ITE1003

DATA BASE MANAGEMENT SYSTEM

F1-SLOT

STORE CHAIN MANAGEMENT SYSTEM

FINAL REVIEW

Faculty: Prof. BIMAL KUMAR RAY

November, 2019

18BIT0041: MAHAK GUPTA

18BIT0182: LAKHAN NAD

18BIT0190: KHUSHI GELDA

18BIT0219: ISHAN SAWHNEY

ABSTRACT:

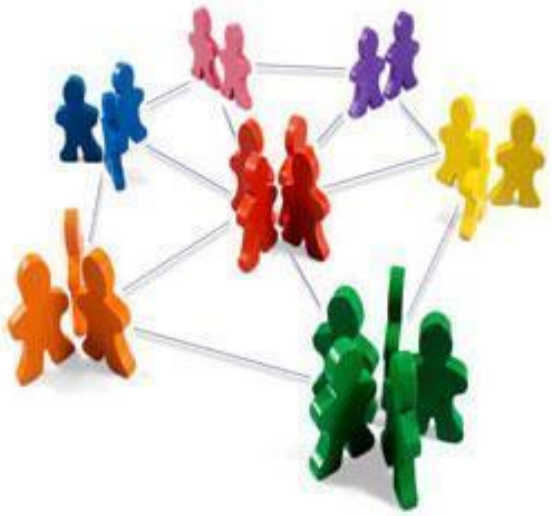
Our project revolves around handling chain of stores.

In the present competitive business world, expansion of business for successful business is a need. A store-owner doing a good business on a particular shop always wishes to expand and add more branches, creating a chain of branches for his store.

But the major concern the owner is always how to keep track of day to day activity of the each branch.Monitor which branch is making profit and loss, so to make sure his business expansion turns to be profitable for him.

We are going to make a data-base management system for handling the day-to-day record of purchase made by a particular branch, stock available in the store, products sold, buyer detail, products detail, generate bill for buyer, store and update of the records of employee working in different branches.

All this functionality will help the owner to keep a record with any misleading and flaws. And manage the chain of stores and grow his business.



DATA REQUIREMENTS:

- We need to store information of various **branches**. This information includes name, location, branch manager name, manager id and a unique id is assigned to each branch. Branch Id should be unique and should be present for each branch.

- **Customer** information is stored for each customer who visits a particular branch. This data is stored for retrieval of customer details when required either for customer queries regarding purchase or branch's need to monitor a customer record. Customer information should include contact number, email, and name. Customer contact number can be used to uniquely identify customer information and it should be given by each customer who makes a purchase.

- Details of **product** which are sold in various branches must be stored. These details can be referred when a product is purchased or sold by a branch. They should include price, name, id and description of product. Id of the product can be used to uniquely identify each product and is generally provided at the time of acquiring the product or at the time of manufacture of the product.

- **Employees** who work in a particular branch must have a unique id. This Id can be used to uniquely identify each employee. Other details like branch they work in, name, salary, contact and their joining date should be stored.

- **Seller** information like their id, contact, email, seller name and their bank details should be stored. Seller id can be used to uniquely identify each seller and the bank details should be given by each seller so that the transaction can take place smoothly.

- When some product is on **sale** in a particular branch then details like discount percent, start date, end date of the sale must be stored to refer while billing. These details may or may not be unique for each product in each branch.

- Details of **stocks** (quantity) available of a particular product in a particular branch must be stored. This information is updated whenever a product is sold or purchased. These details must include quantity of product in consideration and in which branch these stocks are available.

- **Bills** generated on every purchase by a buyer must be stored so the selling history can be traced and referred when necessary. Bills have a unique bill number across all branch, it also has a date and a total amount that can be referred to track the overall profit.

- Which product and in what quantity is sold to which buyer is a necessary data that is needed for reference when a customer comes up with query related to bills or payments. This data should include quantity of product sold, the product id as well as the bill number issued for the purchase.

- The details of the purchase made from the seller must include the date of purchase of product, quantity of product acquired, product details and seller's information. Each transaction can be uniquely identified by the seller Id ,product Id and the date of transaction.

Functional Requirements:

Scenarios of adding new data

- Add new product to supply chain.
- Hire new employee & store his details
- Add new seller and acquire his details.
- Add customer details when a new customer visits a particular branch.
- Add new sale offer.
- Generate bill.
- Acquire new products or more quantity of old products from seller.
- Add new branch details.

Scenarios of updating old data

- Update prices of product.
- Update dates of the sale.
- Change manager of a branch.
- Update contact details of the seller.
- Increase or decrease the discount percentage of an ongoing sale in a particular branch.
- Update stocks when a product is acquired or when a product is sold.
- Update salary of employees.
- Update bank details of the seller.

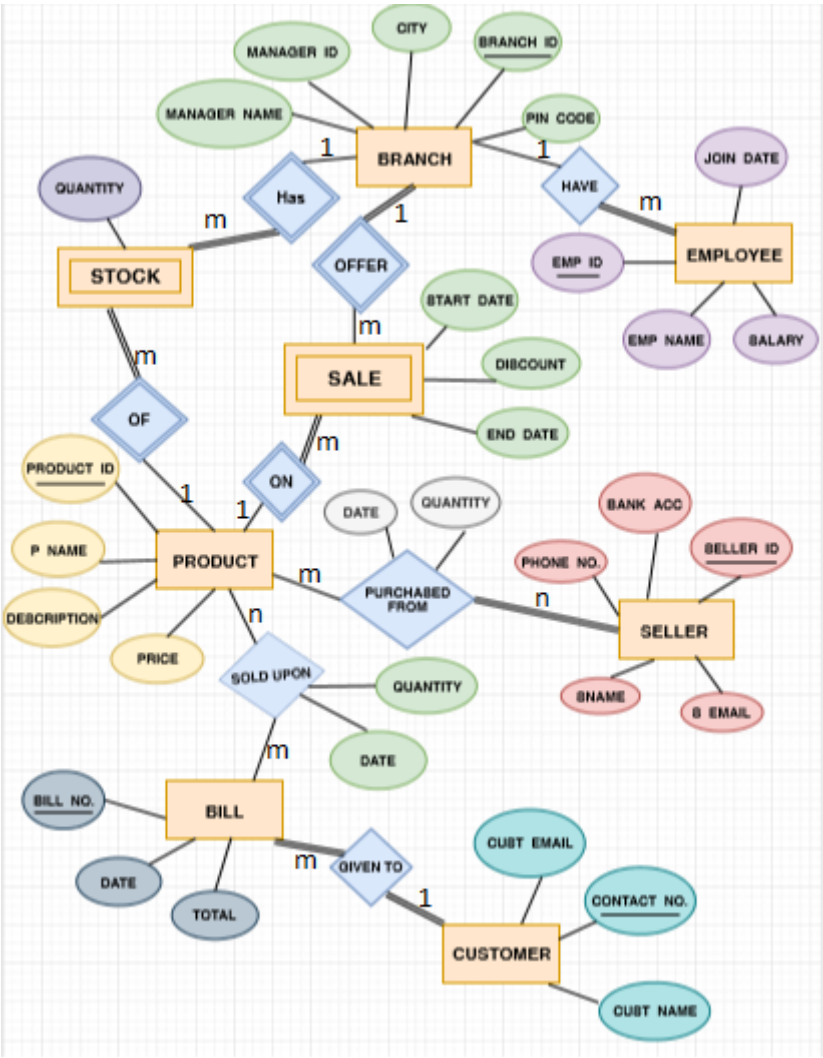
Scenarios of deleting old data

- Delete stock details when quantity is zero.
- Delete sale info after sale's end date.
- Delete customer info if the last purchase was made a year ago.
- Delete bill details and associated data if date in bill is more than a year ago.

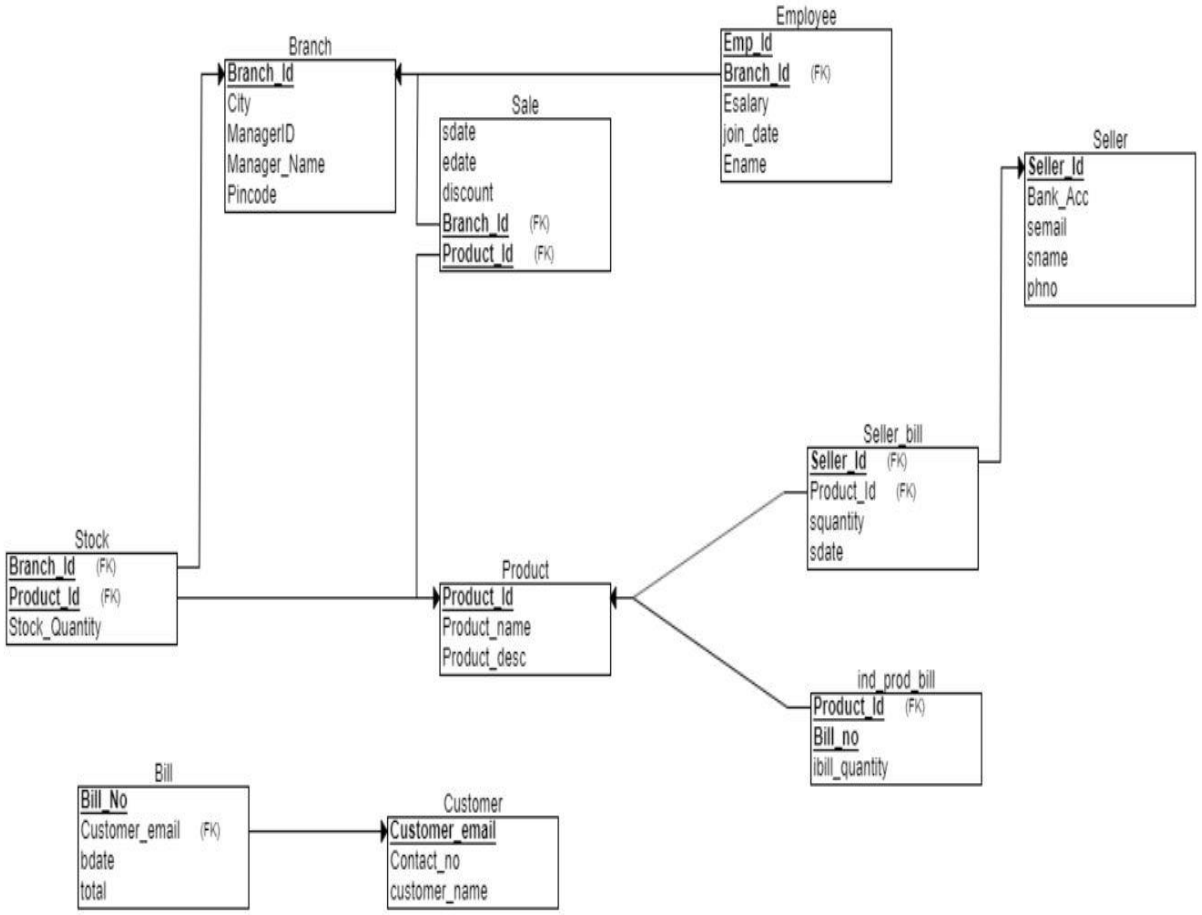
Scenarios of retrieving some data

- Get the last purchase of a customer.
- Get stock details for particular product available in a particular branch.
- Get total profit made in a particular day.
- Details of a customer that visited on a particular date at a particular branch.
- Number of employees working in a particular branch.
- Ongoing sale in a particular branch.
- Most purchased product by the customers on a particular day.
- Details of product acquired on a particular date.
- Get product associated with a particular bill.
- Get average profit among all branches on a particulate day.

ER – DIAGRAM:


CITY
MANAGER ID
BRANCH ID
MANAGER NAME
PIN CODE
BRANCH
1
1
JOIN DATE
QUANTITY
Has
HAVE
m
m
1
EMP ID
EMPLOYEE
STOCK
OFFER
START DATE
m
EMP NAME
SALARY
m
SALE
DISCOUNT
OF
END DATE
m
PRODUCT ID
ON
1
1
DATE
QUANTITY
BANK ACC
P NAME
PRODUCT
m
PHONE NO.
SELLER ID
PURCHASED FROM
n
DESCRIPTION
n
SELLER
PRICE
SOLD UPON
QUANTITY
SNAME
S EMAIL
m
DATE
BILL NO.
BILL
CUST EMAIL
m
GIVEN TO
DATE
1
CONTACT NO.
TOTAL
CUSTOMER
CUST NAME

RELATION SCHEMA DIAGRAM:



CREATE TABLE COMMANDS

BRANCH:

Command Prompt - sqlplus

```
SQL> create table branch (
  2      branch_id       number(5) ,
  3      city            varchar(10) ,
  4      manager_id      number(5),
  5      manager_name    varchar(20),
  6      pin_code        number(6),
  7      primary key(branch_id),
  8      check (pin_code > 100000),
  9      check(manager_id is NOT NULL)
 10  );

Table created.

SQL>
```

EMPLOYEE:

Command Prompt - sqlplus

```
SQL> create table employee(
  2      emp_id  number(5) ,
  3      ename  VARCHAR(20) ,
  4      esalary number(9,2) ,
  5      join_date DATE,
  6      branch_id references branch,
  7      primary key(emp_id,branch_id),
  8      check (length(ename) > 0),
  9      check (esalary between 10000.00 and 99999999.00)
 10  );

Table created.

SQL>
```

PRODUCT:

Command Prompt - sqlplus

```
SQL> create table product (
  2      product_id number(5) ,
  3      product_name varchar(20),
  4      product_desc varchar(40),
  5      price number(5,2),
  6      primary key(product_id),
  7      check (price > 0),
  8      check (length(product_name) > 0)
  9  );

Table created.

SQL>
```

SELLER:

Command Prompt - sqlplus

```
SQL> create table seller(
  2      seller_id number(4) ,
  3      bank_acc number(15),
  4      semail varchar(20),
  5      sname varchar(10),
  6      phno number(10),
  7      primary key(seller_id),
  8      check (phno >= 1000000000),
  9      check (bank_acc >= 1000000000)
 10  );

Table created.

SQL>
```

CUSTOMER:

Command Prompt - sqlplus

```
SQL> create table customer(
  2      customer_email varchar(20),
  3      contact_no number(10),
  4      customer_name varchar(10),
  5      primary key(customer_email),
  6      check (regexp_like(customer_email,'^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$','i')),
  7      check(regexp_like(to_char(contact_no),'^9[0-9]{9}$|^8[0-9]{9}$|^7[0-9]{9}$|^6[0-9]{9}$'))
  8  );

Table created.

SQL>
```

STOCK:

Command Prompt - sqlplus

```
SQL> CREATE TABLE stock(
  2      product_id  references product,
  3      branch_id   references branch,
  4      stock_quantity    number(3),
  5      primary key(product_id,branch_id)
  6  );

Table created.

SQL>
```

SALE:

Command Prompt - sqlplus

```
SQL> create table sale(
  2      product_id  references product,
  3      branch_id references branch,
  4      sdate  date,
  5      edate date,
  6      discount number(4,2),
  7      primary key(product_id,branch_id),
  8      check (sdate < edate),
  9      check (discount between 01.00 and 99.99)
 10  );

Table created.

SQL>
```

SELLER-BILL:

Command Prompt - sqlplus

```
SQL> create table seller_bill(
  2      seller_id references seller,
  3      product_id references product,
  4      squantity number(3),
  5      sdate date,
  6      primary key(seller_id,product_id,sdate),
  7      check (squantity between 9 and 9999)
  8  );

Table created.

SQL>
```

BILL:

Command Prompt - sqlplus

```
SQL> create table bill(
  2      bill_no number(4),
  3      customer_email references customer,
  4      bdate DATE,
  5      total number(9,2),
  6      primary key(bill_no),
  7      check (total > 1.00)
  8  );

Table created.

SQL>
```

PRODUCT WISE BILL:

Command Prompt - sqlplus

```
SQL> create table prod_bill(
  2      product_id references product,
  3      bill_no references bill,
  4      ibill_quantity number(2),
  5      primary key(product_id,bill_no),
  6      check (ibill_quantity between 1 and 100)
  7  );

Table created.

SQL>
```

INSERTION QUERIES

Stock Table:

Run SQL Command Line

```
SQL> insert into stock values (&product_id, &branch_id, &stock_quantity);
Enter value for product_id: 90021
Enter value for branch_id: 11121
Enter value for stock_quantity: 50
old   1: insert into stock values (&product_id, &branch_id, &stock_quantity)
new   1: insert into stock values (90021, 11121, 50)

1 row created.

SQL>
```

Seller_Bill:

Run SQL Command Line

```
SQL> insert into seller_bill values (&seller_id,&product_id,&squantity,'&sdate');
Enter value for seller_id: 70020
Enter value for product_id: 90021
Enter value for squantity: 10
Enter value for sdate: 12-APRIL-2019
old   1: insert into seller_bill values (&seller_id,&product_id,&squantity,'&sdate')
new   1: insert into seller_bill values (70020,90021,10,'12-APRIL-2019')

1 row created.

SQL>
```

Seller:

Run SQL Command Line

```
SQL> insert into seller values (&seller_id,&bank_acc,'&semail','&sname',&phno);
Enter value for seller_id: 70020
Enter value for bank_acc: 123456789
Enter value for semail: HSTUW.YRY@XFAT.CSS
Enter value for sname: TONY
Enter value for phno: 7893828923
old   1: insert into seller values (&seller_id,&bank_acc,'&semail','&sname',&phno)
new   1: insert into seller values (70020,123456789,'HSTUW.YRY@XFAT.CSS','TONY',7893828923)

1 row created.

SQL>
```

Sale:

Run SQL Command Line

```
SQL> insert into sale values (&product_id,&branch_id,'&sdate','&edate',&discount);
Enter value for product_id: 90021
Enter value for branch_id: 11121
Enter value for sdate: 23-JUL-2020
Enter value for edate: 3-AUG-2020
Enter value for discount: 15
old   1: insert into sale values (&product_id,&branch_id,'&sdate','&edate',&discount)
new   1: insert into sale values (90021,11121,'23-JUL-2020','3-AUG-2020',15)

1 row created.

SQL>
```

Product:

Run SQL Command Line

```
SQL> insert into product values (&product_id, '&product_name', '&product_desc', &price);
Enter value for product_id: 90021
Enter value for product_name: COCACOLA
Enter value for product_desc: COLD DRINKS
Enter value for price: 30
old   1: insert into product values (&product_id, '&product_name', '&product_desc', &price)
new   1: insert into product values (90021, 'COCACOLA', 'COLD DRINKS', 30)

1 row created.

SQL>
```

Product_Bill:

Run SQL Command Line

```
SQL> insert into prod_bill values (&product_id, &bill_no, &ibill_quantity);
Enter value for product_id: 90021
Enter value for bill_no: 51
Enter value for ibill_quantity: 1
old   1: insert into prod_bill values (&product_id, &bill_no, &ibill_quantity)
new   1: insert into prod_bill values (90021, 51, 1)

1 row created.

SQL>
```

Employee:

Command Prompt - sqlplus

```
SQL> insert into employee values (&emp_id, '&ename', &esalary, '&join_date', &branch_id);
Enter value for emp_id: 40041
Enter value for ename: KHUSHI
Enter value for esalary: 200000
Enter value for join_date: 23-DEC-2019
Enter value for branch_id: 11121
old   1: insert into employee values (&emp_id, '&ename', &esalary, '&join_date', &branch_id)
new   1: insert into employee values (40041, 'KHUSHI', 200000, '23-DEC-2019', 11121)

1 row created.

SQL>
```

Customer:

Command Prompt - sqlplus

```
SQL> insert into customer values ('&customer_email', &contac_no, '&customer_name');
Enter value for customer_email: HOIFA.LJA@YUSM.FGA
Enter value for contac_no: 8457292939
Enter value for customer_name: HUYYTE
old   1: insert into customer values ('&customer_email', &contac_no, '&customer_name')
new   1: insert into customer values ('HOIFA.LJA@YUSM.FGA', 8457292939, 'HUYYTE')

1 row created.

SQL>
```

Branch:

Command Prompt - sqlplus

```
SQL> insert into branch values (&branch_id,'&city',&managerid,'&manager_name', &pin_code);
Enter value for branch_id: 11121
Enter value for city: GTAVICE
Enter value for managerid: 88898
Enter value for manager_name: KHUSHI
Enter value for pin_code: 123466
old   1: insert into branch values (&branch_id,'&city',&managerid,'&manager_name', &pin_code)
new   1: insert into branch values (11121,'GTAVICE',88898,'KHUSHI', 123466)

1 row created.

SQL>
```

Bill:

Run SQL Command Line

```
SQL> insert into bill values (&bill_no,'&customer_email','&bdate',&total);
Enter value for bill_no: 51
Enter value for customer_email: HOIFA.LJA@YUSM.FGA
Enter value for bdate: 25-JAN-2020
Enter value for total: 50
old   1: insert into bill values (&bill_no,'&customer_email','&bdate',&total)
new   1: insert into bill values (51,'HOIFA.LJA@YUSM.FGA','25-JAN-2020',50)

1 row created.

SQL>
```

DATA UPDATE QUERIES:

1. Update prices of product

SQL Plus

```
SQL> UPDATE product
  2  SET price= 220.2
  3  WHERE product_id=90001;

1 row updated.
```

2. Update dates of the sale

SQL Plus

```
SQL> UPDATE sale
  2  SET sdate=to_date('02-FEB-2000','DD-MON-YYYY')
  3  WHERE product_id=90002 AND branch_id=11111;

0 rows updated.
```

3. Change manager of a branch

SQL Plus

```
SQL> UPDATE branch
  2  SET manager_name='new_manager_name'
  3  WHERE branch_id=11111;

1 row updated.
```

4. Update contact details of the seller

SQL Plus

```
SQL> UPDATE seller
  2  SET pHno=9893965437
  3  WHERE seller_id=70010;

1 row updated.
```

5. Increase or decrease the discount percentage of an ongoing sale in a particular branch

SQL Plus

```
SQL> UPDATE sale
  2  SET discount=discount+5
  3  WHERE branch_id=11111 AND product_id=90001;

1 row updated.
```

6. Update salary of employee

SQL Plus

```
SQL> UPDATE employee
  2  SET esalary=esalary+1000
  3  WHERE emp_id=40028 and branch_id=11111;

0 rows updated.
```

7. Update bank details of the seller

SQL Plus

```
SQL> UPDATE seller
  2  SET bank_acc=703052820
  3  WHERE seller_id=70019;

1 row updated.
```

8. Update Total of Bill from product Bills

SQL Plus

```
SQL> UPDATE bill
  2  SET total = (SELECT total FROM (SELECT bill_no, sum(price * ibill_quantity) AS total FROM product NATURAL JOIN prod_bill GROUP BY bill_no) t
  3  WHERE t.bill_no = bill.bill_no);

50 rows updated.
```

9. Update Stock Quantity

SQL Plus

```
SQL> UPDATE stock
  2  SET stock_quantity = stock_quantity - (SELECT sum(ibill_quantity) from bill NATURAL JOIN prod_bill
  3  WHERE branch_id = stock.branch_id AND product_id = stock.product_id
  4  GROUP BY product_id);

80 rows updated.
```

DATA DELETE QUERIES:

1. Delete customer info if last purchase was a year ago

SQL Plus

```
SQL> /* Delete customer info if last purchase was a year ago */
SQL> DELETE FROM customer
  2  where customer_email in
  3  (SELECT customer_email FROM customer NATURAL JOIN bill
  4  GROUP BY customer_email
  5  HAVING max(bdate) < add_months(SYSDATE,-12));

0 rows deleted.
```

2. Delete Stock if quantity reaches 0

SQL Plus

```
SQL> /* Delete Stock if quantity reaches 0 */
SQL> DELETE FROM stock
  2  WHERE stock_quantity = 0;

0 rows deleted.
```

3. Delete Sale info if its over

SQL Plus

```
SQL> /* Delete Sale info if its over */
SQL> DELETE FROM sale
  2  WHERE edate < SYSDATE;

17 rows deleted.
```

4. Delete bill info if purchase was made a year ago

SQL Plus

```
SQL> /*Deleting Child Records */
SQL> DELETE FROM prod_bill
  2  WHERE EXISTS
  3  (SELECT 1 from bill
  4   WHERE bill_no = prod_bill.bill_no
  5   AND
  6   bdate < add_months(SYSDATE,-12)
  7  );

0 rows deleted.

SQL> /*Deleting Parent Record */
SQL> DELETE FROM bill
  2  WHERE bdate < add_months(SYSDATE,-12);

0 rows deleted.
```

5. Remove Seller Info If no purchase has been made from him in last 1 year

SQL Plus

```
SQL> DELETE FROM seller S WHERE NOT EXISTS
  2  (SELECT max(sdate) FROM seller_bill WHERE seller_id = S.seller_id AND sdate > add_months(SYSDATE,-12)
  3  GROUP BY seller_id);

0 rows deleted.
```

DATA RETRIEVAL QUERIES:

1. Retrieve Branch Wise Data Of All Employees

SQL Plus

```
SQL> /* Retrieve Branch Wise Data Of All Employees */
SQL> SELECT * FROM employee
  2  ORDER BY branch_id;

    EMP_ID ENAME                   ESALARY JOIN_DATE  BRANCH_ID
---------- -------------------- ---------- --------- ----------
     40021 MIKEL                    852695 13-FEB-18      11111
     40011 HOPE                     285090 22-OCT-18      11111
     40031 GARRETT                  891611 28-JUL-18      11111
     40032 DONG                     933714 16-MAY-19      11112
     40022 DORTHY                   673993 24-FEB-19      11112
     40012 LETA                     761475 22-NOV-18      11112
     40023 TOMMY                    812473 15-AUG-18      11113
     40033 NATALIE                  514073 19-JUL-19      11113
     40013 PAULETTE                 233343 03-AUG-19      11113
     40024 SCOT                     228901 19-OCT-18      11114
     40014 MYRTLE                   773241 19-DEC-18      11114

    EMP_ID ENAME                   ESALARY JOIN_DATE  BRANCH_ID
---------- -------------------- ---------- --------- ----------
     40034 JOHNNY                   459042 03-MAR-18      11114
     40015 STEVEN                   441432 29-JAN-19      11115
     40025 KERRY                    552007 18-JUN-18      11115
     40035 ADAM                     389605 08-JUN-18      11115
     40026 SAVANNAH                 350953 23-OCT-18      11116
     40036 THANH                    891611 19-FEB-19      11116
     40016 ISREAL                   389605 30-JUL-18      11116
     40027 TONEY                    459042 24-MAR-19      11117
     40037 JILL                     514073 15-MAR-19      11117
     40017 MARGRET                  852974 03-AUG-18      11117
     40038 OSVALDO                  170977 21-SEP-18      11118

    EMP_ID ENAME                   ESALARY JOIN_DATE  BRANCH_ID
---------- -------------------- ---------- --------- ----------
     40028 ANDREAS                  458280 02-MAY-18      11118
     40018 COLE                     138843 21-MAY-19      11118
     40019 SALLIE                   891611 17-FEB-19      11119
     40039 DOROTHEA                 459042 17-APR-19      11119
     40029 ISAIAH                   514073 07-MAR-19      11119
     40030 LEONA                    983341 10-MAY-19      11120
     40040 DARRICK                  933714 25-SEP-19      11120
     40020 GABRIELA                 793548 30-MAY-19      11120

30 rows selected.
```

2. Upcoming sales in 3 months

SQL Plus

```
SQL> /* Upcoming sales in 3 months */
SQL> SELECT * FROM sale
  2  WHERE add_months(SYSDATE, 3) BETWEEN sdate AND edate;

no rows selected
```

3. Find total sale in a branch within last 3 months

SQL Plus

```
SQL> /* Find total sale in a branch within last 3 months */
SQL> SELECT SUM(total) FROM bill
  2  WHERE branch_id = 11111 AND
  3  bdate BETWEEN add_months(SYSDATE, -3) and SYSDATE;

SUM(TOTAL)
----------
      4770
```

4. Customers who have visited more than thrice in a particular branch

SQL Plus

```
SQL> /* Customers who have visited more than thrice in a particular branch */
SQL> SELECT customer_email, count(*) FROM customer C NATURAL JOIN bill
  2  WHERE branch_id = 11111
  3  GROUP BY customer_email
  4  HAVING COUNT(*) > 3;

no rows selected
```

5. Most purchased product on a particular day

SQL Plus

```
SQL> /* Most purchased product on a particular day */
SQL> SELECT * FROM product WHERE
  2  product_id IN (SELECT product_id FROM (SELECT product_id, sum(ibill_quantity) AS total_qty FROM bill NATURAL JOIN prod_bill
  3  GROUP BY (product_id)
  4  ORDER BY total_qty DESC)
  5  WHERE ROWNUM < 2);

PRODUCT_ID PRODUCT_NAME         PRODUCT_DESC
---------- -------------------- ---------------------------------------
     PRICE
----------
     90005 PEANUT BUTTER        BUTTER PACK 50 GM EACH
        95
```

6. 5 most costly products in supply chain

SQL Plus

```
SQL>
SQL> /* 5 most costly products in supply chain */
SQL> SELECT * FROM (SELECT * FROM product
  2  ORDER BY price DESC)
  3  WHERE ROWNUM < 6;

PRODUCT_ID PRODUCT_NAME         PRODUCT_DESC
---------- -------------------- ---------------------------------------
     PRICE
----------
     90006 FACE MAKE UP         COSMETIC
       200

     90007 MANGO                FRUIT
       180

     90015 PONDS                FACE POWDER
       105


PRODUCT_ID PRODUCT_NAME         PRODUCT_DESC
---------- -------------------- ---------------------------------------
     PRICE
----------
     90005 PEANUT BUTTER        BUTTER PACK 50 GM EACH
        95

     90004 ALMOND MILK          PACKED MILK
        80
```

7. Find details of customers who visited a particular branch

SQL Plus

```
SQL>
SQL> /* Find details of customers who visited a particular branch */
SQL> SELECT * FROM customer WHERE
  2  customer_email IN (SELECT DISTINCT(customer_email) FROM BILL
  3  WHERE branch_id = 11111);

CUSTOMER_EMAIL                                     CONTACT_NO CUSTOMER_N
-------------------------------------------------- ---------- ----------
PAKASTE@MSN.COM                                    8096470743 WILLIAMSON
JESPLEY@SBCGLOBAL.NET                              8501501720  VAUGHAN
SARIDDER@AOL.COM                                   8317216702 BROWN
NORTH@HOTMAIL.COM                                  8166913459 HESS
CROBLES@HOTMAIL.COM                                6094652509 MONTES
```

8. Number of Employees working in various branches

SQL Plus

```
SQL> /* Number of Employees working in various branches */
SQL> SELECT branch_id, count(*) FROM employee
  2  GROUP BY branch_id;

 BRANCH_ID   COUNT(*)
---------- ----------
     11111          3
     11117          3
     11112          3
     11114          3
     11116          3
     11120          3
     11113          3
     11115          3
     11119          3
     11118          3

10 rows selected.
```

9. Details of product bought on a particular day

SQL Plus

```
SQL> /* Details of product bought on a particular day */
SQL> SELECT product_id,sum(squantity)
  2  FROM product NATURAL JOIN seller_bill
  3  WHERE sdate = SYSDATE
  4  GROUP BY product_id;

no rows selected
```

10. Get last purchase dates of customers

SQL Plus

```
SQL>
SQL> /* Get last purchase dates of customers */
SQL> SELECT customer_email, max(bdate) AS LAST_PURCHASE_DATE FROM customer NATURAL JOIN bill
  2  GROUP BY customer_email;

CUSTOMER_EMAIL                                     LAST_PURC
-------------------------------------------------- ---------
JAARNIAL@OUTLOOK.COM                               12-NOV-19
JRKORSON@HOTMAIL.COM                               16-SEP-19
STEWWY@AOL.COM                                     29-SEP-19
SCITEXT@MSN.COM                                    14-JUN-19
MMCCOOL@AOL.COM                                    20-OCT-19
JESPLEY@SBCGLOBAL.NET                              03-AUG-19
SARIDDER@AOL.COM                                   21-AUG-19
NORTH@HOTMAIL.COM                                  23-MAY-19
CROBLES@HOTMAIL.COM                                02-SEP-19
IMIGHTB@COMCAST.NET                                04-JUN-19
EMINENCE@LIVE.COM                                  28-OCT-19

CUSTOMER_EMAIL                                     LAST_PURC
-------------------------------------------------- ---------
PARENTS@HOTMAIL.COM                                28-SEP-19
PAKASTE@MSN.COM                                    27-JUN-19
JOHNBOB@HOTMAIL.COM                                22-JUN-19
WAGNERCH@ME.COM                                    03-JUN-19
ERYNF@HOTMAIL.COM                                  05-SEP-19
TUBAJON@LIVE.COM                                   20-SEP-19
AIBRAHIM@LIVE.COM                                  28-MAY-19
NIGHTHAWK@ATT.NET                                  31-OCT-19
AKOBLIN@YAHOO.COM                                  25-JUN-19

20 rows selected.
```

11. Get Data of all newly joined employees whose salary is greater than 500000 or less than 200000

SQL Plus

```
SQL> SELECT * FROM employee WHERE esalary > 500000
  2  UNION
  3  SELECT * FROM employee WHERE esalary < 200000;

    EMP_ID ENAME                  ESALARY JOIN_DATE  BRANCH_ID
---------- -------------------- ---------- --------- ----------
     40012 LETA                    761475 22-NOV-18      11112
     40014 MYRTLE                  773241 19-DEC-18      11114
     40017 MARGRET                 852974 03-AUG-18      11117
     40018 COLE                    138843 21-MAY-19      11118
     40019 SALLIE                  891611 17-FEB-19      11119
     40020 GABRIELA                793548 30-MAY-19      11120
     40021 MIKEL                   852695 13-FEB-18      11111
     40022 DORTHY                  673993 24-FEB-19      11112
     40023 TOMMY                   812473 15-AUG-18      11113
     40025 KERRY                   552007 18-JUN-18      11115
     40029 ISAIAH                  514073 07-MAR-19      11119

    EMP_ID ENAME                  ESALARY JOIN_DATE  BRANCH_ID
---------- -------------------- ---------- --------- ----------
     40030 LEONA                   983341 10-MAY-19      11120
     40031 GARRETT                 891611 28-JUL-18      11111
     40032 DONG                    933714 16-MAY-19      11112
     40033 NATALIE                 514073 19-JUL-19      11113
     40036 THANH                   891611 19-FEB-19      11116
     40037 JILL                    514073 15-MAR-19      11117
     40038 OSVALDO                 170977 21-SEP-18      11118
     40040 DARRICK                 933714 25-SEP-19      11120

19 rows selected.
```

12. Get details of Seller who sells all the products

SQL Plus

```
SQL> SELECT * FROM seller S WHERE NOT EXISTS
  2  (SELECT product_id FROM product
  3  MINUS
  4  SELECT DISTINCT(product_id) FROM seller_bill WHERE seller_id = S.seller_id);

no rows selected
```

PROCEDURES

1. A PROCEDURE TO DISPLAY BILL WITH LIST OF PRODUCTS IN IT AND THEIR DETAILS.

CODE:

SQL Plus

```
SQL> create or replace procedure DISPLAY_BILL (bill_number number)
  2  as
  3  c_name customer.customer_name%type;
  4  c_email customer.customer_email%type;
  5  bill_date bill.bdate%type;
  6  bill_total bill.total%type;
  7  cursor prod_list is
  8      select product_id, product_name, ibill_quantity, price from prod_bill natural join product
  9      where bill_no = bill_number;
 10  products_data prod_list%rowtype;
 11  ptotal bill.total%type;
 12  begin
 13      select customer_name,customer_email into c_name,c_email from customer
 14      where customer_email in (select customer_email from bill where bill_no = bill_number);
 15      select bdate, total into bill_date,bill_total from bill where bill_no = bill_number;
 16      dbms_output.put_line(chr(10));
 17      dbms_output.put_line('BILL NUMBER: ' || bill_number);
 18      dbms_output.put_line('BILL DATE: ' || bill_date);
 19      dbms_output.put_line('CUSTOMER NAME: ' || c_name);
 20      dbms_output.put_line('CUSTOMER EMAIL: ' || c_email);
 21      dbms_output.put_line(rpad('NAME',25,' ') || rpad('ID',10,' ') || rpad('PRICE',10, ' ') || rpad('QTY.',7, ' ') || rpad('TOTAL',10, ' '));
 22      open prod_list;
 23      loop
 24          fetch prod_list into products_data;
 25          exit when prod_list%NOTFOUND;
 26          ptotal := products_data.price * products_data.ibill_quantity;
 27          dbms_output.put_line(rpad(products_data.product_name,25,' ') || rpad(products_data.product_id,10,' ') ||
 28          rpad(products_data.price,10, ' ') || rpad(products_data.ibill_quantity,7, ' ') || rpad(ptotal,10, ' '));
 29      end loop;
 30      close prod_list;
 31      dbms_output.put_line('BILL TOTAL: ' || bill_total);
 32      dbms_output.put_line(chr(10));
 33  end;
 34  /

Procedure created.
```

OUTPUT:

SQL Plus

```
SQL> set serveroutput on
SQL> EXEC DISPLAY_BILL(50);


BILL NUMBER: 50
BILL DATE: 20-MAR-19
CUSTOMER NAME: CHOI
CUSTOMER EMAIL: STEWWY@AOL.COM
NAME                      ID        PRICE     QTY.   TOTAL
GOLDEN ARCS.              90001     20        9      180
PIZZA_CHEEZE              90010     75        5      375
YOGURT                    90011     25        5      125
CABBAGE                   90020     20        9      180
BILL TOTAL: 860



PL/SQL procedure successfully completed.

SQL>
```

2. A PROCEDURE TO DISPLAY ALL STOCKS IN A PARTICULAR BRANCH

CODE:

SQL Plus

```
SQL> create or replace procedure show_stocks (branchId number)
  2  as
  3  cursor stock_list is
  4      select product_id, stock_quantity from stock
  5      where branch_id = branchId;
  6  stock_data stock_list%rowtype;
  7  pname  product.product_name%type;
  8  pprice product.price%type;
  9  pdesc  product.product_desc%type;
 10  psaled  sale.discount%type;
 11  psalesd sale.sdate%type;
 12  psaleed sale.edate%type;
 13  begin
 14      dbms_output.put_line('Branch Id: ' || branchId);
 15      dbms_output.put_line(rpad('NAME',17,' ')||rpad('ID',7,' ')||rpad('DESCRIPTION',20,' ')||rpad('PRICE',7,' ')||
 16      rpad('Qty',5,' ')||rpad('SALE',6,' ')||rpad('Sale_Start',15,' ')||rpad('Sale_End',15,' '));
 17      open stock_list;
 18      loop
 19          begin
 20              fetch stock_list into stock_data;
 21              exit when stock_list%NOTFOUND;
 22              select product_name,product_desc,price into pname,pdesc,pprice from product where product_id = stock_data.product_id;
 23              select discount,sdate,edate into psaled,psalesd,psaleed from sale where product_id = stock_data.product_id and branch_id = branchId;
 24              exception
 25                  when NO_DATA_FOUND then
 26                  psaled := 0;
 27                  psalesd := null;
 28                  psaleed := null;
 29          end;
 30          dbms_output.put_line(rpad(pname,17,' ')||rpad(stock_data.product_id,7,' ')||rpad(pdesc,20,' ')||
 31          rpad(stock_data.stock_quantity,5,' ')||rpad(pprice,7,' ')||rpad(psaled,6,' ')||rpad(psalesd,15,' ')||rpad(psaleed,15,' '));
 32      end loop;
 33      close stock_list;
 34  end;
 35  /

Procedure created.
```

OUTPUT:

SQL Plus

```
SQL> EXEC SHOW_STOCKS(11111);
Branch Id: 11111
NAME             ID     DESCRIPTION         PRICE  Qty  SALE  Sale_Start
Sale_End
GOLDEN ARCS.     90001  BISCUIT             181  20     36    21-JAN-19
12-JUN-19
BANANA           90002  FRUIT               147  50     0
BISLERI          90003  DRINKING WATER      141  30     0
ALMOND MILK      90004  PACKED MILK         65   80     0
YOGURT           90011  EATABLE MILK PRODUCT168  25     42    23-MAR-19
04-NOV-19
PEAR             90012  FRUIT               60   60     0
WHOLE_BREAD      90013  BREAD               186  30     0
KITKAT           90014  CHOCOLATE           59   45     0

PL/SQL procedure successfully completed.

SQL>
```

FUNCTIONS

1. A FUNCTION TO DISPLAY AVERAGE SALE OF ALL BRANCHES COMBINED.

CODE:

SQL Plus

```
SQL> create or replace function avg_sale(dateavg  in date)
  2  return number
  3  as
  4  average number;
  5  cursor branches is
  6      select branch_id from branch;
  7  branch_data branches%rowtype;
  8  branch_count number;
  9  branch_sum number;
 10  begin
 11      select count(branch_id) into branch_count from branch;
 12      average := 0;
 13      open branches;
 14      loop
 15          begin
 16              fetch branches into branch_data;
 17              exit when branches%NOTFOUND;
 18              select sum(total) into branch_sum from bill where branch_id = branch_data.branch_id and bdate = dateavg;
 19              if branch_sum is null then
 20                  branch_sum := 0;
 21              end if;
 22          end;
 23          average := average + branch_sum;
 24      end loop;
 25      close branches;
 26      average := average / branch_count;
 27      return(average);
 28  end;
 29  /

Function created.

SQL>
```

OUTPUT:

SQL Plus

```
SQL> DECLARE
  2  average_sale number;
  3  begin
  4  average_sale := avg_sale(to_date('03-MAR-19','DD-MON-YY'));
  5  dbms_output.put_line('AVERAGE SALE OF ALL BRANCH IS: ' || average_sale);
  6  end;
  7  /
AVERAGE SALE OF ALL BRANCH IS: 91

PL/SQL procedure successfully completed.
```

2. A FUNCTION TO FIND POPULARITYOF A PRODUCT IN TWO BRANCHES

CODE:

SQL Plus

```
SQL> create or replace function popular_prod(branch1 number, branch2 number, productId number)
  2  return number
  3  as
  4  branch3 number;
  5  qbranch1 number := 0;
  6  qbranch2 number := 0;
  7  temp number;
  8  cursor qb1 is select bill_no from bill where branch_id = branch1;
  9  qb1d qb1%rowtype;
 10  cursor qb2 is select bill_no from bill where branch_id = branch2;
 11  qb2d qb2%rowtype;
 12  begin
 13      open qb1;
 14      loop
 15          temp := 0;
 16          begin
 17              fetch qb1 into qb1d;
 18              exit when qb1%NOTFOUND;
 19              select ibill_quantity into temp from prod_bill where product_id = productId and bill_no = qb1d.bill_no;
 20              exception
 21                  when NO_DATA_FOUND then temp := 0;
 22          end;
 23          qbranch1 := qbranch1 + temp;
 24      end loop;
 25      close qb1;
 26      open qb2;
 27      loop
 28          temp := 0;
 29          begin
 30              fetch qb2 into qb2d;
 31              exit when qb2%NOTFOUND;
 32              select ibill_quantity into temp from prod_bill where product_id = productId and bill_no = qb2d.bill_no;
 33              exception
 34                  when NO_DATA_FOUND then temp := 0;
 35          end;
 36          qbranch2 := qbranch2 + temp;
 37      end loop;
 38      close qb2;
 39      if (qbranch1 > qbranch2) then branch3 := branch1;
 40      elsif (qbranch2 > qbranch1) then branch3 := branch2;
 41      else branch3 := 0;
 42      end if;
 43      return branch3;
 44  end;
 45  /

Function created.
```

OUTPUT:

SQL Plus

```
SQL> DECLARE
  2  branch_popular number;
  3  begin
  4  branch_popular := popular_prod(11111,11112,90001);
  5  dbms_output.put_line('Product 90001 is most popular in branch_id: ' || branch_popular);
  6  end;
  7  /
Product 90001 is most popular in branch_id: 11112

PL/SQL procedure successfully completed.
```

TRIGGERS

1. A TRIGGER TO CHECK IF THE QUANTITY OF STOCSK AFTER UPDATE IS A NON NEGATIVE INTEGER

SQL Plus

```
SQL> create or replace trigger stock_empty
  2  before insert or update on stock
  3  for each row
  4  begin
  5  if(:new.stock_quantity < 0) then
  6      raise_application_error(-21000, 'not that much quantity available');
  7  end if;
  8  end;
  9  /

Trigger created.

SQL> SELECT * FROM STOCK WHERE BRANCH_ID = 11111;

PRODUCT_ID  BRANCH_ID STOCK_QUANTITY
---------- ---------- --------------
     90001      11111            181
     90002      11111            147
     90003      11111            141
     90004      11111             65
     90011      11111            168
     90012      11111             60
     90013      11111            186
     90014      11111             59

8 rows selected.

SQL> UPDATE STOCK
  2  SET STOCK_QUANTITY = -2
  3  WHERE BRANCH_ID = 11111;
UPDATE STOCK
       *
ERROR at line 1:
ORA-21000: error number argument to raise_application_error of -21000 is out of
range
ORA-06512: at "SYSTEM.STOCK_EMPTY", line 3
ORA-04088: error during execution of trigger 'SYSTEM.STOCK_EMPTY'
```

2. A TRIGGER TO CHECK IF SALARY OF EMPLOYEE IS ALWAYS INCREASED AND IF IT IS TRIED TO DECREASE THEN DB WON'T ALLOW SUCH UPDATE

SQL Plus

```
SQL> create or replace trigger salary_update
  2  before update of esalary on employee
  3  for each row
  4  begin
  5  if (:new.esalary < :old.esalary) then
  6      :new.esalary := :old.esalary;
  7  end if;
  8  end;
  9  /

Trigger created.

SQL> SELECT esalary from employee
  2  where emp_id = 40011;

   ESALARY
----------
    285090

SQL> update employee
  2  set esalary = esalary * 0.5;

30 rows updated.

SQL> SELECT esalary from employee
  2  where emp_id = 40011;

   ESALARY
----------
    285090

SQL>
```

3. A TRIGGER WHICH DOESN'T ALLOW UPDATE OF EMPLOYEES SALARY BEFORE SIX MONTHS FROM HIS JOIN DATE.

SQL Plus

```
SQL> create or replace trigger employee_salary_update
  2  before update of esalary on employee
  3  for each row
  4  begin
  5  if(SYSDATE - :old.join_date < 180) then
  6      raise_application_error(-20000,'Invalid Update Option');
  7  end if;
  8  end;
  9  /

Trigger created.

SQL> update employee
  2  set esalary = esalary * 0.9
  3  where emp_id = 40040
  4  ;
update employee
       *
ERROR at line 1:
ORA-20000: Invalid Update Option
ORA-06512: at "SYSTEM.EMPLOYEE_SALARY_UPDATE", line 3
ORA-04088: error during execution of trigger 'SYSTEM.EMPLOYEE_SALARY_UPDATE'
```

4. A TRIGGER TO GIVE CUSTOMERS DISCOUNT IF THEY ARE SHOPPING FOR FIRST TIME.

SQL Plus

```
SQL> create or replace trigger customer_trigger
  2  before insert on bill
  3  for each row
  4  declare
  5  c number;
  6  begin
  7      select count(*) into c from bill where customer_email = :new.customer_email group by customer_email;
  8      exception
  9      when NO_DATA_FOUND then
 10          :new.total := :new.total * 0.80;
 11          dbms_output.put_line('You Got 20% Off Coz This Is Your 1st Shopping At Our Stores.');
 12  end;
 13  /

Trigger created.

SQL> insert into bill values (&bill_no,'&customer_email','&bdate',&total,&branch_id);
Enter value for bill_no: 51
Enter value for customer_email: QWERA.TE@HYSTA.CTY
Enter value for bdate: 23-NOV-19
Enter value for total: 400
Enter value for branch_id: 11111
old   1: insert into bill values (&bill_no,'&customer_email','&bdate',&total,&branch_id)
new   1: insert into bill values (51,'QWERA.TE@HYSTA.CTY','23-NOV-19',400,11111)
You Got 20% Off Coz This Is Your 1st Shopping At Our Stores.

1 row created.
```

-----Thank you-----