MINI PROJECT PRESENTATION

DATABASE MANAGEMENT SYSTEM

Great Learning

POWER AHEAD

TABLE OF CONTENTS

1. Team Members
2. Problem Statement
3. Technology Used
4. Skills Developed
5. Data Description
6. Problem Solving and ER Models
7. Future Steps
8. Conclusion

Problem Statement

The objective of this report is to document the process of creating tables for data in the Bank Retail Case Study and inserting records into those tables while maintaining data integrity. This includes ensuring that all data entered into the tables conforms to the defined data types and constraints, and that all relationships between the tables are accurately established. In this project, we aim to create separate tables for each entity - customer, employee, and account - and integrate them through referential integrity. We also need to ensure data integrity in our database is accurately established.

Technology Used

MySQL is the software used for the project. We chose to use MySQL because it is a relational database management system. It stores data across multiple tables which can then be queried to get desired output. it is also free which would help us negate the cost of procurement.

Skills Developed

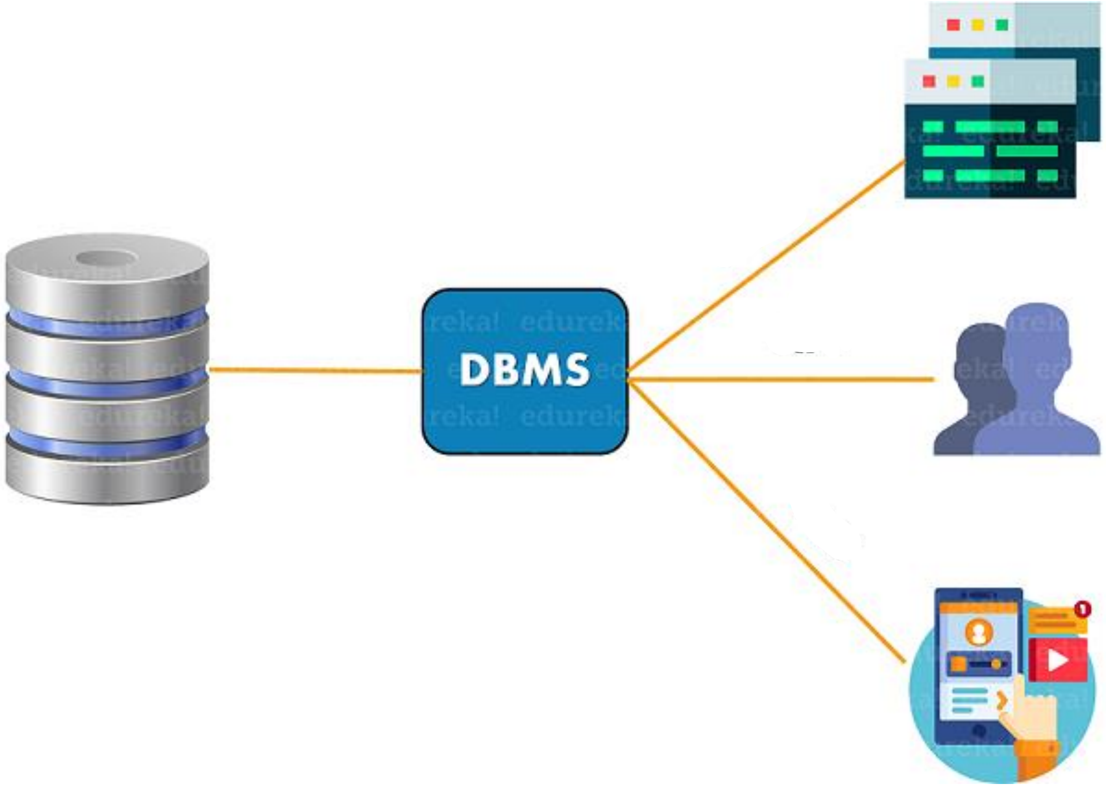
During the implementation of a MySQL we developed many skills:

1. Database creation & Backup
2. DDL operations
3. DML operations
4. DQL operations
5. Views creation
6. Join statements
7. Subqueries.
8. Window functions
9. Case & When statements
10. Trigger statements etc.

Data Description

We are provided with retail banking business model which contains two schemas. Schema Bank contains records of Branch, Employees, Customers and Accounts. Schema Customer contains order items, Carton, order header, address, online customer, shipper, product, product class. Each table has constraints with which they are related.

Problem Solving



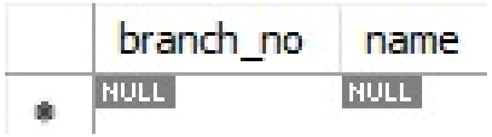
Part - A

1. Create Database BANK And Write SQL Query To Create Above Schema With Constraints

```
CREATE DATABASE dbms_project1;
USE dbms_project1;
```

Part-1. CREATE TABLE BRANCH

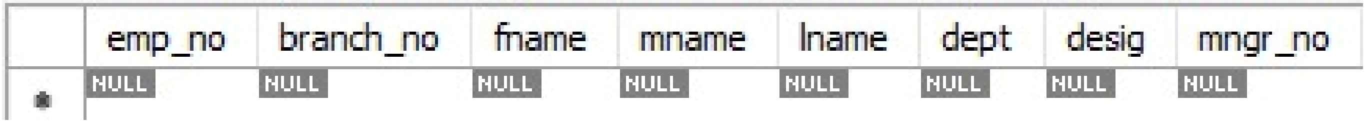
```
CREATE TABLE branch(
branch_no INT AUTO_INCREMENT ,
name CHAR(50) NOT NULL
PRIMARY KEY (branch_no) );
```



	branch_no	name
●	NULL	NULL

Part-2. CREATE TABLE EMPLOYEES

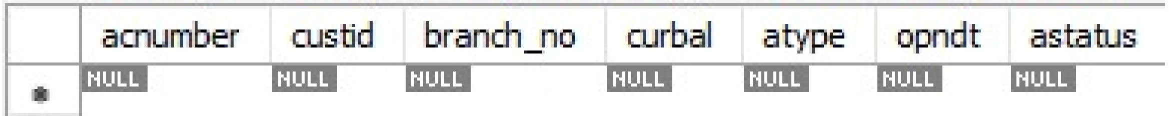
```
CREATE TABLE employees(
emp_no INT AUTO_INCREMENT,
branch_no INT,
fname CHAR(20),
mname CHAR(20),
lname CHAR(20),
dept CHAR(20),
desig CHAR(10),
mngr_no INT  NOT NULL,
PRIMARY KEY (emp_no),
FOREIGN KEY (branch_no) REFERENCES branch (branch_no));
```



	emp_no	branch_no	fname	mname	lname	dept	desig	mngr_no
●	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Part-3. CREATE TABLE accounts

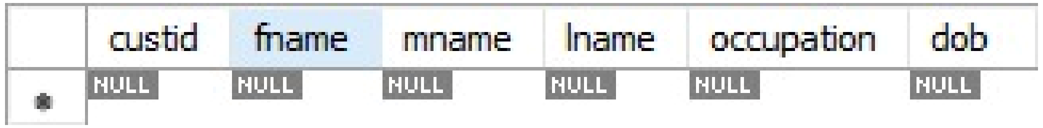
```
CREATE TABLE accounts(
acnumber INT,
custid INT,
branch_no INT,
curbal INT,
atype CHAR(10),
opndt DATE,
astatus CHAR(10),
PRIMARY KEY (acnumber),
FOREIGN KEY (custid) REFERENCES customers(custid),
FOREIGN KEY (branch_no) REFERENCES
branch(branch_no));
```



	acnumber	custid	branch_no	curbal	atype	opndt	astatus
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

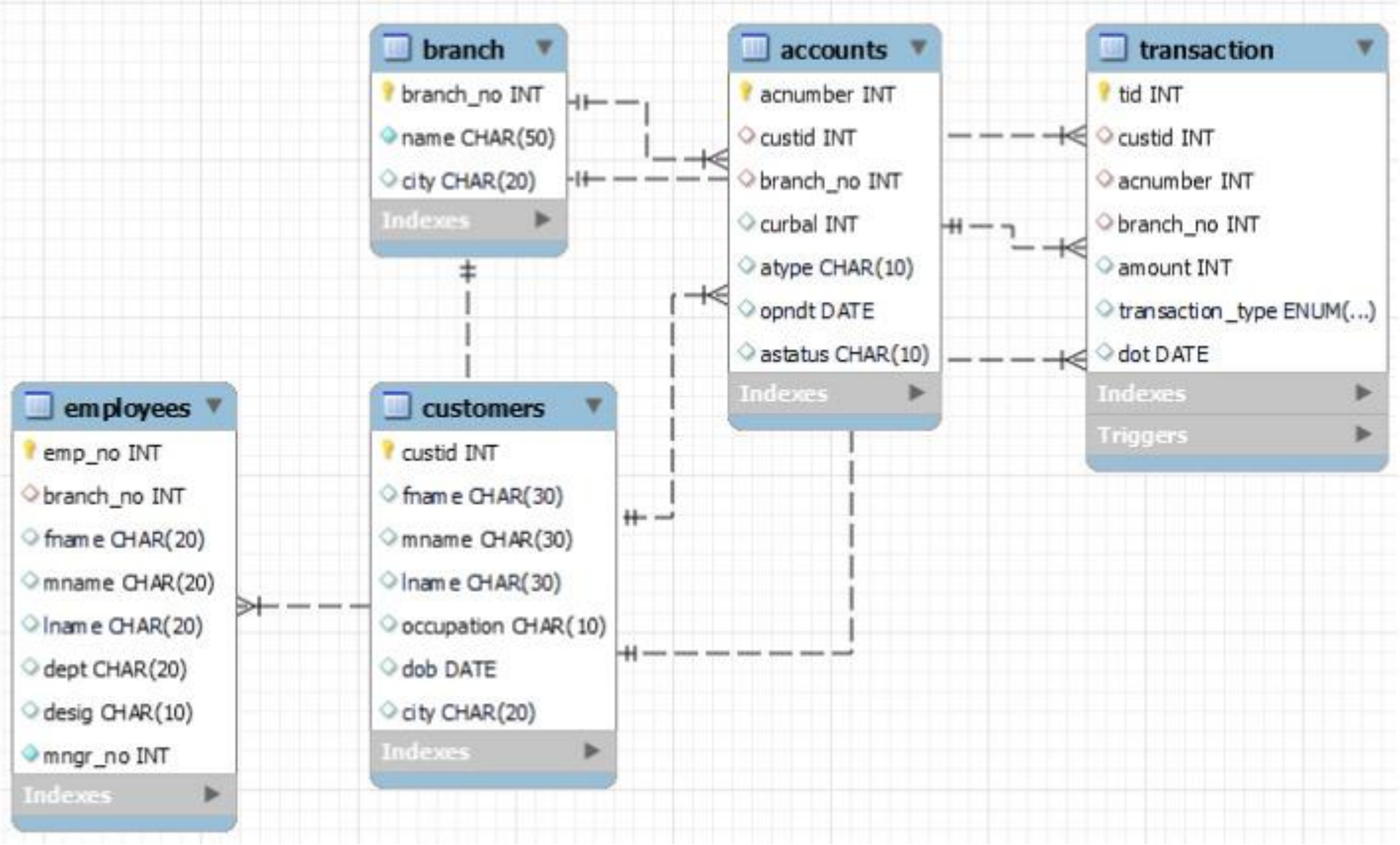
Part-4. CREATE TABLE customers

```
CREATE TABLE customers(
custid INT AUTO_INCREMENT,
fname CHAR(30),
mname CHAR(30),
lname CHAR(30),
occupation CHAR(10),
dob DATE,
PRIMARY KEY (custid));
```



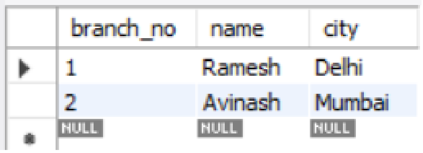
	custid	fname	mname	lname	occupation	dob
*	NULL	NULL	NULL	NULL	NULL	NULL

ER DIAGRAM


branch
branch_no INT
name CHAR(50)
city CHAR(20)
Indexes
accounts
acnumber INT
custid INT
branch_no INT
curbal INT
atype CHAR(10)
opndt DATE
astatus CHAR(10)
Indexes
transaction
tid INT
custid INT
acnumber INT
branch_no INT
amount INT
transaction_type ENUM(...)
dot DATE
Indexes
Triggers
employees
emp_no INT
branch_no INT
fname CHAR(20)
mname CHAR(20)
lname CHAR(20)
dept CHAR(20)
desig CHAR(10)
mngr_no INT
Indexes
customers
custid INT
fname CHAR(30)
mname CHAR(30)
lname CHAR(30)
occupation CHAR(10)
dob DATE
city CHAR(20)
Indexes

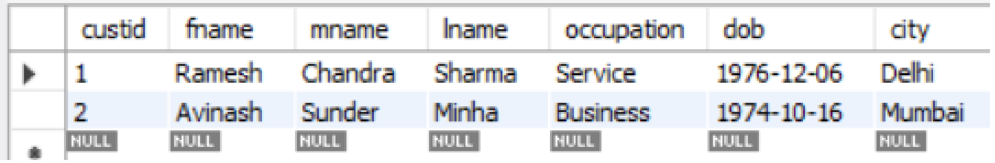
2. Inserting Records Into Created Tables

```
INSERT INTO branch VALUES (1, 'Delhi'), (2, 'Mumbai’);
```



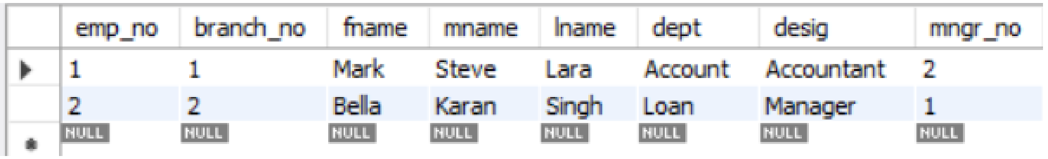
branch_no	name	city
1	Ramesh	Delhi
2	Avinash	Mumbai
NULL	NULL	NULL

```
INSERT INTO customers VALUES
(1,'Ramesh','Chandra', 'Sharma', 'Service', '1976-12-06’),
(2,'Avinash','Sunder', 'Minha', 'Business', '1974-10-16’);
```



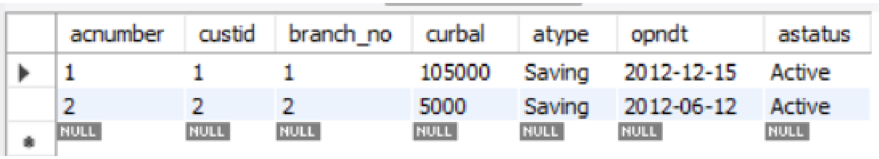
custid	fname	mname	lname	occupation	dob	city
1	Ramesh	Chandra	Sharma	Service	1976-12-06	Delhi
2	Avinash	Sunder	Minha	Business	1974-10-16	Mumbai
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
INSERT INTO employees VALUES
(1, 1, 'Mark','Steve', 'Lara', 'Account', 'Accountant', 2),
(2, 2, 'Bella','James', 'Ronald', 'Loan', 'Manager', 1);
```



emp_no	branch_no	fname	mname	lname	dept	desig	mngr_no
1	1	Mark	Steve	Lara	Account	Accountant	2
2	2	Bella	Karan	Singh	Loan	Manager	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
INSERT INTO accounts VALUES
(1, 1, 1, 100000, 'Saving', '2012-12-15','Active'),
(2, 2, 2, 5000, 'Saving', '2012-06-12','Active');
```

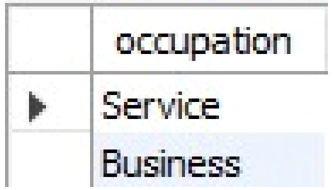


acnumber	custid	branch_no	curbal	atype	opndt	astatus
1	1	1	105000	Saving	2012-12-15	Active
2	2	2	5000	Saving	2012-06-12	Active
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Great Learning
POWER AHEAD

3. Select Unique Occupation From Customer Table

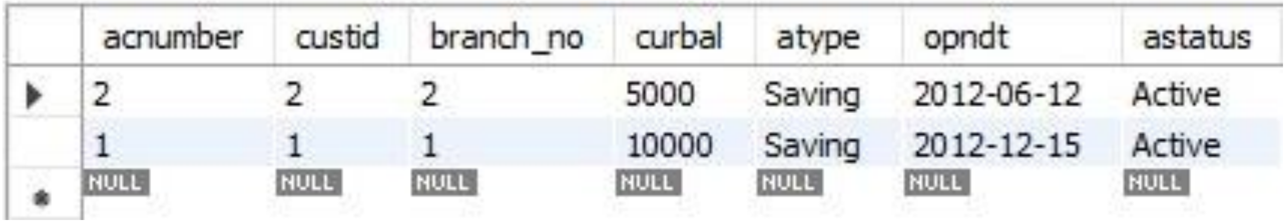
```
SELECT DISTINCT occupation FROM customers;
```



	occupation
▶	Service
	Business

4. Sort Accounts According To Current Balance

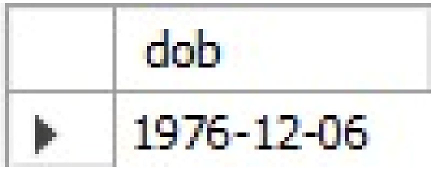
```
SELECT * FROM accounts ORDER BY curbal ASC;
```



	acnumber	custid	branch_no	curbal	atype	opndt	astatus
▶	2	2	2	5000	Saving	2012-06-12	Active
	1	1	1	10000	Saving	2012-12-15	Active
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

5. Find The Date Of Birth Of Customer Name ‘Ramesh’

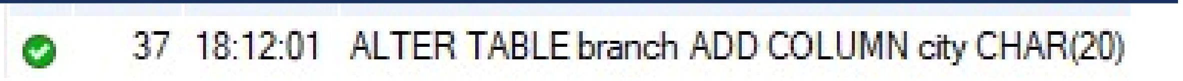
```
SELECT dob FROM customers WHERE fname
LIKE 'Ramesh';
```



	dob
▶	1976-12-06

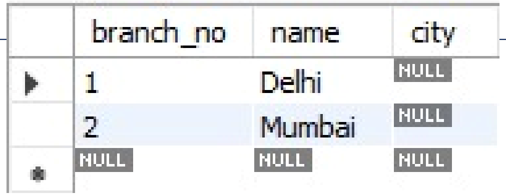
6. Add Column City To Branch Table

```
ALTER TABLE branch ADD COLUMN city CHAR(20);
```



```
37 18:12:01 ALTER TABLE branch ADD COLUMN city CHAR(20)
```

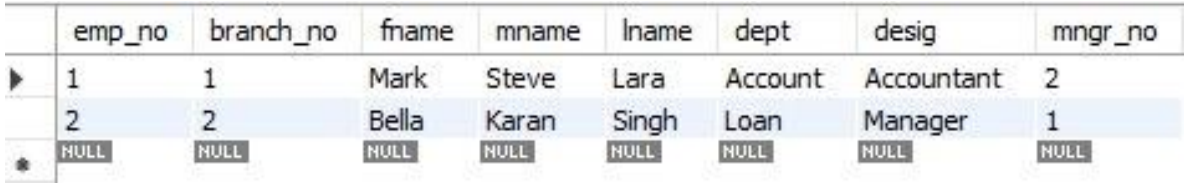
```
SELECT * FROM branch;
```



	branch_no	name	city
▶	1	Delhi	NULL
	2	Mumbai	NULL
*	NULL	NULL	NULL

7. Update the mname and lname of Employee ‘Bella’ and set to ‘Karan’, ‘Singh’

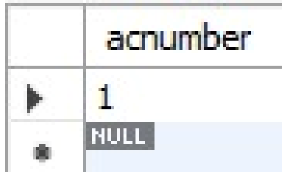
```
UPDATE employees
SET mname = 'Karan', lname = 'Singh'
WHERE fname LIKE 'Bella' ;
```



	emp_no	branch_no	fname	mname	lname	dept	desig	mngr_no
▶	1	1	Mark	Steve	Lara	Account	Accountant	2
	2	2	Bella	Karan	Singh	Loan	Manager	1
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

8. Select Accounts Opened Between '2012-07-01' AND '2013-01-01'

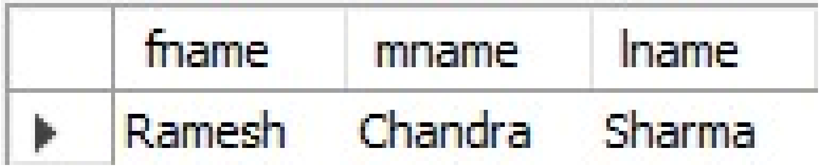
```
SELECT acnumber FROM accounts
WHERE opndt BETWEEN '2012-07-01' AND '2013-01-01';
```



	acnumber
▶	1
*	NULL

9. List The Names Of Customers Having ‘A’ As The Second Letter In Their Names

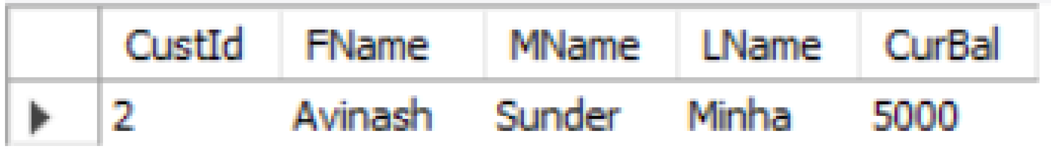
```
SELECT fname, mname, lname FROM customers
WHERE fname LIKE ‘_a ;
```



	fname	mname	lname
▶	Ramesh	Chandra	Sharma

10. Find The Lowest Balance From Customer And Account Table

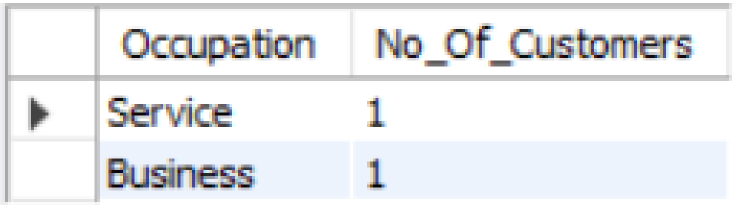
```
SELECT cust.custid, cust.fname, cust.mname, cust.lname, acc.curbal
FROM customers AS cust
INNER JOIN accounts acc
USING (custid)
ORDER BY curbal ASC LIMIT 1;
```



	CustId	FName	MName	LName	CurBal
▶	2	Avinash	Sunder	Minha	5000

11. Give The Count Of Customer For Each Occupation

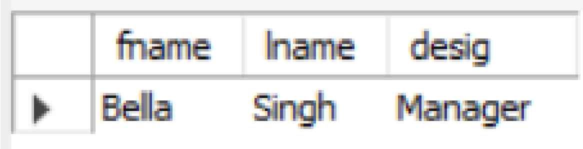
```
SELECT  Occupation, COUNT(*) AS No_Of_Customers
FROM Customers GROUP BY Occupation;
```



	Occupation	No_Of_Customers
▶	Service	1
	Business	1

12. Write A Query To Find The Name (First_name, Last_name) of the Employees Who Are Managers.

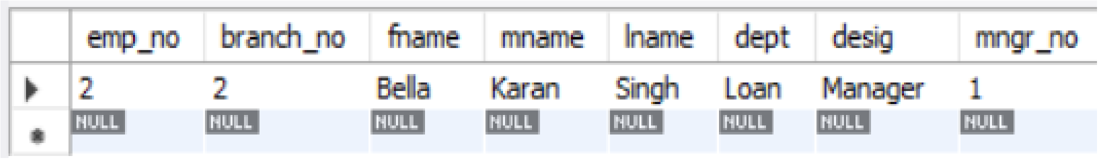
```
SELECT fname, lname, desig
FROM employees
WHERE desig LIKE 'manager' ;
```



	fname	lname	desig
▶	Bella	Singh	Manager

13. Select the details of the employee who work for department ‘loan’ or ‘credit’

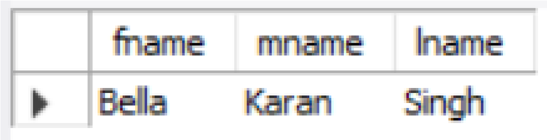
```
SELECT *
FROM employees
WHERE dept LIKE 'Loan' OR 'Credit' ;
```



	emp_no	branch_no	fname	mname	lname	dept	desig	mngr_no
▶	2	2	Bella	Karan	Singh	Loan	Manager	1
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

14. List Name Of All Employees Whose Name Ends With A

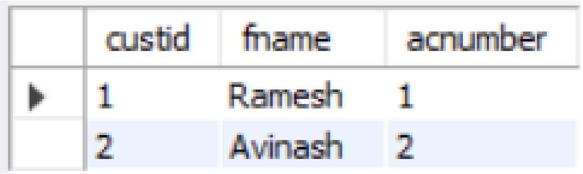
```
SELECT fname, mname, lname
FROM employees
WHERE fname LIKE '%a' ;
```



	fname	mname	lname
▶	Bella	Karan	Singh

15. Write a query to display the customers number, customers firstname, account number for the customers' who are born after 15th of any month.

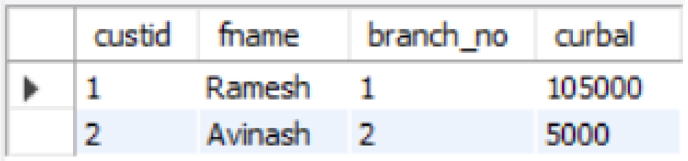
```
SELECT a.custid, a.fname, b.acnumber
FROM customers a
JOIN accounts b
ON a.custid = b.custid
WHERE DATE(dob) > 15;
```



	custid	fname	acnumber
▶	1	Ramesh	1
	2	Avinash	2

16.Write a query to display the customers' number, customers' firstname, branch id and balance amount for people using JOIN

```
SELECT a.custid, a.fname, b.branch_no, b.curbal
FROM customers a
JOIN accounts b
ON a.custid = b.custid;
```

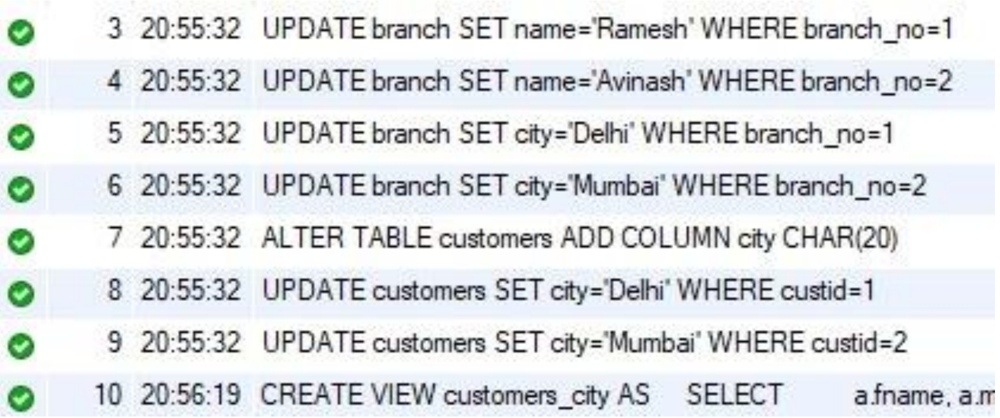


	custid	fname	branch_no	curbal
▶	1	Ramesh	1	105000
	2	Avinash	2	5000

17. Create a virtual table to store the customers who are having the accounts in the same city as they live

Step-1 : Updating the data to execute query

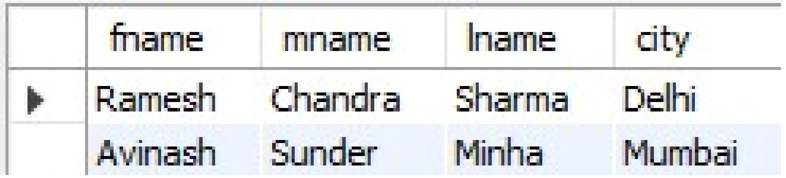
```
UPDATE branch SET name='Ramesh' WHERE branch_no=1;
UPDATE branch SET name='Avinash' WHERE branch_no=2;
UPDATE branch SET city='Delhi' WHERE branch_no=1;
UPDATE branch SET city='Mumbai' WHERE branch_no=2;
ALTER TABLE customers ADD COLUMN city CHAR(20);
UPDATE customers SET city='Delhi' WHERE custid=1;
UPDATE customers SET city='Mumbai' WHERE custid=2;
```



```
✓  3  20:55:32  UPDATE branch SET name='Ramesh' WHERE branch_no=1
✓  4  20:55:32  UPDATE branch SET name='Avinash' WHERE branch_no=2
✓  5  20:55:32  UPDATE branch SET city='Delhi' WHERE branch_no=1
✓  6  20:55:32  UPDATE branch SET city='Mumbai' WHERE branch_no=2
✓  7  20:55:32  ALTER TABLE customers ADD COLUMN city CHAR(20)
✓  8  20:55:32  UPDATE customers SET city='Delhi' WHERE custid=1
✓  9  20:55:32  UPDATE customers SET city='Mumbai' WHERE custid=2
✓ 10  20:56:19  CREATE VIEW customers_city AS    SELECT     a.fname, a.m
```

Step-2 : Creating a view using the updated data

```
CREATE VIEW customers_city AS
        SELECT a.fname, a.mname, a.lname, b.city
        FROM customers a JOIN  branch b
        ON a.fname = b.name
        WHERE a.city = b.city;

SELECT * FROM customers_city;
```

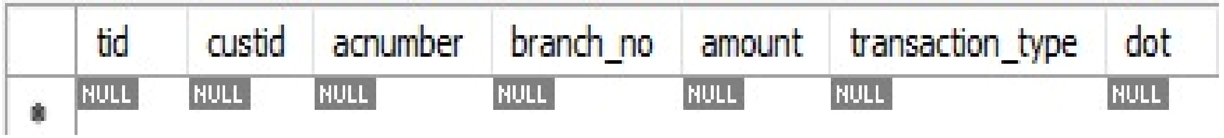


	fname	mname	lname	city
▶	Ramesh	Chandra	Sharma	Delhi
	Avinash	Sunder	Minha	Mumbai

- customers_city view

18. Create a Transaction Table.

```
CREATE TABLE transaction
(tid INT AUTO_INCREMENT PRIMARY KEY,
custid INT,
acnumber INT,
branch_no INT,
amount INT,
transaction_type ENUM ('withdraw','deposit’),
dot DATE,
FOREIGN KEY (custid) REFERENCES
customers(custid),
FOREIGN KEY (acnumber) REFERENCES
accounts(acnumber),
FOREIGN KEY (branch_no) REFERENCES
branch(branch_no));
```



	tid	custid	acnumber	branch_no	amount	transaction_type	dot
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

a. Write trigger to update balance in account table on Deposit or Withdraw in transaction table
b. Insert values in transaction table to show trigger success

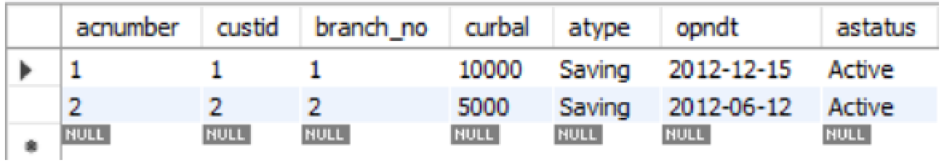
Part a:

```
DELIMITER //

CREATE TRIGGER transaction_update
AFTER INSERT ON transaction FOR EACH ROWBEGIN
UPDATE accounts SET curbal = curbal + NEW.amount *
CASE
    WHEN NEW.transaction_type ='withdraw’
        THEN '-1’
    ELSE '1’
    END
WHERE custid=NEW.custid;
END
//DELIMITER ;
```

Part b:

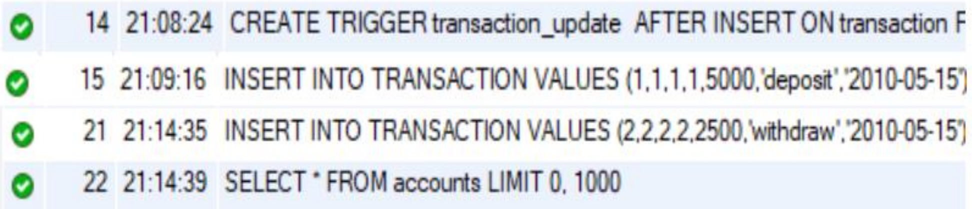
```
INSERT INTO TRANSACTION VALUES
(1,1,1,1,5000,'deposit','2010-05-15');
```

Account Table Before Update



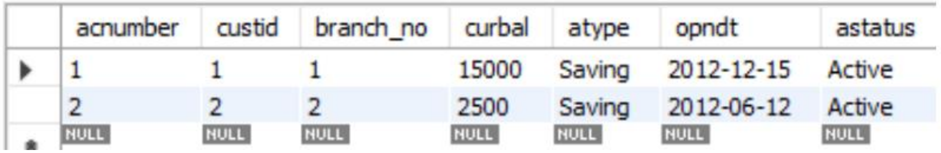
	acnumber	custid	branch_no	curbal	atype	opndt	astatus
▶	1	1	1	10000	Saving	2012-12-15	Active
	2	2	2	5000	Saving	2012-06-12	Active
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Creating of Trigger and Inserting Values



```
✓ 14 21:08:24 CREATE TRIGGER transaction_update  AFTER INSERT ON transaction F
✓ 15 21:09:16 INSERT INTO TRANSACTION VALUES (1,1,1,1,5000,'deposit','2010-05-15')
✓ 21 21:14:35 INSERT INTO TRANSACTION VALUES (2,2,2,2,2500,'withdraw','2010-05-15')
✓ 22 21:14:39 SELECT * FROM accounts LIMIT 0, 1000
```

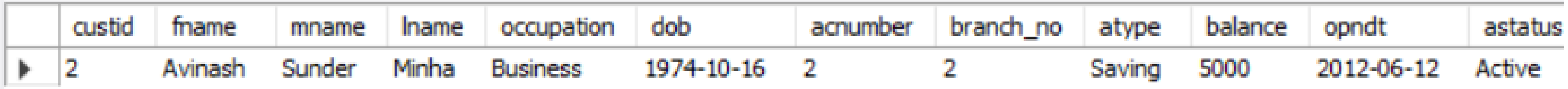
Account Table Update After Trigger



	acnumber	custid	branch_no	curbal	atype	opndt	astatus
▶	1	1	1	15000	Saving	2012-12-15	Active
	2	2	2	2500	Saving	2012-06-12	Active
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

19. Write a query to display the details of customers with second highest balance

```
SELECT a.custid, a.fname, a.mname, a.lname, a.occupation, a.dob, b.acnumber,
         b.branch_no, b.atype, b.curbal AS balance, b.opndt, b.astatus
FROM customers a
JOIN accounts b
ON a.custid = b.custid
ORDER BY curbal DESC
LIMIT 1 OFFSET 1;
```

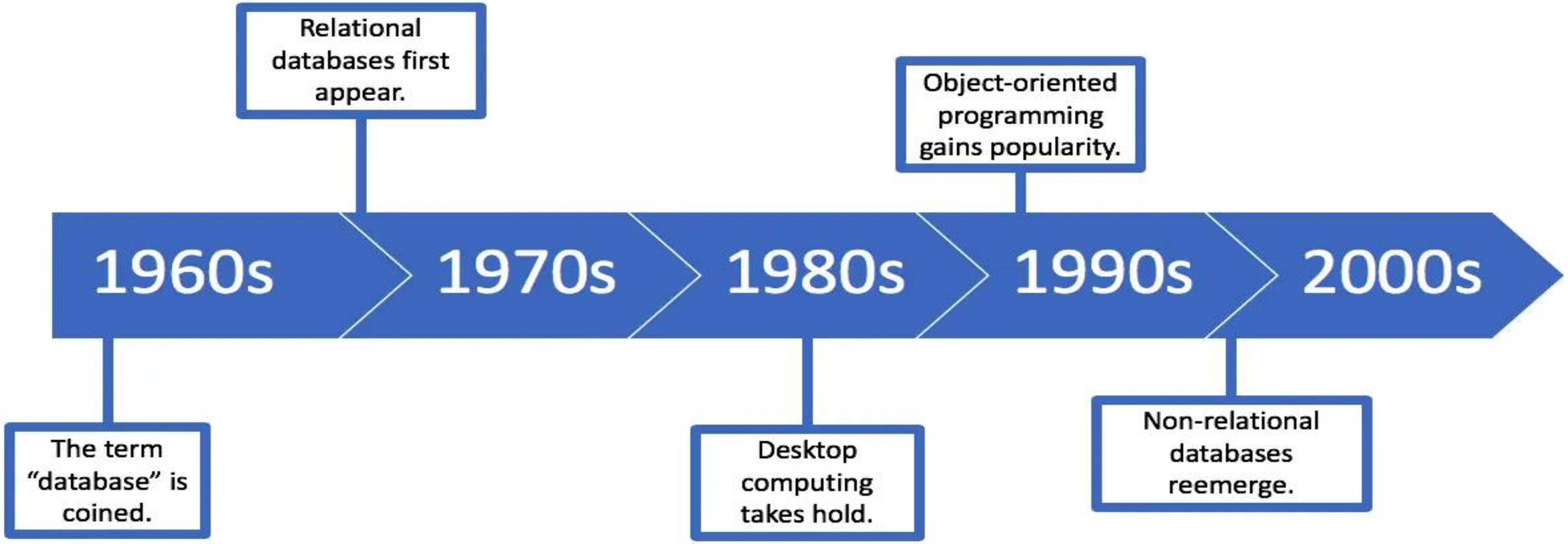


	custid	fname	mname	lname	occupation	dob	acnumber	branch_no	atype	balance	opndt	astatus
▶	2	Avinash	Sunder	Minha	Business	1974-10-16	2	2	Saving	5000	2012-06-12	Active

20. Take backup of the database created in this case study

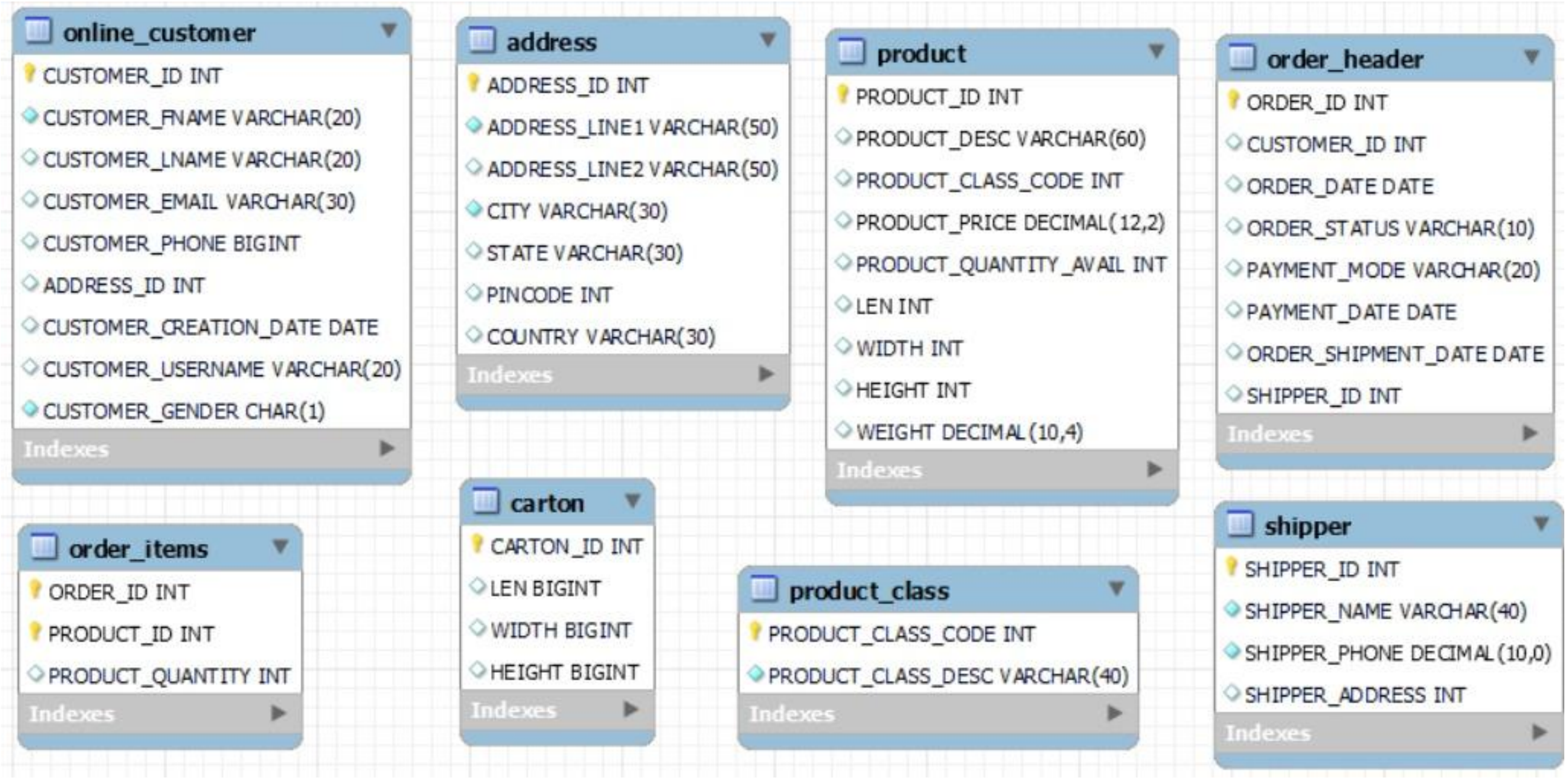
To create a backup of the Database steps are:

SERVER -> EXPORT -> SELECT SCHEMA -> BROWSE LOCATION -> START EXPORT


Relational databases first appear.
Object-oriented programming gains popularity.
1960s
1970s
1980s
1990s
2000s
The term “database” is coined.
Desktop computing takes hold.
Non-relational databases reemerge.

PART - B

ER DIAGRAM



online_customer

- CUSTOMER_ID INT
- CUSTOMER_FNAME VARCHAR(20)
- CUSTOMER_LNAME VARCHAR(20)
- CUSTOMER_EMAIL VARCHAR(30)
- CUSTOMER_PHONE BIGINT
- ADDRESS_ID INT
- CUSTOMER_CREATION_DATE DATE
- CUSTOMER_USERNAME VARCHAR(20)
- CUSTOMER_GENDER CHAR(1)

address

- ADDRESS_ID INT
- ADDRESS_LINE1 VARCHAR(50)
- ADDRESS_LINE2 VARCHAR(50)
- CITY VARCHAR(30)
- STATE VARCHAR(30)
- PINCODE INT
- COUNTRY VARCHAR(30)

product

- PRODUCT_ID INT
- PRODUCT_DESC VARCHAR(60)
- PRODUCT_CLASS_CODE INT
- PRODUCT_PRICE DECIMAL(12,2)
- PRODUCT_QUANTITY_AVAIL INT
- LEN INT
- WIDTH INT
- HEIGHT INT
- WEIGHT DECIMAL(10,4)

order_header

- ORDER_ID INT
- CUSTOMER_ID INT
- ORDER_DATE DATE
- ORDER_STATUS VARCHAR(10)
- PAYMENT_MODE VARCHAR(20)
- PAYMENT_DATE DATE
- ORDER_SHIPMENT_DATE DATE
- SHIPPER_ID INT

order_items

- ORDER_ID INT
- PRODUCT_ID INT
- PRODUCT_QUANTITY INT

carton

- CARTON_ID INT
- LEN BIGINT
- WIDTH BIGINT
- HEIGHT BIGINT

product_class

- PRODUCT_CLASS_CODE INT
- PRODUCT_CLASS_DESC VARCHAR(40)

shipper

- SHIPPER_ID INT
- SHIPPER_NAME VARCHAR(40)
- SHIPPER_PHONE DECIMAL(10,0)
- SHIPPER_ADDRESS INT

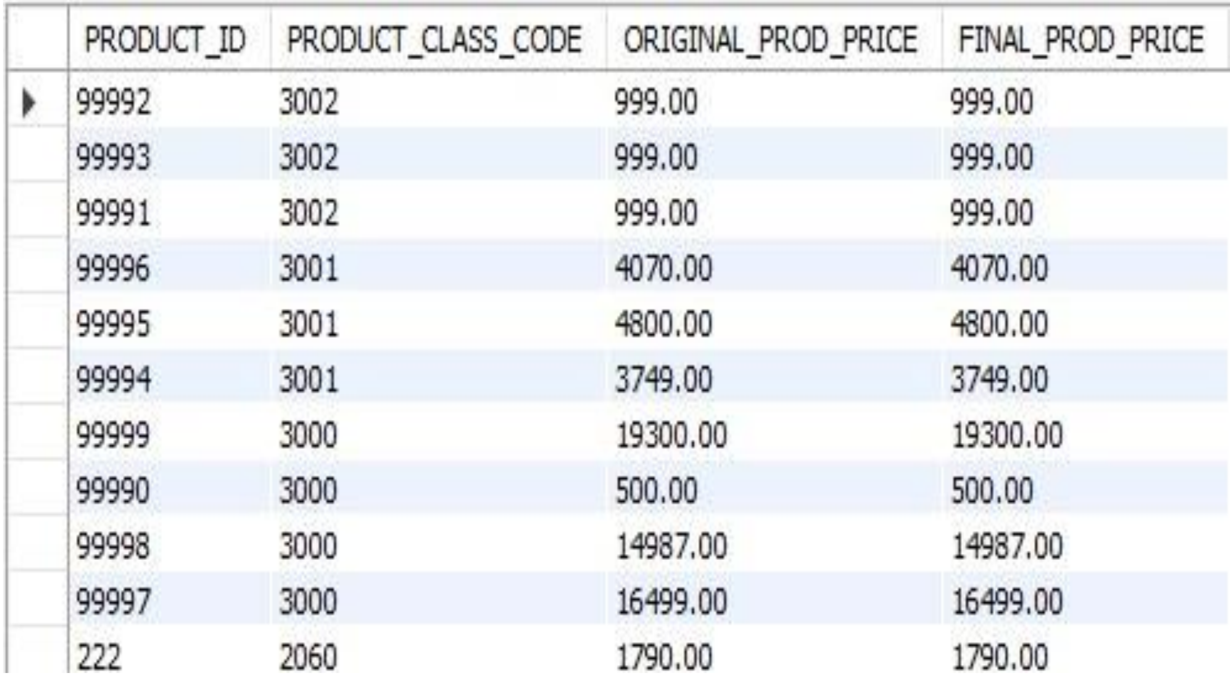
1. Display the Product Details as per the following Criteria and Sort them in Descending Order of Category:

a. If The Category Is 2050, Increase The Price By 2000

b. If The Category Is 2051, Increase The Price By 500

c. If The Category Is 2052, Increase The Price By 600

```
SELECT product_id, product_class_code,
product_price AS original_prod_price,
CASE
    WHEN product_class_code = 2050 THEN product_price+2000
    WHEN product_class_code = 2051 THEN product_price+500
    WHEN product_class_code = 2052 THEN product_price+600
    WHEN product_class_code > 2052 THEN product_price
END AS final_prod_price
FROM product ORDER BY product_class_code DESC;
```

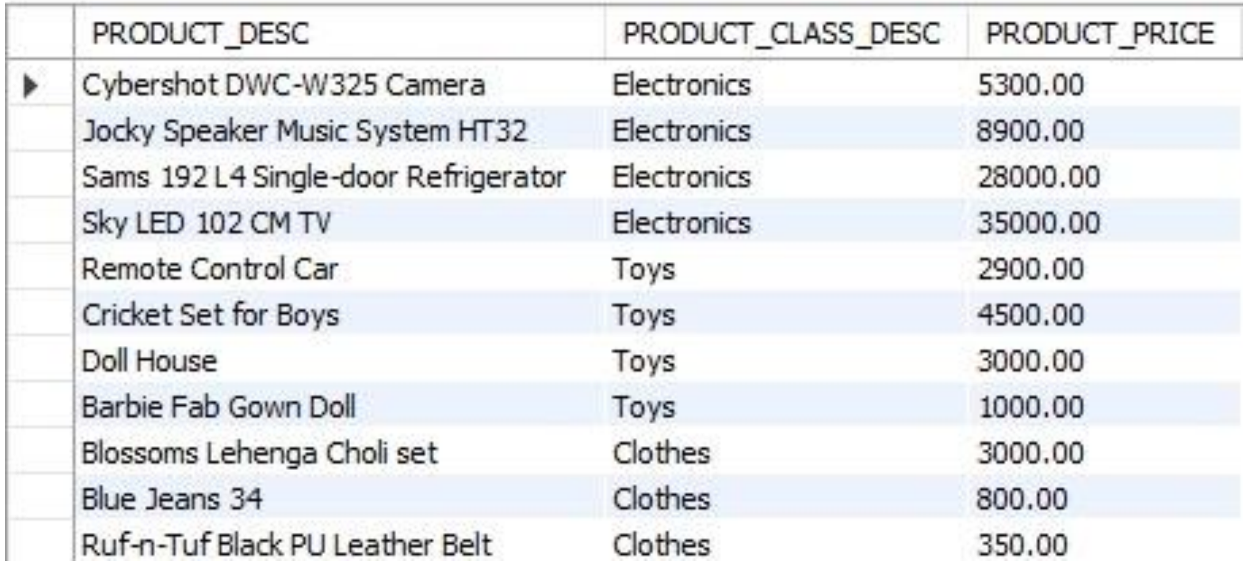


PRODUCT_ID	PRODUCT_CLASS_CODE	ORIGINAL_PROD_PRICE	FINAL_PROD_PRICE
99992	3002	999.00	999.00
99993	3002	999.00	999.00
99991	3002	999.00	999.00
99996	3001	4070.00	4070.00
99995	3001	4800.00	4800.00
99994	3001	3749.00	3749.00
99999	3000	19300.00	19300.00
99990	3000	500.00	500.00
99998	3000	14987.00	14987.00
99997	3000	16499.00	16499.00
222	2060	1790.00	1790.00

*60 rows returned

2. List the Product Description, Class Description And Price of All Products Which Are Shipped.

```
SELECT a.product_desc, b.product_class_desc, a.product_price

FROM  product AS a

JOIN product_class AS b

ON a.product_class_code = b.product_class_code

WHERE product_id IN

        (SELECT  product_id FROM order_items WHERE order_id IN

                (SELECT order_id FROM order_header WHERE order_status LIKE 'Shipped' ) ) ;
```



PRODUCT_DESC	PRODUCT_CLASS_DESC	PRODUCT_PRICE
Cybershot DWC-W325 Camera	Electronics	5300.00
Jocky Speaker Music System HT32	Electronics	8900.00
Sams 192 L4 Single-door Refrigerator	Electronics	28000.00
Sky LED 102 CM TV	Electronics	35000.00
Remote Control Car	Toys	2900.00
Cricket Set for Boys	Toys	4500.00
Doll House	Toys	3000.00
Barbie Fab Gown Doll	Toys	1000.00
Blossoms Lehenga Choli set	Clothes	3000.00
Blue Jeans 34	Clothes	800.00
Ruf-n-Tuf Black PU Leather Belt	Clothes	350.00

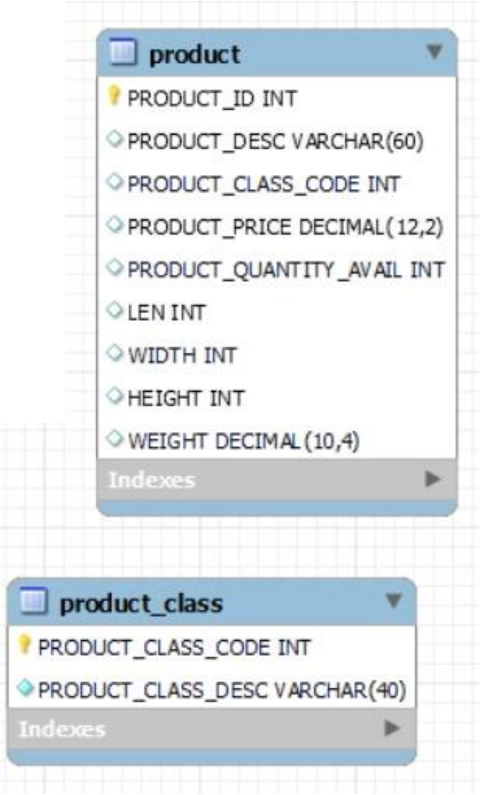
*47 rows returned

3. Show Inventory Status Of Products As Below As Per Their Available Quantity:

A. For Electronics And Computer Categories,
If Available Quantity is: < 10, 'Low Stock',
11 < qty < 30, 'In Stock',
> 31, 'Enough Stock'

B. For Stationery And Clothes Categories,
If qty < 20, 'Low Stock',
21 < qty < 80, 'In Stock',
> 81, 'Enough Stock'

C. Rest Of The Categories,
If qty < 15, 'Low Stock',
16 < qty < 50, 'In Stock',
> 51, 'Enough Stock'

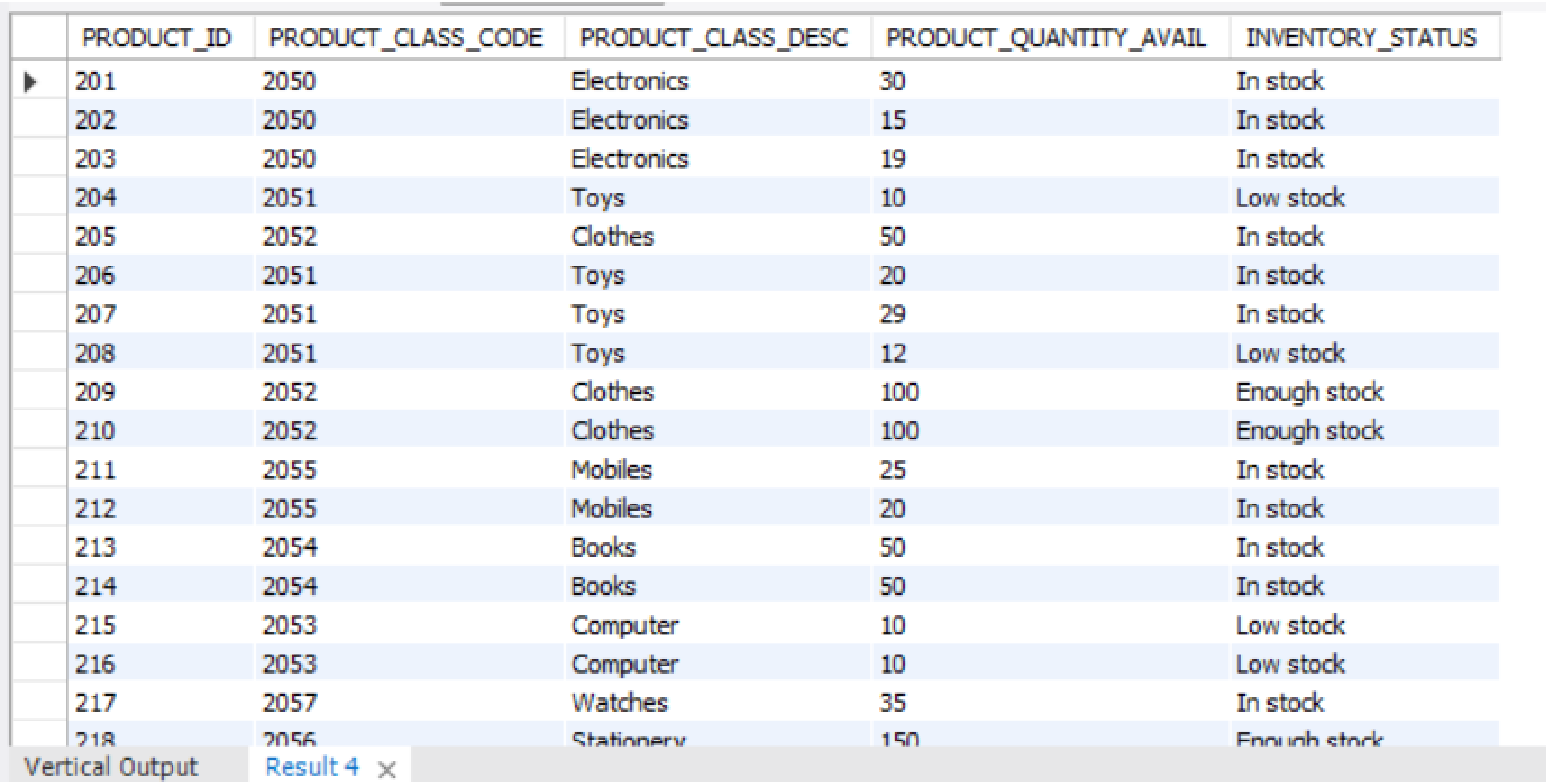


product
PRODUCT_ID INT
PRODUCT_DESC VARCHAR(60)
PRODUCT_CLASS_CODE INT
PRODUCT_PRICE DECIMAL(12,2)
PRODUCT_QUANTITY_AVAIL INT
LEN INT
WIDTH INT
HEIGHT INT
WEIGHT DECIMAL(10,4)
Indexes

product_class
PRODUCT_CLASS_CODE INT
PRODUCT_CLASS_DESC VARCHAR(40)
Indexes

For All Categories, If Available Quantity Is 0, Show 'Out Of Stock'.

```
SELECT a.product_id, a.product_class_code, b.product_class_desc, a.product_quantity_avail,
CASE
    WHEN product_quantity_avail = 0 THEN 'Out of stock’
    WHEN product_class_desc like 'electronics' or 'computer’ THEN
         CASE
         WHEN a.product_quantity_avail <= 10   THEN 'low stock’
         WHEN a.product_quantity_avail > 10 AND a.product_quantity_avail <= 30   THEN 'in stock'
         WHEN a.product_quantity_avail > 31   THEN 'enough stock'
         END
     WHEN product_class_desc like 'stationery' or 'clothes’ THEN
        CASE
        WHEN a.product_quantity_avail <= 20   THEN 'low stock’
        WHEN a.product_quantity_avail > 20 AND a.product_quantity_avail <= 80   THEN 'in stock'
        WHEN a.product_quantity_avail > 80   THEN 'enough stock’
        END
    ELSE CASE
        WHEN a.product_quantity_avail <= 15 THEN 'low stock’
        WHEN a.product_quantity_avail > 15 AND a.product_quantity_avail <= 50 THEN 'in stock’
        WHEN a.product_quantity_avail > 50 THEN 'enough stock’
        END
END AS inventory_status
FROM     product AS a   JOIN    product_class AS b
ON a.product_class_code = b.product_class_code
ORDER BY product_id;
```

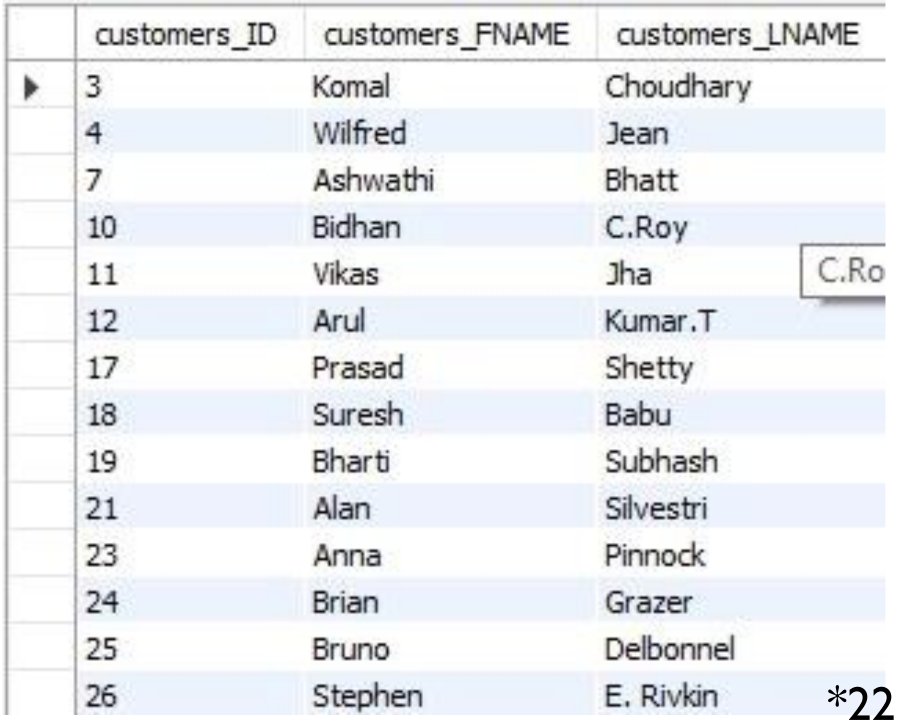


	PRODUCT_ID	PRODUCT_CLASS_CODE	PRODUCT_CLASS_DESC	PRODUCT_QUANTITY_AVAIL	INVENTORY_STATUS
▶	201	2050	Electronics	30	In stock
	202	2050	Electronics	15	In stock
	203	2050	Electronics	19	In stock
	204	2051	Toys	10	Low stock
	205	2052	Clothes	50	In stock
	206	2051	Toys	20	In stock
	207	2051	Toys	29	In stock
	208	2051	Toys	12	Low stock
	209	2052	Clothes	100	Enough stock
	210	2052	Clothes	100	Enough stock
	211	2055	Mobiles	25	In stock
	212	2055	Mobiles	20	In stock
	213	2054	Books	50	In stock
	214	2054	Books	50	In stock
	215	2053	Computer	10	Low stock
	216	2053	Computer	10	Low stock
	217	2057	Watches	35	In stock
	218	2056	Stationery	150	Enough stock

*60 rows returned

4. List Customers From Outside Karnataka Who Haven’t Bought Any Toys Or Books

```
SELECT customers_id, customers_fname, customers_lname  FROM online_customers
WHERE address_id IN (SELECT address_id FROM address WHERE STATE NOT LIKE 'Karnataka’)
    AND customers_ID IN (SELECT customers_ID FROM  order_header WHERE order_id IN
        (SELECT order_id FROM order_items WHERE product_id IN
            (SELECT product_id FROM product  WHERE product_class_code IN
                (SELECT product_class_code FROM product_class WHERE product_class_desc NOT LIKE 'Toys' OR
                    'Books’ ) ) ) ) ;
```



customers_ID	customers_FNAME	customers_LNAME
3	Komal	Choudhary
4	Wilfred	Jean
7	Ashwathi	Bhatt
10	Bidhan	C.Roy
11	Vikas	Jha
12	Arul	Kumar.T
17	Prasad	Shetty
18	Suresh	Babu
19	Bharti	Subhash
21	Alan	Silvestri
23	Anna	Pinnock
24	Brian	Grazer
25	Bruno	Delbonnel
26	Stephen	E. Rivkin

*22 rows returned

FUTURE STEPS

1. In the future, we can enhance the project by using transactional commands (TCL) to set a limit for each user and grant access to only authorized individuals. This will help us to ensure that only the appropriate individuals have access to the database, which will enhance the security and privacy of the system.
2. Furthermore, we can prevent loss of data and maintain the atomicity of the data by using rollback and commit commands. This will ensure that if any error occurs during a transaction, the data changes can be rolled back to their previous state, preventing data loss.
3. Additionally, we can isolate tables before inserting or updating using various isolation levels such as read uncommitted, read committed, repeatable read, and serializable. This will prevent conflicts and ensure that the data remains consistent and accurate throughout the transaction.
4. Overall, these future steps will help to enhance the security, privacy, and reliability of the database, making it a more robust and effective system.

CONCLUSION

1. The objective of this report is to document the process of creating tables for data in the Bank Retail Case Study and inserting records into those tables while maintaining data integrity. This includes ensuring that all data entered into the tables conforms to the defined data types and constraints, and that all relationships between the tables are accurately established.
2. In this project, we have created separate tables for each entity - customer, employee, and account - and integrated them through referential integrity. We ensured data integrity in our database is accurately established. We have inserted few records into these tables and performed data retrieval using queries. This helped us fulfill some of the probable client-side requests from the database. Overall, the project was successful in achieving the objectives of creating and maintaining data integrity in the Bank Retail Case Study.

3. One limitation of this project is that the database does not account for privacy concerns. The database contains sensitive information such as customer and employee details, which may not be suitable for unrestricted access. To overcome this limitation, we can create multiple views for each table and use those views to fetch records, which will reduce the access of the user towards the data. By using views, we can ensure that only the authorized individuals have access to the data that they require. This will enhance the privacy and security of the database and protect sensitive information from unauthorized access.

THANK YOU