Data Base Management System

Project Report

Project: Supermarket Management

Submitted By:

Bhavesh Rughwani-16BCP040

Smit Sanghavi-16BCP044

Jigar Shah-16BCP047

Supragya Lal-16BCP054

Tirth Patel -16BCP056

Index

Benefits of Data Base Management System over File System

Concurrent Access Anomalies

- It basically means that two different users try to access a particular database and try to make changes in it according to their requirement.
- In a gist , In the context of supermarket, concurrent access anomalies will not be an issue since , in a super market there won't be any possibility of two customers buying a product with same product id at two different cash counters.
- Suppose we have a section of super market which stores stationary items.
- Here , We are enlisting details about stock of n books
- In the table given below , as we can see that there are two customers who are buying the same product.
- But as after ,the book being sold to the customer id145296 so at that same time the book cannot be sold to any other customer with any other customer id.

Product Id	Brand	Customer Id	Product Name	Quantity
121263	Classmate	145296	Diary	2
122263	Classmate	145296	Notebook	1
132456	Navneet	145369	Sketchbook	3
145189	Boss	145369	Scrapbook	1
122263	Classmate	145297	Notebook	1
145290	Camlin	145300	Notebook	5

Data Isolation

- Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.
- A lower isolation level increases the ability of many users to access the same data at the same time ,but increase he number of concurrency effects (such as dirty reads or lost updates) users might encounter.
- Conversely, a higher isolation level reduces the types of concurrency effects that users may encounter, but requires more system resources and increases the chances that one transaction will block another.
- For example , in context to supermarket, when a user is creating a purchase order and has created the header ,but not the purchase order lines, is the header available for other systems/users ,carrying out concurrent operations (such as a report on purchase orders), to see?

Integrity Problems

- The data values stored in the database must satisfy certain types of consistency constraints.
- Suppose the university maintains an account for each department, and records the balance amount in each account.
- Suppose also that the university requires that the account balance of a department may never fall below zero.
- Developers enforce these constraints in the system by adding appropriate code in the various application programs.
- However, when new constraints are added, it is difficult to change the programs to enforce them.
- The problem is compounded when constraints involve several data items from different files.
- An example being textual data entered where a date-time value is required. Rules for data derivation are also applicable, specifying how a data value is derived based on algorithm, contributors and conditions. It also specifies the conditions on how the data value could be re-derived.
- In a Super Market, when one is preparing the bill, the same time inventory record will be updated, so data integrity provides access but data will be allowed to change as per algorithm only, other can access but cannot change.

DIFFICULTY IN ACCESSING DATA

- When there is very huge amount of data, it is always a time consuming task to search for particular information from the file system.
- Conventional file-processing environments so not allow needed data to be retrieved in a convenient and efficient manner
- In classical file organization the data is stored in files.
- Whenever data has to be retrieved as per requirements, a new application program has to be written which, is a very tedious process.
- Consider following table. In, File system if one wants to find ID of costumer who bought a product of class mate then we need to ask programmer to make an application which can fulfill our purpose or extract information manually
- Both of these alternatives are time consuming
- Further, if we want to find some other information like product ID of products of Classmate brand then we have to again ask the programmer to write the application program or extract information manually.
- Both these ways are quite tedious.

Product Id	Brand	Customer Id	Product Name	Quantity
121263	Classmate	145296	Diary	2
122263	Classmate	145296	Notebook	1
132456	Navneet	145369	Sketchbook	3
145189	Boss	145369	Scrapbook	1
122263	Classmate	145297	Notebook	1
145290	Camlin	145300	Notebook	5

SECURITY PROBLEMS

- Data security means prevention of data accession by unauthorized users.
- Every user of database system should not be allowed to access all the data
- There is no centralized control of data in classical file organization.
- Moreover, application programs are added to file-processing system in an ad hoc manner, enforcing security constraints is difficult.
- Here,in context to supermarket store, if all users are allowed to access all data then, there are chances of data getting altered which can create a problem.

ATOMICITY PROBLEMS

- Atomicity is the first stage of the ACID model (Atomicity, Consistency, Isolation, Durability), which is a set of guidelines for ensuring the accuracy of database transactions.
- Either all operations of the transaction are reflected properly in the database, or none are.
- If all the operations are executed successfully,then the entire transaction should be saved to the database
- It is essential that a database system that claims to offer atomicity be able to do so even in the face of failure in power supply or the underlying operating system or application that uses the database.
- Suppose we have a section of super market which stores stationary items.
- Here , We are enlisting details about stock of n books.
- So, basically the table gives the information about the things which the customer have bought it's quantity etc.
- TABLE NAME :CUSTOMER_STATIONARY

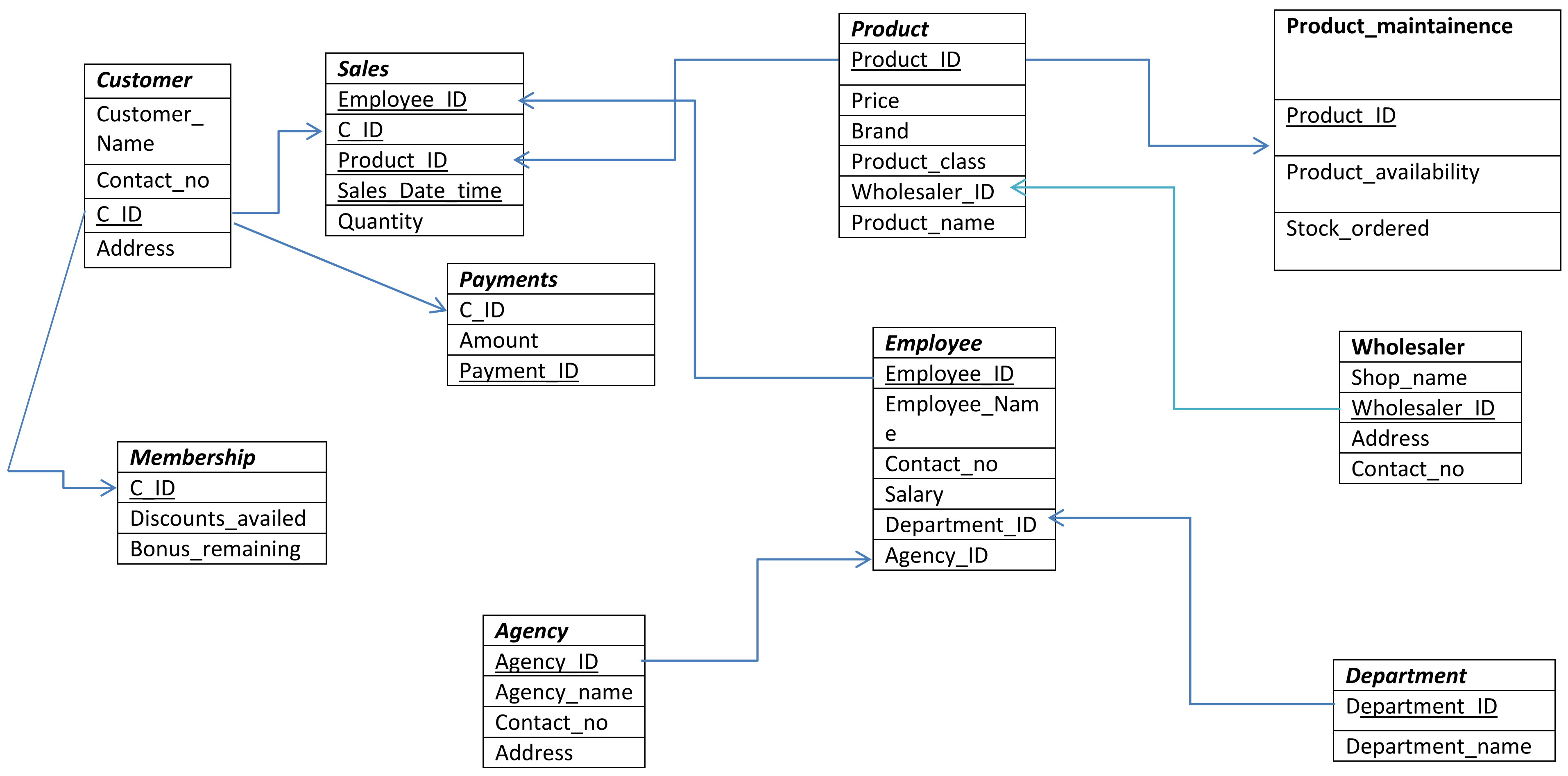
Product Id	Brand	Customer Id	Product Name	Quantity
121263	Classmate	145296	Diary	2
122263	Classmate	145296	Notebook	1
132456	Navneet	145369	Sketchbook	3
145189	Boss	145369	Scrapbook	1
122263	Classmate	145297	Notebook	1
145290	Camlin	145300	Notebook	5

- In the table given below the seller updates the stock so that he can keep a track of products to be bought.
- But now if we write two statements in any database(say MySQL) to first update the CUSTOMER_STATIONARY TABLE and then update the SELLER_STATIONARY TABLE, then this transaction should be completed fully or should not take place.

Data redundancy and inconsistency

- Since different programmers create the files and application programs over a long period, the various files are likely to have different structures and the programs may bewritten in several programming languages.
- Moreover, the same information may be duplicated in several places
- It may lead to data inconsistency. That is, the various copies of the same data may no longer agree.
- For example in context to supermarket, if there is a list of customers in two different shops then in file system there are chances that both these shops reflect different customer ID for same customer.

Relational model



Tables

- Product(<u>Product_ID</u>, Product_Name, Brand, Product_class, Price)
- Employee(<u>Employee_ID</u>, Name, Contact_no, Salary)
- Customer(<u>C_ID</u>, Customer_Name, Contact_no, Address)
- Sales(<u>C_ID, Employee_ID, Product_ID, Time)</u>
- Payments(Payment_ID,Amount,C_ID)
- Agency(<u>Agency_ID</u>,Agency_name,Contact_no,Address)
- Wholesaler(<u>Wholesaler_ID</u>,Shop_name,Address,Contact_no)
- Department(<u>Department_ID</u>,Department_Name)
- Product_maintainence(<u>Product_ID</u>,Product_availability,Stock_ordered)
- Membership(C_ID,Discounts_availed,Bonus_remaining,)

Keys

1) Super Keys:-

- *Employee*
  (i) (Employee_ID)
  (ii)(Employee_ID, Phone No.)
  (iii)(Employee ID, Name, Phone No.)
- *Product*
  (i)(Product ID) (ii)(Product ID, Brand, Product class)

  (iii)(Product ID, Product Name, Brand, Product Class)
- *Customer*
  (ii)(C_ID)
  (ii)(C_ID, Phone No.)
  (iii)(C_ID, Name, Phone No.)
- *Sales*
  (i) (Sales_time,C_ID, Product_ID,Employee_ id)

- *Wholesaler*
  (i) (Wholesaler_id,contactno.)
  (ii)(Wholesaler_id,contactno,shopname)
- *Customer*
  (i) (C_ID,name,Phoneno.)
  (ii) (C_ID,name)
- *Payments*
  (i)Payment_id
  (ii)(payment_id,C_ID)
- *Membership_Account*
  (i)(C_ID,Discounts_availed)
  (ii)(Discounts_availed,Bonus_remaining)
- *Department*
  (i)(*Department_name,Department_id)*
  (ii)Department_name
- *Agency*
  (i)(Agency_id,Agencyname,Contactno.)
  (ii)agency_id
  (iii)(Agency_id,Agencyname)

***Candidate Keys**:-*

- *Employee*
  (Employee_ID)
- *Product*
  (Product_ID)
- *Customer*
  (C_ID)
- *Sales: -*
  No Candidate keys
- *Wholesaler*
  (i) (Wholesaler_id,contactno,shopname)

- *Payments*
  (i)(payment_id,C_ID)

- *Membership_Account*
  (i)(C_ID,Discounts_availed)
  (ii)(Discounts_availed,Bonus_remaining)

- *Department*
  (i)(Department_name,Department_id)

- *Agency*
  (i)(Agency_id,Agencyname,Contactno.)
  (ii)(Agency_id,Agencyname)

***Primary Keys**: -*

- *Employee*
  (Employee_ID)
- *Product*

  (Product_ID)

- *Customer*
  (C_ID)
- *Sales :-*
  Time
  C_ID
  Product_ID
  Employee_ id

- *Wholesaler*

  Wholesaler_id

- *Customer*

C_ID

- *Payments*

  Payment_id

- *Membership_Account*

  C.ID

Discounts_availed

Bonus_remaining

- *Department*

Department_name

- *Agency*

  Agency_id

***Foreign Keys**: -*

- *Employee*

  Department_ID
- *Product*

  Wholesaler_id

- *Customer*

No keys
- *Sales: -*
  1. *Employee ID*
  2. *C_ ID*
  3. *Product ID*

- *Wholesaler*

  No keys

- *Customer*

  No Keys

- *Payments*

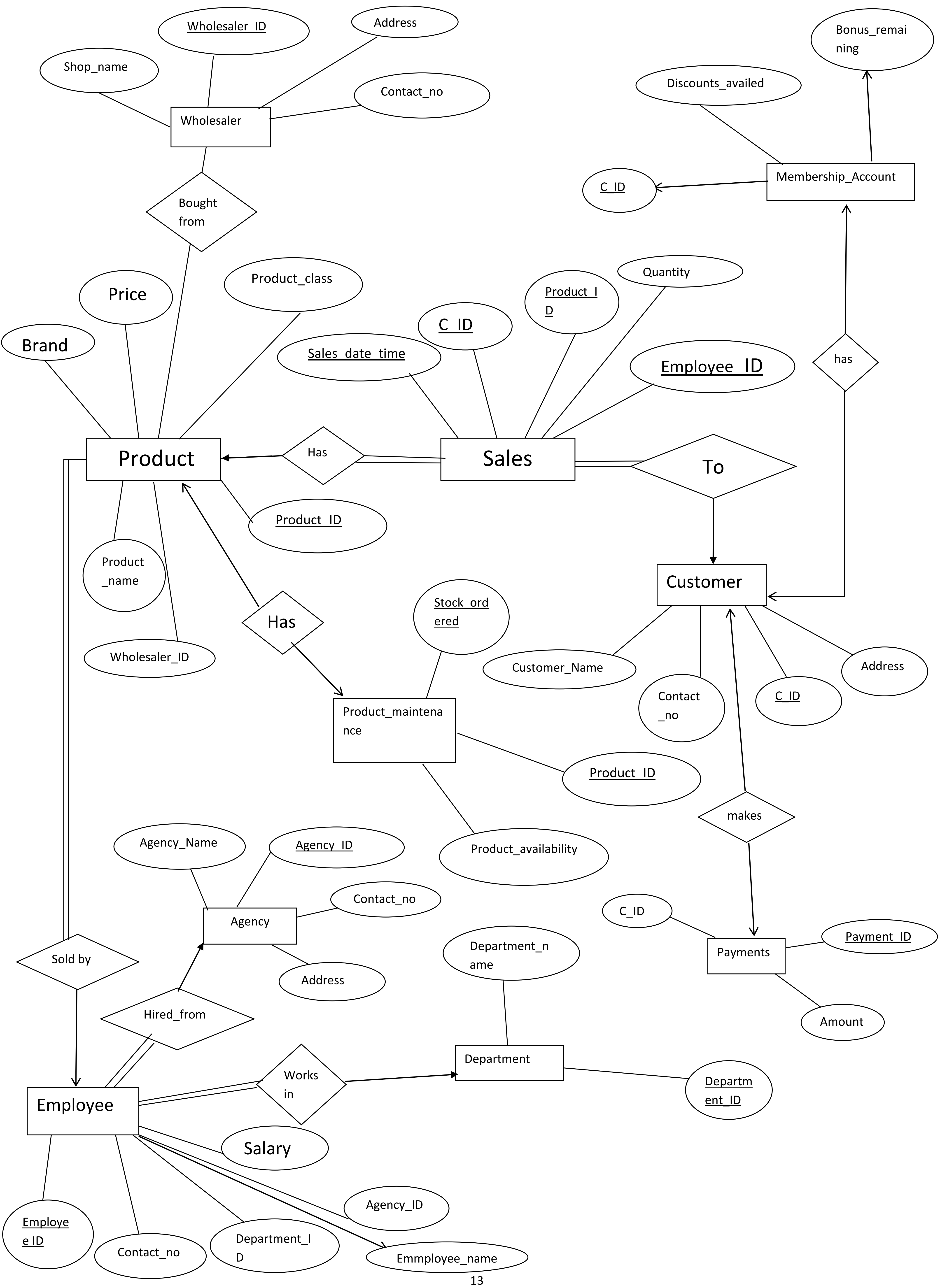
  C_ID
- *Membership_Account*

  C_ ID

- *Department*
  No Keys

- *Agency*
  No Keys

E-R Model


Wholesaler_ID
Address
Shop_name
Contact_no
Wholesaler
Bought from
Bonus_remaining
Discounts_availed
Membership_Account
C_ID
has
Price
Product_class
Brand
Product_ID
Quantity
C_ID
Sales_date_time
Employee_ID
Product
Has
Sales
To
Product_ID
Customer
Product _name
Has
Stock_ordered
Wholesaler_ID
Customer_Name
Address
Contact _no
C_ID
Product_maintenance
Product_ID
makes
Agency_Name
Agency_ID
Product_availability
Contact_no
Agency
C_ID
Department_name
Payment_ID
Sold by
Payments
Address
Hired_from
Amount
Works in
Department
Department_ID
Employee
Salary
Agency_ID
Employee ID
Contact_no
Department_ID
Emmployee_name

Queries

16BCP040

1.SELECTION(σ)-select name RAGHU from the EMPLOYEE table

σname=”RAGHU”(EMPLOYEE)

2.PROJECTION(π) –project product id and brand from the Product table

Πproduct_id,brand(Product)

3.UNION(U)-select C. ID from customer and Union with product id from product

ΠC_ ID(CUSTOMER) U ΠProduct_ID(Product)

4.CARTESIAN PRODUCT(×) –select names where name=’Ramesh’ from the cartesian product table employee and customer

σname=’Ramesh’ (EMPLOYEE× CUSTOMER)

5.NATURAL JOIN (⋈)– natural join of SALES and CUSTOMER

ΠC_ID (SALES ⋈ CUSTOMER)

6.SET DIFFERENCE(-)-select tuples from EMPLOYEE that are not present in CUSTOMER

Π department_ID(DEPARTMENT)-Πdepartment_ID(EMPLOYEE)

7.composition of two form-project customer id and address of customers who bought something

ΠC_ID(SALES ⋈ CUSTOMER)

8.composition of three form-project product id and class of products with brand name puma which are sold

ΠProduct Id,Product class (σBRAND=’PUMA’ (SALES ⋈ PRODUCT))

16BCP056

1.SELECTION(σ)-select customer id 102011 from the CUSTOMER table

σ_{C_ID}=102011(CUSTOMER)

2.PROJECTION(π) –project customer id and Address from the CUSTOMER table

ΠC_ID,Address(CUSTOMER)

3.UNION(U)-select account no from customer and Union with salary table from employee

$\Pi_{Contact\_no}$(CUSTOMER) U Πsalary(EMPLOYEE)

4.CARTESIAN PRODUCT(×) –select salary less than 10k from the join table employee and SALES

σsalary<10k(EMPLOYEE× SALES)

5.NATURAL JOIN (⋈)– Take natural join of EMPLOYEE and CUSTOMER

ΠEmployee_ID (EMPLOYEE ⋈ SALES)

6.SET DIFFERENCE(-) :Find tuples present in Employee but not in customer table

ΠWholesaler_ID(WHOLSALER)- Π department_ID(PRODUCT)

7.composition of two form:project customer id and address of employee who sold something

$\Pi_{employee\_ID}$,Address(EMPLOYEE ⋈ SALES)

8.composition of three form:project employee id and phone no of employees who has salary grater than 10000 and sold something

$\Pi_{employee\_ID}$,Contact_ no (σsalary<10k(EMPLOYEE⋈SALES))

16BCP047

1.SELECTION(σ)-select store_id from the store table

σcontact_no=9876564709(WHOLESALER)

2.PROJECTION(π) –project product_id and sales_date from the sales table

$\Pi_{product\_id}$,Sales_date(SALES)

3.UNION(U)-select Payment_ID from Payments and Union with C_ID from customer table

Π_{C_ID}(CUSTOMER) U $\Pi_{Payment\_ID}$(PAYMENTS)

4.CARTESIAN PRODUCT(×) –select salary less than 50k from the join table employee and customer

σsalary<50k(CUSTOMER× EMPLOYEE)

5.NATURAL JOIN (⋈)– natural join of STORE and CUSTOMER

STORE ⋈ CUSTOMER

6.SET DIFFERENCE(-) :Tuples present in product but not in sales table

Πproduct_ID(PRODUCT)-Πproduct_ID(SALES)

7.composition of two form:project customer name and product_ID of product sold to customers

Πcustomer_name,product_ID(SALES⋈CUSTOMER)

8.composition of three form:project name and salary of employee named raj who sold something

$\Pi_{employee_name,}$salary (σname=RAJ(EMPLOYEE⋈SALES))

16BCP044

1.SELECTION(σ)-select product id from the product table

σ Department_name="Security" (DEPARTMENT)

2.PROJECTION(π) –project price and brand from the product table

Πagency_name,agency_ID(AGENCY)

3.UNION(U)-select product id from Product and Union with Customer_ID from customer table

Πproduct_ID(SALES) U Π_{C_ID}(CUSTOMER)

4.CARTESIAN PRODUCT(×) –select name='Ram' from the Cartesian product of table employee and customer

σName='Ram'(CUSTOMER× SALES)

5.NATURAL JOIN (⋈)– natural join of PRODUCT and CUSTOMER

Πemployee_ID(AGENCY ⋈EMPLOYEE)

6.SET DIFFERENCE(-):tuples present in agency but not in product table

Пagency_ID(AGENCY)-Пagency_ID(EMPLOYEE)

7.composition of two form:customer name and address of customers who bought some product.

Пcustomer name,address(PRODUCT⋈ CUSTOMER)

8.composition of three form:select and project salary of employees who sold something and have salary greater than 15000

Пsalary (σsalary=15000(EMPLOYEE⋈SALES))

<u>16BCP054</u>

-

1.SELECTION(σ)-select salary greater than 5000 from the employee table

σ salary>5000 (employee)

2.PROJECTION(π) –project emp_ID and salary from the employee table

Пemployee_ID,salary(Empoyee)

3.UNION(U)-select phone no. from Employee and Union with product_id from sales table

ПContact_n(Customer) U $\Pi_{product\_id}$(sales)

4.CARTESIAN PRODUCT(×) – join table sales and customer

CUSTOMER× SALES

5.NATURAL JOIN (⋈)– natural join of PRODUCT and CUSTOMER

Пbrand(PRODUCT ⋈ SALES)

6.SET DIFFERENCE(-) :Tuples present in department table but not in employee table

Пproduct_ID(PRODUCT)-Пproduct_ID(PRODUCT_MAINTAINENCE)

7.composition of two form:project wholesaler_ID of wholesaler from whom stock was bought

$\Pi_{Wholesaler\_ID}$(WHOLESALER ⋈PRODUCT)

8.composition of three form:select and project phone no of employees allocated a department

Πphone no. (σphone no.(EMPLOYEE⋈DEPARTMENT))

SQL queries

16BCP044

1.display list of product with product id=10243 and price=2000

- SELECT * FROM PRODUCT WHERE PRODUCT_ID=10243 AND PRICE=2000;

2.display name of wholesalers shop from which something is bought.

- SELECT SHOP_NAME FROM WHOLESALER,PRODUCT WHERE PRODUCT.WHOLESALER_ID=WHOLESALER.WHOLESALER_ID;

3.display phone number of employee belonging to security department and have salary greater than 7000

- SELECT PHONENO FROM DEPARTMENT NATURAL JOIN EMPLOYEE WHERE DEPARTMENT_NAME='SECURITY' AND SALARY>7000;

4.display customer name in alphabetical order who bought product with product id=10838

- SELECT CUSTOMER_NAME FROM CUSTOMER NATURAL JOIN SALES WHERE PRODUCT_ID=10838 ORDER BY CUSTOMER_NAME;

5.display name and employee id of employee grouped by agency from which they are hired

- SELECT EMPLOYEE_NAME,EMPLOYEE_ID,AGENCY_NAME FROM EMPLOYEE NATURAL JOIN AGENCY GROUP BY AGENCY_NAME;

16BCP054

1.display name of customers with 0 amount of discounts availed and have bought something of brand nike.

- SELECT CUSTOMER_NAME FROM CUSTOMER,SALES,PRODUCT,MEMBERSHIP WHERE CUSTOMER.C_ID=MEMBERSHIP.C_ID AND CUSTOMER.C_ID=SALES.C_ID AND SALES.PRODUCT_ID=PRODUCT.PRODUCT_ID AND DISCOUNTS_AVAILED=0 AND PRODUCT.BRAND='NIKE';

2.display product id of all products and employee id who sold them

- SELECT PRODUCT_ID,EMPLOYEE_ID FROM PRODUCT,EMPLOYEE,SALES WHERE PRODUCT.PRODUCT_ID=SALES.PRODUCT_ID AND EMPLOYEE.EMPLOYEE_ID=SALES.EMPLOYEE_ID;

3.group product id and product name by class of their product in alphabetical manner.

- SELECT PRODUCT_ID,PRODUCT_NAME,PRODUCT_CLASS FROM PRODUCT GROUP BY PRODUCT_CLASS ORDER BY PRODUCT_NAME;

4.display all product name and product id with null availability.

- SELECT PRODUCT_ID,PRODUCT_NAME FROM PRODUCT,PRODUCT_MAINTAINENCE WHERE PRODUCT_AVAILABILITY IS NULL;

5.display name of agency from which max number of employees are hired

SELECT AGENCY_NAME FROM AGENCY ,EMPLOYEE WHERE EMPLOYEE.AGENCY_ID=AGENCY.AGENCY_ID AND (SELECT COUNT(AGENCY_ID) FROM EMPLOYEE GROUP BY AGENCY_ID)>(SELECT COUNT(AGENCY_ID) FROM AGENCY GROUP BY AGENCY_ID);

6.display department wise average salary of employees.

- SELECT DEPARTMENT_NAME,AVG(SALARY) FROM EMPLOYEE,DEPARTMENT GROUP BY DEPARTMENT_NAME HAVING EMPLOYEE.DEPARTMENT_ID=DEPARTMENT.DEPARTMENT_ID;

16BCP040

1)Display IDs of all the agencies

select agency_id

from agency;

2)Display shopname whose wholesaler ID is "11163"

select Shopname

from wholesaler

where wholesaler_id="11163";

3)Display name and contact no. of employees in each department whose salary is less than 5000

select name,contact no.

from employee

group by department_id

having salary<5000;

4)Display customer_id and its discounts availed where bonus remaining is greater than 500

select C_ID,discounts_availed

from membership account

where bonus_remaining>500;

5)Display department_id and department_name where no. of employees is greater than 20

select Department_id,Department_name

from Department

where No.ofemployees>20;

16BCP047

1) DISPLAY customer id tuple.

select c.id

from customer;

2) DISPLAY employees where their salary is greater than 5k.

select name

from employee

where salary>5k;

3) DISPLAY brand whose price is greater than 1k.

select brand

from product

where price>1k;

4)DISPLAY address of customer.

select address

from customer;

5)DISPLAY customer id which has pay amount greater than 10k.

select c_id

from payment

where amount>10k;

EXIST:-

1)return true and list department name which has employee who has salary greater then 90k

```
SELECT deparment_name
FROM department
WHERE EXISTS (select employee_id from employee where
department_id=department.department_id and salary > 90k)
```

2)return true and list customer name   who pays > 10k

```
select customer_name
from customer
where exist (select payment_id from payments where c_id=customer.c_id and amount > 10k)
```

ANY:-

1)RETURN TRUE AND LIST product name which has availibility > 100

```
SELECT Product_Name
FROM Product
WHERE Product_ID = ANY (SELECT Product_ID FROM product_maintainence WHERE
product_availability > 100)
```

2)LIST CUSTOMER NAME WHO AVAILED DISCOUNT

```
SELECT CUSTOMER_NAME
FROM CUSTOMER
WHERE C_ID=ANY(SELECT C_ID FROM maintainence WHERE discounts_availed > 5)
```

ALL:-

1) list product name which sold after 20 march 2014

```
select product_name
from product
where product_id=ALL(select product_id from sales where sales_date < 20-3-2014)
```

2)list department name which HAS ALL employee HAVE SALARY greater THEN 20K

```
select department_name
from department
where department_id=ALL(select department_id from employee where salary > 20k)
```

- 1) Find average salary of empoyees in each department--query

```
select avg(salary) as avg_sal
from Department_ID
group by Department_ID
```

- 2)Find departent with average salary greater then 20000--query

select *department_name, avg (salary) as avg salary*

from *Dpeartment*

group by *departmrnt_name*

having avg (*salary) > 20000;*

- 1.create a view to show all employees in deartment of security
- Create view security as
- Select employee_ID,Employee_name
- From employee natural join dpartmeent
- Where department_name="Security
- 2.create a view to show all wholesalers who sell products with product class electronics
- Create view electronics_wholesaers as
- Select wholesaer_name,Contact_no
- From wholesaler natural join product
- Where product_class="Electronics"

EXPLANATION OF

-DDL -SQL -DML -KEYS -DCL

What is sql?

SQL stands for Structured Query Language

- IBM developed the original version of SQL, originally called Sequel, as part of the system R project in the early 1970s
- SQL lets you define and manipulate databases

What is DDL?

DDL is stands for Data Definition Language

DDL provides commands for- :

- Defining relation schemas,
- deleting relations
- modifying relation schemas.

What is DML?

DML is short name of Data Manipulation Language

The SQL DML provides the ability to

- Query information from the database
- Insert tuples
- Delete tuples
- Modify tuples in the database.
- There are basically two types:

• Procedural DMLs require a user to specify *what data are needed and how to* get those data.

• Declarative DMLs (also referred to as nonprocedural DMLs) require a user tospecify *what data are needed without specifying how to get those data.*

What is DCL?

- A data control language (DCL) is a syntax similar to a computer programming language used to control access to data stored in a database . In particular, it is a component of Structured Query Language (SQL).
- Examples of DCL commands include:
- GRANT to allow specified users to perform specified tasks.
- REVOKE to cancel previously granted or denied permissions.

Keys

- Let *R denote the set of attributes in the schema of relation r. If we* say that a subset *K of R is a superkey for r , we are restricting consideration to* instances of relations *r in which no two distinct tuples have the same values on* all attributes in *K. That is, if t1 and t2 are in r and t1 = t2, then t1.K = t2.K.*
- If *K is a superkey, then so* is any superset of *K. We are often interested in superkeys for which no proper* subset is a superkey. Such minimal superkeys are called **candidate keys.**
- We shall use the term **primary key to denote a candidate key that is chosen**
- by the database designer as the principal means of identifying tuples within a
- relation.
- A relation, say *r1, may include among its attributes the primary key of another*
- relation, say *r2. This attribute is called a* ***foreign key from r1, referencing r2.***
- The relation *r1 is also called the* ***referencing relation of the foreign key dependency,***
- and *r2 is called the* ***referenced relation of the foreign key.***

AGGREGATE FUNCTIONS AND QUERIES

1)Product_ID for which Payment_ID is C11829

```
select Product_id
from sales
where
C_ID=(select C_ID
from payments
where payment_id='C11829');
```

2)Retrieve names and ids of employees who are not in cashier and cleaning department

select employee_id,employee_name

from employee

where department_id is not in (select department_id

from department where department='Cashier' and department='Cleaning'

Use of group by, Having, Order by features of SQL

1)Display name and contact no. of employees in each department whose salary is less than 5000

select name, contact_no

from employee

group by department_id

having salary<5000;

2)Display customer name in alphabetical order who bought product with product_id=10838

select customer_name

from customer natural join sales

where product_id=10838 order by customer_name;

Different types of Join operations, Exist, Any, All and relevant features of

SQL

1)return true and list department name which has employee who has salary less then 90k

SELECT department_name

FROM department

WHERE EXISTS (select employee_id from employee where

department_id=department.department_id and salary < 90000)

2)return true and list customer name   who pays > 10000

select customer_name

from customer

where exist (select payment_id from payments where c_id=customer.c_id and amount > 10000)

3)RETURN TRUE AND LIST product name which has availibility

SELECT Product_Name

FROM Product

WHERE Product_ID = ANY (SELECT Product_ID FROM product_maintenance WHERE product_availability ='YES')

4)list customer name who availed discount

SELECT CUSTOMER_NAME

FROM CUSTOMER

WHERE C_ID=ANY(SELECT C_ID FROM maintenance WHERE discounts_availed > 5)

Views

Whats view?

In database, a view is the result set of a stored query on the data, which the database users can query just as they would in a persistent database collection object. This pre-established query command is kept in the database dictionary. Unlike ordinary base tables in a relational database, a view does not form part of the physical schema: as a result set, it is a virtual table computed or collated dynamically from data in the database when access to that view is requested. Changes applied to the data in a relevant underlying table are reflected in the data shown in subsequent invocations of the view.

1.create a view to show all employees in deartment of security

Create view security as

Select employee_ID,Employee_name

From employee natural join dpartment

Where department_name="Security"

2.create a view to show all wholesalers who sell products with product class electronics

Create view electronics_wholesalers as

Select wholesaler_name,Contact_no

From wholesaler natural join product

Where product_class="Electronics"

Use of Transaction control commands, Commit, Rollback, Save point features of SQL

Commit command

Commit command is used to permanently save any transaction into database.

Following is Commit command's syntax,

Commit;

Rollback command

This command restores the database to last commited state. It is also use with savepoint command to jump to a savepoint in a transaction.

Following is Rollback command's syntax,

Rollback to< savepoint-name>;

Savepoint command

savepoint command is used to temporarily save a transaction so that you can rollback to that point whenever necessary.

Following is savepoint command's syntax,

Savepoint<savepoint-name>;

INSERT into Payments values(A93693,10000);

commit;

UPDATE payments set C_ID=A93682 where payment is 10000;

savepoint A;

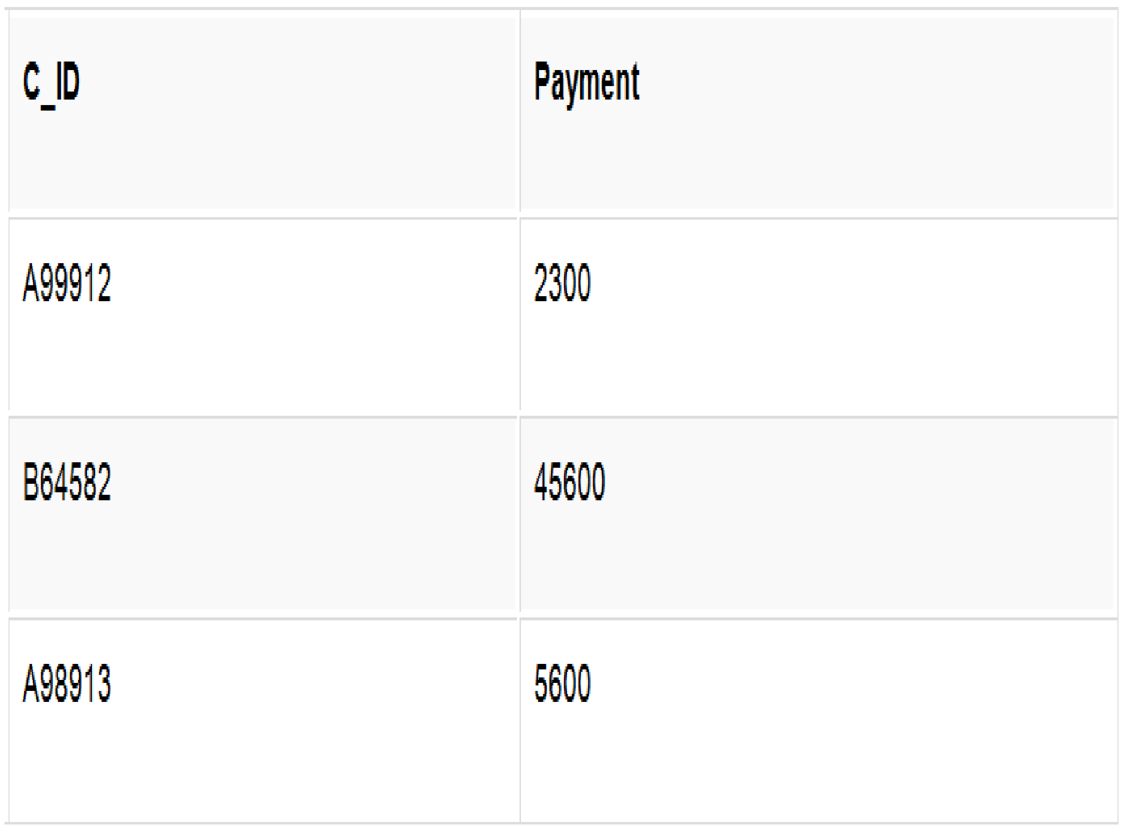
INSERT into Payments values(A94682,2200);

savepoint B;

INSERT into Payments values(A95582,12200);

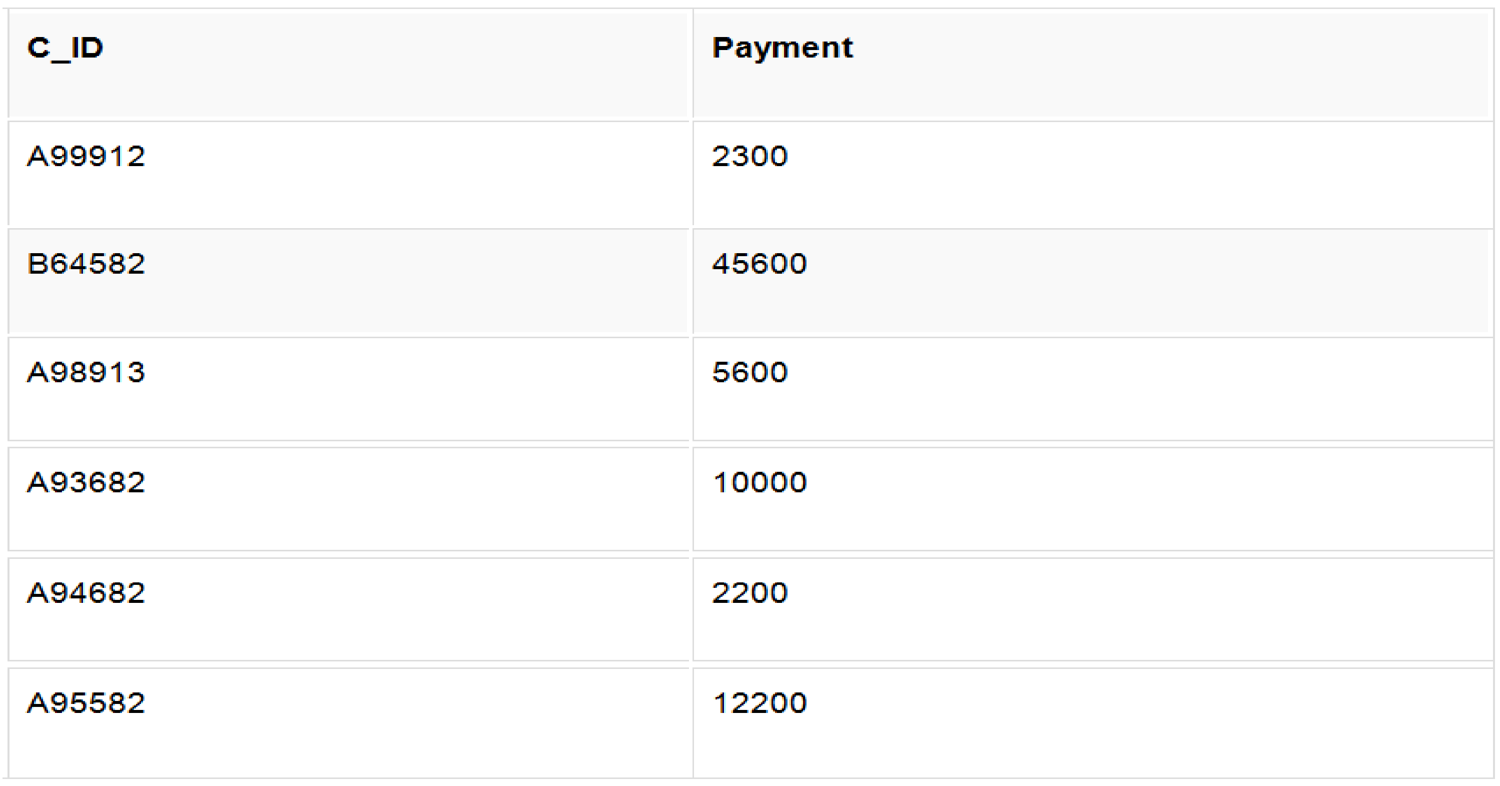
savepoint**C**;

INSERT into Payments values(A94692,21200);



C_ID	Payment
A99912	2300
B64582	45600
A98913	5600

- The resultant table will look like,

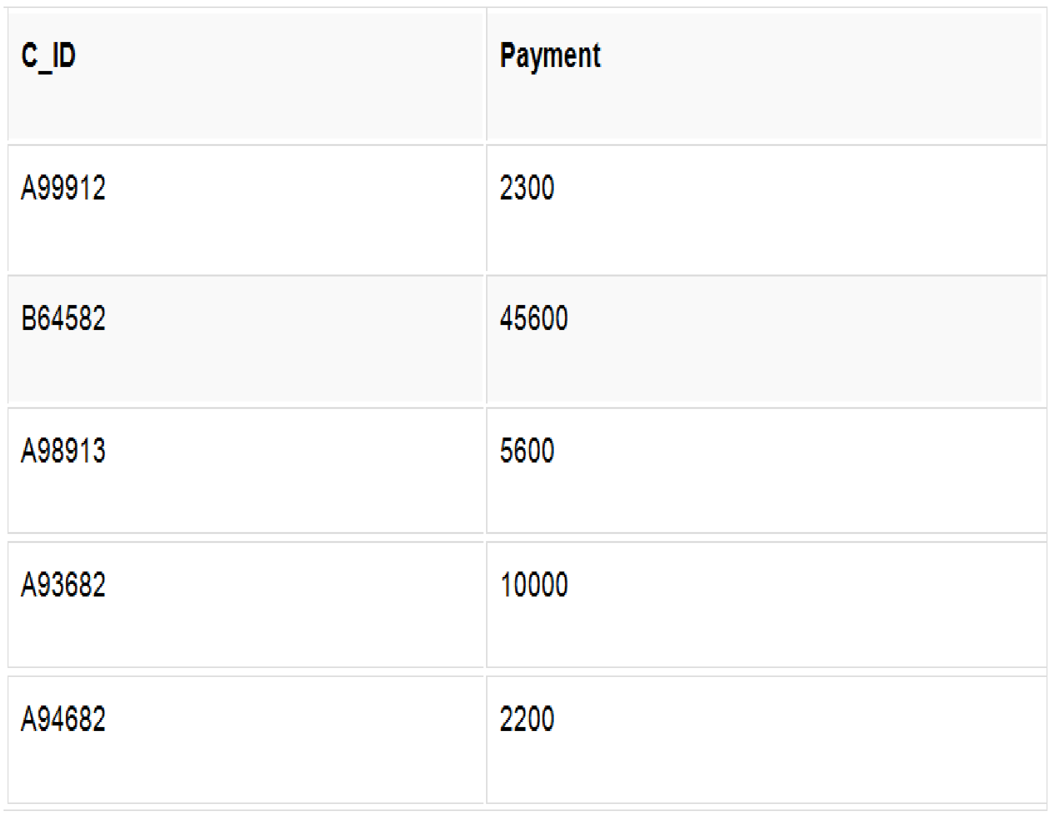


C_ID	Payment
A99912	2300
B64582	45600
A98913	5600
A93682	10000
A94682	2200
A95582	12200

- Now **rollback** to **savepoint B**

rollback to B;

SELECT * from class; The following will be the resultant table

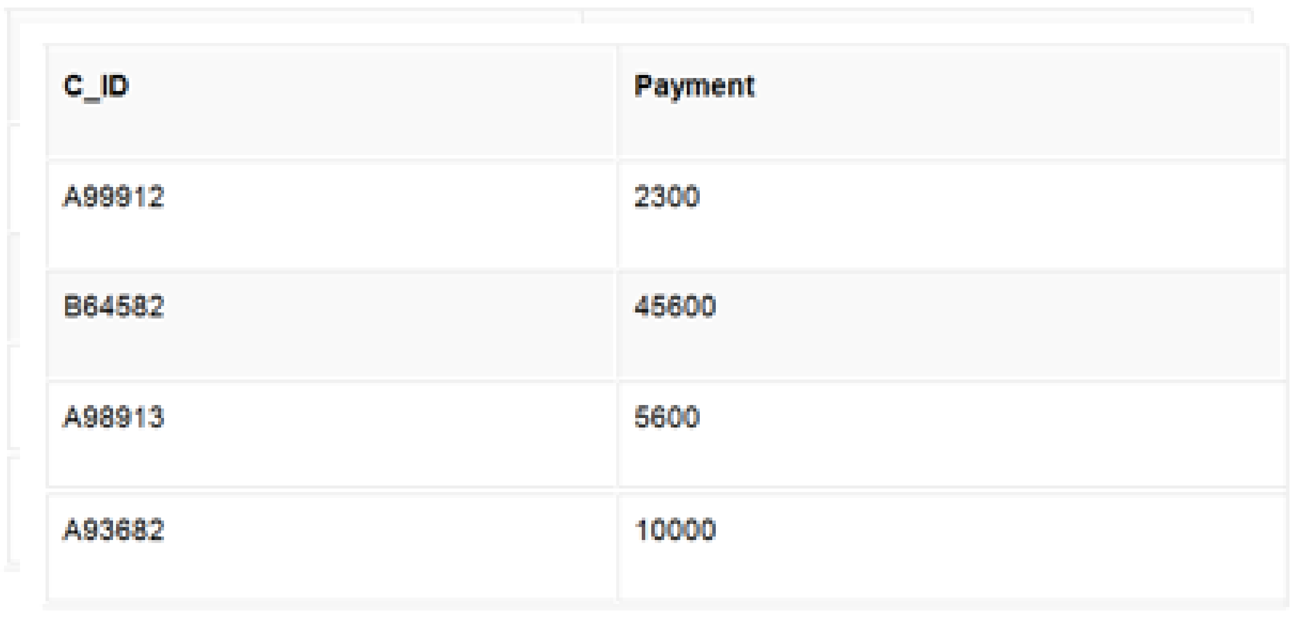


C_ID	Payment
A99912	2300
B64582	45600
A98913	5600
A93682	10000
A94682	2200

Now **rollback** to **savepoint A**

rollback to A;

SELECT * from class; Below is the resultant table



C_ID	Payment
A99912	2300
B64582	45600
A98913	5600
A93682	10000

Database Normalization

Table Customer:

- C_ID →Customer_name,Contact_no,Address

Minimal Cover

C_ID →Customer_name

C_ID →Address

C_ID →Contact_no

BCNF form

Candidate key-C_ID

Table Sales

- C_ID,Product_ID,Employee_ID,Sales_Date_time→Quantity

Minimal cover

C_ID,Product_ID,Employee_ID,Sales_Date_time→Quantity

BCNF form

Candidate key-C_ID,Product_ID,Employee_ID,Sales_Date

Membership Table

C_ID→discounts_availed,Bonus_remaining

Minimal cover

C_ID→Bonus_remaining

C_ID→discounts_availed

BCNF form

Candidate key-C_ID

Payments Table

Payment_ID→C_ID,amount

Minimal cover

Payment_ID→C_ID

Payment_ID→amount

BCNF form

candidate key-Payment_ID

Agency Table

Agency__ID→agency_name,Contact_no,Address

Minimal cover

Agency__ID→agency_name

Agency__ID→Contact_no

Agency__ID→Address

BCNF form

Candidate key-Agency_ID

Employee Table

Employee_ID→Emplouee_name,Contact_no,Salary,Agency_ID,Department_ID

Minimal cover

Employee_ID→Contact_no

Employee_ID→Salary

Employee_ID→Agency_ID

Employee_ID→Department_ID

Employee_ID→Employee_name

BCNF form

Candidate key-Empoyee_ID

Product Table

Product_ID→price, brand,product_class,wholesaler_ID,product_name

Minimal cover

Product_ID→product_name

Product_ID→wholesaler_ID

Product_ID→product_class

Product_ID→ brand

Product_ID→price,

BCNF form

Candidate key-Product_ID

Product_maintainence Table

Product_ID→Product_availability,stock_ordered

Minimal cover

Product_ID→stock_ordered

Product_ID→Product_availability

BCNF form

Candidate key-Product_ID

Wholesaler Table

Wholesaler_ID→Shop_name,Contact_no,Address

Minimal cover

Wholesaler_ID→Contact_no

Wholesaler_ID→Address

Wholesaler_ID→Shop_name

BCNF form

Candidate key-Wholesaler_ID

Department Table

Department_ID→Department_name

Department_name→deartment_ID

BCNF form

Candidate key-department_name,Department_ID

Introduction to Embedded SQL, PL SQL Concepts

What is Embedded SQL?

As a result, database applications are usually developed by combining capabilities of a high-level programming language with SQL.

The simplest approach is to embed SQL statements directly into the source code file(s) that will be used to create an application. This technique is referred to as embedded SQL programming.

sqlca.h – header file to be included.

High-level programming language compilers cannot interpret, SQL statements.

Hence , source code files containing Embedded SQL must be preprocessed before compiling.

Thus each SQL statement coded in a high-level programming language source code file must be prefixed with the keywords **EXEC SQL** and terminated with either a semicolon or the keywords **END_EXEC**.

Likewise, the Database Manager cannot work directly with high-level programming language variables.

Instead, it must use special variables known as **host variables** to move data between an application and a database.

Two types of Host variables:-

Input Host Variables – Transfer data to database

Output Host Variables – receives data from database

- Host variables are ordinary programming language variables.
- To be set apart, they must be defined within a special section known as a **declare section**.

```
EXEC SQL BEGIN DECLARE SECTION
Char Employee_id[7]
Double salary;
EXEC SQL END DECLARE SECTION
```

- Each host variable must be assigned a unique name even though declared in different declaration section.

Implementation of Cursors, Stored Procedures, Stored Function, Triggers

TRIGGERS

A database trigger is procedural code that is automatically executed in response to certain events on a particular table or view in a database. The trigger is mostly used for maintaining the integrity of the information on the database. For example, when a new record (representing a new worker) is added to the employees table, new records should also be created in the tables of the taxes, vacations and salaries. Triggers can also be used to log historical data, for example to keep track of employees' previous salaries.

The four main types of triggers are:

Row Level Trigger: This gets executed before or after any column value of a row changes

Column Level Trigger: This gets executed before or after the specified column changes

For Each Row Type: This trigger gets executed once for each row of the result set affected by an insert/update/delete

For Each Statement Type: This trigger gets executed only once for the entire result set, but also fires each time the statement is executed.

Schema-level triggers:-

- After Creation
- Before Alter
- After Alter
- Before Drop
- After Drop
- Before Insert

SYNTAX:-

```
CREATE TRIGGER name { BEFORE | AFTER } { event [ OR ... ] }
   ON TABLE [ FOR [ EACH ] { ROW | STATEMENT } ]
   EXECUTE PROCEDURE funcname ( arguments )
```

CURSOR

- a database cursor is a control structure that enables traversal over the records in a database. Cursors facilitate subsequent processing in conjunction with the traversal, such as retrieval, addition and removal of database records. The database cursor characteristic of traversal makes cursors akin to the programming language concept of iterator.
- Cursors are used by database programmers to process individual rows returned by database system queries. Cursors enable manipulation of whole result sets at once. In this scenario, a cursor enables the rows in a result set to be processed sequentially.
- In SQL procedures, a cursor makes it possible to define a result set (a set of data rows) and perform complex logic on a row by row basis. By using the same mechanics, a SQL procedure can also define a result set and return it directly to the caller of the SQL procedure or to a client application.
- A cursor can be viewed as a pointer to one row in a set of rows. The cursor can only reference one row at a time, but can move to other rows of the result set as needed.
- TO USE CURSOR:-
- Declare a cursor that defines a result set.
- Open the cursor to establish the result set.
- Fetch the data into local variables as needed from the cursor, one row at a time.
- Close the cursor when done
- A programmer makes a cursor known to the DBMS by using a DECLARE ... CURSOR statement and assigning the cursor a (compulsory) name:
- 1.DECLARE cursor_name CURSOR IS SELECT ... FROM ...
- Before code can access the data, it must open the cursor with the OPEN statement. Directly following a successful opening, the cursor is positioned before the first row in the result set.
- 2.OPEN cursor_name
- Programs position cursors on a specific row in the result set with the FETCH statement. A fetch operation transfers the data of the row into the application.
- 3.FETCH cursor_name INTO ...

Procedures in PL/SQL

- A stored procedure in PL/SQL is nothing but a series of declarative SQL statements which can be stored inside the database catalog. A procedure can be thought of as a function or a method. They can be invoked through triggers, other procedures, or applications on Java, PHP etc. Some MySQL versions even support recursive procedures, i.e. procedures which call themselves in their body.

Advantage

- They result in performance improvement of the application. If a procedure is being called frequently in an application in a single connection, then the compiled version of the procedure is delivered.
- They reduce the traffic between the database and the application, since the lengthy statements are already fed into the database and need not be sent again and again via the application.
- They add to code reusability, similar to how functions and methods work in other languages such as C/C++ and Java.

Disadvantage

- Stored procedures can cause a lot of memory usage. The database administrator should decide an upper bound as to how many stored procedures are feasible for a particular application.
- MySQL does not provide the functionality of debugging the stored procedures.

SYNTAX FOR STORED FUNCTION:-

```
--- Author: <Author,,Name>

-- Create date: <Create Date,,>

-- Description:<Description,,>

-- ============================================

Create       Proc  SP_Name  (

Param1 DataType,  param2 DataType,  Param3 DataType,

Paramn DataType

)

As

Begin

Body Of Store Procedure

End
```

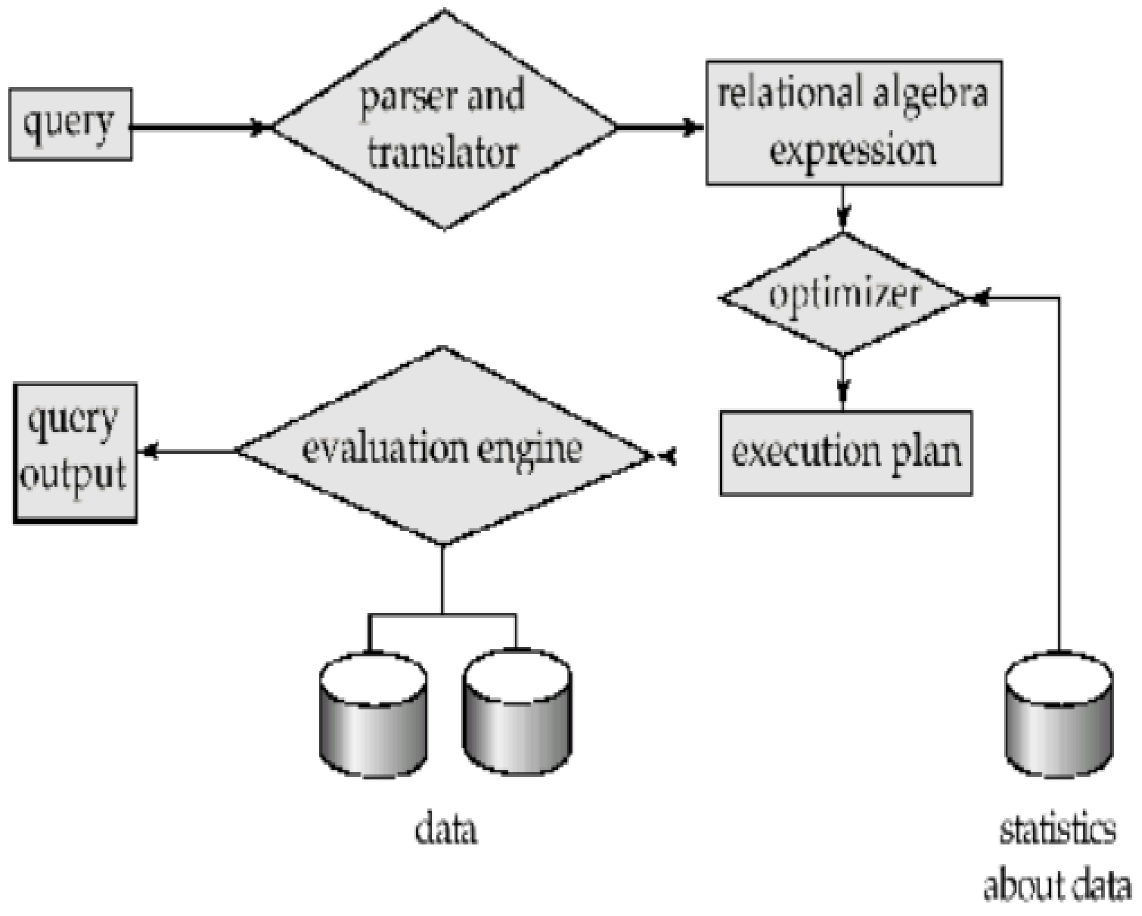
Analysis of query cost, creating indices and evaluating their effect on query Evaluation Plans and cost

Measures of Query Cost

- Cost is generally measured as total elapsed time for answering query.
- Many factors contribute to time cost
- *disk accesses, CPU*, or even network *communication*
- Typically disk access is the predominant cost, and is also relatively easy to estimate Measured by taking into account
- Number of blocks written * average-block-write-cost
- Cost to write a block is greater than cost to read a block
- ⁻data is read back after being written to ensure that the write was successful
- Costs depends on the size of the buffer in main memory
    - Having more memory reduces need for disk access
    - Amount of real memory available to buffer depends on other concurrent OS processes, and hard to determine ahead of actual execution
    - We often use worst case estimates, assuming only the minimum amount of memory needed for the operation is available
- Real systems take CPU cost into account, differentiate between sequential and random I/O, and take buffer size into account
- We do not include cost to writing output to disk in our cost formulae

Basic Steps in Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



- Parsing and translation
    - translate the query into its internal form.This is then translated into relational algebra.
    - Parser checks syntax, verifies relations
- Evaluation
    - The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query.

CREATING INDEXING:-

- Indexing is a data structure technique used to find data more quickly and efficiently in a table.
- An index is a small table having only two columns. The first column contains a copy of the primary or candidate key of a table and the second column contains a set of pointers holding the address of the disk block where that particular key value can be found. The users cannot see the indexes, they are just used to speed up searches/queries.
- Indexes are of 3 types:
- . Primary indexes : In primary index, there is a one-to-one relationship between the entries in the index table and the records in the main table. Primary index is defined on an ordered data file. The data file is ordered on a key field. The key field is generally the primary key of the relation.
- 2. Secondary indexes : Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values .While creating the index, generally the index table is kept in the primary memory (RAM) and the main table, because of its size is kept in the secondary memory (Hard Disk).
- 3. Clustering indexes : Clustering index is defined on an ordered data file. The data file is ordered on a non-key field e.g. if we are asked to create an index on a non-unique key, such as Dept-id. There could be several employees in each department. Here we use a clustering index, where all employees belonging to the same Dept-id are considered to be within a single cluster, and the index pointers point to the cluster as a whole.