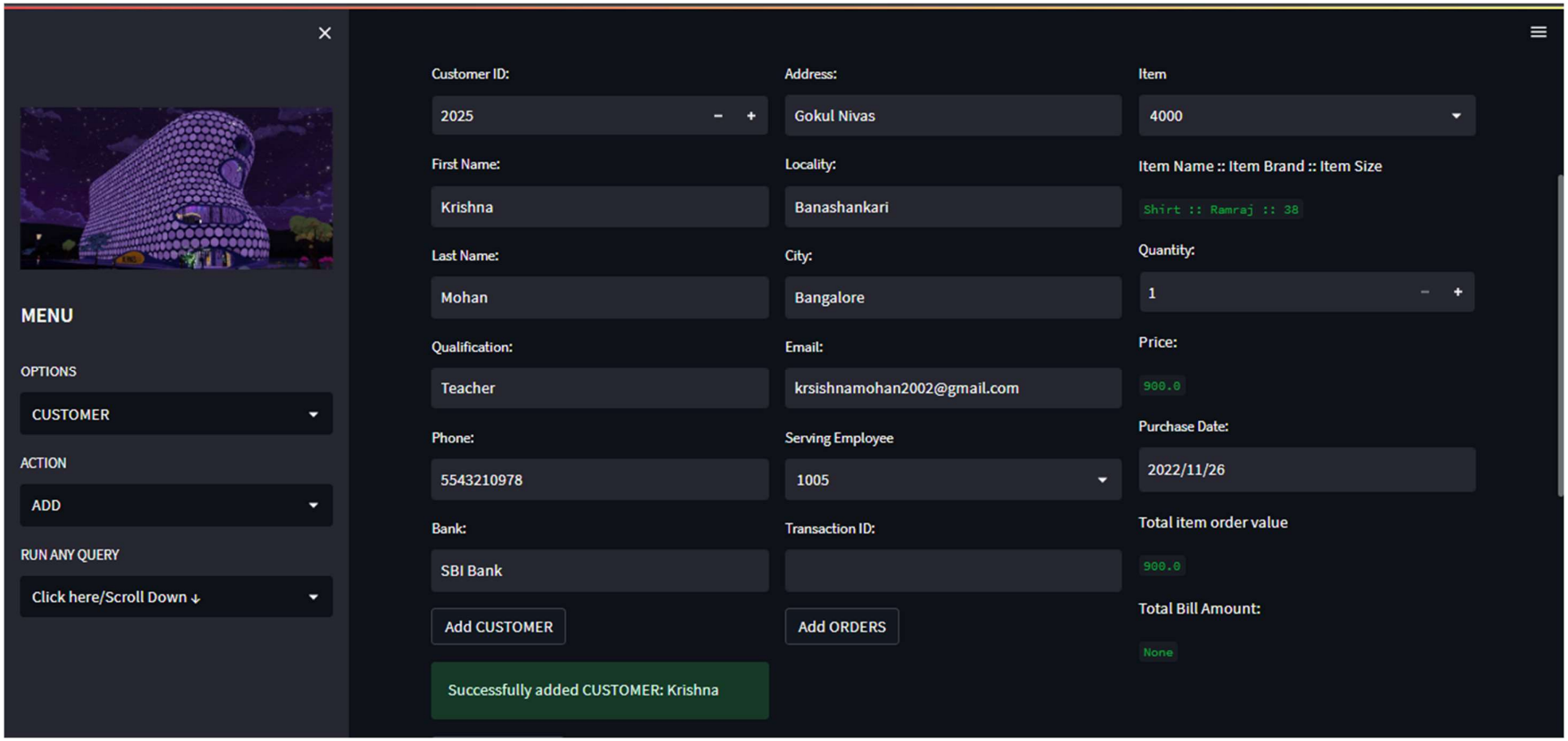
13. OUTPUT SCREENSHOTS OF FRONTEND

MAIN PAGE :

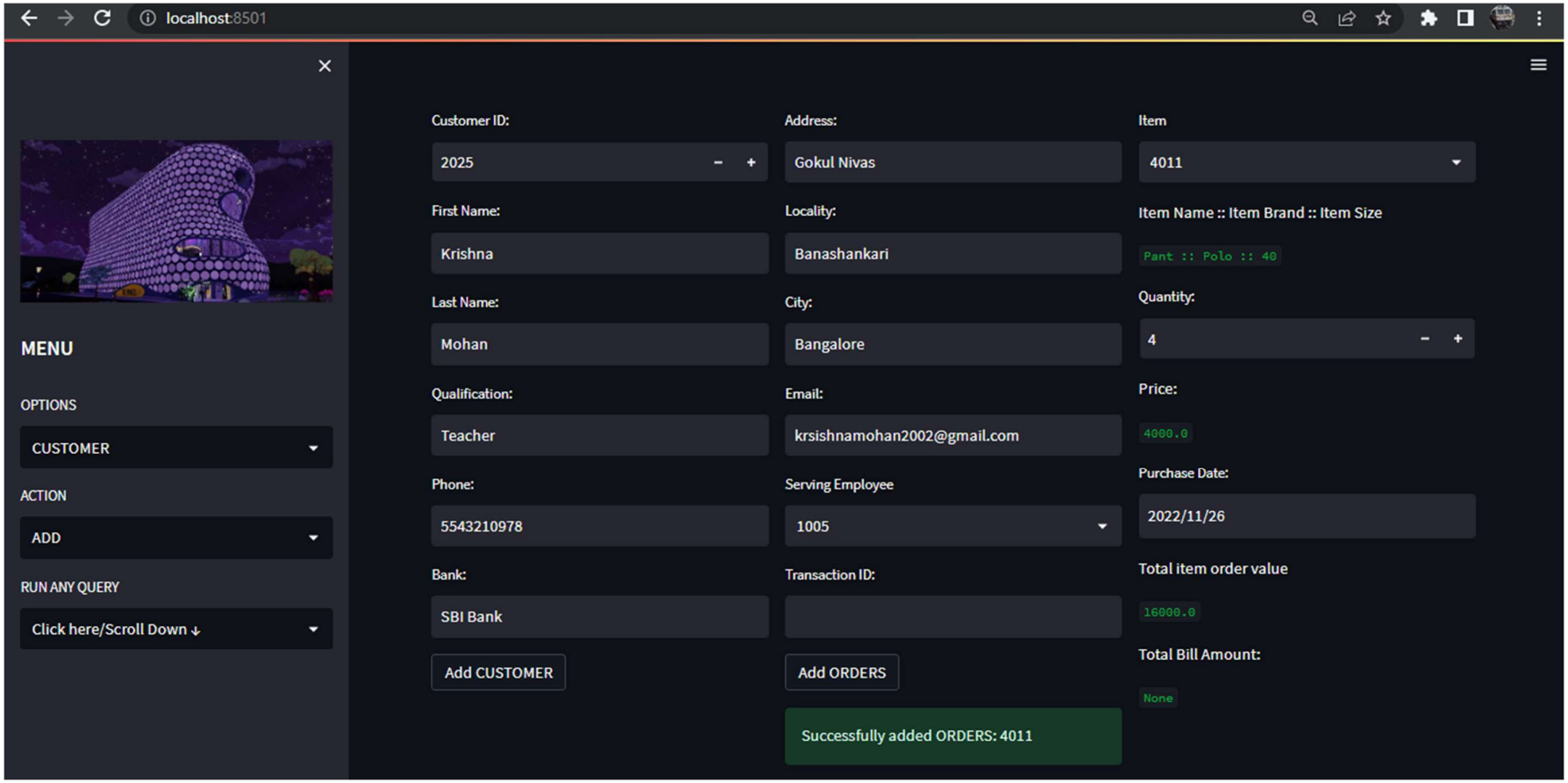


a. CUSTOMER PAGE(Includes CRUD operations on Customers, Orders, Bill table in the backend)

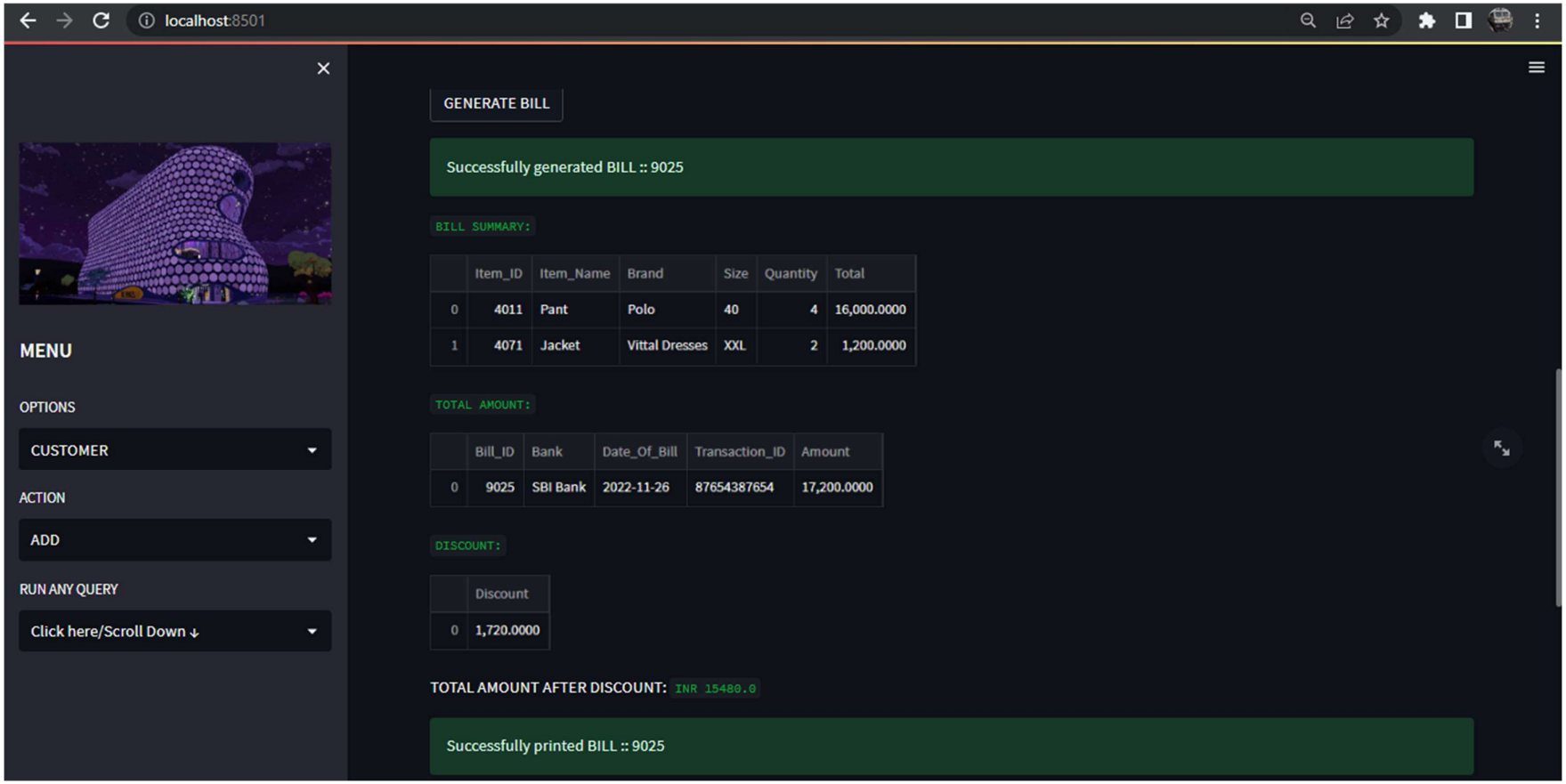
- Added customer successfully



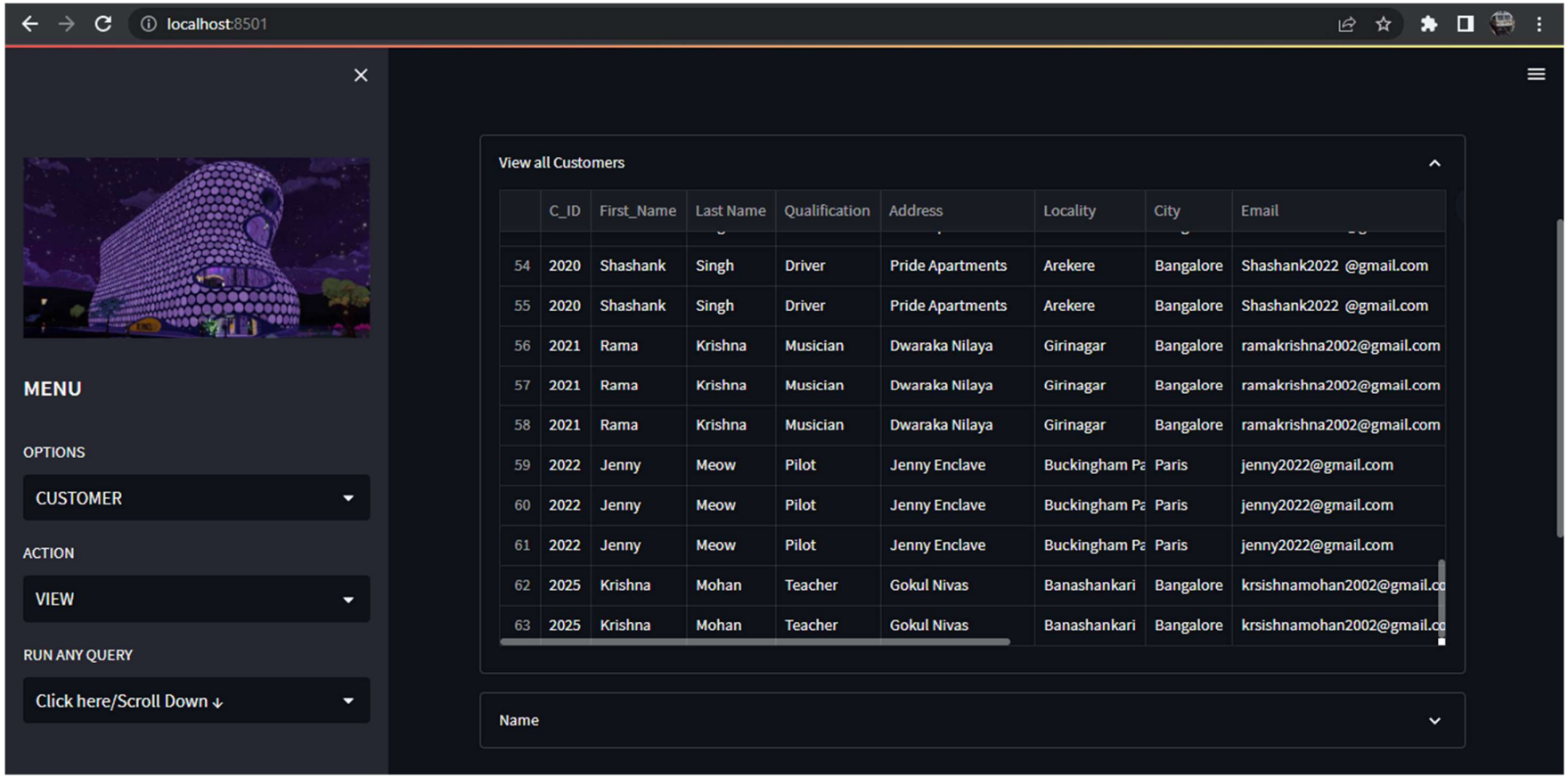
- Adding 2 orders for the customer (Here only 1 is shown)



- Generating bill for the selected order and customer

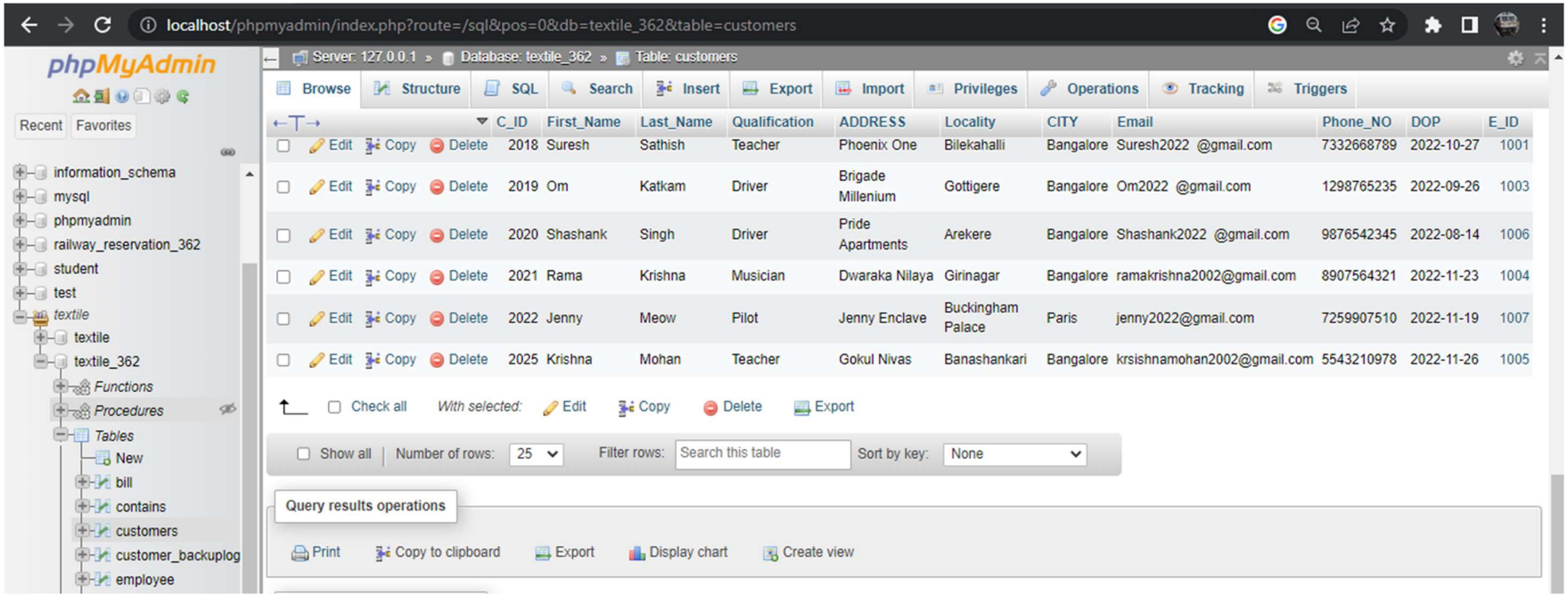


- Reading the customers and his order details



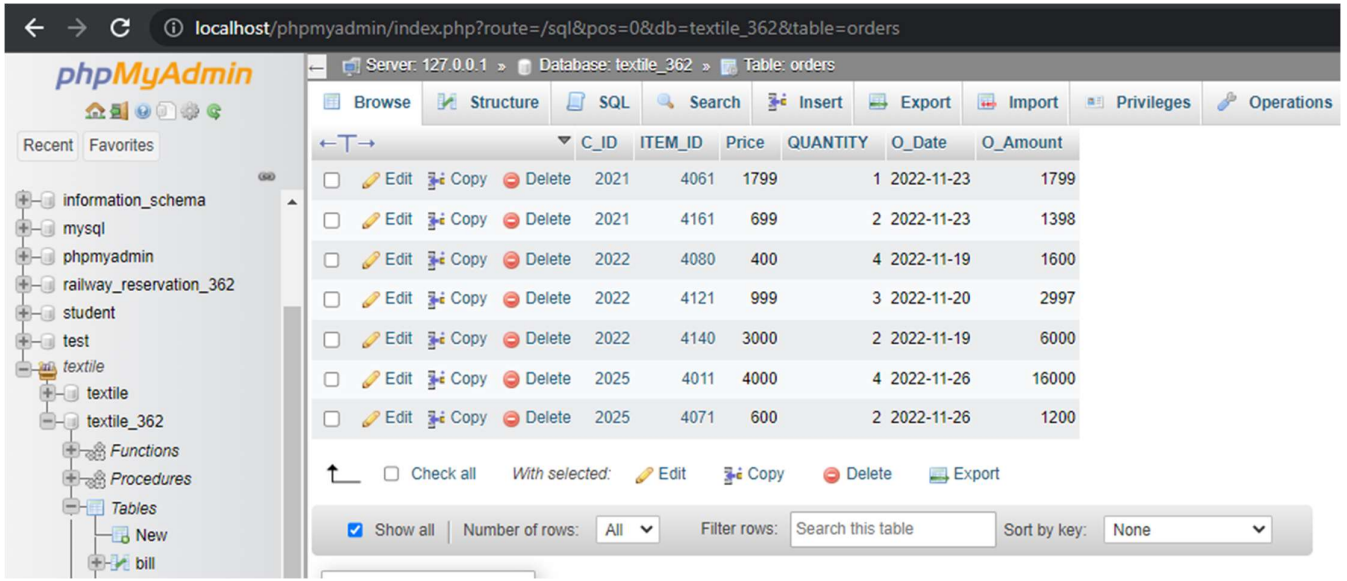
	C_ID	First_Name	Last Name	Qualification	Address	Locality	City	Email
54	2020	Shashank	Singh	Driver	Pride Apartments	Arekere	Bangalore	Shashank2022 @gmail.com
55	2020	Shashank	Singh	Driver	Pride Apartments	Arekere	Bangalore	Shashank2022 @gmail.com
56	2021	Rama	Krishna	Musician	Dwaraka Nilaya	Girinagar	Bangalore	ramakrishna2002@gmail.com
57	2021	Rama	Krishna	Musician	Dwaraka Nilaya	Girinagar	Bangalore	ramakrishna2002@gmail.com
58	2021	Rama	Krishna	Musician	Dwaraka Nilaya	Girinagar	Bangalore	ramakrishna2002@gmail.com
59	2022	Jenny	Meow	Pilot	Jenny Enclave	Buckingham Pa	Paris	jenny2022@gmail.com
60	2022	Jenny	Meow	Pilot	Jenny Enclave	Buckingham Pa	Paris	jenny2022@gmail.com
61	2022	Jenny	Meow	Pilot	Jenny Enclave	Buckingham Pa	Paris	jenny2022@gmail.com
62	2025	Krishna	Mohan	Teacher	Gokul Nivas	Banashankari	Bangalore	krsishnamohan2002@gmail.co
63	2025	Krishna	Mohan	Teacher	Gokul Nivas	Banashankari	Bangalore	krsishnamohan2002@gmail.co

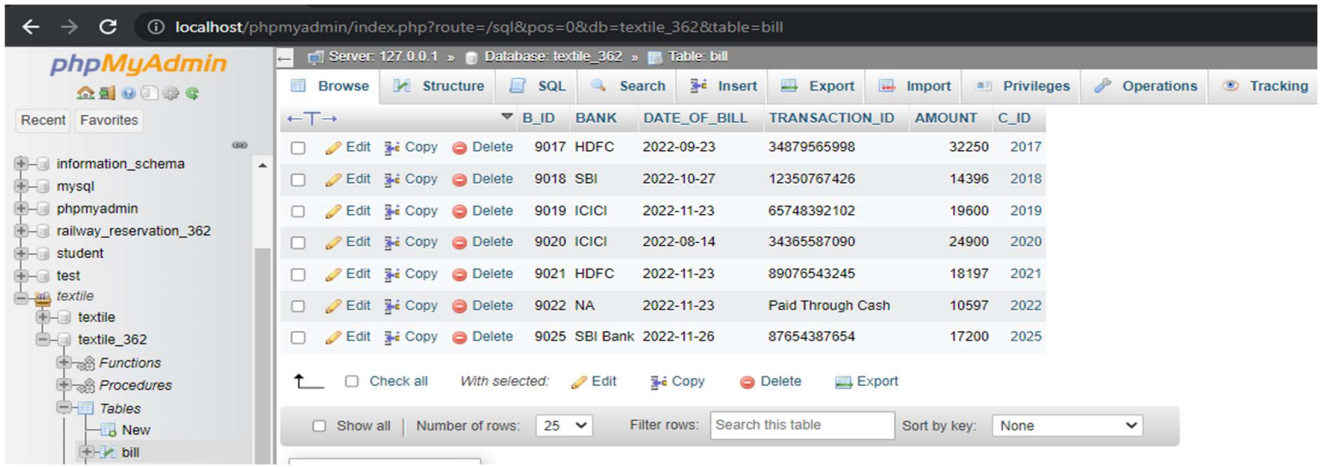
Changes reflected in the customers table



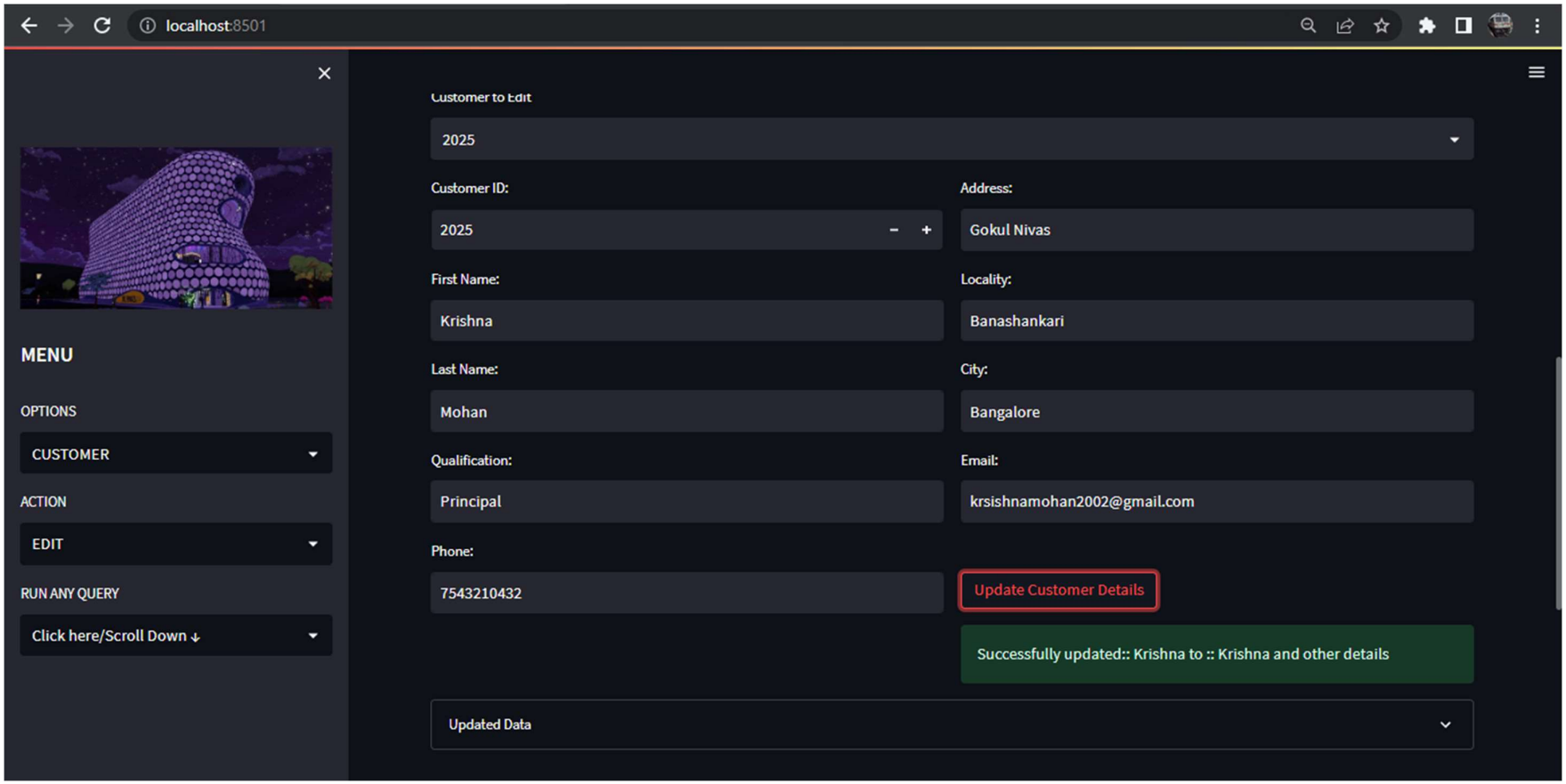
- Changes reflected in the orders table

- Changes reflected in the orders and the bill table

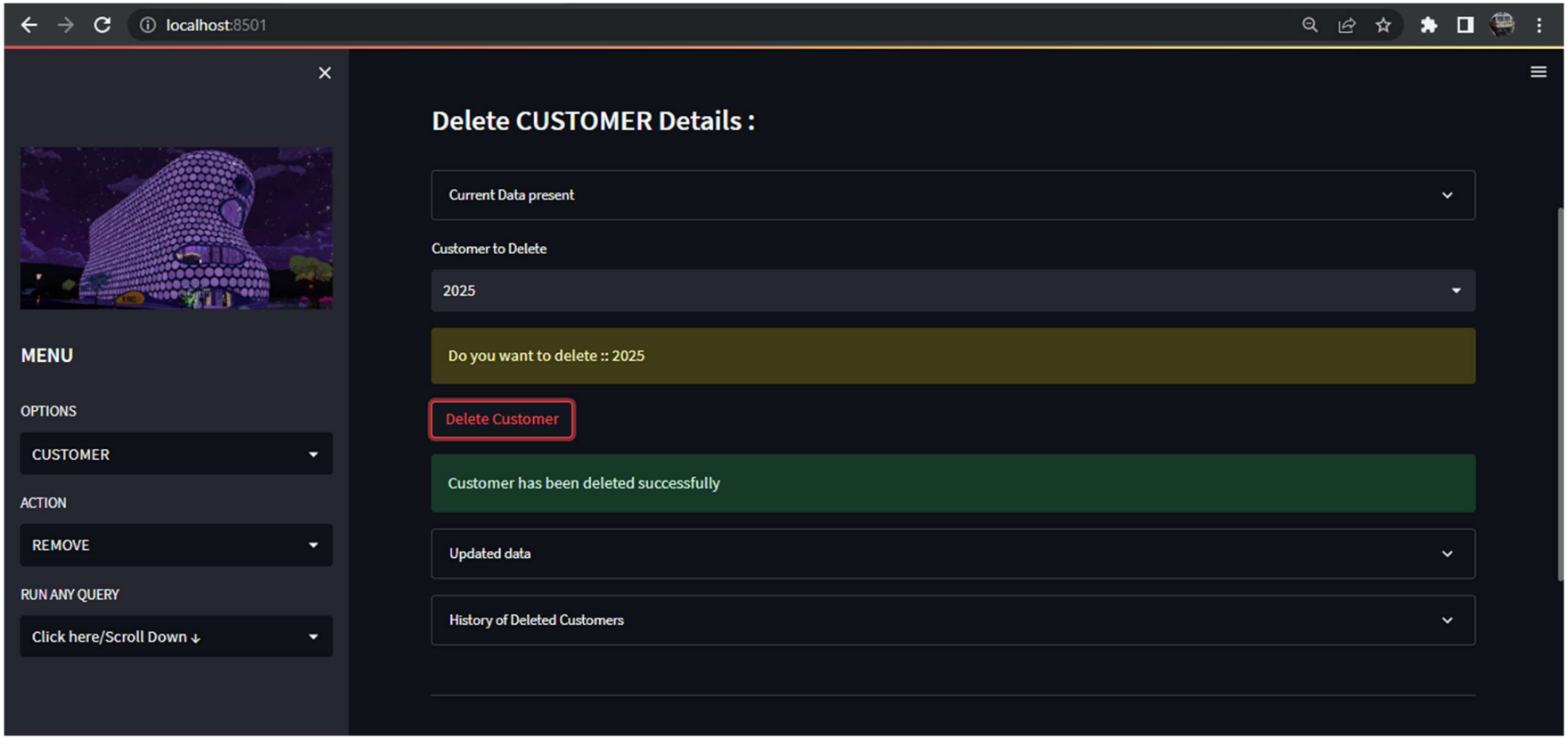




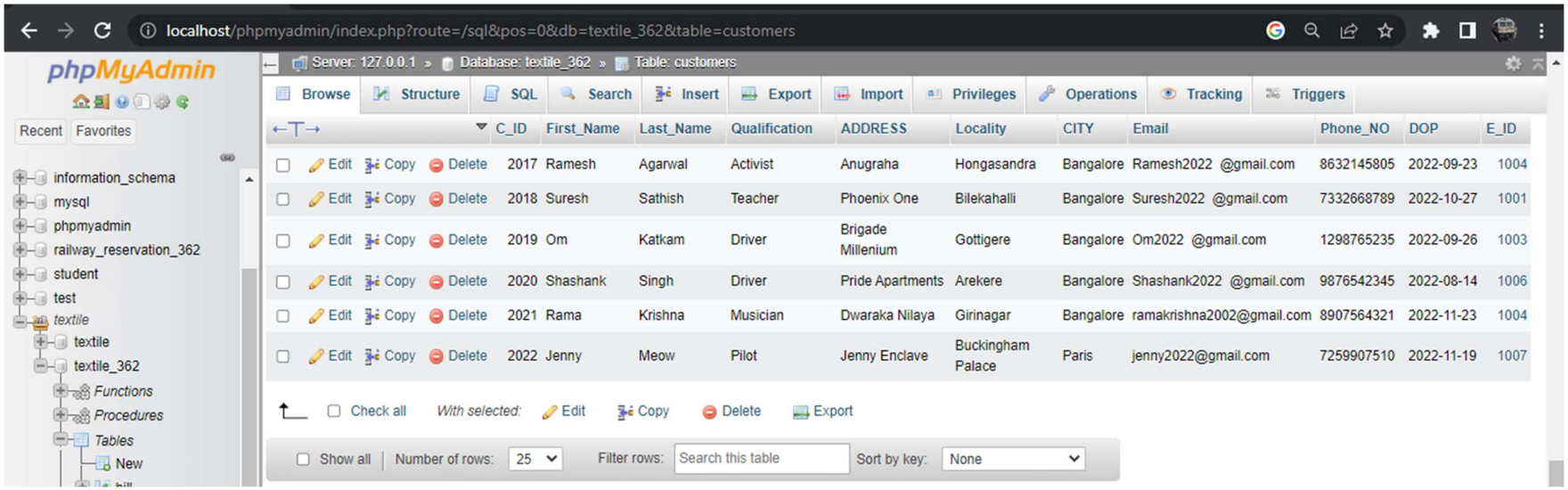
- Updating the details of the customer



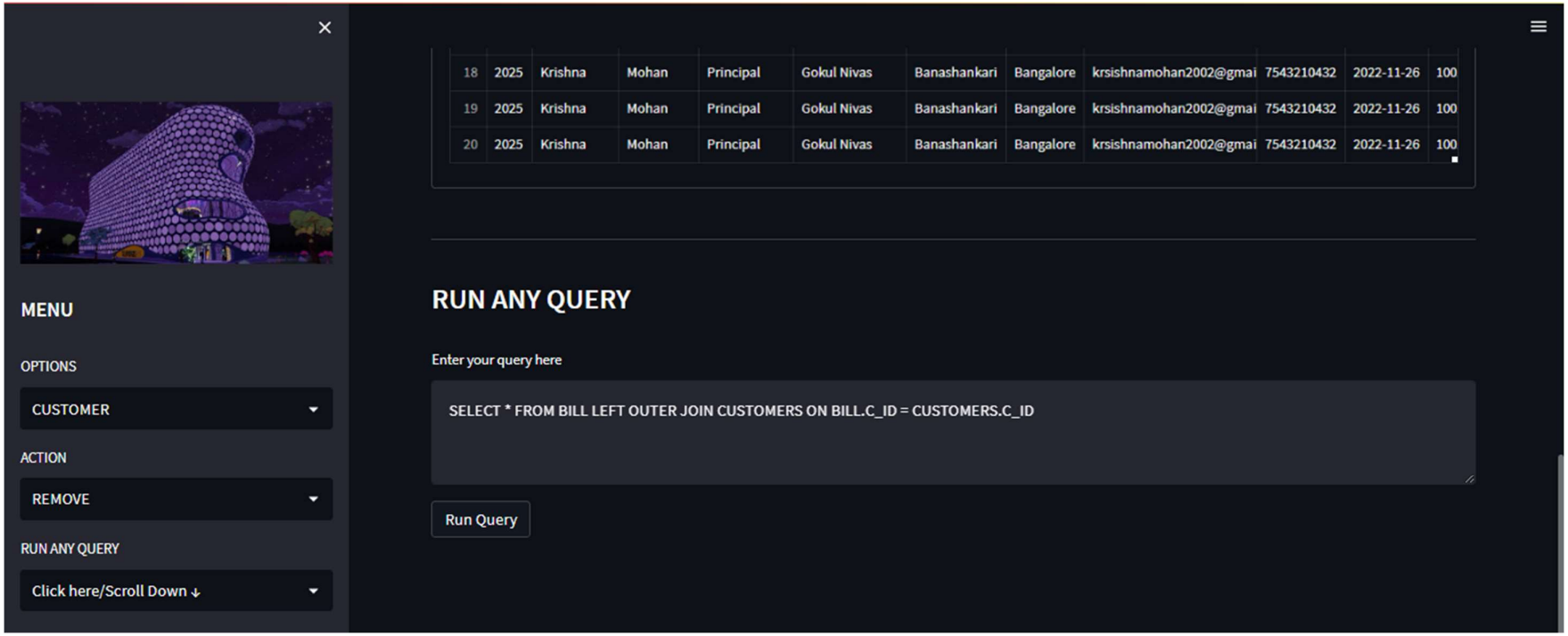
- Deleting the customer



- Updated details in the backup



- Deleted customer in the customer backuplog table.



Similarly frontend is available for all the tables with the following options **EMPLOYEE(Combines Employee table)**, **ITEMS-STOCK(Combines Items ,Item_Category, Contains, Store tables)** and **SUPPLIER-SHIP(Combines Shipment, Ships, Suppliers table)**. **All the CRUD operations can be applied on all the mentioned tables.**

Please use the frontend for all the further CRUD operations on the above-mentioned tables and menu.

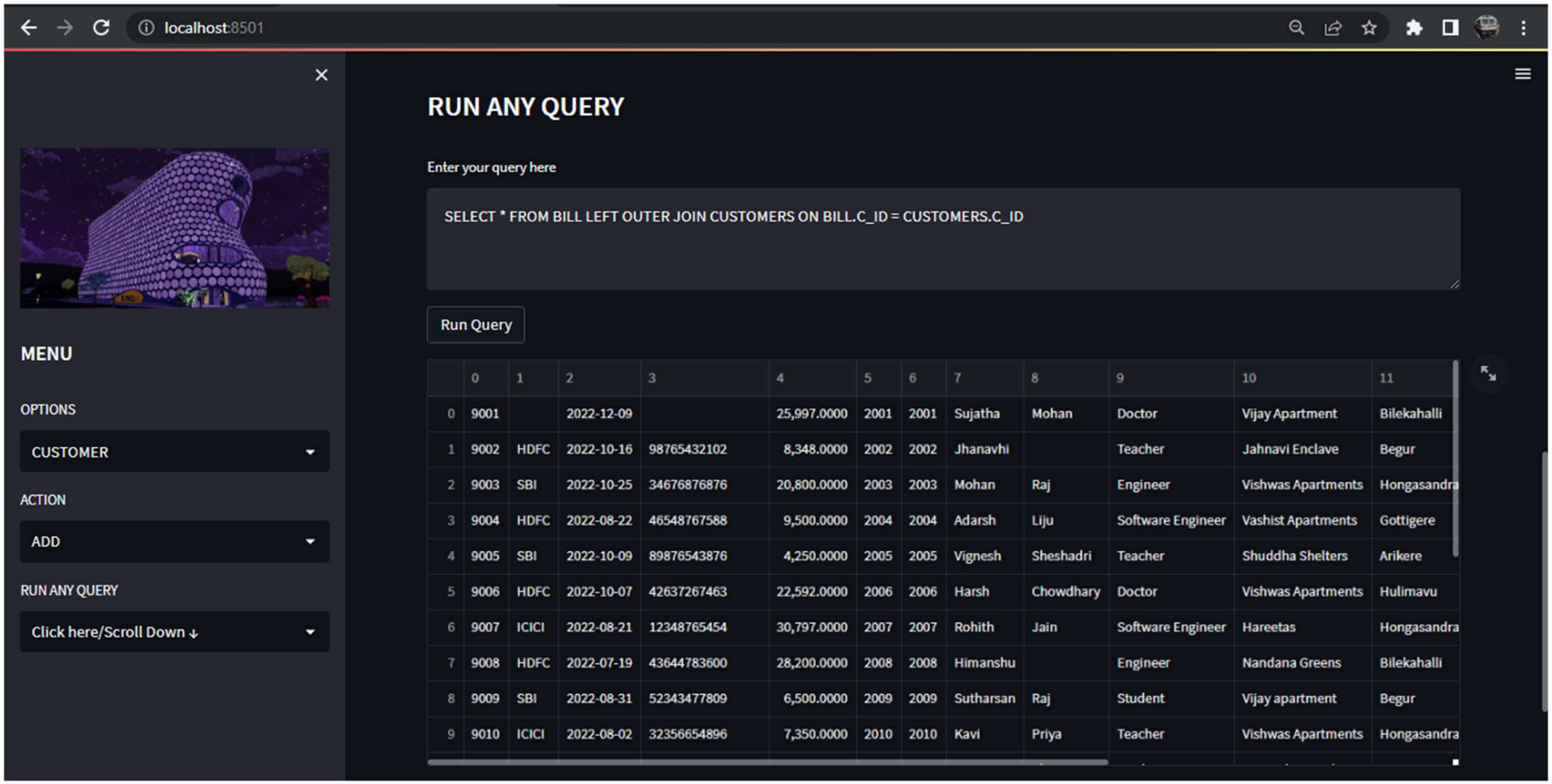
In the front end , the **OPTIONS** refer to the menu as mentioned above namely

- **CUSTOMER (Combines Customers, Orders, Bill tables)**
- **EMPLOYEE(Combines Employee table)**
- **ITEMS-STOCK(Combines Items ,Item_Category, Contains, Store tables)**
- **SUPPLIER-SHIP(Combines Shipment, Ships, Suppliers table)**

In the front end , the **ACTIONS** refer to the CRUD operations namely

- **ADD**
- **READ**
- **UPDATE**
- **REMOVE**

- **Running any query in the front-end app**



	0	1	2	3	4	5	6	7	8	9	10	11
0	9001		2022-12-09		25,997.0000	2001	2001	Sujatha	Mohan	Doctor	Vijay Apartment	Bilekahalli
1	9002	HDFC	2022-10-16	98765432102	8,348.0000	2002	2002	Jhanavhi		Teacher	Jahnavi Enclave	Begur
2	9003	SBI	2022-10-25	34676876876	20,800.0000	2003	2003	Mohan	Raj	Engineer	Vishwas Apartments	Hongasandra
3	9004	HDFC	2022-08-22	46548767588	9,500.0000	2004	2004	Adarsh	Liju	Software Engineer	Vashist Apartments	Gottigere
4	9005	SBI	2022-10-09	89876543876	4,250.0000	2005	2005	Vignesh	Sheshadri	Teacher	Shuddha Shelters	Arikere
5	9006	HDFC	2022-10-07	42637267463	22,592.0000	2006	2006	Harsh	Chowdhary	Doctor	Vishwas Apartments	Hulimavu
6	9007	ICICI	2022-08-21	12348765454	30,797.0000	2007	2007	Rohith	Jain	Software Engineer	Hareetas	Hongasandra
7	9008	HDFC	2022-07-19	43644783600	28,200.0000	2008	2008	Himanshu		Engineer	Nandana Greens	Bilekahalli
8	9009	SBI	2022-08-31	52343477809	6,500.0000	2009	2009	Sutharsan	Raj	Student	Vijay apartment	Begur
9	9010	ICICI	2022-08-02	32356654896	7,350.0000	2010	2010	Kavi	Priya	Teacher	Vishwas Apartments	Hongasandra


Thank
You!