UNIT-II
Chapter -I

Syllabus:

Formal Relational Query Languages: The Relational Algebra, The Tuple Relational Calculus, The Domain Relational Calculus.

Relational Algebra

- Relational algebra is a procedural query language. It consists of a set of operations that take one or two relations as input and produce a new relation as their result.
- The fundamental operations in relational algebra are select, project, union, set difference, Cartesian product, and rename.
- In addition to the fundamental operations, there are several other operations namely, set intersection, natural join, and assignment.

Fundamental Operations

- The select, project, and rename operations are called unary operations because they operate on one relation.
- The other three operations operate on pairs of relations and are, therefore, called binary operations.

1. **The Select Operation**

   The select operation selects tuples that satisfy a given predicate. We use the lowercase Greek letter sigma (σ) to denote selection. The predicate appears as a subscript to σ. The argument relation is in parentheses after the σ.

   Notation: **σ p(r)**

   Where:

   σ is used for selection prediction
   r is used for relation
   p is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like =, ≠, ≥, <, >, ≤.

   Examples:

   i. Select the tuples of the instructor relation where the instructor is in the Physics department.
   we write:

$$\sigma_{dept\_name = \text{"Physics"}}(instructor)$$

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Fig. The *instructor* relation

The result of the above selection statement is:

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

ii. Find all instructors with salary greater than $90,000.

$$\sigma_{salary > 90000}\ (instructor)$$

iii. Find instructors in Physics with a salary greater than $90,000

$$\sigma_{dept\_name = \text{“Physics”} \wedge salary > 90000}\ (instructor)$$

2. The Project Operation

The project operation is a unary operation that returns the relation with required attributes and the rest of the attributes are eliminated from the relation. Projection is denoted by the uppercase Greek letter pi (∏). We list those attributes that we wish to appear in the result as a subscript to ∏. The argument relation follows in parentheses.

Notation: ∏ A1, A2, A3 (r)

Where **A1**, **A2**, **A3** are used as an attribute-name of relation **r**.

Examples:

i. Display the ID, name, and salary attributes from instructor relation

$$\Pi_{ID,\ name,\ salary}(instructor)$$

The result of above projection statement is

ID	*name*	*salary*
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

ii. Find the name of all instructors in the Physics department.

$$\Pi_{name}\ (\sigma_{dept\_name = \text{“Physics”}}\ (instructor))$$

3. The Union Operation

The Union operation is a binary operation. It is denoted by U. *R U S* returns a relation instance containing all tuples that occur in *either* relation instance *R* or relation instance *S* (or both). *R* and *S* must be *union-compatible*, and the schema of the result is defined to be identical to the schema of *R*. The union operation R ∪ S to be valid, we require that two conditions hold:

i. The relations R and S must be of the same arity. That is, they must have the same number of attributes.
ii. The domains of the i^{th} attribute of R and the i^{th} attribute of S must be the same, for all i.

Example:

Find the set of all courses taught in the Fall 2009 semester, the Spring 2010 semester, or both.

$$\Pi_{course\_id}\ (\sigma_{semester = \text{“Fall”} \land year=2009}\ (section)) \cup$$
$$\Pi_{course\_id}\ (\sigma_{semester = \text{“Spring”} \land year=2010}\ (section))$$

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

Fig. The *Section* relation

The result of above statement is

course_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

4. The Set-Difference Operation

The set-difference operation, denoted by −, allows us to find tuples that are in one relation but are not in another. The expression R − S produces a relation containing those tuples in R but not in S. As with the union operation, we must ensure that set differences are taken between *compatible* relations.

Example: Find all the courses taught in the Fall 2009 semester but not in Spring 2010 semester.

$$\Pi_{course\_id}\ (\sigma_{semester = \text{“Fall”} \land year=2009}\ (section))\ -$$
$$\Pi_{course\_id}\ (\sigma_{semester = \text{“Spring”} \land year=2010}\ (section))$$

The result of the above statement is

course_id
CS-347
PHY-101

5. The Cartesian-Product Operation

The Cartesian-product operation, denoted by a cross (×), allows us to combine information from any two relations. We write the Cartesian product of relations R and S as R × S. If there are n1 tuples in relation R and n2 tuples in relation S then R X S contains n1*n2 tuples. *R* × *S* returns a relation instance whose schema contains all the fields of *R* (in the same order as they appear in *R*) followed by all the fields of *S* (in the same order as they appear in *S*).

6. The Rename Operation

- The **rename** operator, denoted by the lowercase Greek letter rho (***ρ***) is used to rename the output relation. Given a relational-algebra expression *E*, the expression $\rho_x(E)$ returns the result of expression *E* under the name *x*.
- A relation *r* by itself is considered a (trivial) relational-algebra expression. Thus, we can also apply the rename operation to a relation *r* to get the same relation under a new name.
- A second form of the rename operation is as follows:
  Assume that a relational algebra expression *E* has arity *n*. Then, the expression $\rho_{x(A1, A2,...,An)}(E)$ returns the result of expression *E* under the name *x*, and with the attributes renamed to *A*1, *A*2, . . . , *An*.

Additional Relational-Algebra Operations

1. The Set-Intersection Operation

- The Intersection operation is a binary operation. It is denoted by ∩. *R ∩ S* returns a relation instance containing all tuples that occur in both relation instance *R* and relation instance *S*. *R* and *S* must be *union-compatible.*
- Note that we can rewrite any relational-algebra expression that uses set intersection by replacing the intersection operation with a pair of set-difference operations as: $r \cap s = r - (r - s)$

Example: Find the set of all courses taught in both the Fall 2009 and the Spring 2010 semesters.

$$\Pi_{course\_id}\ (\sigma_{semester = \text{“Fall”} \wedge year=2009}\ (section)) \cap \\ \Pi_{course\_id}\ (\sigma_{semester = \text{“Spring”} \wedge year=2010}\ (section))$$

2. The Natural-Join Operation

The *natural join* is a binary operation that allows us to combine certain selections and a Cartesian product into one operation. It is denoted by the **join** symbol. $\bowtie$ The natural-join operation forms a Cartesian product of its two arguments, performs selection-forcing equality on those attributes that appear in both relation schemas, and finally removes duplicate attributes.

Example: Find the names of all instructors together with the *course id* of all courses they taught.

$$\Pi_{name,\ course\_id}\ (instructor\ \bowtie\ teaches)$$

Since the schemas for *instructor* and *teaches* have the attribute *ID* in common, the natural-join operation considers only pairs of tuples that have the same value on *ID*. It combines each such pair of tuples into a single tuple on the union of the two schemas; that is, (*ID*, *name*, *dept name*, *salary*, *course id*).

- The ***theta join*** operation is a variant of the natural-join operation that allows us to combine a selection and a Cartesian product into a single operation. Consider relations $r\ (R)$ and $s(S)$, and let Θ be a predicate on attributes in the schema $R \cup S$. The **theta join** operation r s is defined $\bowtie_\theta$ as follows:

$$r \bowtie_\theta s = \sigma_\theta(r \times s)$$

3. The Assignment Operation

The **assignment** operation, denoted by ←, works like an assignment in a programming language. We could write r s $\bowtie$ as:

$$\begin{aligned} temp1 &\leftarrow R \times S \\ temp2 &\leftarrow \sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n}(temp1) \\ result &= \Pi_{R \cup S}(temp2) \end{aligned}$$

The evaluation of an assignment does not result in any relation being displayed to the user. Rather, the result of the expression to the right of the ← is assigned to the relation variable on the left of the←. This relation variable may be used in subsequent expressions.

4. Outer join Operations

- The **outer join** operation works in a manner similar to the natural join operation but preserves those tuples that would be lost in a join by creating tuples in the result containing null values.
- There are actually three forms of the operation: *left outer join*, denoted ⟕; *right outer join*, denoted ⟖ ; and *full outer join*, denoted ⟗ . All three forms of outer join compute the join, and add extra tuples to the result of the join.
- The **left outer join** (⟕) takes all tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation, and adds them to the result of the natural join. All information from the left relation is present in the result of the left outer join.
- The **right outer join** (⟖) is symmetric with the left outer join: It pads tuples from the right relation that did not match any from the left relation with nulls and adds them to the result of the natural join. All information from the right relation is present in the result of the right outer join.
- The **full outer join** (⟗) does both the left and right outer join operations, padding tuples from the left relation that did not match any from the right relation, as well as tuples from the right relation that did not match any from the left relation, and adding them to the result of the join.

Extended Relational-Algebra Operations

Extended relational-algebra operations provide the ability to write queries that cannot be expressed using the basic relational algebra operations. There are two extended relational algebra operations: Generalized Projection and Aggregation.

1. **Generalized Projection:**

- It extends the projection operation by allowing operations such as arithmetic and string functions to be used in the projection list.

- The generalized projection operation has the form:

$$\Pi_{F_1, F_2, \ldots, F_n}(E)$$

  Where E is any relational-algebra expression, and each of F1, F2, . . . , Fn is an arithmetic expression involving constants and attributes in the schema of E.

- Example: $$\Pi_{ID, name, dept\_name, salary \div 12}(instructor)$$

  gives the *ID, name, dept name,* and the monthly salary of each instructor

2. **Aggregation:**

- The second extended relational-algebra operation is the aggregate operation *G*, which permits the use of aggregate functions such as min, max, sum, average, or count on sets of values.
- The general form of the aggregation operation $\mathcal{G}$ is as follows:

  $_{G1, G2, \ldots, Gn}\mathcal{G}_{F1(A1),\ F2(A2), \ldots,\ Fm(Am)}(E)$
- where *E* is any relational-algebra expression; *G*1, *G*2, . . . , *Gn* constitute a list of attributes on which to group; each *Fi* is an aggregate function; and each *Ai* is an attribute name.
- The meaning of the operation is as follows:

  The tuples in the result of expression *E* are partitioned into groups in such a way that:
  1. All tuples in a group have the same values for *G*1, *G*2, . . . , *Gn*.
  2. Tuples in different groups have different values for *G*1, *G*2, . . . , *Gn*.

Examples:

i. Find out the sum of salaries of all instructors

$\mathcal{G}_{\textbf{sum}(salary)}(instructor\ )$

ii. Find the total number of instructors who teach a course in the Spring 2010 semester.

$\mathcal{G}_{\textbf{count-distinct}(ID)}(\_{semester = \text{“Spring”} \land year = 2010}(teaches))$

iii. Find the average salary in each department

$_{dept\ name}\mathcal{G}_{\textbf{average}(salary)}(instructor)$

Relational Calculus

- Relational calculus is an alternative to relational algebra. In contrast to algebra, which is procedural, calculus is nonprocedural, or *declarative.*
- Relational calculus has had a big influence on the design of commercial query languages such as SQL and, especially, Query-by-Example (QBE).
- The variant of the calculus that we present in detail is called the tuple relational calculus (TRC). Variables in TRC take on tuples as values.
- In another variant, called the domain relational calculus (DRC), the variables range over field values.
- TRC has had more of an influence on SQL, while DRC has strongly influenced QBE.

Tuple Relational Calculus

- A tuple variable is a variable that takes on tuples of a particular relation schema as values. That is, every value assigned to a given tuple variable has the same number and type of fields.
- A tuple relational calculus query has the form ***{ T | p(T) },*** where *T* is a tuple variable and *p*(*T*) denotes a *formula* that describes *T* .
- The result of this query is the set of all tuples *t* for which the formula *p*(*T*) evaluates to true with *T* = *t*.
- A tuple variable is said to be a *free variable* unless it is quantified by a ∃ or ∀.

$$t \in instructor \land \exists\, s \in department(t[dept\_name] = s[dept\_name])$$

- In the above example, t is a free variable. Tuple variable s is said to be a bound variable.
- A tuple-relational-calculus formula is built up out of *atoms*. An atom has one of the following forms:
  - s∈ *r*, where *s* is a tuple variable and *r* is a relation.
  - *s*[*x*] *Θ u*[*y*], where *s* and *u* are tuple variables, *x* is an attribute on which *s* is defined, *y* is an attribute on which *u* is defined, and *Θ* is a comparison operator (<, ≤, =, _=, >, ≥).
  - *s*[*x*] *Θ c*, where *s* is a tuple variable, *x* is an attribute on which *s* is defined, *Θ* is a comparison operator, and *c* is a constant in the domain of attribute *x*.
- We build up formulae from atoms by using the following rules:
  - An atom is a formula.
  - If *P*1 is a formula, then so are ¬*P*1 and (*P*1).
  - If *P*1 and *P*2 are formulae, then so are *P*1 ∨ *P*2, *P*1 ∧ *P*2, and *P*1 ⇒ *P*2.
  - If *P*1(*s*) is a formula containing a free tuple variable *s*, and *r* is a relation, then ∃ *s* ∈ *r* (*P*1(*s*)) and ∀ *s* ∈ *r* (*P*1(*s*)).

  are also formulae.
- In the tuple relational calculus, the following equivalences are available:

1. $P1 \land P2$ is equivalent to $\neg(\neg(P1) \lor \neg(P2))$.

2. $\forall\, t \in r\,(P1(t))$ is equivalent to $\neg\,\exists\, t \in r\,(\neg P1(t))$.

3. $P1 \Rightarrow P2$ is equivalent to $\neg(P1) \lor P2$.

- If a TRC expression returns the finite number of tuples then we say that is safe otherwise unsafe.

Domain Relational Calculus

- A domain variable is a variable that ranges over the values in the domain of some attribute (e.g., the variable can be assigned an integer if it appears in an attribute whose domain is the set of integers).
- A DRC query has the form $\{\langle x_1, x_2, \ldots, x_n\rangle \mid p(\langle x_1, x_2, \ldots, x_n\rangle)\}$, where each x_i is either a *domain variable* or a constant and $p(\langle x_1, x_2, \ldots, x_n\rangle)$ denotes a DRC formula whose only free variables are the variables among the x_i, $1 \le i \le n$. The result of this query is the set of all tuples $\langle x_1, x_2, \ldots, x_n\rangle$ for which the formula evaluates to true.
- A DRC formula is defined in a manner that is very similar to the definition of a TRC formula. The main difference is that the variables are now domain variables. Let op denote an operator in the set $\{<, >, =, \le, \ge, =\}$ and let X and Y be domain variables.
- An atom in the domain relational calculus has one of the following forms:
  • $< x1, x2, \ldots, xn > \in r$, where r is a relation on n attributes and $x1, x2, \ldots, xn$ are domain variables or domain constants.
  • $x\ \Theta\ y$, where x and y are domain variables and Θ is a comparison operator ($<, \le, =, !=, >, \ge$). We require that attributes x and y have domains that can be compared by Θ.
  • $x\ \Theta\ c$, where x is a domain variable, Θ is a comparison operator, and c is a constant in the domain of the attribute for which x is a domain variable.
- We build up formulae from atoms by using the following rules:
  • An atom is a formula.
  • If $P1$ is a formula, then so are $\neg P1$ and $(P1)$.
  • If $P1$ and $P2$ are formulae, then so are $P1 \lor P2$, $P1 \land P2$, and $P1 \Rightarrow P2$.
  • If $P1(x)$ is a formula in x, where x is a free domain variable, then $\exists\, x\,(P1(x))$ and $\forall\, x\,(P1(x))$ are also formulae.
- **Examples:**

1. Find the instructor *ID, name, dept name,* and *salary* for instructors whose salary is greater than $80,000.

   $\{< i, n, d, s > \mid < i, n, d, s > \in instructor \land s > 80000\}$

2. Find all instructor *ID* for instructors whose salary is greater than $80,000.

   $\{< n > \mid \exists\, i, d, s\,(< i, n, d, s > \in instructor \land s > 80000)\}$

3. Find the set of all courses taught in the Fall 2009 semester, the Spring 2010 semester, or both:

   $\{< c > \mid \exists\, s\,(< c, a, s, y, b, r, t > \in section \land s = \text{“Fall”} \land y = \text{“2009”}$
   $\lor \exists u\,(< c, a, s, y, b, r, t > \in section$

$\wedge\ s =$ "Spring" $\wedge\ y =$ "2010"

4. Find all students who have taken all courses offered in the Biology department:

$\{< i > \mid \exists\ n, d, t\ (< i, n, d, t > \in student) \wedge$

$\forall\ x, y, z, w\ (< x, y, z, w > \in course \wedge z =$ "Biology" $\Rightarrow$

$\exists\ a, b\ (< a, x, b, r, p, q > \in takes \wedge < c, a > \in depositor\ ))\}$