UNIT – II (DBMS)

SYLLABUS

The Relational Algebra and Relational Calculus:

Unary Relational Operations: SELECT and PROJECT - Relational Algebra Operations from Set Theory - Binary Relational Operations: JOIN and DIVISION – Additional Relational Operations - The Tuple Relational Calculus - The Domain Relational Calculus

SQL-99:

Schema Definition, Constraints, Queries, and Views: SQL Data Definition and Data Types - Specifying Constraints in SQL - Schema Change Statements in SQL - Basic Queries in SQL - More Complex SQL Queries - INSERT, DELETE, and UPDATE Statements in SQL - Views (Virtual Tables) in SQL

The Relational Algebra and Relational Calculus

Relational Algebra:

- The basic set of operations for the relational model is the relational algebra.
- A sequence of relational algebra operations forms a relational algebra expression, whose result will also be a relation that represents the result of a database query (or retrieval request).
- The relational algebra is very important for several reasons.
  1. It provides a formal foundation for relational model operations.
  2. It is used as a basis for implementing and optimizing queries in the query processing and optimization modules that are integral parts of relational database management systems (RDBMSs).
  3. Third, some of its concepts are incorporated into the SQL standard query language for RDBMSs.
- The relational algebra operations can be divided into two groups.
  - One group includes set operations from mathematical set theory. Set operations include UNION, INTERSECTION, SET DIFFERENCE, and CARTESIAN PRODUCT (also known as CROSS PRODUCT).
  - The other group consists of operations developed specifically for relational databases—these include SELECT, PROJECT, and JOIN, among others

Tables:

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

1.1 Unary Relational Operations: SELECT and PROJECT

- The SELECT and PROJECT operator are **unary**; that is, it is applied to a **single relation (table)**.

1.1.1 SELECT Operation:

- The SELECT operation is used to **retrieve a subset of the tuples from a relation that satisfies a selection condition**.
- **Syntax:**

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

- Where the symbol **σ (sigma)** is used to denote the SELECT operator,
  selection condition is a Boolean expression (condition) specified on the attributes of relation R.
- For example, to select the EMPLOYEE tuples whose department is 4, SELECT operation as follows:

  $\sigma_{Dno=4}(EMPLOYEE)$

- The SELECT operation can also be visualized as a horizontal partition of the relation into two sets of tuples:
  - Tuples that satisfy the condition and are selected.
  - Tuples that do not satisfy the condition and are discarded.
- The **<selection condition>** is applied **independently to each individual tuple t in R**.
- The Boolean expression specified in **<selection condition>** is made up of a number of clauses of the form.

  <attribute name> <comparison op> <constant value>

  Or

  <attribute name> <comparison op> <attribute name>

- Where **<attribute name>** is the name of an attribute of R

  <comparison op> is normally one of the operators {=, <, ≤, >, ≥, ≠}

  For **ordered values** (numeric or date), the comparison operators are {=, <, ≤, >, ≥, ≠}

  For **unordered values** (names, colors), the comparison operators are {=, ≠}

  <constant value> is a constant value from the attribute domain.

- **Example:**

  $\sigma_{(Dno=4\ AND\ Salary>25000)\ OR\ (Dno=5\ AND\ Salary>30000)}(EMPLOYEE)$

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5

- The number of tuples in the resulting relation is always less than or equal to the number of tuples in R. That is, $|\sigma_c(R)| \leq |R|$ for any condition C.
- The **degree of the relation** is the **number of attributes in a relation**.
- Notice that the SELECT operation is **commutative**; that is,

  $\sigma_{<cond1>}(\sigma_{<cond2>}(R)) = \sigma_{<cond2>}(\sigma_{<cond1>}(R))$

- we can always combine a **cascade** (or **sequence**) of SELECT operations into a single SELECT operation with a conjunctive (AND) condition; that is,

  $\sigma_{<cond1>}(\sigma_{<cond2>}(\ldots(\sigma_{<condn>}(R))\ldots)) = \sigma_{<cond1>\ AND\ <cond2>\ AND \ldots AND\ <condn>}(R)$

- In SQL, the SELECT condition is typically specified in the WHERE clause of a query.
- For example, the following operation:

  $\sigma_{Dno=4\ AND\ Salary>25000}(EMPLOYEE)$

- would correspond to the following SQL query:

```
SELECT *
FROM EMPLOYEE
WHERE Dno=4 AND Salary>25000;
```

1.1.2 PROJECT Operation:

- The PROJECT operation is used to **retrieve only certain attributes (columns) in a relation (table).**
- The PROJECT operation is display the attributes **in the same order as they appear in the list.**
- **Syntax:**

$$\pi_{<\text{attribute list}>}(R)$$

- Where **π** (**pi**) is the symbol used to represent the PROJECT operation,
  <attribute list> is the desired sub-list of attributes from the relation R.
- For example, to list each employee's first and last name and salary, we can use the PROJECT operation as follows:

$$\pi_{\text{Lname, Fname, Salary}}(\text{EMPLOYEE})$$

- The PROJECT operation can be **visualized as a vertical partition** of the relation into two relations:
  - One has the needed columns (attributes) and contains the result of the operation.
  - The other contains the discarded columns.
- Degree is equal to the number of attributes in <attribute list>.
- If the attribute list includes only non-key attributes of R, duplicate tuples are likely to occur.
- The PROJECT operation removes any duplicate tuples from relation and retrieve a valid set of distinct tuples is known as **duplicate elimination**.
- For example, consider the following PROJECT operation:

$$\pi_{\text{Sex, Salary}}(\text{EMPLOYEE})$$

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

- Notice that the tuple <'F', 25000> appears only once in above Figure, even though this combination of values appears twice in the EMPLOYEE relation.

$$\pi_{<\text{list1}>}(\pi_{<\text{list2}>}(R)) = \pi_{<\text{list1}>}(R)$$

- As long as <list2> contains the attributes in <list1>; otherwise, the left-hand side is an incorrect expression.
- It is also noteworthy that **commutativity does not hold** on PROJECT.
- In SQL, the PROJECT attribute list is specified in the SELECT clause of a query.
- For example, the following operation:

$$\pi_{\text{Sex, Salary}}(\text{EMPLOYEE})$$

- would correspond to the following SQL query:

```
SELECT DISTINCT Sex, Salary
FROM EMPLOYEE
```

Write a relational algebra expression in single line known as an **in-line expression**.

Example: π Fname, Lname, Salary(σ Dno= 5 (EMPLOYEE))

1.1.3 RENAME Operation

- To rename the attributes in a relation, we simply list the new attribute names in parentheses.
- Rename operation is denoted with the symbol ρ (rho)
- Example:

  TEMP ← $\sigma_{Dno=5}$ (EMPLOYEE)

  R (First_name, Last_name, Salary) ← $\pi_{Fname, Lname, Salary}$(TEMP)

TEMP

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston,TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston,TX	M	40000	888665555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble,TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

R

First_name	Last_name	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

- If no renaming is applied, the names of the attributes in the resulting relation of a SELECT operation are the same as those in the original relation and in the same order.
- We can rename either the **relation name** or the **attribute names**, or **both**—as a unary operator.
- The general RENAME operation when applied to a relation R of degree n is denoted by any of the following three forms:

  ρ_S (R) or $\rho_{(B\,1, B\,2, ..., Bn)}$(R) or $\rho_{S\,(B\,1, B\,2, ..., Bn)}$(R)
- Where the symbol ρ (rho) is used to denote the RENAME operator, S is the new relation name, and B1 , B2 , ..., Bn are the new attribute names.
- The first expression renames the relation only, the second renames the attributes only, and the third renames both the relation and its attributes.
- Renaming in SQL is accomplished by aliasing using AS, as in the following example:

```
SELECT E.Fname AS First_name, E.Lname AS Last_name, E.Salary AS Salary
FROM EMPLOYEE AS E
WHERE E.Dno=5;
```

1.2 Relational Algebra Operations from Set Theory

1.2.1 The UNION, INTERSECTION, and MINUS Operations

- Set theoretic operations are used to merge the elements of two sets in various ways, including UNION, INTERSECTION, and SET DIFFERENCE (also called MINUS or EXCEPT).
- These are binary operations; that is, each is applied to two relations.
- The two relations on which any of these three operations are applied must follow two rules.
  - ✓ Two relations must have **the same number of attributes**.
  - ✓ Two relations must have **union compatibility** or **type compatibility (**i.e., same type of tuples**)**.
- Two relations R(A1 , A2 , ..., An) and S(B1 , B2 , ..., Bn) are said to be union compatible (or type compatible) if they have the same degree n and if dom(Ai) = dom(Bi).
- We can define the three operations **UNION**, **INTERSECTION**, and **SET DIFFERENCE** on two union-compatible relations R and S as follows:

  ❖ **UNION:** The result of this operation, **denoted by R ∪ S**, is a relation that includes **all tuples that are either in R or in S or in both R and S**. **Duplicate tuples are eliminated**.

  ❖ **INTERSECTION:** The result of this operation, **denoted by R ∩ S**, is a relation that includes **all tuples that are in both R and S**.

  ❖ **SET DIFFERENCE (or MINUS):** The result of this operation, **denoted by R – S**, is a relation that includes **all tuples that are in R but not in S**.

  Figure: The set operations UNION, INTERSECTION, and MINUS.
  (a) Two union-compatible relations. (b) STUDENT ∪ INSTRUCTOR.
  (c) STUDENT ∩ INSTRUCTOR. (d) STUDENT − INSTRUCTOR.
  (e) INSTRUCTOR − STUDENT

(a) STUDENT

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

(b)

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

(d)

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

- Notice that both UNION and INTERSECTION are **commutative operations**.
  $R \cup S = S \cup R$ and $R \cap S = S \cap R$
- The MINUS operation is **not commutative**.
  $R - S \neq S - R$
- Note that INTERSECTION can be expressed in terms of union and set difference as follows:
  $R \cap S = ((R \cup S) - (R - S)) - (S - R)$
- There are multiset operations (UNION ALL, INTERSECT ALL, and EXCEPT ALL) that **do not eliminate duplicates.**

1.2.2 The CARTESIAN PRODUCT (CROSS PRODUCT) Operation:

- CROSS PRODUCT or CROSS JOIN—which is **denoted by ×.**
- This is also a **binary set operation**, but the relations on which it is **applied do not have to be union compatible or type compatibility**.
- The CARTESIAN PRODUCT creates tuples with the combined attributes of two relations.
- CROSS PRODUCT operation produces a new element by **combining every member (tuple) from one relation (set) with every member (tuple) from the other relation (set).**
- The result of R(A1 , A2 , ..., An) × S(B1 , B2 , ..., Bm) is a relation Q with degree n + m attributes Q(A1 , A2 , ..., An , B1 , B2 , ..., Bm), in that order.

Example:

Dept

Deptno	Dname	Loc
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Salgrade

Losal	Hisal	Grade
700	1200	1
1201	1400	2
1401	2000	3
2001	3000	4
3001	9999	5

Dept X Salgrade

Deptno	Dname	Loc	Losal	Hisal	Grade
10	ACCOUNTING	NEW YORK	700	1200	1
10	ACCOUNTING	NEW YORK	1201	1400	2
10	ACCOUNTING	NEW YORK	1401	2000	3
10	ACCOUNTING	NEW YORK	2001	3000	4
10	ACCOUNTING	NEW YORK	3001	9999	5
20	RESEARCH	DALLAS	700	1200	1
20	RESEARCH	DALLAS	1201	1400	2
20	RESEARCH	DALLAS	1401	2000	3
20	RESEARCH	DALLAS	2001	3000	4
20	RESEARCH	DALLAS	3001	9999	5
30	SALES	CHICAGO	700	1200	1
30	SALES	CHICAGO	1201	1400	2
30	SALES	CHICAGO	1401	2000	3
30	SALES	CHICAGO	2001	3000	4
30	SALES	CHICAGO	3001	9999	5
40	OPERATIONS	BOSTON	700	1200	1
40	OPERATIONS	BOSTON	1201	1400	2
40	OPERATIONS	BOSTON	1401	2000	3
40	OPERATIONS	BOSTON	2001	3000	4
40	OPERATIONS	BOSTON	3001	9999	5

1.3 Binary Relational Operations: JOIN and DIVISION

6.3.1 The JOIN Operation

- The JOIN operation, denoted by ⋈, is used to combine related tuples from two relations into single "longer" tuples.
- There are several join operations: **THETA JOIN, EQUIJOIN, NATURAL JOIN**

THETA JOIN:

- A JOIN operation with general join condition is called a **THETA JOIN**.
- Syntax:

  $R \bowtie_{<\text{join condition}>} S$ or $R \bowtie_{\theta} S$
- Where R and S are two relations R (A1 , A2 , ..., An) and S(B1 , B2 , ..., Bm).
- where each <condition> is of the form $A_i\ \theta\ B_j$, Ai is an attribute of R, B_j is an attribute of S, A_i and B_j have the same domain, and θ (theta) is one of the comparison operators $\{=, <, \leq, >, \geq, \neq\}$.
- Example: to retrieve the employee names and their grades

  $\pi_{\text{Ename , Grade}}(\text{Emp} \bowtie_{\text{emp.sal>=salgrade.losal AND emp.sal>=salgrade.hisal Salgrade}} \text{Salgrade})$
- Tuples whose join attributes are NULL or for which the join condition is FALSE do not appear in the result.

EQUIJOIN:

- A JOIN, where the only comparison operator used is =, is called an EQUIJOIN.
- EQUIJOIN we **always have one or more pairs of attributes that have identical values in every tuple**.
- Example: $\pi_{\text{Ename , Deptno, Dname}}(\text{Emp} \bowtie_{\text{emp.deptno=dept.deptno}} \text{Dept})$

NATURAL JOIN:

- NATURAL JOIN denoted by *****.
- A NATURAL JOIN requires that the two join **attributes** (or **each pair of join attributes**) **have the same name** in both relations.
- NATURAL JOIN was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition.
- Example: $\pi_{\text{Ename , Deptno, Dname}}(\text{Emp} \bowtie_{\text{emp.deptno=dept.deptno}} \text{Dept})$
- If this is not the case, a renaming operation is applied first.

  Example: e1←EMP e2←EMP mgr←e2.Empno

  $\pi_{\text{e1.Ename , e2.Ename}}(\text{e1} \bowtie_{\text{e1.mgr=mgr}} \text{e2})$
- **Join attribute:** an **attribute is used to join the both relations** is called join attribute.
- In JOIN, only combinations of tuples satisfying the join condition appear in the result, whereas in the CARTESIAN PRODUCT all combinations of tuples are included in the result.
- The expected size of the join result divided by the maximum size $n_R * n_S$ leads to a ratio called **join selectivity**, which is a property of each join condition.

1.3.2 The DIVISION Operation

- The DIVISION operation, denoted by ÷
- The DIVISION include the tuples appear in R in combination with every tuple in S.
- Note that in the formulation of the DIVISION operation, the tuples in the denominator relation S restrict the numerator relation R by selecting those tuples in the result that match all values present in the denominator.
- The DIVISION operation is applied to two relations R(Z) ÷ S(X), where the attributes of R are a subset of the attributes of S.
- **Example** is Retrieve the names of employees who work on all the projects that 'John Smith' works on.

  $\text{SMITH} \leftarrow \sigma_{\text{Fname= 'John' AND Lname= 'Smith'}}(\text{EMPLOYEE})$

  $\text{SMITH\_PNOS} \leftarrow \pi_{\text{Pno}}(\text{WORKS\_ON} \bowtie_{\text{Essn= Ssn}} \text{SMITH})$

  $\text{SSN\_PNOS} \leftarrow \pi_{\text{Essn, Pno}}(\text{WORKS\_ON})$

- Apply the DIVISION operation to the two relations, which gives the desired employees' Social Security numbers:

SSNS(Ssn) ← SSN_PNOS ÷ SMITH_PNOS
RESULT ← πFname, Lname(SSNS * EMPLOYEE)

(a)

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SMITH_PNOS

Pno
1
2

SSNS

Ssn
123456789
453453453

(b)

R

A	B
a1	b1
a2	b1
a3	b1
a4	b1
a1	b2
a3	b2
a2	b3
a3	b3
a4	b3
a1	b4
a2	b4
a3	b4

S

A
a1
a2
a3

T

B
b1
b4

Figure 1.8 The DIVISION operation. (a) Dividing SSN_PNOS by SMITH_PNOS. (b) T ← R ÷ S

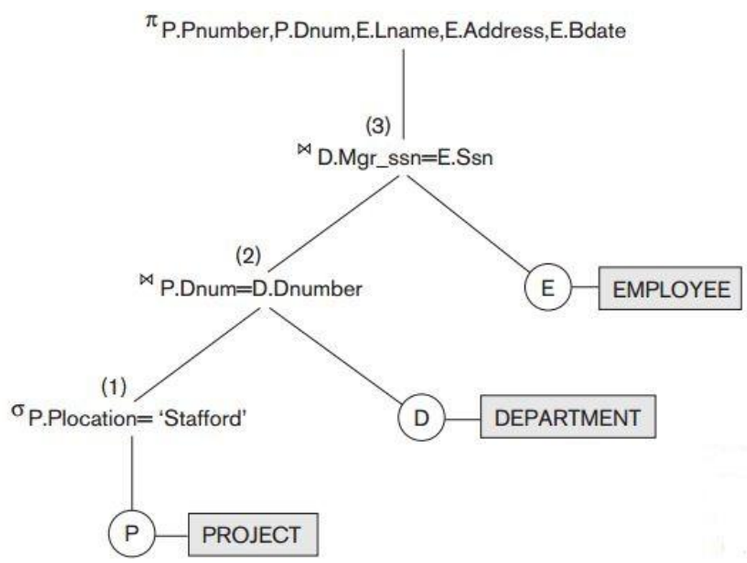
- The DIVISION operation can be expressed as a sequence of π, ×, and – operations as follows:

T1 ← πY (R)
T2 ← πY ((S × T1) – R)
T ← T 1 – T2

1.3.3 Notation for Query Trees

- **Query tree** is a tree data structure that corresponds to a relational algebra expression.
- It represents the input relations of the query as leaf nodes of the tree, and rep-resents the relational algebra operations as internal nodes.
- An execution of the query tree consists of executing an internal node operation whenever its operands (represented by its child nodes) are available, and then replacing that internal node by the relation that results from executing the operation. The execution terminates when the root node is executed and produces the result relation for the query.
- Example:
  For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

$\pi_{Pnumber, Dnum, Lname, Address, Bdate}(($
$(\sigma_{Plocation='Stafford'}(PROJECT)) \bowtie_{Dnum=Dnumber} (DEPARTMENT)) \bowtie_{Mgr\_ssn=Ssn} (EMPLOYEE))$



1.4 Additional Relational Operations

1.4.1 Generalized Projection

- The generalized projection operation extends the projection operation by allowing functions of attributes to be included in the projection list.
- Syntax:

$$\pi_{F\,1,\,F2,\,...,\,Fn}\,(\,R)$$

- Where F1 , F2 , ..., Fn are functions over the attributes in relation R and may involve arithmetic operations and constant values.
- As an example, consider the relation

  EMPLOYEE (Ssn, Salary, Deduction, Years_service)
- A report may be required to show

  Net Salary = Salary – Deduction,

  Bonus = 2000 * Years_service, and

  Tax = 0.25 * Salary.
- Then a generalized projection combined with renaming may be used as follows:

$$\text{REPORT} \leftarrow \rho_{(\,Ssn,\,Net\_salary,\,Bonus,\,Tax)}\,(\,\pi_{\,Ssn,\,Salary\,-\,Deduction,\,2000\,*\,Years\_service,\,0.25\,*\,Salary}(\,\text{EMPLOYEE}))$$

1.4.2 Aggregate Functions and Grouping

- Aggregate functions are **SUM, AVERAGE, MAXIMUM, MINIMUM, and COUNT**.
- **SUM:** function is used to find sum of tuples or values.
- **AVERAGE:** function is used to find average of tuples or values.
- **MAXIMUM:** function is used to find maximum value from the tuples or values.
- **MINIMUM:** function is used to find minimum value from the tuples or values.
- **COUNT:** function is used for counting tuples or values.
- **GROUP:** grouping the tuples in a relation by the value of some of their attributes and then applying an aggregate function independently to each group.
- We can define an AGGREGATE FUNCTION operation, using the symbol ℑ (pronounced **script F**)
- Syntax:

$$\textbf{<grouping attributes>}\ \Im\ \textbf{<function list>}\ (\,R)$$

- Where **<grouping attributes>** is a list of attributes of the relation specified in R, **<function list>** is a list of (<function> <attribute>) pairs.
- **Example:** List Count of employees in each department and their average salary.

$$(\,_{Dno}\,\Im\,_{COUNT\ Ssn,\ AVERAGE\ Salary}\,(\,\text{EMPLOYEE}))$$

Dno	Count_ssn	Average_salary
5	4	33250
4	3	31000
1	1	55000

- If **no renaming is applied**, then the attribute names be the **concatenation of the function name with the attribute name in the form <function>_<attribute>**.
- If **rename is applied** then corresponding rename attributes are displayed.

$$\rho_{\,R\,(\,Dno,\,No\_of\_employees,\,Average\_sal)}\,(\,_{Dno}\,\Im\,_{COUNT\ Ssn,\ AVERAGE\ Salary}\,(\,\text{EMPLOYEE}))$$

R

Dno	No_of_employees	Average_sal
5	4	33250
4	3	31000
1	1	55000

- If **no grouping attributes are specified**, the functions are applied to all the tuples in the relation, so the resulting relation has a **single tuple only**.
- Example:

$$\Im_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE})$$

Count_ssn	Average_salary
8	35125

- **Note:** duplicates are not eliminated when an aggregate function is applied.

1.4.3 OUTER JOIN Operations

- **Outer joins** were developed for the case where the user wants to **keep all the tuples in R, or all those in S, or all those in both relations** in the result of the JOIN, regardless of whether or not they have matching tuples in the other relation.
- This satisfies the need of queries in which tuples from two tables are to be combined by matching corresponding rows, but without losing any tuples for lack of matching values.
- There are three type of outer joins are there
  1. **LEFT OUTER JOIN** The LEFT OUTER JOIN operation keeps every tuple in the ***first***, or ***left***, relation R in R⟕S.
  2. **RIGHT OUTER JOIN** The RIGHT OUTER JOIN operation keeps every tuple in the ***second***, or ***right***, relation S in R⟖S.
  3. **FULL OUTER JOIN** The FULL OUTER JOIN operation keeps both tuple in the ***left*** *and* ***right***, relations in R⟗S.
- **If no matching tuple** is found in relations then **filled or padded with NULL values**.
- **For example**, list of all employee names as well as the name of the departments they manage.
- If employee manage a department then department name will be result; if they do not manage placed with NULL value. We can apply an operation LEFT OUTER JOIN

$$\text{TEMP} \leftarrow (\text{EMPLOYEE} \overset{}{\bowtie}_{\text{Ssn= Mgr\_ssn}} \text{DEPARTMENT})$$

$$\text{RESULT} \leftarrow \pi_{\text{Fname, Minit, Lname, Dname}}(\text{TEMP})$$

RESULT

Fname	Minit	Lname	Dname
John	B	Smith	NULL
Franklin	T	Wong	Research
Alicia	J	Zelaya	NULL
Jennifer	S	Wallace	Administration
Ramesh	K	Narayan	NULL
Joyce	A	English	NULL
Ahmad	V	Jabbar	NULL
James	E	Borg	Headquarters

1.4.4 The OUTER UNION Operation

- The OUTER UNION operation was developed to take the **union of tuples** from two relations that have some common attributes, but are **not union (type) compatible**.
- This operation will take the UNION of tuples in two relations R(X, Y) and S(X, Z) that are **partially compatible**, meaning that only some of their attributes, say X, are union compatible.
- The attributes that are union compatible are represented only once in the result, and those attributes that are not union compatible from either relation are also kept in the result relation T(X, Y, Z).
- It is therefore the same as a **FULL OUTER JOIN** on the common attributes.

1.5 Examples of Queries in Relational Algebra

Query 1. Retrieve the name and address of all employees who work for the 'Research' department.

Ans: RESEARCH_DEPT ← $\sigma_{Dname='Research'}$(DEPARTMENT)

RESEARCH_EMPS ← (RESEARCH_DEPT $\bowtie_{Dnumber=Dno}$ EMPLOYEE)

RESULT ← $\pi_{Fname, Lname, Address}$(RESEARCH_EMPS)

As a single in-line expression, this query becomes:

$\pi_{Fname, Lname, Address}$ ($\sigma_{Dname='Research'}$(DEPARTMENT $\bowtie_{Dnumber=Dno}$(EMPLOYEE))

Query 2. For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

Ans: STAFFORD_PROJS ← $\sigma_{Plocation='Stafford'}$(PROJECT)

CONTR_DEPTS ← (STAFFORD_PROJS $\bowtie_{Dnum=Dnumber}$ DEPARTMENT)

PROJ_DEPT_MGRS ← (CONTR_DEPTS $\bowtie_{Mgr\_ssn=Ssn}$ EMPLOYEE)

RESULT ← $\pi_{Pnumber, Dnum, Lname, Address, Bdate}$(PROJ_DEPT_MGRS)

Query 3. Find the names of employees who work on all the projects controlled by department number 5.

Ans: DEPT5_PROJS ← $\rho_{(Pno)}$ ($\pi_{Pnumber}$($\sigma_{Dnum=5}$ (PROJECT)))

EMP_PROJ ← $\rho_{(Ssn, Pno)}$ ($\pi_{Essn, Pno}$(WORKS_ON))

RESULT_EMP_SSNS ← EMP_PROJ ÷ DEPT5_PROJS

RESULT ← $\pi_{Lname, Fname}$(RESULT_EMP_SSNS * EMPLOYEE)

Query 4. Make a list of project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

Ans: SMITHS(Essn) ← π_{Ssn} ($\sigma_{Lname='Smith'}$(EMPLOYEE))

SMITH_WORKER_PROJS ← π_{Pno}(WORKS_ON * SMITHS)

MGRS ← $\pi_{Lname, Dnumber}$(EMPLOYEE $\bowtie_{Ssn=Mgr\_ssn}$ DEPARTMENT)

SMITH_MANAGED_DEPTS(Dnum) ← $\pi_{Dnumber}$ ($\sigma_{Lname='Smith'}$(MGRS))

SMITH_MGR_PROJS(Pno) ← $\pi_{Pnumber}$(SMITH_MANAGED_DEPTS * PROJECT)

RESULT ← (SMITH_WORKER_PROJS ∪ SMITH_MGR_PROJS)

π_{Pno} (WORKS_ON $\bowtie_{Essn=Ssn}$(π_{Ssn} ($\sigma_{Lname='Smith'}$(EMPLOYEE))) ∪

π_{Pno} (($\pi_{Dnumber}$ ($\sigma_{Lname='Smith'}$($\pi_{Lname, Dnumber}$(EMPLOYEE))) $\bowtie_{Ssn=mgr\_ssn}$ DEPARTMENT)) $\bowtie_{Dnumber=Dnum}$ PROJECT)

Query 5. List the names of all employees with two or more dependents.

Ans: T1(Ssn, No_of_dependents) ← $_{Essn}\Im_{COUNT\ Dependent\_name}$(DEPENDENT)

T2 ← $\sigma_{No\_of\_dependents>2}$ (T1)

RESULT ← $\pi_{Lname, Fname}$(T2 * EMPLOYEE)

Query 6. Retrieve the names of employees who have no dependents.

This is an example of the type of query that uses the MINUS (SET DIFFERENCE) operation.

Ans: ALL_EMPS ← π_{Ssn}(EMPLOYEE)

EMPS_WITH_DEPS(Ssn) ← π_{Essn}(DEPENDENT)

EMPS_WITHOUT_DEPS ← (ALL_EMPS – EMPS_WITH_DEPS)

RESULT ← $\pi_{Lname, Fname}$(EMPS_WITHOUT_DEPS * EMPLOYEE)

As a single in-line expression, this query becomes:

$\pi_{Lname, Fname}((\pi_{Ssn}$(EMPLOYEE) – ρ Ssn(π_{Essn}(DEPENDENT))) * EMPLOYEE)

Query 7. List the names of managers who have at least one dependent.

Ans: MGRS(Ssn) ← π_{Mgr_ssn}(DEPARTMENT)

EMPS_WITH_DEPS(Ssn) ← π_{Essn}(DEPENDENT)

MGRS_WITH_DEPS ← (MGRS ∩ EMPS_WITH_DEPS)

RESULT ← $\pi_{Lname, Fname}$(MGRS_WITH_DEPS * EMPLOYEE)

Relational Calculus:

- The relational calculus is a formal language, based on the branch of **mathematical logic called predicate calculus**.
- The relational calculus provides a ***higher-level declarative language*** for specifying relational queries.
- In a relational calculus expression, there is ***no order of operations*** to specify how to retrieve the query result—only ***what information the result should contain***.
- This is the main distinguishing feature between relational algebra and relational calculus.
- There are two variations of relational calculus-**tuple relational calculus** and **domain relational calculus**.

1.6 The Tuple Relational Calculus

- The relational calculus is considered to be a **nonprocedural language** i.e., **no need to write a sequence of operations to specify a retrieval request in a particular order** of applying the operations.
- A calculus expression specifies **what is to be retrieved rather than how to retrieve it**.
- Relational calculus is important for two reasons-
  - First, it has a firm basis in mathematical logic.
  - Second, the standard query language (SQL) for RDBMSs has some of its foundations in the tuple relational calculus.

1.6.1 Tuple Variables and Range Relations

- The tuple relational calculus is based on specifying a **number of tuple variables**.
- **Each tuple variable usually ranges over a particular database relation**, meaning that the variable may take as its value any individual tuple from that relation.
- Syntax:

  {t | COND(t)}
- Where **t** is a tuple variable and **COND(t)** is a conditional (Boolean) expression involving t that evaluates to either TRUE or FALSE for different assignments of tuples to the variable t.
- For example, to find all employees whose salary is above $50,000, we can write the following tuple calculus expression:

  {t | EMPLOYEE(t) AND t .Salary>50000}

- We can also retrieve only some of the attributes(say the first and last names) as

  {t.Fname, t.Lname | EMPLOYEE(t) AND t .Salary>50000}
- we need to specify the following information in a tuple relational calculus expression:
  - For each tuple variable t, the **range relation** R of t means R(t). **If we do not specify a range relation,** then the variable t will **range over all possible tuples "in the universe"** as it is not restricted to any one relation.
  - A condition to select particular combinations of tuples from range relations, the condition is evaluated for every possible combination of tuples to identify the **selected combinations** for which the condition evaluates to TRUE.
  - A **set of attributes** to be retrieved, the **requested attributes**. The values of these attributes are retrieved for each selected combination of tuples.

1.6.2 Expressions and Formulas in Tuple Relational Calculus

- A general **expression of the tuple relational calculus** is of the form

  $$\{t_1 . A_j, t_2 . A_k , ..., t_n .A_m \mid COND(t1 , t2 , ..., tn , tn +1, tn +2, ..., tn+m)\}$$

- where t_1 , t_2 , ..., t_n , t_{n+1}, ..., t_{n+m} are tuple variables, each A_i is an attribute of the relation on which t_i ranges, and **COND is a condition or formula**.
- A formula is **made up of predicate calculus atoms**, which can be one of the following:
  1. An atom of the form $R(t_i)$, where R is a relation name and t_i is a tuple variable.
  2. An atom of the form t_i. A op t_j. B.

     Where **op** is one of the comparison operators in the set $\{=, <, \leq, >, \geq, \neq\}$,

     t_i and t_j are tuple variables,

A is an attribute of the relation on which t_i ranges, and
B is an attribute of the relation on which t_j ranges.

3. An atom of the form ti. A op c or c op tj. B, where c is a constant value.

- A formula (Boolean condition) is made up of one or more atoms connected via the logical operators AND, OR, and NOT and is defined recursively by Rules 1 and 2 as follows:
  - Rule 1: Every atom is a formula.
  - Rule 2: If F1 and F2 are formulas, then so are (F1 AND F 2), (F1 OR F 2), NOT (F1).
    a. (F1 AND F 2) is TRUE if both F1 and F2 are TRUE; otherwise, it is FALSE.
    b. (F1 OR F 2) is FALSE if both F1 and F2 are FALSE; otherwise, it is TRUE.
    c. NOT (F1) is TRUE if F1 is FALSE; it is FALSE if F1 is TRUE.
    d. NOT (F2) is TRUE if F2 is FALSE; it is FALSE if F2 is TRUE.

1.6.3 The Existential and Universal Quantifiers

- **Quantifiers** are two types, they are the **universal quantifier (∀)** and the **existential quantifier (∃)**.
- We need to define the concepts of **free** and **bound tuple variables** in a formula.
- Informally, a tuple variable t is **bound if it is quantified**, meaning that it appears in an **(∃t)** or **(∀t)** clause; **otherwise, it is free.**
- we define a tuple variable in a formula as **free** or **bound** according to the following rules:
  - An occurrence of a tuple variable in a formula F that is **an atom is free in F**.
  - An occurrence of a tuple variable t is free or bound in a formula made up of logical connectives — (F1 **AND** F 2), (F1 **OR** F 2), **NOT**(F1), and **NOT**(F2) -- depending on whether it is free or bound in F1 or F2 (if it occurs in either).
- **Truth values** for formulas with quantifiers are described below-
  - The formula (∃t)(F) is TRUE if the formula **F evaluates to TRUE for some (at least one) tuple assigned to free occurrences of t in F**; otherwise, (∃t)(F) is FALSE.
  - The formula (∀t)(F) is TRUE if the formula F evaluates to TRUE for every tuple (in the universe) assigned to free occurrences of t in F; otherwise, (∀t)(F) is FALSE.

1.6.4 Sample Queries in Tuple Relational Calculus

Query 1. List the name and address of all employees who work for the 'Research' department.

Ans: { t.Fname, t.Lname, t.Address | EMPLOYEE(t) AND (∃d)(DEPARTMENT(d) AND d .Dname='Research' AND d .Dnumber= t.Dno)}

Query 2. For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, birth date, and address.

Ans: { p.Pnumber, p.Dnum, m.Lname, m.Bdate, m.Address | PROJECT(p) AND EMPLOYEE(m) AND p .Plocation='Stafford' AND ((∃d)(DEPARTMENT(d) AND p .Dnum= d.Dnumber AND d .Mgr_ssn= m.Ssn))}

Query 3. List the name of each employee who works on **some** project controlled by department number 5. This is a variation of Q3 in which all is changed to **some**. In this case we need two join conditions and two existential quantifiers.

Ans: { e. Lname, e. Fname | EMPLOYEE(e) AND ((∃x)(∃w)(PROJECT(x) AND WORKS_ON(w) AND x.Dnum=5 AND w .Essn= e.Ssn AND x.Pnumber= w.Pno))}

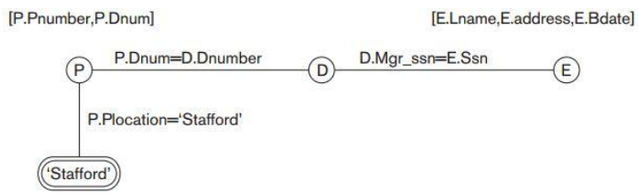
Query 4. Make a list of project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as manager of the controlling department for the project.

{ p.Pnumber | PROJECT(p) AND (((∃e)(∃w)(EMPLOYEE(e)
AND WORKS_ON(w) AND w .Pno= p.Pnumber
AND e.Lname='Smith' AND e.Ssn= w.Essn))
OR
((∃m)(∃d)(EMPLOYEE(m) AND DEPARTMENT(d)
AND p .Dnum= d.Dnumber AND d .Mgr_ssn= m.Ssn
AND m .Lname='Smith'))))}

1.6.5 Notation for Query Graphs

- Graphical representation of a query is called a query graph.
- F Relations in the query are represented by **relation nodes**, which are displayed as **single circles**.
- Constant values, typically from the query selection conditions, are represented by **constant nodes**, which are displayed as **double circles or ovals**.
- **Selection and join conditions are represented by the graph edges** (the lines that connect the nodes), as shown in below figure.
- The attributes to be retrieved from each relation are displayed in square brackets above each relation.
- Example: For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, birth date, and address.

Ans: { p.Pnumber, p.Dnum, m.Lname, m.Bdate, m.Address | PROJECT(p) AND EMPLOYEE(m) AND p .Plocation='Stafford' AND ((∃d)(DEPARTMENT(d) AND p .Dnum= d.Dnumber AND d .Mgr_ssn= m.Ssn))}



1.6.6 Transforming the Universal and Existential Quantifiers

- It is possible to transform a universal quantifier into an existential quantifier, and vice versa, to get an equivalent expression.
- One general transformation can be described informally as follows:
- Transform one type of quantifier into the other with negation (preceded by NOT); AND and OR replace one another; a negated formula becomes unnegated; and an unnegated formula becomes negated.
- Some special cases of this transformation can be stated as follows, where the ≡ symbol stands for equivalent to:

(∀x) (P(x)) ≡ NOT (∃x) (NOT (P(x)))

(∃x) (P(x)) ≡ NOT (∀x) (NOT (P(x)))

(∀x) (P(x) AND Q (x)) ≡ NOT (∃x) (NOT (P(x)) OR NOT (Q(x)))

(∀x) (P(x) OR Q (x)) ≡ NOT (∃x) (NOT (P(x)) AND NOT (Q(x)))

(∃x) (P(x)) OR Q (x)) ≡ NOT (∀x) (NOT (P(x)) AND NOT (Q(x)))

(∃x) (P(x) AND Q (x)) ≡ NOT (∀x) (NOT (P(x)) OR NOT (Q(x)))

Notice also that the following is TRUE, where the ⇒ symbol stands for implies:

(∀x)(P(x)) ⇒ (∃x)(P(x))

NOT (∃x)(P(x)) ⇒ NOT (∀x)(P(x))

1.6.7 Using the Universal Quantifier in Queries

Query 3. List the names of employees who work on all the projects controlled by department number 5.

One way to specify this query is to use the universal quantifier as shown:

Ans: { e. Lname, e. Fname | EMPLOYEE(e) AND ((∀x)(NOT(PROJECT(x)) OR NOT (x. Dnum=5) OR ((∃w)(WORKS_ON(w) AND w .Essn= e.Ssn AND x .Pnumber= w. Pno))))}

Query 6. List the names of employees who have no dependents.

Ans: { e.Fname, e.Lname | EMPLOYEE(e) AND (NOT (∃d)(DEPENDENT(d) AND e .Ssn= d.Essn))}

Using the general transformation rule, we can rephrase Q6 as follows:

Ans: { e.Fname, e.Lname | EMPLOYEE(e) AND ((∀d)(NOT(DEPENDENT(d)) OR NOT(e.Ssn= d.Essn)))}

Query 7. List the names of managers who have at least one dependent.

Ans: { e.Fname, e.Lname | EMPLOYEE(e) AND ((∃d)(∃ρ)(DEPARTMENT(d) AND DEPENDENT(ρ) AND e .Ssn= d.Mgr_ssn AND ρ.Essn= e.Ssn))}

This query is handled by interpreting managers who have at least one dependent as managers for whom there exists some dependent.

1.6.8 Safe Expressions

- A **safe expression** in relational calculus is one that is guaranteed to yield a finite number of tuples as its result; otherwise, the expression is called **unsafe**.
- For example, the expression

  {t | NOT (EMPLOYEE(t))}
- Above expression is **unsafe** because it yields **all tuples in the universe that are not EMPLOYEE tuples**, which are infinitely numerous.