DBMS Project Report

PES University

Database Management Systems

UE18CS252

Submitted By

|  |  |  |
| --- | --- | --- |
| **SRN:** PES1201800190 | **Name:** Sakshi Shetty | **Evaluation date and time:** |

|  |  |
| --- | --- |
| **Functional Dependencies**  **Identifying Keys Based on FDs**  **Normalisation & testing for lossless join property**  **DDL: Table creation with all constraints**  **Triggers**  **SQL Queries**  **Viva / modifications (Unit III / IV concepts)** | ***2***  ***2***  ***2+2***  ***2+2***  ***2***  ***2***  ***2+2*** |

<< **CLOTHES STORE MINIWORLD** >>

|  |
| --- |
| **Problem statement:**  A **clothes store database** needs to store information about its *outlets* (outlet ID, address, contact and email) and the *employees* in each outlet (ID, contact, age, gender, designation, salary and outlet ID). The *customers* relation stores the customer’s ID, name, contact, gender and email and the *orders* relations maps an order ID to each customer ID along with the items, timestamp, outlet ID and the ID of the employee assigned to each customer. Another relation named *items* stores the description of the purchased item (name, size, gender, price). Finally, the relation *bills* keeps track of bill ID, corresponding order, customer and cashier (employee) ID and the amount to be paid along with the payment type.  Drive link containing all my materials for this project:  [ISA 2](https://drive.google.com/drive/folders/1x6KolIowUgpWFDEUKF53NPgQSfW07zop?usp=sharing)  **Contents:** This report in docx format, the same report in PDF format, “FrontEnd.py”, “Createtables&Triggers.sql”, “Insert.sql”, “Queries&LosslessJoin.sql”, and a folder Containing the images used in this document. |

# **Database Schema: (show all the tables and the constraints)**

*NOTE: This is the INITIAL database BEFORE normalisation* *(FIGURE 1, FIGURE 2)*:

1. **ER diagram and Schema**:

|  |  |
| --- | --- |
|  |  |

1. **CONSTRAINTS**:

For data types of every attribute, scroll to section “DDL” below.

The constraints I’ve applied are as follows:

* All primary keys have **NOT NULL** and **UNIQUE** constraints (i.e. no ID can ever be null).
* Details such as contact, address, email, name, size and price of item are also **NOT NULL**. Age attribute for employees is **NOT NULL**.
* **CHECK CONSTRAINT** ensures that only ‘F’ or ‘M’ are entered for GENDER, AGE is always greater than 1 and less than 100, and PAYMENT\_TYPE is ‘CASH’/’CARD’/’UPI’.
* **Referential integrity - FOREIGN KEY CONSTRAINT:**

Referential integrity refers to the accuracy and consistency of data within a relationship. In relationships, data is linked between two or more tables. So, referential integrity requires that whenever a foreign key value is used, it must reference a valid, existing primary key in the parent table. The foreign keys in my schema are as follows:

* *EMPLOYEES:* DSGN\_ID is a foreign key that references DSGN\_ID in the DESIGNATION relation. This is to keep track of what designation each employee has and map it to the list of available designations in DESIGNATION. Similarly, OUTLET\_ID references the OUTLETS relation and keeps track of the outlet in which each employee works.
* *ORDERS:* CUST\_ID references the CUSTOMERS table and EMP\_ID references the employee table to keep track of which employee assisted which customer for every order.
* *ORDER\_DETAILS:* ORDER\_ID and ITEM\_ID are the foreign keys referencing ORDERS and ITEMS respectively to keep track which item pertains to which order (which in turn indicates which customer has purchased which item due to transitivity!)
* *BILLS:* ORDER\_ID and CASHIER\_ID are the foreign keys referencing ORDERS and EMPLOYEES respectively to keep track of which order each bill belongs to and which employee handled the purchase and transaction.

**Functional Dependencies: (List based on your application constraints)**

Before normalisation, we can map the functional dependencies that exist in this schema to gain clarity on what can be made the candidate keys for each relation. Further, upon applying **attribute closure**, we can achieve more clarity on choosing the respective primary keys.

OUTLETS:

(OUTLET\_ID)+ = {OUTLET\_ID, ADDRESS, EMAIL, CONTACT}

However, since all of them are unique (no 2 outlets can have the same address/contact/mail), we see that:

(ADDRESS)+ = {OUTLET\_ID, ADDRESS, EMAIL, CONTACT}

(EMAIL)+ = {OUTLET\_ID, ADDRESS, EMAIL, CONTACT}

(CONTACT)+ = {OUTLET\_ID, ADDRESS, EMAIL, CONTACT}

Which means that any of these can be made the primary key, so they’re all candidate keys. I have chosen OUTLET\_ID as the primary key.

EMPLOYEES:

(EMP\_ID)+ = {EMP\_ID, EMP\_NAME, EMP\_CONTACT, GENDER, AGE, DSGN, SALARY, OUTLET\_ID}

(EMP\_CONTACT)+ = {EMP\_ID, EMP\_NAME, EMP\_CONTACT, GENDER, AGE, DSGN, SALARY, OUTLET\_ID}

These 2 are candidate keys, out of which I have chosen EMP\_ID as the primary key.

Only designation indicates the salary (such issues will be solved during normalisation)

(DSGN)+ = {DSGN, SALARY}

Note that OUTLET\_ID is a foreign key from the OUTLETS relation.

CUSTOMERS:

(CUST\_ID)+ = {CUST\_ID, CUST\_NAME, CUST\_CONTACT, GENDER, EMAIL}

(CUST\_CONTACT)+ = {CUST\_ID, CUST\_NAME, CUST\_CONTACT, GENDER, EMAIL}

(EMAIL)+ = CUST\_ID, CUST\_NAME, CUST\_CONTACT, GENDER, EMAIL}

All 3 of these can derive every other attribute, which means that every other attribute is functionally dependent on these 3 due to their uniqueness. Hence they’re candidate keys, out of which I have chosen CUST\_ID as the primary key.

ORDERS:

(ORDER\_ID)+ = {ORDER\_ID, CUST\_ID, ITEMS, TIME\_STAMP, OUTLET\_ID, EMP\_ID}

(EMP\_ID)+ = {EMP\_ID, OUTLET\_ID}

I have chosen ORDER\_ID as the primary key. Note that CUST\_ID and EMP\_ID are foreign keys referencing CUSTOMERS and EMPLOYEES respectively.

ITEMS:

(ITEM\_NAME)+ = {ITEM\_NAME, SIZE, GENDER, PRICE}

Only ITEM\_NAME can uniquely determine the other attributes in this relation. This is the primary key here.

BILLS:

(BILL\_ID)+ = {BILL\_ID, ORDER\_ID, CUST\_ID, CASHIER\_ID, AMOUNT, PAYMENT\_TYPE}

(ORDER\_ID)+ = {BILL\_ID, ORDER\_ID, CUST\_ID, CASHIER\_ID, AMOUNT, PAYMENT\_TYPE}

(CUST\_ID)+ = {BILL\_ID, ORDER\_ID, CUST\_ID, CASHIER\_ID, AMOUNT, PAYMENT\_TYPE}

Thus, the set of candidate keys is {BILL\_ID, ORDER\_ID, CUST\_ID}, out of which I have chosen BILL\_ID as the primary key, because ORDER\_ID and CUST\_ID are foreign keys and any violation of referential integrity can be avoided by not making one of them a primary key. Additionally, CASHIER\_ID can’t be a candidate or a primary key because 1 cashier can handle multiple different bills.

**Candidate keys: (Justify how did you get these as keys) /**

**Identifying Keys based on FDs**

As stated above in the “Functional Dependencies” section, the candidate keys for each relation are:

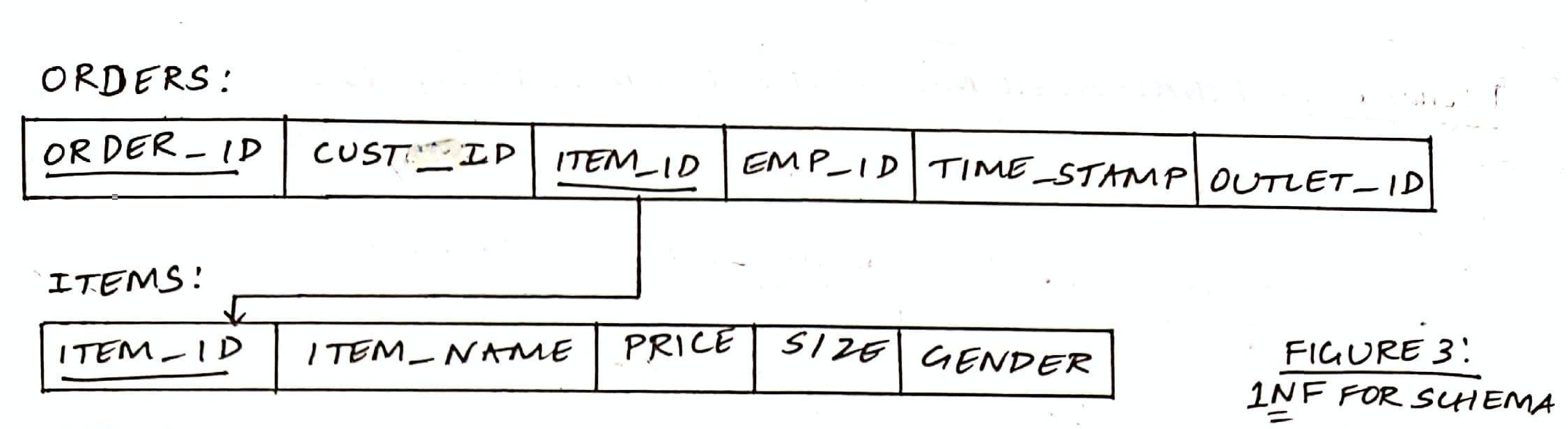
* OUTLETS: The set of all attributes is the candidate key set - {OUTLET\_ID, ADDRESS, EMAIL, CONTACT} - since each of them are unique on their own and can uniquely determine other attributes as well. I chose OULTET\_ID as the primary key.
* EMPLOYEES: Candidate key set is {EMP\_ID, EMP\_CONTACT} due to both being unique, and I have chosen EMP\_ID as the primary key.
* CUSTOMERS: Candidate key set is {CUST\_ID, CUST\_CONTACT, EMAIL} out of which I have chosen CUST\_ID to be the primary key.
* ORDERS: Only ORDER\_ID is the candidate and primary key because none of the other attributes need to be unique. For example, the same customer can have many different order IDs due to choosing more than 1 item.
* ITEMS: Only ITEM\_NAME can uniquely determine the other attributes hence it’s the candidate as well a primary key.
* BILLS: The set of candidate keys is {BILL\_ID, ORDER\_ID, CUST\_ID} but I have chosen BILL\_ID as the primary key to avoid any possible problems that may arise with the other 2 attributes since they are foreign keys.

**Normalization and testing for lossless join property:**

A) 1NF - FIRST NORMAL FORM:

If a relation contains a composite or multi-valued attribute, it violates first normal form or a relation is in first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is **a single valued attribute**.

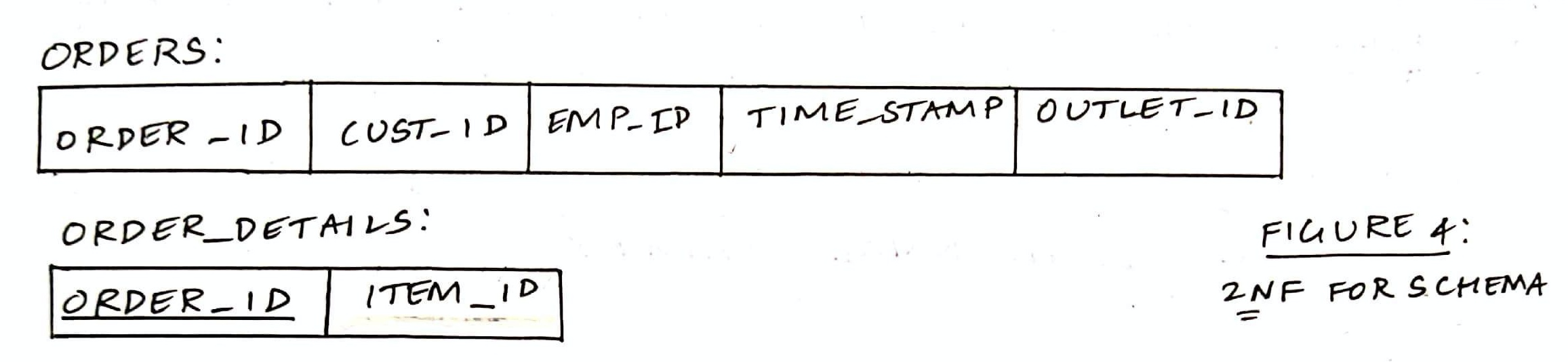
In ORDERS, the attribute ITEMS can have multiple values - for example, “Jeans, Kurta, Shirt”. To get rid of this and ensure atomicity, I changed ITEMS in ORDERS to ITEM\_ID and included ITEM\_ID as a foreign key in the ITEMS relation *(FIGURE 3)*.



B) 2NF - SECOND NORMAL FORM:

To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency. A relation is in 2NF if it has **no partial dependency,** i.e.**,** no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

In ORDERS, the candidate key after achieving 1NF is {ORDER\_ID, CUST\_ID, ITEM\_ID}. However, the TIME\_STAMP attribute only depends on ORDER\_ID and CUST\_ID. Thus to achieve 2NF, I split ORDERS into 2 tables as follows *(FIGURE 4)*.

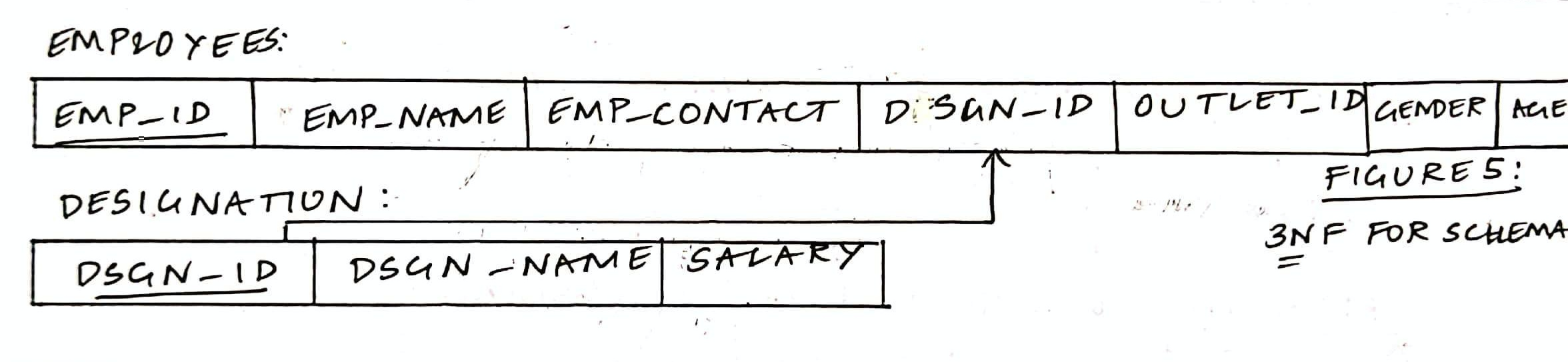


C) 3NF - THIRD NORMAL FORM

A relation is in third normal form, if there is **no transitive dependency** for non-prime attributes as well as it is in second normal form. A relation is in 3NF if **at least one of the following condition holds** in every non-trivial functional dependency X –> Y:

* X is a super key.
* Y is a prime attribute (each element of Y is part of some candidate key).

In EMPLOYEE, EMP\_ID -> DSGN and DSGN -> SALARY, but EMP\_ID and SALARY do not display functional dependency. To resolve this and achieve 3NF, I replaced DSGN with DSGN\_ID as a foreign key referencing a new relation DESIGNATION as follows *(FIGURE 5)*:



D) REMOVING REDUNDANCIES

In ORDERS, ORDER\_ID -> EMP\_ID and EMP\_ID -> OUTLET\_ID. So, there’s no need to have this attribute in the relation and hence it can be dropped *(FIGURE 6)*.



Now, **the schema has been normalised up to the third form** and all other unnecessary redundancies have been omitted. Thus, the final and new schema of the database is as shown below *(FIGURE 7)*.



**TESTING FOR** **LOSSLESS JOIN**:

According to the textbook:

**Claim 2 (Preservation of Nonadditivity in Successive Decompositions):** If a

decomposition D = {R1, R2, … , Rm} of R has the nonadditive (lossless) join

property with respect to a set of functional dependencies F on R, and if a

decomposition Di = {Q1, Q2, … , Qk} of Ri has the nonadditive join property with

respect to the projection of F on Ri, then the decomposition D2 = {R1, R2, … ,

Ri−1, Q1, Q2, … , Qk, Ri+1, … , Rm} of R has the nonadditive join property with

respect to F.

For instance, I have the BILLS = {BILL\_ID, ORDER\_ID, CASHIER\_ID, AMOUNT, PAYMENT\_TYPE} relation. Adhering to the functional dependencies and decomposition based on 3NF that I’ve mentioned in the previous sections, the decomposition of this table based on FDs leads to F = {BILL\_ID -> {ORDER\_ID, CASHIER\_ID}, BILL\_ID -> {AMOUNT, PAYMENT\_TYPE}}.

Therefore, upon splitting BILLS as above we get:

|  |  |
| --- | --- |
| CREATE TABLE R1 (  BILL\_ID INT, AMOUNT FLOAT, PAYMENT\_TYPE VARCHAR(4)  );  INSERT INTO R1  SELECT BILL\_ID, AMOUNT, PAYMENT\_TYPE FROM BILLS;  GO |  |
| CREATE TABLE R2 (  BILL\_ID INT, ORDER\_ID INT, CUST\_ID INT  );  INSERT INTO R2  SELECT BILL\_ID, ORDER\_ID, CASHIER\_ID FROM BILLS;  GO |  |
| SELECT \* FROM R2 JOIN R1 ON R2.BILL\_ID = R1.BILL\_ID ORDER BY R2.BILL\_ID;  GO |  |

**Due to normalisation** (BILLS is in 3NF) we can see that **only those that match in both tables** are displayed and there are only values that satisfy the functional dependencies as they should. Achieving 3NF automatically makes the decomposition and joining of every table lossless. Thus, **lossless join has been achieved** by ensuring that the schema has been normalised to 3NF.

**DDL:**

1. **CREATE** STATEMENTS WITH ALL CONSTRAINTS APPLIED

**CREATE TABLE OUTLETS (**

**OUTLET\_ID INT NOT NULL PRIMARY KEY,**

**UNIQUE(OUTLET\_ID),**

**OUTLET\_ADDRESS VARCHAR(100) NOT NULL,**

**EMAIL VARCHAR(50) NOT NULL,**

**CONTACT VARCHAR(10) NOT NULL**

**);**

**CREATE TABLE DESIGNATION (**

**DSGN\_ID INT NOT NULL PRIMARY KEY,**

**DSGN\_NAME VARCHAR(15) NOT NULL,**

**SALARY FLOAT NOT NULL**

**);**

**CREATE TABLE EMPLOYEES (**

**EMP\_ID INT NOT NULL PRIMARY KEY,**

**UNIQUE(EMP\_ID),**

**EMP\_NAME VARCHAR(25) NOT NULL,**

**EMP\_CONTACT VARCHAR(10) NOT NULL,**

**GENDER CHARACTER(1),**

**CHECK (GENDER='F' OR GENDER='M'),**

**AGE INT NOT NULL,**

**CHECK (AGE>1 AND AGE<100),**

**DSGN\_ID INT NOT NULL,**

**FOREIGN KEY (DSGN\_ID) REFERENCES DESIGNATION(DSGN\_ID),**

**OUTLET\_ID INT NOT NULL,**

**FOREIGN KEY (OUTLET\_ID) REFERENCES OUTLETS(OUTLET\_ID)**

**);**

**CREATE TABLE CUSTOMERS (**

**CUST\_ID INT NOT NULL PRIMARY KEY,**

**CUST\_NAME VARCHAR(25) NOT NULL,**

**CUST\_CONTACT VARCHAR(10),**

**EMAIL VARCHAR(50),**

**GENDER CHARACTER(1),**

**CHECK (GENDER='F' OR GENDER='M')**

**);**

**CREATE TABLE ORDERS (**

**ORDER\_ID INT NOT NULL PRIMARY KEY,**

**CUST\_ID INT NOT NULL,**

**FOREIGN KEY (CUST\_ID) REFERENCES CUSTOMERS(CUST\_ID),**

**EMP\_ID INT NOT NULL,**

**FOREIGN KEY (EMP\_ID) REFERENCES EMPLOYEES(EMP\_ID),**

**TIME\_STAMP DATETIME**

**);**

**CREATE TABLE ITEMS (**

**ITEM\_ID INT NOT NULL PRIMARY KEY,**

**ITEM\_NAME VARCHAR(30) NOT NULL,**

**GENDER CHARACTER(1),**

**CHECK (GENDER='F' OR GENDER='M'),**

**SIZE VARCHAR(4) NOT NULL,**

**PRICE FLOAT NOT NULL**

**);**

**CREATE TABLE ORDER\_DETAILS (**

**ORDER\_ID INT NOT NULL PRIMARY KEY,**

**FOREIGN KEY (ORDER\_ID) REFERENCES ORDERS(ORDER\_ID),**

**ITEM\_ID INT NOT NULL,**

**FOREIGN KEY (ITEM\_ID) REFERENCES ITEMS(ITEM\_ID)**

**);**

**CREATE TABLE BILLS (**

**BILL\_ID INT NOT NULL PRIMARY KEY,**

**ORDER\_ID INT NOT NULL,**

**FOREIGN KEY (ORDER\_ID) REFERENCES ORDERS(ORDER\_ID),**

**CASHIER\_ID INT NOT NULL,**

**FOREIGN KEY (CASHIER\_ID) REFERENCES EMPLOYEES(EMP\_ID),**

**AMOUNT FLOAT,**

**PAYMENT\_TYPE VARCHAR(4),**

**CHECK (PAYMENT\_TYPE = 'CASH' OR PAYMENT\_TYPE = 'CARD' OR PAYMENT\_TYPE = 'UPI')**

**);**

1. **INSERTION**:

INSERT INTO OUTLETS VALUES

('1','Rajajinagar','lifestyle.rajajinagar@gmail.com','080234338'),

('2','Jayanagr','lifestyle.jayanagar@gmail.com','082766804'),

('3','Vijayanagar','lifestyle.vijayanagar@gmail.com','0802342367'),

('4','Indiranagar','lifestyle.indiranagar@gmail.com','0802834516');

INSERT INTO DESIGNATION VALUES

('11','MANAGER','250000'),

('12','ACCOUNTANT','80000'),

('13','LAWYER','200000'),

('14','REGULAR','25000'),

('15','SECURITY','8500');

INSERT INTO EMPLOYEES VALUES

('21','Amulya Rao','9611710889','F','35','11','1'),

('22','Karthik Ramakishnan','8897742010','M','31','12','1'),

('23','Naina Agnihotri','7756330217','F','42','13','1'),

('24','Aradhika Menon','8867549930','F','23','14','1'),

('25','Rahul Tandon','6367259981','M','27','14','1'),

('26','Hrishikesh Shet','9127763091','M','26','14','1'),

('27','Zaira Mundrawala','6465875320','F','24','14','1'),

('28','Latha Veeramma','9876540981','F','33','14','1'),

('29','Aishwarya Hegde','8897065948','F','30','14','1'),

('30','Rahil Sheikh','7765893091','M','48','15','1'),

('31','Maithri Iyengar','6289605531','F','40','11','2'),

('32','Vashisht Shah','8769504376','M','39','12','2'),

('33','Zoran Syed','8756435192','M','33','13','2'),

('34','Deshna Patil','7568493021','F','29','14','2'),

('35','Ridhima Kasbekar','8190675439','F','31','14','2'),

('36','Sheryl Rebello','6574937639','F','25','14','2'),

('37','Markhand Arora','7659874630','M','30','14','2'),

('38','Akash Bhat','7685048322','M','26','14','2'),

('39','Maanvi Mallya','6578390261','F','36','14','2'),

('40','Raghu Hoskote','8956473822','M','52','15','2'),

('41','Aaryan Shetty','6789567830','M','38','11','3'),

('42','Swathi Nair','7869546372','F','41','12','3'),

('43','Trisha Swaika','6574889032','F','37','13','3'),

('44','Akshatha Maheshwari','9808076541','F','24','14','3'),

('45','Madhoolika Swaraj','8796657748','F','29','14','3'),

('46','Rakshith Holla','7869546300','M','36','14','3'),

('47','Dhanyatha Reddy','8796554409','F','26','14','3'),

('48','Deemahi Joshi','6759608737','F','25','14','3'),

('49','Shanaya Pithawala','8796055432','F','34','14','3'),

('50','Dinesh Singh','7685900867','M','56','15','3'),

('51','Sanath Shettigar','6457683399','M','47','11','4'),

('52','Sumukh Bharadwaj','6758890011','M','39','12','4'),

('53','Aanchal Pratap','8769906574','F','32','13','4'),

('54','Rohith Mukherjee','7659876079','M','36','14','4'),

('55','Meenal Sanghvi','9898776510','F','28','14','4'),

('56','Reesha Shenoy','7686004328','F','37','14','4'),

('57','Siddharth Naik','7659511220','M','27','14','4'),

('58','Lalit Vaishnav','6325490844','M','33','14','4'),

('59','Irene Lasrado','7693324907','F','29','14','4'),

('60','Nikhil Thirthahalli','9032514730','M','49','15','4');

INSERT INTO CUSTOMERS VALUES

('101','Saachi Sangankal','7609845377','saachi@yahoo.com','F'),

('102','Sachin Bhosle','8760098021','sachin@yahoo.com','M'),

('103','Aaliya Mehra','6275640911','aaliya@yahoo.com','F'),

('104','Sunaina Ghosh','7299648100','sunaina@yahoo.com','F'),

('105','Bharath Bohra','9856443703','bharath@yahoo.com','M'),

('106','Arya Khurana','9034327665','arya@yahoo.com','F'),

('107','Milind Bhose','7775940210','milind@yahoo.com','M'),

('108','Karan Rai','6364542291','karan@yahoo.com','M'),

('109','Vedika Hosahalli','8988887654','vedika@yahoo.com','F'),

('110','Aaron Coelho','7564983340','aaran@yahoo.com','M');

INSERT INTO ORDERS VALUES

('201','101','24',getdate()),

('202','101','24',getdate()),

('203','102','35',getdate()),

('204','103','36',getdate()),

('205','103','36',getdate()),

('206','104','37',getdate()),

('207','104','37',getdate()),

('208','105','48',getdate()),

('209','105','48',getdate()),

('210','106','46',getdate()),

('211','107','44',getdate()),

('212','108','55',getdate()),

('213','109','57',getdate()),

('214','109','57',getdate()),

('215','110','58',getdate());

INSERT INTO ITEMS VALUES

('301','Black Dress','F','S','1299'),

('302','Red Shirt','F','S','899'),

('303','Navy Blazer','M','M','2499'),

('304','Paisley Kurta','F','L','699'),

('305','Purple Leggings','F','XL','449'),

('306','Plaid Skirt','F','XS','1499'),

('307','Brown Turtleneck','F','XS','999'),

('308','Pyjama','M','XXL','799'),

('309','Vest','M','XXL','199'),

('310','Silk Salwar','F','M','899'),

('311','Olive Green Shirt','M','M','1099'),

('312','Beige Trousers','M','S','2749'),

('313','Pink Crop Top','F','XS','649'),

('314','Skinny Jeans','F','XS','2549'),

('315','Polo T Shirt','M','L','1699');

INSERT INTO ORDER\_DETAILS VALUES

('201','301'),

('202','302'),

('203','303'),

('204','304'),

('205','305'),

('206','306'),

('207','307'),

('208','308'),

('209','309'),

('210','310'),

('211','311'),

('212','312'),

('213','313'),

('214','314'),

('215','315');

INSERT INTO BILLS VALUES

('401','201','28','2198','CASH'),

('402','203','28','2499','CASH'),

('403','204','29','1198','CARD'),

('404','206','38','2498','UPI'),

('405','208','39','998','CARD'),

('406','210','39','899','CASH'),

('407','211','45','1099','CASH'),

('408','212','49','2749','CASH'),

('409','213','56','2198','UPI'),

('410','215','59','1699','UPI');

**Triggers:**

A trigger is a special type of stored procedure that automatically runs when an event occurs in the database server. DML triggers run when a user tries to modify data through a data manipulation language (DML) event, such as INSERT, UPDATE, or DELETE statements on a table.

I have used an **AFTER INSERT trigger** named “OfferType” to decide if a customer is eligible to receive a gift voucher. Here, my trigger is fired when INSERT operation takes place on the BILLS relation and stores the BILL\_ID and ORDER\_ID of those customers whose bill amount exceeds 2500 (indicated by the SELECT statements in the figure). This information is pushed into a fresh relation named OFFERS which keeps the list of bill and order IDs of those that **won a voucher** by shopping for more than Rupees 2500.

CREATE TABLE OFFERS (

BILL\_ID INT,

ORDER\_ID INT,

*--AMOUNT FLOAT,*

OFFER VARCHAR(25)

);

GO

CREATE TRIGGER OfferType ON BILLS

FOR INSERT

AS DECLARE @Bid INT, @Oid INT, @Offer VARCHAR(25);

SELECT @Bid = i.BILL\_ID FROM INSERTED i WHERE i.AMOUNT>2500;

SELECT @Oid = i.ORDER\_ID FROM INSERTED i WHERE i.AMOUNT>2500;

SET @Offer = 'Gift Voucher worth 1000 Rs';

*--SELECT @Amt = i.AMOUNT FROM INSERTED i WHERE i.AMOUNT>3000;*

*--SET @Offer ='Gift Voucher' WHERE i.Amount > 4000 FROM INSERTED i;*

INSERT INTO OFFERS(BILL\_ID,ORDER\_ID,OFFER) VALUES (@Bid,@Oid,@Offer);

PRINT 'AFTER INSERT trigger has been fired; offers have been updated accordingly.'

*--GO*

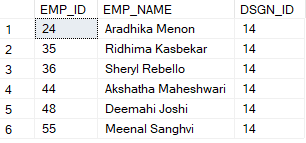
For the values that I have inserted into BILLS, the OFFERS table looks like this after the trigger has been fired:



# **SQL Queries:**

1. Select only female employees who are assigned to the customers currently shopping today across all outlets of the store *(nested)*:

There are 2 ways of doing this - one using nested queries and another using *correlation*:



|  |  |
| --- | --- |
| SELECT E.EMP\_ID, E.EMP\_NAME, E.DSGN\_ID  FROM EMPLOYEES AS E  WHERE E.EMP\_ID IN  (SELECT ORDERS.EMP\_ID FROM ORDERS  WHERE E.GENDER = 'F');  *--OR, CONVERTING THE SAME INTO CORRELATION QUERY:*  SELECT DISTINCT E.EMP\_ID, E.EMP\_NAME, E.DSGN\_ID  FROM EMPLOYEES AS E, ORDERS AS O  WHERE E.EMP\_ID=O.EMP\_ID AND E.GENDER = 'F'; |  |

1. Display the names and IDs of all employees who are working today, i.e only the employees who carried out orders for customers or billed them at the cash counter *(nested)*:

|  |  |
| --- | --- |
| SELECT DISTINCT EMPLOYEES.EMP\_ID, EMPLOYEES.EMP\_NAME  FROM EMPLOYEES  WHERE EMPLOYEES.EMP\_ID IN  (SELECT EMPLOYEES.EMP\_ID  FROM EMPLOYEES, ORDERS  WHERE EMPLOYEES.EMP\_ID = ORDERS.EMP\_ID)  OR EMPLOYEES.EMP\_ID IN  (SELECT EMPLOYEES.EMP\_ID  FROM EMPLOYEES, BILLS  WHERE EMPLOYEES.EMP\_ID = BILLS.CASHIER\_ID);  GO |  |

1. Display the total bill amount in a day *(aggregate function: SUM)*:

SELECT SUM(AMOUNT) FROM BILLS;

*O/P = 19035*

1. Display the average age of male employees *(aggregate function: AVG)*:

SELECT AVG(AGE) FROM EMPLOYEES WHERE GENDER = 'M';

*O/P = 37*

1. Display the age of the youngest manager for this brand in the city *(aggregate function: MIN)*:

SELECT MIN(AGE)

FROM EMPLOYEES

WHERE EMPLOYEES.DSGN\_ID = '11';

*O/P = 35*