Relational Algebra: Selection and Projection, Set Operations, Renaming, Joins, Division, More Examples of Algebra Queries.

SQL: Queries, Constraints, Triggers: The Form of a Basic SQL Query, UNION, INTERSECT and EXCEPT, Nested Queries, Aggregate Operators, Null Values, Complex Integrity Constraints in SQL, Triggers and Active Databases, Designing Active Databases.

In DBMS there are two types of query languages. Query is a condition or statement it is used to retrieve the data from the database. Query language is also called high level language. Query languages are two types

1. Procedural query language.
2. Non Procedural query language.

Procedural Query Language:

The user used to specific procedure to retrieve the data from database called procedural query language.

Ex: Relational algebra.

Non-Procedural Query Language:

The user not used to specific procedure to retrieve the data from database is called non procedural query language.

Ex: Relational calculus.

Relational Algebra:

❖ Relational algebra is a procedural query language. Each relational query describes a step-by-step procedure for computing the desired answer, based on the order in which operators are applied in the query .It consisting of set of operators that are used to take one or more relations as input and produce new relation as output.

❖ Basic operators are

1. Unary operator.
2. Binary operator.

Unary operator:

Unary operator applied operation on single relations.

Ex: selection, projection, renaming etc…

Binary operator:

Binary operator applied operation on a two relations.

Ex: set operators, joins, division etc…

****

Selection:

Selection Operator (σ) is a unary operator in relational algebra that performs a selection operation. Selection operation retrieving the data from database using records or tuples.

Syntax: $\sigma_{<\text{selection condition}>}(R)$

Example: $\sigma_{<\text{rating}} > 7(\text{sailors})$.

- ❖ Selection operator can perform using logical (and, or, not) and comparison operators(<,>,<=,>=,==,!=).
- ❖ Selection operator can retrieve the selected records.
- ❖ Selection operator always selects the entire tuple. It can not select a section or part of a tuple.
- ❖ Selection operator can also run with projection.
- ❖ Degree of the relation from a selection operation is same as degree of the input relation.
- ❖ Selection operator performs horizontal partitioning of the relation.

Projection:

The **projection operator** π is one of the unary operators in relational algebra (RA) and is used to project columns from a relation. It can select specific columns from a given relation and hide all the other columns.

Syntax: $\pi_{attribute\ list}(R)$

Example: π sname,age**(sailors).**

- The projection operator is also known as vertical partitioning.
- The degree of output relation (number of columns present) is equal to the number of attributes mentioned in the attribute list.
- Projection operator automatically removes all the duplicates while projecting the output relation.
- Projection operator does not obey commutative property.
- Projection operator performs vertical partitioning of the relation.

****

Set Operations:

SQL set operators are used to combine the results obtained from two or more queries into a single result. The queries which contain two or more sub queries are known as compounded queries.

There are four major types of SQL operators, namely:

- Union
- Union all
- Intersect
- Minus

1. **Union:**

- The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
- In the union operation, all the number of data type and columns must be same in both the tables on which UNION operation is being applied.
- The union operation eliminates the duplicate rows from its result set.

Example: **π** id, name, salary**(employee) U π** id, name, salary**(emp);**

Example: Select id, name. salary from employee1

union

Select id, name. salary from employee2;

2. **Union all:**

Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

Example: **π** id, name, salary**(employee) U_{all} π** id, name, salary**(emp);**

Example: Select id, name. salary from employee1

Union all

Select id, name. salary from employee2;

3. **Intersect:**
   - ❖ It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
   - ❖ In the Intersect operation, the number of data type and columns must be the same.

It has no duplicates and it arranges the data in ascending order by default.

Example: π id, name, salary**(employee)** $\cap$ π id, name, salary**(emp);**
Example: Select id, name. salary from employee1
Intersect
Select id, name. salary from employee2;

4. **Minus**
   - ❖ It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.

It has no duplicates and data arranged in ascending order by default.

Example: π id, name, salary**(employee)** - π id, name, salary**(emp);**
Example: Select id, name. salary from employee1
Minus
Select id, name. salary from employee2

****

Renaming:

Rename is used to change the object name /table name in database name. The **rename operator** ρ is one of the unary operators in relational algebra and is used to rename relations in a DBMS.

Syntax: ρ *new_relation_name***(new_*relational name*).**
Example : ρ *employee* **(*emp*).**
Example: Rename **employee** to **emp.**
Renaming can be used by three methods, which are as follows −

- ❖ Changing name of the relation.
  Example: Rename **employee** to **emp**
- ❖ Changing name of the attribute.
  Example: ρnewname,newbranch(∏name,branch(**student**))
- ❖ Changing both.

****

Joins:

Joins are used to display multiple data types of data from multiple tables. Joins are two types

1. Physical join
2. Logical join

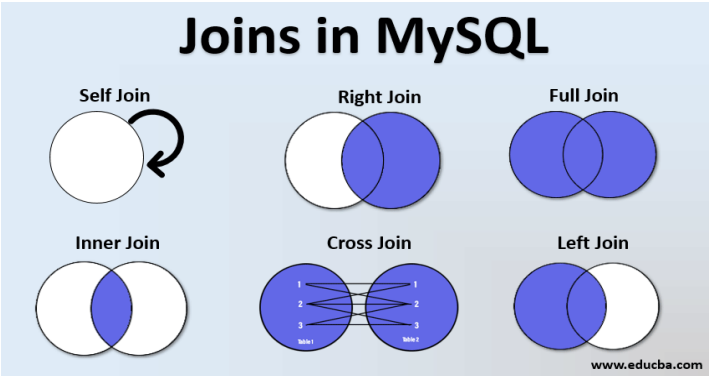
Physical join: physical join is used to establish the connection physically on tables using referential constraints/integrity constraints.
Logical join: Logical join is used to establish the connection logically on tables. Logical join two types

1. **Combined data**: It can be joined using set operators.(same data type of data and same number columns)

These are

- Union
- Union all
- Intersect
- Minus

2. **Combinational data**: It can be joined using join operation.(different data type of data and different number columns).These are
   - Cross join
   - Equal join
   - Inner join
   - Self join
   - Outer join
     - Left outer.
     - Right outer.
     - Full outer.



Cross join:

It will display combination of data from multiple tables. In this join, each value in the first table is mapped with all values in the second table. It will display all possible combinations of data from multiple tables.

Example:

select eid, ename, dname from employee, depart;

Equal Join:

A cross join is known as equi join if we specify join condition using '=' operator. It will display only matched data from all tables. A condition is known as join condition if it is specified between primary key of one table and foriegn key of other table.

Example: select eid,ename,dname from employee,depart
where employee.deptid=depart.deptid**;**

Inner Join:

The INNER JOIN keyword selects records that have matching values in both tables. The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns.

Example: select employee.eid, depart.deptid from employee
inner join
depart on (depart.deptid=employee.deptid);

Outer join:

These are used to display all data from one table and only matched data from other table.

Types of outer joins:

1) **Left outer join** : left join Display all the data from left table and only matched data from right table.

Example: select employee.eid, depart.deptid from employee
left outer join
depart on (depart.deptid=employee.deptid);

2) Right outer join : Right join Display complete data from right table and only matched data from left table.

Example: select employee.eid, depart.deptid from employee
right outer join
depart on (depart.deptid=employee.deptid)

3) Full outer join : Full join: Display --matched data from both the tables
--unmatched data from left table
--unmatched data from right table

Example: select employee.eid, depart.deptid from employee
full outer join
depart on (depart.deptid=employee.deptid);

Self Join: A table which is joined itself is known as self join. In this case we can use alias names for single table. Here the alias names are temporary.

Ex:

```
select e1.*
from employee e1, employee e2
where e1.ename='rekha'
and
e1.deptid=e2.deptid;
```

****

Division:

The DIVISION operation is a binary relational operation that divides one set of rows into another set of rows based on specified conditions.

- It is similar to a JOIN operation, but the resulting table contains only the rows that belong to the first set and satisfy the division condition.
- The division operator is used for queries which involve the 'all'.R1 ÷ R2 = tuples of R1 associated with all tuples of R2.

R

ColA	ColB
F	1
F	2
F	3
E	1
E	3
S	1
S	2

S

ColB
1
2

$R \div S =$

ColA
F
S

Example: select *from customers
Whereexists
(select * from orders where customers.customer_id = orders.customer_id);

****

More Examples of Algebra Queries.

We use the Sailors, Reserves, and Boats schema for all our examples

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance $B1$ of Boats

1. Find the names and ages of sailors who are rating above 7
   ∏ sname,age **σ_{rating} > 7(sailors).**
2. Find the names of sailors who have reserved 103 boat
   ∏ sname**σ <bid=103> (sailors X Reserves).**
3. Find the names of sailors who ha've reserved a red boat
   ∏ sname**σ <color='red' (Boats) > (sailors X Reserves).**
4. Find the colors of boats reserved by Lubber
   ∏ colors**σ <sname='Lubber' (sailors) > (sailors X Reserves X Boats)**
5. Find the names of sailors who have reserved at least one boat
   ∏ sname **(sailors X Reserves).**
6. Find the names of sailors who have reserved a red and green boat
   ∏ sname**σ <color='red' (Boats) U color='green' (Boats) > (sailors X Reserves X Boats).**
7. Find the sids of sailors with age over 20 who have not Reserved a Red boat
   ∏ sid <**σ (age=20 (Sailors)) - σ (color='red' (Boats)) > (sailors X Reserves X Boats).**
8. Find the names of sailors 'Who have reserved all boats
   ∏ sname **(sailors X Reserves).**

****

The Form of a Basic SQL Query:

Structured Query Language (SQL) is the most widely used relational database language. A conceptual evaluation strategy is a way to evaluate the query that is intended to be easy to understand rather than efficient. A DBMS would typically execute a query in a different and more efficient way.

The basic form of an SQL query is follows:

Example:

SELECT [DISTINCT]
select-list FROM from-list
WHERE qualification

- Select statement can be used retrieve all the information table.

  Syntax:

  Select *from table where condition.

  Example:

  Select * from student where city ='Kakinada';

- Select statement can be use retrieve selected attributes from the table

  Syntax:

  Select attribute1, attribute2,….from table where condition.

  Example:

  Select id, name, course from student;

- Select statement can be use retrieve selected records from the table

  Syntax:

  Select attribute1, attribute2,….from table where condition.

  Example:

  Select id, name, course from student where city ='Kakinada';

****

UNION, INTERSECT:

Union

- The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
- In the union operation, all the number of data type and columns must be same in both the tables on which UNION operation is being applied.
- The union operation eliminates the duplicate rows from its result set.

Example: π id, name, salary**(employee) U** π id, name, salary**(emp);**

Example: Select id, name. salary from employee1
union
Select id, name. salary from employee2;

Union all:

Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

Example: π id, name, salary**(employee)** $\mathbf{U_{all}}$ π id, name, salary**(emp);**

Example: Select id, name. salary from employee1
Union all
Select id, name. salary from employee2;

Intersect:

- It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
- In the Intersect operation, the number of data type and columns must be the same.

It has no duplicates and it arranges the data in ascending order by default.

Example: π id, name, salary**(employee)** $\cap$ π id, name, salary**(emp);**
Example: Select id, name. salary from employee1
Intersect
Select id, name. salary from employee2;

****

EXCEPT:

The **EXCEPT** operator in SQL is used to retrieve the unique records that exist in the first table, not the common records of both tables. This operator acts as the opposite of the SQL UNION operator.

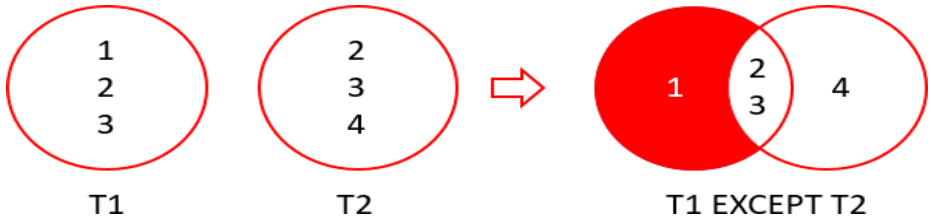
Rules for SQL EXCEPT:

- In all SELECT statements, the number of columns and orders in the tables must be the same.
- The corresponding column's data types should be either the same or compatible.
- The fields in the respective columns of two SELECT statements cannot be the same.

Syntax:

```
SELECT column_lists from table_name1
EXCEPT
SELECT column_lists from table_name2;
```

Example:



Example:

```
select name, hobby, age
from students
where age between 20 and 30

except

select name, hobby, age
from associates
where age between 20 and 30
```

****

Nested Queries:

A nested query in SQL contains a query inside another query. The outer query will use the result of the inner query. For instance, a nested query can have two SELECT statements, one on the inner query and the other on the outer query.

```
SELECT ID, NAME FROM EMPLOYEES
WHERE ID IN (SELECT EMPLOYEE_ID FROM AWARDS)
```

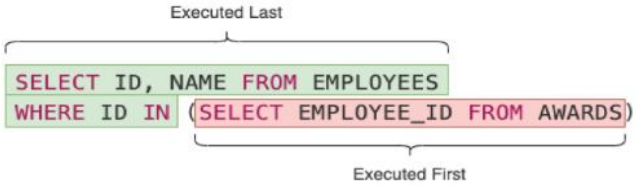
Nested Query

Nested queries in SQL can be classified into two different types:

- Independent Nested Queries.
- Co-related Nested Queries.

- **Independent Nested Queries:**
  In independent nested queries, the execution order is from the innermost query to the outer query. An outer query won't be executed until its inner query completes its execution. The outer query uses the result of the inner query. Operators such as **IN**, **NOT IN**, **ALL**, and **ANY** are used to write independent nested queries.

- The **IN** operator checks if a column value in the outer query's result is **present** in the inner query's result. The **NOT IN** operator checks if a column value in the outer query's result is **not present** in the inner query's result.
- The **ALL** operator compares a value of the outer query's result with **all the values** of the inner query's result and returns the row if it matches all the values.
- The **ANY** operator compares a value of the outer query's result with all the inner query's result values and returns the row if there is a match with **any value**.



Example:

```
SELECT eid, ename FROM employee
WHERE eid IN (SELECT eid FROM awards);
```

Example:

```
SELECT eid, ename FROM employee
WHERE eid NOT IN (SELECT eid FROM awards);
```

Example:

```
SELECT * FROM employee
WHERE desg = 'Asst.prof'
AND salary > ALL (
SELECT salary FROM employee WHERE desg = 'HOD');
```

Example:

```
SELECT * FROM employee
WHERE desg = 'Asst.prof'
AND salary > ANY (
SELECT salary FROM employee WHERE desg = 'HOD');
```

- **Co-related Nested Queries:**
  In co-related nested queries, the inner query uses the values from the outer query to execute the inner query for every row processed by the outer query. The co-related nested queries run slowly because the inner query is executed for every row of the outer query's result.

Example:

```
SELECT * FROM employee emp1
WHERE salary > (
SELECT AVG(salary)
FROM employee emp2
WHERE emp1.desg = emp2.desg);
```

****

Aggregate Operators:

Aggregate Functions in DBMS: Aggregate functions are those functions in the DBMS which takes the values of multiple rows of a single column and then form a single value by using a query. These functions allow the user to summarizing the data.

In Database Management System, following are the five aggregate functions:

1. AVG
2. COUNT
3. SUM
4. MIN
5. MAX

- **AVG Function**

  This function takes the values from the given column and then returns the average of the values.

  Example:

  select avg(pcost) from product;

- **COUNT Function**

  This aggregate function returns the total number of values in the specified column. This function can work on any type of data, i.e., numeric as well as non-numeric. This function does not count the NULL values.

  Example:

  select count(name) from student;

- **SUM Function**

  This aggregate function sums all the non-NULL values of the given column. Like the AVG function, this function also works only on the numeric data.

  Example:

  select sum(pcost) from product;

- **MAX Function**

  This function returns the value, which is maximum from the specified column.

  Example:

  select max(pcost) from product;

- **MIN Function**

  This function returns the value, which is minimum from the specified column.

  Example:

  select min(pcost) from product;

****

Null Values:

Null values are special values in DBMS that represent values which are unknown and are always different from zero value. For example age of a particular student is not available in the age column of student table then it is represented as null but not as zero.

SQL provides special operators and functions to deal with data involving null values

- **IS NULL operator**:

  All operations upon null values present in the table must be done using this 'is null' operator **.**we cannot compare null value using the assignment operator

  Example:

  select * from emp
  where comm **is null**

- **IS NOT NULL operator:**

  All operations upon not null values present in the table must be done using this 'is not null' operator **.**

  Example:

  select * from emp
  where comm **is not null**

- **NOT NULL Constraint :**

  Not all constraints prevents a column to contain null values Once **not null is applied to a particular column, you cannot enter null values to that column** and restricted to maintain only some proper value other than null.A **not-null constraint cannot be applied at table level**

  Example:

```
create table student
(
  id   int  not null,
  name varchar (20)    not null,
  age  int     not null,
  address  char (25) ,
  salary   decimal (18, 2),
  primary key (id));
```

- **NVL() Function:**

  Using NVL function you can substitute a value in the place of NULL values.

  Example:

```
select nvl (comm, 500) from employees
where     salary>1000;
```

****

Complex Integrity Constraints in SQL:

We can specify complex constraints over a single table using table constraints, which have the form CHECK conditional-expression.

We use the Sailors, Reserves, and Boats schema for all our examples

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance $B1$ of Boats

1. Find the names and ages of sailors who are rating above 7
   SELECT S.sid, S.sname, S.rating, S.age
   FROM Sailors AS S
   WHERE S.rating > 7
2. Find the names of sailors who have reserved 103 boat
   SELECT S.sname FROM Sailors S, Reserves R
   WHERE S.sid=R.sid AND R.bid= 103;
3. Find the names of sailors who ha've reserved a red boat
   SELECT DISTINCT s.sname FROM Sailors S, Reserves R, Boats B
   WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
4. Find the colors of boats reserved by Lubber
   SELECT B.color
   FROM Sailors S, Reserves R, Boats B
   WHERE S.sname='Lubber' AND S.sid=R.sid AND R.bid = B.bid
5. Find the names of sailors who have reserved at least one boat
   SELECT S.sname
   FROM Sailors S, Reserves R
   WHERE S.sid = R.sid
6. Find the names of sailors who have reserved a red and green boat
   SELECT S.sname
   FROM Sailors S, Reserves R, Boats B
   WHERE S.sid = R.sid AND R.bid = B.bid
   AND (B.color = 'red' OR B.color = 'green')
7. Find the sids of sailors with age over 20 who have not Reserved a Red boat
   SELECT S.sname
   FROM Sailors S, Reserves R, Boats B
   WHERE B.color != 'red' and B.bid = R.bid and S.sid = R.sid and S.sid > 20
8. Find the names of sailors 'Who have reserved all boats
   SELECT S.sname FROM Sailors S, Reserves R
   WHERE S.sid=R.sid .

Triggers:

A trigger is a procedure that is automatically invoked by the DBMS in response to specified changes to the database, and is typically specified by the DBA. A database that has a set of associated triggers is called an active database.

A trigger description contains three parts:

- Event: A change to the database that activates the trigger. ..
- Condition: A query or test that is run when the trigger is activated. ..
- Action: A procedure that is executed when the trigger is activated and its condition is true.

Use of trigger:

Triggers may be used for any of the following reasons −

- To implement any complex business rule, that cannot be implemented using integrity constraints.
- Triggers will be used to audit the process. For example, to keep track of changes made to a table.
- Trigger is used to perform automatic action when another concerned action takes place.

Types of triggers:

The different types of triggers are explained below −

- **Statement level trigger** − It is fired only once for DML statement irrespective of number of rows affected by statement. Statement-level triggers are the default type of trigger.

- **Before-triggers** − At the time of defining a trigger we can specify whether the trigger is to be fired before a command like INSERT, DELETE, or UPDATE is executed or after the command is executed. Before triggers are automatically used to check the validity of data before the action is performed. For instance, we can use before trigger to prevent deletion of rows if deletion should not be allowed in a given case.
- **After-triggers** − It is used after the triggering action is completed. For example, if the trigger is associated with the INSERT command then it is fired after the row is inserted into the table.
- **Row-level triggers** − It is fired for each row that is affected by DML command. For example, if an UPDATE command updates 150 rows then a row-level trigger is fired 150 times whereas a statement-level trigger is fired only for once.

Create database trigger

To create a database trigger, we use the CREATE TRIGGER command. The details to be given at the time of creating a trigger are as follows −

- Name of the trigger.
- Table to be associated with.
- When trigger is to be fired: before or after.
- Command that invokes the trigger- UPDATE, DELETE, or INSERT.
- Whether row-level triggers or not.
- Condition to filter rows.

Syntax:

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
ON table_name
[FOR EACH ROW]
WHEN (condition)
[trigger_body]
```

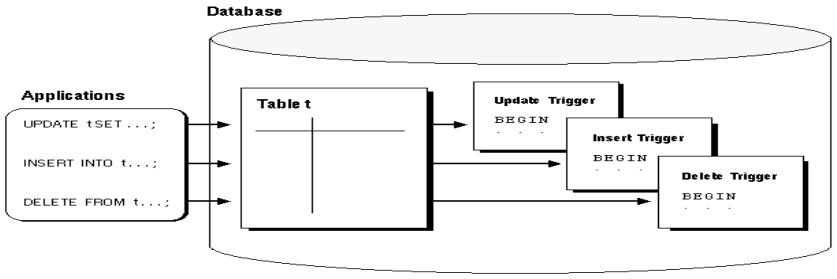
Example:

```
create trigger student_name
after INSERT
on student
for each row
BEGIN
  UPDATE student set full_name = first_name || ' ' || last_name;
END;
```

****

Active Databases:

An active Database is **a database consisting of a set of triggers**. An active database is an automated interface that performs certain functions that are dependent on specific inputs of information. Programmers and administrators can manipulate active database systems to execute transactions according to predefined relationships.



Features of Active Database:

- It possesses all the concepts of a conventional database i.e. data modeling facilities, query language, etc.
- It supports all the functions of a traditional database like data definition, data manipulation, storage management, etc.
- It supports the definition and management of ECA rules.
- It detects event occurrence.
- It must be able to evaluate conditions and execute actions.

Advantages of Active Database:

- It enhances traditional database functionalities with powerful rule processing capabilities.
- Enable a uniform and centralized description of the business rules relevant to the information system.
- Avoids redundancy of checking and repair operations.
- A suitable platform for building a large and efficient knowledge base and expert systems.

Designing Active Databases

Triggers offer a powerful mechanism for dealing with changes to a database, but they must be used with caution. The effect of a collection of triggers can be very complex, and maintaining an active database can become very difficult.

- **Why Triggers Can Be Hard to Understand** :
  In an active database system, when the DBMS is about to execute a statement that modifies the databa.se, it checks whether some trigger is activated by the statement. If so, the DBMS processes the trigger by evaluating its condition part, and then (if the condition evaluates to true) executing its action part. If a statement activates more than one trigger, the DBMS typically processes all of them.
- **Constraints versus Triggers :**
  A common use of triggers is to maintain database consistency, and in such cases, we should always consider whether using an integrity constraint (e.g., a foreign key constraint) achieves the same goals. The meaning of a constraint is not defined operationally, unlike the effect of a trigger. This property makes a constraint easier to understand, and also gives the DBMS more opportunities to optimize execution.
- **Other Uses of Triggers** :Triggers can alert users to unusual events ,For example, we may want to check whether a customer placing an order h&s made enough purchases in the past month to qualify for an additional discount;

****