CHAPTER-4

The Relation Algebra implementation using SQL

❖ Introduction

- Relational databases are the most widely used databases in commercial applications like as banking system, railway-reservation system, etc.
- Relational databases are based on the relational model.

❖ Historical Perspective of Relational Model

- A relational data model was first introduced by Dr. Edgar F. Codd of IBM Research in 1970.
- He represented the overview of the relational model in his paper entitled "A Relational Model of Data for Large Shared Data Banks"
- Popular commercial RDBMS for large databases include Oracle, Microsoft SQL Server, Sybase SQL Server, and IBM's DB2.

❖ Structure of Relational Databases

- **Domain :**
  - A domain of an attribute is a set of permitted values for that attribute
  - For example, if the bank is organized in three branches, named 'vvn', 'ksad' and 'anand', then, the domain of attribute *bname* is {'vvn', 'ksad', 'anand'}.
  - A value for a branch name must be one of these three values in any relation.

Attribute	Domain Name	Type	Description	Definition
ano	Account number	String	Set of strings starting from 'A' followed by unique number.	{str | str is a string as per description}
bal	Balance	Number	Possible values of balance for a bank account.	{n | n ≥ 0 }
bname	Branch name	String	Set of all possible branch names.	{'anand', 'ksad', 'vvn'}

  - Similarly, a domain for column *ano* in Account table is a set of strings starting from 'A followed by unique numerical value, such as {A01, A02, A03, A04, A05}.
  - A domain is *atomic* if elements of domain are considered to be indivisible units.
  - For example, a domain for *bname* is an atomic domain.
  - A domain for some attribute, say *address* may not be atomic, as address can be divided into sub-parts such as society, city, and pincode.
  - A domain of an attribute should be atomic
  - An atomic domain is preferable rather than a non-atomic domain.

➢ Keys of Relations

- Tuples in a relational model represents individual entities or relationships.
- Tuples are distinguished based on data values stored
- No any two tuples in a relation should have exactly the same values for all attributes.
- There cannot be two identical tuples in a relation.
- Several *keys* have been defined, as illustrated below.
  - Super Key
  - Relation Key
  - Candidate Key
  - Primary Key
  - Alternate Key
  - Foreign Key

- **Super Key**
  - ♦ "A super key is a set of attributes which contains key."
  - ♦ For example, consider a relation student with attributes en_no, name, address, and contact_no.
  - ♦ In this { en_no *} and {name, address}* attribute can distinguish each tuple from another.
  - ♦ So, { en_no *} and {name, address}* are key attributes for student relation and by combining any non-key attribute to this we form a super key.

- **Relation Key**
  - ♦ "A super key for which no subset is a super key is called a relation key."
  - ♦ In other words, relation key is a minimal super key.
  - ♦ A relation key is also referred as 'key' only.
  - ♦ A relation key is sufficient to identify each and every tuple uniquely within a relation.
  - ♦ A relation key cannot have null or duplicate values, as well as it does not contain any redundant attribute.
  - ♦ There can be more than one relation keys for the same relation. For example, for student relation, *{*en_no *}* as well as *{name, address}* are relation keys.
- **Candidate Key**
  - ♦ "minimal set of attributes which uniquely identifies a tuple in a relation is called candidate key"
  - ♦ For example, for student relation, *{*en_no *}* as well as *{name, address}* are relation keys. And so, they each are called candidate keys.

- **Primary Key**
  - "A primary key is a set of one or more attributes that allows to identify each tuple *uniquely*
  - Primary key can't hold null.
  - Each table have single primary key.
  - For example, from two candidate key *{*en_no *}* and *{name,address}*.if database designers choose candidate key en_no to identify all students uniquely, then, en_no is a primary key of the relation student.

- **Alternate Key**
  - "An alternate key is a candidate key that is **not** chosen by the database designer to identify tuples uniquely in a relation."
  - For example, if database designers choose candidate key { en_no *}* to identify all students uniquely, then, another candidate key *{name, address}* is an alternate key.

- **Foreign Key**
  - "Primary key and foreign key is used to connect two or more table."
  - Foreign key is one or more attribute in a relation is matches with the primary key of another relation.
  - Table with the primary key is parent table
  - Table with the foreign key is the child table.

❖ RELATIONAL ALGEBRA

- Relational algebra is a collection of relational operations performed on relations and producing relations as results.
- Relational algebra expression is a sequence of relational algebra operations.
- Six basic operators
  - select: σ
  - project: $\prod$
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - Rename: ρ

- The operators take one or two relations as inputs and produce a new relation as a result.
- Relational algebra is a Procedural language.

❖ The Select Operation : (σ)

- **Operation** : Selects tuples from a relation that satisfy a given condition
- **Symbol:** σ (Sigma)
- **Notation**: σ <*selection condition*>(R)
- **Operators:** The following operators can be used to form a condition.

  = , ? , <, =, >, =, A (AND), V (OR) Find out all
- **Example:** Find out all students which belong to branch named "computer".
- Select operation is the horizontal partition.
- The following query returns all tuples which contains " computer " as a course name

$$\sigma_{\text{COURSE} = \text{“comp”}}(\text{student})$$

EN_NO	NAME	COURSE
101	RAVI	COMP
102	RAJ	COMP
105	NEHAL	COMP

❖ The Project Operation (∏):

- **Operation** : Selects specified attributes of a relation.
- **Symbol** : ∏ **(pi)**
- **Notation** : $\prod_{(\text{attribute list})}$ <R>
- Duplicate rows removed from result, since relations are sets
- The projection of R, denoted by $\prod_{(A, B,...)}R$ is the set of specified attributes A, B, ... of the relation R, i.e. a vertical subset of R.
- EXAMPLE:- $\prod_{(\text{STUDENT}, \text{EXAM})}$ ASSESSMENTS is

  OUTPUT:-

STUDENT	EXAM
A101	60
A101	50
J326	70

- **Example-3** : List out all account numbers having balance less than 7000.
  - $\prod_{(\text{ano})}(\sigma$ balance < 7000 (Account))

Output :

ano
A01
A02
A05

Figure 3.7: All account numbers having balance < 7000.

❖ The Union Operation (U)

- The union operation requires that the involved relations must have the same attributes, i.e. the relations are **Union-compatible**.
- **Operation** : Selects tuples those are in either or both of the relations.
- **Symbol** : U (union)
- **Notation** : Relation1 U Relation2
- **Requirement** :
  - Union must be taken between *compatible* relations.
  - Relations R and S are compatible, if –
    - Both have same number of attributes
    - Domains of i^{th} attribute of R and S are same.-
- For example, in Fig 1.0, STUDENTS-1 ∪ STUDENTS-2

TABLE: STUDENTS-1

EN_NO	**STUDENT-NAME**	**COURSE**
101	**RAVI**	COMP
102	**RAJ**	COMP
105	**NEHAL**	COMP

TABLE: STUDENTS-2

EN_NO	STUDENT-NAME	COURSE
206	REA	CIVIL

EN_NO	STUDENT-NAME	COURSE
101	RAVI	COMP
102	RAJ	COMP
206	REA	CIVIL
105	NEHAL	COMP

❖ The Set-Intersection Operation (∩)

- **Operation:** Selects tuples those are in both relations.

- **Symbol** : ∩ (intersection)
- **Notation** : Relation1 ∩ Relation2
- **Requirement** :
  - Set-intersection must be taken between *compatible* relations.
  - Relations R and S are compatible, if –
    - Both have same number of attributes
    - Domains of i^{th} attribute of R and S are same.
- **Example-6:** List out all persons who are customers as well as employees.
- **Query** : $\prod_{(name)}$ (STUDENT-1) ∩ $\prod_{(name)}$ (STUDENT-2)

TABLE: STUDENTS-1

EN_NO	NAME	COURSE
101	RAVI	COMP
102	RAJ	COMP
105	NEHAL	COMP

TABLE: STUDENTS-2

EN_NO	NAME	COURSE
206	REA	CIVIL
208	RAJ	CIVIL

OUTPUT:-

NAME
RAJ

❖ The Set-Difference (Minus) Operation (-) :

- **Operation:** Selects tuples those are in one relation but not in another relation.
- **Symbol** : - (minus)
- **Notation** : Relation 1 - Relation2
- **Requirement** :
  - Set-difference must be taken between *compatible* relations.
  - Relations R and S are compatible, if -
    - Both have same number of attributes
    - Domains of i^{th} attribute of R and S are same.

Example: List out all persons who are customers but not employees.

$\prod_{(name)}$ (Customer) - $\prod_{(name)}$ (Employee)

TABLE: CUSTOMER

CUST_NO	NAME
101	RAVI
102	RAJ
105	NEHAL

TABLE: EMPLOYEE

EMP_NO	NAME
101	RAVI
102	RAJ
105	VIDHI

OUTPUT:-

NAME
NEHAL

❖ The Cartesian-Product Operation (X)

- **Operation:** Combines information of two relations. It is also known as Cross-product operation and similar to mathematical Cartesian-product operation.
- **Symbol** : X (cross)
- **Notation** : Relation1 X Relation2
- **Resultant Relation** :
  - If Relation 1 and Relation2 have n1 and n2 attributes respectively, then resultant relation will have n1 + n2 attributes, combining attributes from both the input relations.
  - If both relations have some attribute having same name, it can be distinguished by combining 'Relation-name • Attribute-name'.
  - If Relation 1 and Relation2 have n1 and n2 tuples respectively, then resultant relation will have n1 * n2 attributes, combining each possible pair of tuples from both the input relations.
  - **Example-8:** Combine only consistent information from Account and Branch relation.
  - **Query** : (Customer X employee)

TABLE: CUSTOMER

CUST_NO	NAME
101	RAVI
102	RAJ
105	NEHAL

TABLE: EMPLOYEE

EMP_NO	NAME
101	RAVI
102	RAJ

OUTPUT:-

CUST_NO	NAME	EMP_NO	NAME

101	RAVI	101	RAVI
101	RAVI	102	RAJ
102	RAJ	101	RAVI
102	RAJ	102	RAJ
105	NEHAL	101	RAVI
105	NEHAL	102	RAJ

❖ JOINOPERATION:-

- Join operation is used to retrieve related rows from two relations.
- It is binary relation.
- Following are type of JOIN operation
    a. Inner join(natural join or equi join)
    b. Outer join
        i. Left outer join
        ii. Right outer join
        iii. Full outer join
    c. Self join
    d. Cross join

❖ The Natural Join Operation (INNER JOIN OR EQUI JOIN) (⋈)

- **Symbol** : ⋈ (JOIN).
- Notation : Relation 1 ⋈ Relation 2
- The natural join operation combines following three operations into one operation. The natural join operations -
    - Forms a Cartesian product on relations,
    - Performs a selection on common attributes to remove unnecessary tuples, and
    - Removes duplicate attributes.

Ex:

- Table name: **Client**

NAME	ID
Rahul	10
Vishal	20

- Table name: **Salesman**

ID	CITY
30	Bombay
20	Madras
40	Bombay

- Client $\bowtie_{(ID=ID)}$ Salesman

NAME	ID	CITY
Vishal	20	madras

Natural join operation shows only consistent and useful information. It removes unnecessary tuples as well as duplicate attributes. This makes the retrieval of information from multiple relations very easy and convenient.

❖ Outer Join Operation :

- Outer join operation avoids loss of information.
- Perform the join operation and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Outer join Uses *null* values:
  - *null* signifies that the value is unknown or does not exist
- The outer join operation can be divided into three different forms :
  - Left outer join ($\bowtie$)
  - Right outer join ($\bowtie$)
  - Full outer join ($\bowtie$)

- **Left outer join (⟕)**
  - The left outer join contains all the tuples of the left relation even there is no matching tuple in the right relation.
  - For such kind of tuple, the attributes of right relation will be added with null in resultant relation
  - **Ex:** client ⟕ salesman

NAME	ID	CITY
Rahul	10	Null
Vishal	20	madras

- **Right outer join (⟖)**
  - The right outer join contains all the tuples of the right relation even though there is no matching tuple in the left relation.
  - For such kind of tuple, the attributes of left relation will be added with null in resultant relation.
  - **Ex:** client ⟖ salesman

NAME	ID	CITY
Null	30	Bombay
Vishal	20	madras
Null	40	Bombay

- **Full outer join (⟗)**
  - The full outer join contains all the tuples of both of the relations. It also pads null values whenever required.
  - **Ex:** client ⟗ salesman

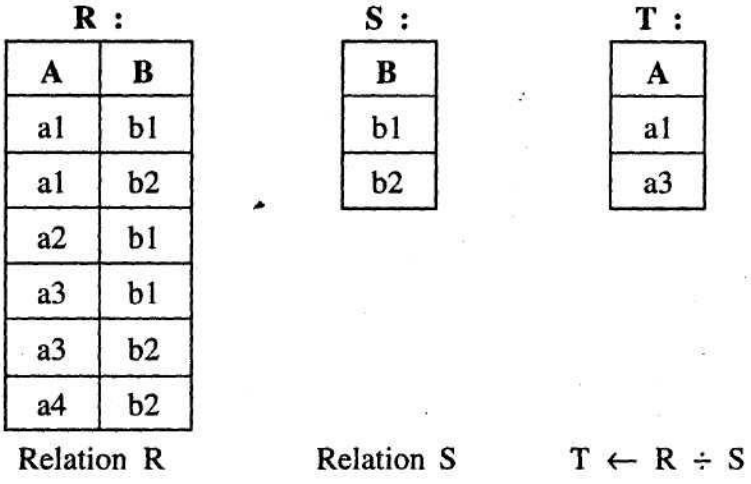
NAME	ID	CITY
Rahul	10	Null
Null	30	Bombay
Vishal	20	madras
Null	40	Bombay

❖ Division Operation (÷)

➢ The division operation is for a special kind of query. It is useful in queries that include "for all" phrase.

- **Symbol:** ÷ (division).
- **Notation:** Relation1 ÷ Relation2

The following example explains the division operation.



R :

A	B
a1	b1
a1	b2
a2	b1
a3	b1
a3	b2
a4	b2

Relation R

S :

B
b1
b2

Relation S

T :

A
a1
a3

T ← R ÷ S

Figure 3.23 : Illustrating the Division operation

- Observe that for all tuples of S, all possible pairs - (al, b1) and (al, b2) are present in R for value al of attribute A. So, al is included in resultant relation T.
- Similarly, for all tuples of S, all possible pairs - (a3, b1) and (a3, b2) are present in R for value a3 of attribute A. So, a3 is also included in T.
- But, for a2, the combination (a2, b1) is present in R, but, (a2, b2) is not present. So, a2 is not included in T.
- Similarly, for a4, the combination (a4, b2) is present in R, but, (a4, b1) is not present. So, a4 is not included in T.

❖ RENAME OPERATION:-

- **Operation:** Rename operation is used to rename the attribute.
- **Symbol:** **ρ**(rho)
- **Notation :**
  - $\rho_{s(a1,a2,\ldots,an)}(R)$
  - $\rho_{s}(R)$
  - $\rho_{(a1,a2,\ldots,an)}(R)$
- **Example1:** name of the relation renamed from student to student_info.
  - $\rho_{student\_info}(student)$
- **Example2:** attributes of R renamed to a1,a2,…,a3

- $\rho_{(a1,a2,\ldots,an)}(R)$

❖ Implementing relational algebra using SQL

- We can convert any relational algebra in SQL statement.

❖ Set operators:

- UNION, UNION ALL, INTERSECT AND MINUS ARE set operators.
- All set operator have equal precedence.
- If sql statement contains multiple set operation than oracle database will evaluate them from the left to right.

1. UNION and UNION ALL Operation

- Union operation is used to combine result of many queries
- Union merges the output of two or more queries in single table.
- It removes duplicate rows in both the query and display only common records.
- Difference between UNION and UNION ALL is that, UNION ALL also display duplicate data but UNION don't display duplicate data.

Ex: SELECT Dno from department UNION
SELECT Dno from employee;

Ex: SELECT Dno from department UNION ALL
SELECT Dno from employee;

2. INTERSECT Operation

- INTERSECT returns the results of two or more queries.
- It will returns only common records from the result of both query.

Ex: SELECT Dno from department INTERSECT
SELECT Dno from employee;

3. MINUS Operation

- MINUS operator will return all rows in the first SELECT statement that are not present in second SELECT statement.

Ex: SELECT Dno from department MINUS
SELECT Dno from employee;

❖ JOINS:

- Primary key and foreign key used to join tables.

- The table in which foreign key is definrd is called child table
- The table in which primary key is defined is called parent table
- Purpose of is to bind data across tables without repeating the data.
- Join between two table only posssible when table have common column with same datatype and size.
- Types of join:
  - Inner join(Equi join or natural join)
  - Non equi join
  - Outer join
  - Self join
  - Cross join

1. Inner join(Equi join)

- Inner join also known as equi join.
- Equi join is most common join in SQL.
- In this join tables are joined over common columns using = (equivalent) operator.
- **Syntax: (Theta Style)**

```
SELECT ColumnName1, ColumnName2,…ColumnNameN
FROM TableName1, TableName2
WHERE TableName1.column name =
TableName2.column_name
WHERE Condition;
```

- **Ex:**

SELECT * from employee,department where employee.dno=department.dno;

- **Syntax: (ANSI Style)**

```
SELECT ColumnName1, ColumnName2,…ColumnName N
FROM TableName1
INNER JOIN TableName2
ON TableName1.ColumnName1=TableName2.ColumnName2
WHERE Condition;
```

- **Ex:**

SELECT * from employee inner join department on employee.dno= department.dno;

2. Non equi join

- NON EQUI JOIN uses comparison operator instead of the equal sign like **>, <, >=, <=** along with conditions.
- **Syntax:**

```
SELECT *
FROM table name1, table name2
WHERE table_name1.column [> |  < |  >= | <= ]
table_name2.colum
```

- **Ex:**

```
Select * From employee1, employee2
Where employee1.salary>employee2.salary;
```

3. **Outer join:**
   - **I. Left outer join**
   - **II. Right outer join**
   - **III. Full outer join**

I. Left outer join:

- LEFT JOIN joins two tables and gets all matching rows of two tables for which the condition is true, plus rows from the left table that do not match any row in the right table.
- **Syntax:**

```
SELECT *
FROM table1
LEFT OUTER JOIN table2
ON table1.column_name=table2.column_name;
```

- **Ex:**

```
SELECT *
FROM employee
LEFT OUTER JOIN department
ON employee.dno = department.dno;
```

II. Right outer join

- The RIGHT JOIN, joins two tables and gets rows based on a condition, which is matching in both the tables and the unmatched rows will also be available from right table.
- **Syntax:**

```
SELECT *
FROM table1
RIGHT OUTER JOIN table2
ON table1.column_name=table2.column_name;
```

- **Ex:**

```
SELECT *
FROM employee
RIGHT OUTER JOIN department
ON employee.dno = department.dno;
```

IV. Full outer join

- The FULL OUTER JOIN combines the results of both left and right outer joins and returns all rows from the tables on both sides.

- **Syntax:**

```
SELECT *
FROM table1
FULL OUTER JOIN table2
ON table1.column_name=table2.column_name;
```

- **Ex:**

```
SELECT *
FROM employee
FULL OUTER JOIN department
ON employee.dno = department.dno;
```

4. Self join

- A self join is a join in which a table is joined with itself.
- In self join The table has a FOREIGN KEY which references its own PRIMARY KEY. To join a table itself means that each row of the table is combined with itself and with every other row of the table.
- **Syntax:**

```
SELECT a.column_name, b.column_name...
FROM table1 a, table1 b
WHERE a.common_filed = b.common_field;
```

- **Ex:**

```
SELECT a.emp_id AS "Emp_ID",a.emp_name AS "Employee Name",
b.emp_id AS "Supervisor ID",b.emp_name AS "Supervisor Name"
FROM employee a, employee b
WHERE a.emp_supv = b.emp_id;
```

❖ Subquery

- A subquery is a query within a query.
- It is a one SQL statement inside another SQL statement.
- It is also known as nested query.

- A subquery is used to return data that will be used in the main query as a condition to restrict the data to be retrieved.
- Subqueries used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.
- Subqueries must be enclosed within parentheses.

❖ **Subqueries with the SELECT Statement**

- Subqueries are most frequently used with the SELECT statement.

Table: CUSTOMERS

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000
2	Khilan	25	Delhi	1500
3	kaushik	23	Kota	2000
4	Chaitali	25	Mumbai	6500
5	Hardik	27	MP	8500
6	Muffy	24	Indore	4500

- **EX:** select the cutomers who's salary is greater than 4500.

```
SELECT *
FROM CUSTOMER
WHERE ID IN (SELECT ID
FROM CUSTOMER
WHERE SALARY > 4500) ;
```

❖ **Subqueries with the INSERT Statement**

- Subqueries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table.
- **Ex:** insert in cust table from customer table who's salary is greater than 4500.

```
INSERT INTO CUST
SELECT * FROM CUSTOMERS
WHERE ID IN (SELECT ID
FROM CUSTOMERS
 WHERE SALARY > 4500) ;
```

❖ **Subqueries with the UPDATE Statement**

- The subquery can be used in with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

- **Ex:** update the salary of customer who's age is greater than 27.

```
UPDATE CUSTOMERS
```

```
SET SALARY = SALARY * 0.25
WHERE AGE IN (SELECT AGE FROM CUSTOMERS
WHERE AGE >= 27 );
```

❖ **Subqueries with the DELETE Statement**

- The subquery can be used in conjunction with the DELETE statement.
- **Ex:** delete the customers who's age is greater than 27.

```
DELETE FROM CUSTOMERS
WHERE AGE IN (SELECT AGE FROM CUSTOMERS
WHERE AGE >= 27 );
```

❖ **Correlated subquery:**

- When inner query and outer query are mutually dependent, it is known as correted query.
- For every row processed by the inner query, the query is also processed.
- The inner query depends on the outer query before it can be processed.

Ex: Display department names of such departments that have one or more employee.

```
SELECT d.dept_name FROM DEPARTMENT d
WHERE d.dept_no IN (SELECT e.dept_no FROM Employee e
WHERE e.dept_no=d.dept_no);
```

❖ **Restrictions on subquery**

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator..
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

❖ **REPORTS**

- Generally, SQP displays output by using its default formatting.
- This format have default column names, default justifying, default size of pages etc.
- This default behavior can be changed by using various commands and systematic output can be generated.

- Following are the various report formatting commands.

1. TTITLE:
   - Sets top title for each page of a report.
   - Syntax: TTITLE [left] {text} [right] {text} [center] {text}
   - EX: TITLE left 'STUDENT_DATA' right 'page no.' SQL.PNO

2. BTITLE:
   - Sets BOTTOM title for each page of a report.
   - Syntax: BTITLE [left] {text} [right] {text} [center] {text}
   - EX: BTITLE left 'page no.' SQL.PNO 'ends…'

3. SKIP:
   - Skips as many line or pages as specified.
   - Syntax: SKIP {NUMBER} {PAGE}
   - EX: SKIP 2 /* SKIPS 2 LINES */
   - EX: SKIP page /* SKIPS ONE PAGE*/

4. COLUMN:
   - SQL gives variety of instructions on column heading and format.
   - Syntax: COLUMN{columnname}
     [FORMAT] {format instruction}
     [JUSTIFY] {JUSTIFICATION}
     [HEADING] {TEXT}
   - FORMAT option is useful in applying editing to numeric fields, date field, variable-length character field.
   - EX: COLUMN name FORMAT a10 JUSTIFY center HEADING student_name

5. BREAK ON:
   - Break on tells where to put spaces between section of a report.
   - BREAK point can be defined at column level, on a row, on a page or on a report.
   - Syntax: BREAK ON {columnname} [options]
   - EX: BREAK ON NAME SKIP 2
     EX: BREAK ON report
     EX: BREAK ON page

6. COMPUTE SUM:
   - COMPUTR CALCULATE AND PRINT RESULT OF GROUP FUNCTIOS FOR A SET OF ROWS.
   - Various standard functions such as MAX, MIN, SUM, AVG and COUNT are UTILZED.
   - Syntax : COMPUTE {FUNCTION} OF {COLUMN} ON {BREAK}

- EX: COMPUTE_ SUM OF SARAY ON PAGE

7. SQL.PNO:
   - display the page number
   - syntax: SQL.PNO

8. SET PAUSE:
   - Makes screen display stop in between pages of display.
   - Syntax: SET PAUSE [ON/OFF]

9. SET LINESIZE:
   - Sets number of characters displayed in a single line.
   - Syntax: SET LINESIZE [NO]
   - EX: SET LINESIZE 80

10. SET PAGESIZE
    - Sets maximum number of lines per page.
    - Syntax: SET PAGESIZE [NO]
    - EX: SET PAGESIZE 30