***Relational Query Languages***

* ***Query languages:*** Allow manipulation and retrieval of data from a database.
* Query Languages is not a programming languages
  + QLs not intended to be used for complex calculations.
  + QLs support easy, efficient access to large data sets.
* Relational model supports simple, powerful query languages.

Formal Relational Query Languages

* There are varieties of Query languages used by relational DBMS for manipulating relations.
* Some of them are **procedural**
  + User tells the system exactly ***what*** and ***how*** to manipulate the data
* Others are **non-procedural**
  + User states ***what*** data is needed rather than ***how*** it is to be retrieved.

Two mathematical Query Languages form the basis for Relational languages

* + - *Relational Algebra*:
    - *Relational Calculus*:
* We may describe the ***relational algebra as procedural language***: it can be used to tell the DBMS how to build a new relation from one or more relations in the database.
* We may describe ***relational calculus as a non procedural language***: it can be used to formulate the definition of a relation in terms of one or more database relations.
* Formally the relational algebra and relational calculus are equivalent to each other.
* ***For every expression in the algebra, there is an equivalent expression in the calculus.***
* Both are non-user friendly languages. They have been used as the basis for other, higher-level data manipulation languages for relational databases.

***A query is applied to relation instances, and the result of a query is also a relation instance.***

* + *Schemas* of input relations for a query are fixed
  + The schema for the *result* of a given query is also fixed! Determined by definition of query language constructs.

***Relational Algebra***

The basic set of operations for the relational model is known as the relational algebra. These operations enable a user to specify basic retrieval requests.

The result of the retrieval is a new relation, which may have been formed from one or more relations. The **algebra operations** thus produce new relations, which can be further manipulated using operations of the same 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 que ry (or retrieval request).

* Relational algebra is a theoretical language with operations that work on one or more relations to define another relation without changing the original relation.
* The output from one operation can become the input to another operation (nesting is possible)
* **There are different basic operations that could be applied on relations on a database based on the requirement.** 
  + ***Selection* ( σ )** Selects a subset of rows from a relation.
  + ***Projection ( π )*** Deletes unwanted columns from a relation.
  + ***Renaming:*** assigningintermediate relation for a single operation
  + ***Cross-Product ( x )*** Allows us to combine two relations.
  + ***Set-Difference ( - )*** Tuples in relation1, but not in relation2.
  + ***Union (∪ )*** Tuples in relation1 or in relation2.
  + ***Intersection (∩)*** Tuples in relation1 and in relation2
  + ***Join***  Tuples joined from two relations based on a condition
* Using these we can build up sophisticated database queries.

***Table1:***

*Sample table used to illustrate different kinds of relational operations. The relation contains information about employees, IT skills they have and the school where they attend each skill. The primary key for this table is EmpId and Skill ID since a single employee can have multiple skills and a single skill be acquired by many employees.*

*School address is the address of a school for which the address of the main office will be considered in cases where a single school has many branches at different locations.*

Employee

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***EmpID*** | ***FName*** | ***LName*** | ***SkillID*** | ***Skill*** | ***SkillType*** | ***School*** | ***SchoolAdd*** | ***SkillLevel*** |
| 12 | Abebe | Mekuria | 2 | SQL | Database | AAU | Sidist\_Kilo | 5 |
| 16 | Lemma | Alemu | 5 | C++ | Programming | Unity | Gerji | 6 |
| 28 | Chane | Kebede | 2 | SQL | Database | AAU | Sidist\_Kilo | 10 |
| 25 | Abera | Taye | 6 | VB6 | Programming | Helico | Piazza | 8 |
| 65 | Almaz | Belay | 2 | SQL | Database | Helico | Piazza | 9 |
| 24 | Dereje | Tamiru | 8 | Oracle | Database | Unity | Gerji | 5 |
| 51 | Selam | Belay | 4 | Prolog | Programming | Jimma | Jimma City | 8 |
| 94 | Alem | Kebede | 3 | Cisco | Networking | AAU | Sidist\_Kilo | 7 |
| 18 | Girma | Dereje | 1 | IP | Programming | Jimma | Jimma City | 4 |
| 13 | Yared | Gizaw | 7 | Java | Programming | AAU | Sidist\_Kilo | 6 |

1. ***Selection***

* Selects subset of tuples/rows in a relation that satisfy ***selection condition***.
* Selection operation is a unary operator (it is applied to a single relation)
* The Selection operation is applied to each tuple individually
* The degree of the resulting relation is the same as the original relation but the cardinality (no. of tuples) is less than or equal to the original relation.
* The Selection operator is commutative.
* Set of conditions can be combined using Boolean operations (**∧(AND)**, **∨(OR**), and **~(NOT)**)
* No duplicates in result!
* *Schema* of result identical to schema of (only) input relation.
* *Result* relation can be the *input* for another relational algebra operation! (*Operator* *composition.*)
* It is a filter that keeps only those tuples that satisfy a qualifying condition (those satisfying the condition are selected while others are discarded.)

***Notation:***

*** <Selection Condition> <Relation Name>***

**Example**: Find all Employees with skill type of Database.

*** < SkillType =”Database”> (Employee)***

This query will extract every tuple from a relation called Employee with all the attributes where the SkillType attribute with a value of “Database”.

The resulting relation will be the following.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***EmpID*** | ***FName*** | ***LName*** | ***SkillID*** | ***Skill*** | ***SkillType*** | ***School*** | ***SchoolAdd*** | ***SkillLevel*** |
| 12 | Abebe | Mekuria | 2 | SQL | Database | AAU | Sidist\_Kilo | 5 |
| 28 | Chane | Kebede | 2 | SQL | Database | AAU | Sidist\_Kilo | 10 |
| 65 | Almaz | Belay | 2 | SQL | Database | Helico | Piazza | 9 |
| 24 | Dereje | Tamiru | 8 | Oracle | Database | Unity | Gerji | 5 |

If the query is all employees with a SkillType *Database* and School *Unity* the relational algebra operation and the resulting relation will be as follows.

*** < SkillType =”Database” AND School=”Unity”> (Employee)***

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***EmpID*** | ***FName*** | ***LName*** | ***SkillID*** | ***Skill*** | ***SkillType*** | ***School*** | ***SchoolAdd*** | ***SkillLevel*** |
| 24 | Dereje | Tamiru | 8 | Oracle | Database | Unity | Gerji | 5 |

1. ***Projection***

* Selects certain attributes while discarding the other from the base relation.
* The PROJECT creates a vertical partitioning – one with the needed columns (attributes) containing results of the operation and other containing the discarded Columns.
* Deletes attributes that are not in *projection list*.
* *Schema* of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
* Projection operator has to eliminate *duplicates*!
  + Note: real systems typically don’t do duplicate elimination unless the user explicitly asks for it.
* If the Primary Key is in the *projection* *list*, then duplication will not occur
* Duplication removal is necessary to insure that the resulting table is also a relation.

***Notation:***

*** <Selected Attributes> <Relation Name>***

**Example**: To display Name, Skill, and Skill Level of an employee, the query and the resulting relation will be:

*** <FName, LName, Skill, Skill\_Level> (Employee)***

|  |  |  |  |
| --- | --- | --- | --- |
| ***FName*** | ***LName*** | ***Skill*** | ***SkillLevel*** |
| Abebe | Mekuria | SQL | 5 |
| Lemma | Alemu | C++ | 6 |
| Chane | Kebede | SQL | 10 |
| Abera | Taye | VB6 | 8 |
| Almaz | Belay | SQL | 9 |
| Dereje | Tamiru | Oracle | 5 |
| Selam | Belay | Prolog | 8 |
| Alem | Kebede | Cisco | 7 |
| Girma | Dereje | IP | 4 |
| Yared | Gizaw | Java | 6 |

If we want to have the Name, Skill, and Skill Level of an employee with Skill SQL and SkillLevel greater than 5 the query will be:

***<FName, LName, Skill, Skill\_Level> (<****Skill****=”SQL” ∧ SkillLevel>5>(Employee))***

|  |  |  |  |
| --- | --- | --- | --- |
| ***FName*** | ***LName*** | ***Skill*** | ***SkillLevel*** |
| Chane | Kebede | SQL | 10 |
| Almaz | Belay | SQL | 9 |

1. ***Rename Operation***

* We may want to apply several relational algebra operations one after the other. The query could be written in two different forms:

1. Write the operations as a single relational algebra expression by nesting the operations.
2. Apply one operation at a time and create intermediate result relations. In the latter case, we must give names to the relations that hold the intermediate results🡺Rename Operation

If we want to have the Name, Skill, and Skill Level of an employee with salary greater than 1500 and working for department 5, we can write the expression for this query using the two alternatives:

1. A single algebraic expression:

The above used query is using a single algebra operation, which is:

***<FName, LName, Skill, Skill\_Level> (<****Skill****=”SQL” ∧ SkillLevel>5>(Employee))***

1. Using an intermediate relation by the Rename Operation:

***Step1: Result1 🡨 <DeptNo=5 ∧ Salary>1500>(Employee)***

***Step2: Result🡨 <FName, LName, Skill, Skill\_Level>(Result1)***

Then Result will be equivalent with the relation we get using the first alternative.

1. ***Set Operations***

The three main set operations are the Union, Intersection and Set Difference. The properties of these set operations are similar with the concept we have in mathematical set theory. The difference is that, in database context, the elements of each set, which is a Relation in Database, will be tuples. The set operations are Binary operations which demand the two operand Relations to have type compatibility feature.

**Type Compatibility**

Two relations R1 and R2 are said to be Type Compatible if:

* + 1. The operand relations R1(A1, A2, ..., An) and R2(B1, B2, ..., Bn) have the same number of attributes, and
    2. The domains of corresponding attributes must be compatible; that is, Dom(Ai)=Dom(Bi) for i=1, 2, ..., n.

To illustrate the three set operations, we will make use of the following two tables:

Employee

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***EmpID*** | ***FName*** | ***LName*** | ***SkillID*** | ***Skill*** | ***SkillType*** | ***School*** | ***SkillLevel*** |
| 12 | Abebe | Mekuria | 2 | SQL | Database | AAU | 5 |
| 16 | Lemma | Alemu | 5 | C++ | Programming | Unity | 6 |
| 28 | Chane | Kebede | 2 | SQL | Database | AAU | 10 |
| 25 | Abera | Taye | 6 | VB6 | Programming | Helico | 8 |
| 65 | Almaz | Belay | 2 | SQL | Database | Helico | 9 |
| 24 | Dereje | Tamiru | 8 | Oracle | Database | Unity | 5 |
| 51 | Selam | Belay | 4 | Prolog | Programming | Jimma | 8 |
| 94 | Alem | Kebede | 3 | Cisco | Networking | AAU | 7 |
| 18 | Girma | Dereje | 1 | IP | Programming | Jimma | 4 |
| 13 | Yared | Gizaw | 7 | Java | Programming | AAU | 6 |

***RelationOne:*** *Employees who attend Database Course*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***EmpID*** | ***FName*** | ***LName*** | ***SkillID*** | ***Skill*** | ***SkillType*** | ***School*** | ***SkillLevel*** |
| 12 | Abebe | Mekuria | 2 | SQL | Database | AAU | 5 |
| 28 | Chane | Kebede | 2 | SQL | Database | AAU | 10 |
| 65 | Almaz | Belay | 2 | SQL | Database | Helico | 9 |
| 24 | Dereje | Tamiru | 8 | Oracle | Database | Unity | 5 |

***RelationTwo :*** *Employees who attend a course in AAU*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***EmpID*** | ***FName*** | ***LName*** | ***SkillID*** | ***Skill*** | ***SkillType*** | ***School*** | ***SkillLevel*** |
| 12 | Abebe | Mekuria | 2 | SQL | Database | AAU | 5 |
| 94 | Alem | Kebede | 3 | Cisco | Networking | AAU | 7 |
| 28 | Chane | Kebede | 2 | SQL | Database | AAU | 10 |
| 13 | Yared | Gizaw | 7 | Java | Programming | AAU | 6 |

* 1. ***UNION Operation***

The result of this operation, denoted by R U S, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuple is eliminated.

The two operands must be *"type compatible"*

***Eg: RelationOne* U** ***RelationTwo***

*Employees who attend Database in any School or who attend any course at AAU*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***EmpID*** | ***FName*** | ***LName*** | ***SkillID*** | ***Skill*** | ***SkillType*** | ***School*** | ***SkillLevel*** |
| 12 | Abebe | Mekuria | 2 | SQL | Database | AAU | 5 |
| 28 | Chane | Kebede | 2 | SQL | Database | AAU | 10 |
| 65 | Almaz | Belay | 2 | SQL | Database | Helico | 9 |
| 24 | Dereje | Tamiru | 8 | Oracle | Database | Unity | 5 |
| 94 | Alem | Kebede | 3 | Cisco | Networking | AAU | 7 |
| 13 | Yared | Gizaw | 7 | Java | Programming | AAU | 6 |

* 1. ***INTERSECTION Operation***

The result of this operation, denoted by R ∩ S, is a relation that includes all tuples that are in both R and S. The two operands must be *"type compatible"*

***Eg: RelationOne* ∩** ***RelationTwo***

*Employees who attend Database Course at AAU*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***EmpID*** | ***FName*** | ***LName*** | ***SkillID*** | ***Skill*** | ***SkillType*** | ***School*** | ***SkillLevel*** |
| 12 | Abebe | Mekuria | 2 | SQL | Database | AAU | 5 |
| 28 | Chane | Kebede | 2 | SQL | Database | AAU | 10 |

* 1. ***Set Difference (or MINUS) Operation***

The result of this operation, denoted by R - S, is a relation that includes all tuples that are in R but not in S.

The two operands must be *"type compatible"*

***Eg: RelationOne -*** ***RelationTwo***

*Employees who attend Database Course but didn’t take any course at AAU*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***EmpID*** | ***FName*** | ***LName*** | ***SkillID*** | ***Skill*** | ***SkillType*** | ***School*** | ***SkillLevel*** |
| 65 | Almaz | Belay | 2 | SQL | Database | Helico | 9 |
| 24 | Dereje | Tamiru | 8 | Oracle | Database | Unity | 5 |

***Eg: RelationTwo -*** ***RelationOne***

*Employees who attend Database Course but didn’t take any course at AAU*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***EmpID*** | ***FName*** | ***LName*** | ***SkillID*** | ***Skill*** | ***SkillType*** | ***School*** | ***SkillLevel*** |
| 12 | Abebe | Mekuria | 2 | SQL | Database | AAU | 5 |
| 94 | Alem | Kebede | 3 | Cisco | Networking | AAU | 7 |
| 28 | Chane | Kebede | 2 | SQL | Database | AAU | 10 |
| 13 | Yared | Gizaw | 7 | Java | Programming | AAU | 6 |

The resulting relation for; R1 ∪ R2, R1 ∩ R2, or R1-R2 has the same attribute names as the first operand relation R1 (by convention).

***Some Properties of the Set Operators***

Notice that both union and intersection are commutative operations; that is

***R ∪ S = S ∪ R, and R ∩ S = S ∩ R***

Both union and intersection can be treated as n-nary operations applicable to any number of relations as both are associative operations; that is

***R ∪ (S ∪ T) = (R ∪ S) ∪ T, and (R ∩ S) ∩ T = R ∩ (S ∩ T)***

The minus operation is not commutative; that is, in general

***R - S ≠ S – R***

1. ***CARTESIAN Operation (Cross Product)***

This operation is used to combine tuples from two relations in a combinatorial fashion. That means, every tuple in Relation1(R) one will be related with every other tuple in Relation2 (S).

* In general, the result of ***R(A1, A2, . . ., An) x S(B1,B2, . . ., Bm)*** is a relation ***Q*** with degree ***n + m*** attributes ***Q(A1, A2, . . ., An, B1, B2, . . ., Bm)***, in that order.
* Where ***R*** has ***n*** attributes and ***S*** has ***m*** attributes.
* The resulting relation ***Q*** has one tuple for each combination of tuples—one from R and one from S.
* Hence, if ***R*** has ***n*** tuples, and ***S*** has ***m*** tuples, then | R x S | will have ***n\* m*** tuples.

Example:

Employee

|  |  |  |
| --- | --- | --- |
| **ID** | **FName** | **LName** |
| 123 | Abebe | Lemma |
| 567 | Belay | Taye |
| 822 | Kefle | Kebede |

Dept

|  |  |  |
| --- | --- | --- |
| **DeptID** | **DeptName** | **MangID** |
| 2 | Finance | 567 |
| 3 | Personnel | 123 |

Then the Cartesian product between Employee and Dept relations will be of the form:

***Employee X Dept***:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **FName** | **LName** | **DeptID** | **DeptName** | **MangID** |
| 123 | Abebe | Lemma | 2 | Finance | 567 |
| 123 | Abebe | Lemma | 3 | Personnel | 123 |
| 567 | Belay | Taye | 2 | Finance | 567 |
| 567 | Belay | Taye | 3 | Personnel | 123 |
| 822 | Kefle | Kebede | 2 | Finance | 567 |
| 822 | Kefle | Kebede | 3 | Personnel | 123 |

Basically, even though it is very important in query processing, the Cartesian Product is not useful by itself since it relates every tuple in the First Relation with every other tuple in the Second Relation. Thus, to make use of the Cartesian Product, one has to use it with the Selection Operation, which discriminate tuples of a relation by testing whether each will satisfy the selection condition.

In our example, to extract employee information about managers of the departments (Managers of each department), the algebra query and the resulting relation will be.

***<****ID****, FName, LName, DeptName > (<ID=MangID>(Employee X Dept))***

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **FName** | **LName** | **DeptName** |
| 123 | Abebe | Lemma | Personnel |
| 567 | Belay | Taye | Finance |

1. ***JOIN Operation***

The sequence of Cartesian product followed by select is used quite commonly to identify and select related tuples from two relations, a special operation, called ***JOIN***. Thus in JOIN operation, the Cartesian Operation and the Selection Operations are used together.

JOIN Operation is denoted by a  symbol.

This operation is very important for any relational database with more than a single relation, because it allows us to process relationships among relations.

The general form of a join operation on two relations

R(A1, A2,. . ., An) and S(B1, B2, . . ., Bm) is:

R <***join*** ***condition***>S ***is equivalent to*** ***<selection condition>(R X S)***

***where <join condition>*** *and* ***<selection condition>*** *are the same*

Where, R and S can be any relation that results from general relational algebra expressions.

Since JOIN is an operation that needs two relation, it is a Binary operation.

This type of JOIN is called a ***THETA JOIN (θ - JOIN)***

***Where θ is the logical operator used in the join condition.***

***θ Could be { <, ≤ , >, ≥, ≠, = }***

***Example:***

Thus in the above example we want to extract employee information about managers of the departments, the algebra query using the JOIN operation will be.

***Employee***< ***ID=MangID***>***Dept***

* 1. ***EQUIJOIN Operation***

The most common use of join involves join conditions with equality comparisons only ( = ). Such a join, where the only comparison operator used is called an EQUIJOIN. In the result of an EQUIJOIN we always have one or more pairs of attributes (whose names need not be identical) that have identical values in every tuple since we used the equality logical operator.

***For example, the above JOIN expression is an EQUIJOIN since the logical operator used is the equal to operator ( =).***

* 1. ***NATURAL JOIN Operation***

We have seen that in EQUIJOIN one of each pair of attributes with identical values is extra, a new operation called **natural join** was created to get rid of the second (or extra) attribute that we will have in the result of an EQUIJOIN condition.

The standard definition of natural join requires that the two join attributes, or each pair of corresponding join attributes, have the same name in both relations. If this is not the case, a renaming operation on the attributes is applied first.

* 1. ***OUTER JOIN Operation***

OUTER JOIN is another version of the JOIN operation where non matching tuples from a relation are also included in the result with NULL values for attributes in the other relation.

There are two major types of OUTER JOIN.

1. ***RIGHT OUTER JOIN***: where non matching tuples from the second (Right) relation are included in the result with NULL value for attributes of the first (Left) relation.
2. ***LEFT OUTER JOIN***: where non matching tuples from the first (Left) relation are included in the result with NULL value for attributes of the second (Right) relation.

Notation for Left Outer Join:

R <Join Condition > S

When two relations are joined by a JOIN operator, there could be some tuples in the first relation not having a matching tuple from the second relation, and the query is interested to display these non matching tuples from the first or second relation. Such query is represented by the OUTER JOIN.

* 1. ***SEMIJOIN Operation***

SEMI JOIN is another version of the JOIN operation where the resulting Relation will contain those attributes of only one of the Relations that are related with tuples in the other Relation. The following notation depicts the inclusion of only the attributes form the first relation (R) in the result which are actually participating in the relationship.