1/19/2020

November Eleven

ID: ATR/7217/10

Department: ITSC

Section: IT

Subject: Database Administration

Abstract



**Relational Algebra and Relational Calculus**

**Introduction**

A **Relational Database management System**(RDBMS) is a database management system based on the relational model introduced by E.F Codd. In relational model, data is stored in **relations**(tables) and is represented in form of **tuples**(rows). **RDBMS** is used to manage Relational database. **Relational database** is a collection of organized set of tables related to each other, and from which data can be accessed easily. Relational Database is the most commonly used database these days.

* **Table** In Relational database model, a **table** is a collection of data elements organised in terms of rows and columns. A table is also considered as a convenient representation of **relations**. But a table can have duplicate row of data while a true **relation** cannot have duplicate data. Table is the most simplest form of data storage.
* **Tuple** A single entry in a table is called a **Tuple** or **Record** or **Row**. A **tuple** in a table represents a set of related data.
* **Attribute** A table consists of several records(row), each record can be broken down into several smaller parts of data known as **Attributes**. The above **Employee** table consist of four attributes, **ID**, **Name**, **Age** and **Salary**.
* **Attribute Domain** When an attribute is defined in a relation(table), it is defined to hold only a certain type of values, which is known as **Attribute Domain**.
* **Relation Schema** A relation schema describes the structure of the relation, with the name of the relation(name of table), its attributes and their names and type.
* **Relation Key** A relation key is an attribute which can uniquely identify a particular tuple(row) in a relation(table).

**Relational Integity Constraints**

Every relation in a relational database model should abide by or follow a few constraints to be a valid relation, these constraints are called as **Relational Integrity Constraints**.

The three main Integrity Constraints are:

1. Key Constraints
2. Domain Constraints
3. Referential integrity Constraints

* **Key Constraints** We store data in tables, to later access it whenever required. In every table one or more than one attributes together are used to fetch data from tables. The **Key Constraint** specifies that there should be such an attribute(column) in a relation(table), which can be used to fetch data for any tuple(row).The Key attribute should never be **NULL** or same for two different row of data.
* **Domain Constraint** Domain constraints refers to the rules defined for the values that can be stored for a certain attribute. We cannot store **Address** of employee in the column for **Name**. Similarly, a mobile number cannot exceed 10 digits.
* **Referential Integrity Constraint** If a table reference to some data from another table, then that table and that data should be present for referential integrity constraint to hold true.

**Procedural Query language:** In procedural query language, user instructs the system to perform a series of operations to produce the desired results. Here users tells what data to be retrieved from database and how to retrieve it.

**Non-procedural query language:** In Non-procedural query language, user instructs the system to produce the desired result without telling the step by step process. Here users tells what data to be retrieved from database but doesn’t tell how to retrieve it.

**Relational Algebra**

Every database management system must define a query language to allow users to access the data stored in the database. **Relational Algebra** is a procedural query language used to query the database tables to access data in different ways.

In relational algebra, input is a relation(table from which data has to be accessed) and output is also a relation(a temporary table holding the data asked for by the user).

Relational Algebra works on the whole table at once, so we do not have to use loops etc to iterate over all the rows(tuples) of data one by one. All we have to do is specify the table name from which we need the data, and in a single line of command, relational algebra will traverse the entire given table to fetch data for you.

**Basic/Fundamental Operations:**

1. Select (σ)
2. Project (∏)
3. Union (∪)
4. Set Difference (-)
5. Cartesian product (X)
6. Rename (ρ)

**Derived Operations:**

1. Natural Join (⋈)
2. Left, Right, Full outer join (⟕, ⟖, ⟗)
3. Intersection (∩)
4. Division (÷)

Reference Employee table:

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Name | Age | Salary |
| 1 | Adam | 34 | 13000 |
| 2 | Alex | 28 | 15000 |
| 3 | Stuart | 20 | 18000 |
| 4 | Ross | 42 | 19020 |

**Select Operation (σ)**

This is used to fetch rows(tuples) from table(relation) which satisfies a given condition.

**Syntax:** σp(r)

Where, σ represents the Select Predicate, r is the name of relation(table name in which you want to look for data), and p is the prepositional logic, where we specify the conditions that must be satisfied by the data. In prepositional logic, one can use **unary** and **binary** operators like =, <, > etc, to specify the conditions.

Let's take an example of the Student table we specified above in the Introduction of relational algebra, and fetch data for **students** with **age** more than 17.

σage > 17 (Student)

This will fetch the tuples(rows) from table **Student**, for which **age** will be greater than **17**.

You can also use, and, or etc operators, to specify two conditions, for example,

σage > 17 and gender = 'Male' (Student)

This will return tuples(rows) from table **Student** with information of male students, of age more than 17.(Consider the Student table has an attribute Gender too.)

**Project Operation (∏)**

Project operation is used to project only a certain set of attributes of a relation. In simple words, If you want to see only the **names** all of the students in the **Student** table, then you can use Project Operation.

It will only project or show the columns or attributes asked for, and will also remove duplicate data from the columns.

**Syntax:** ∏A1, A2...(r)

where A1, A2 etc are attribute names(column names).

For example,

∏Name, Age(Student)

Above statement will show us only the **Name** and **Age** columns for all the rows of data in **Student** table.

**Union Operation (∪)**

This operation is used to fetch data from two relations(tables) or temporary relation(result of another operation).

For this operation to work, the relations(tables) specified should have same number of attributes(columns) and same attribute domain. Also the duplicate tuples are autamatically eliminated from the result.

**Syntax:** A ∪ B

where A and B are relations.

For example, if we have two tables **RegularClass** and **ExtraClass**, both have a column **student** to save name of student, then,

∏Student(RegularClass) ∪ ∏Student(ExtraClass)

Above operation will give us name of **Students** who are attending both regular classes and extra classes, eliminating repetition.

**Set Difference (-)**

This operation is used to find data present in one relation and not present in the second relation. This operation is also applicable on two relations, just like Union operation.

**Syntax:** A - B

where A and B are relations.

For example, if we want to find name of students who attend the regular class but not the extra class, then, we can use the below operation:

∏Student(RegularClass) - ∏Student(ExtraClass)

**Cartesian Product (X)**

This is used to combine data from two different relations(tables) into one and fetch data from the combined relation.

**Syntax:** A X B

For example, if we want to find the information for Regular Class and Extra Class which are conducted during morning, then, we can use the following operation:

σtime = 'morning' (RegularClass X ExtraClass)

For the above query to work, both **RegularClass** and **ExtraClass** should have the attribute **time**.

**Rename Operation (ρ)**

This operation is used to rename the output relation for any query operation which returns result like Select, Project etc. Or to simply rename a relation(table)

**Syntax:** ρ(RelationNew, RelationOld)

Let us consider two tables named as **A** and **B Table B:**

**Table A:**

|  |  |  |
| --- | --- | --- |
| RollNo | Name | Marks |
| 1 | Henok | 98 |
| 3 | Dawit | 79 |
| 4 | Sara | 88 |

|  |  |  |
| --- | --- | --- |
| RollNo | Name | Marks |
| 1 | Henok | 98 |
| 2 | Meron | 87 |
| 3 | Dawit | 79 |
| 4 | Sara | 88 |

**Intersection Operator (∩)**

Intersection operator is denoted by ∩ symbol and it is used to select common rows (tuples) from two tables (relations).

Let’s say we have two relations R1 and R2 both have same columns and we want to select all those tuples(rows) that are present in both the relations, then in that case we can apply intersection operation on these two relations R1 ∩ R2.

**Intersection** works on the relation as **'this and that'**. In relational algebra, **A ∩ B** returns a relation instance that contains every tuple that occurs in relation to instance **A** and relation instance **B** (both together). Here, **A** and **B** need to be union-compatible, and the schema of both result and **A** must be identical.

**Syntax:**

SELECT \* FROM A INTERSECT SELECT \* FROM B;

**Divide**

Divide operator is used for the queries that contain the keyword ALL.

**For e.g. –** Find all the students who has chosen additional subjects Machine Learning and Data Mining.

**Join**

Join operation is as its name suggest, to join or combine two or more relations’ information. Join can also be defined as a cross-product followed by selection and projection. There are several varieties of Join operation. Let’s discuss all of them one by one.

**Condition Join**

When you want to join two relations based on the given condition, it is termed as Condition Join. It is denoted by the symbol **⋈c**.

**For e.g. –** Select the students from Student1 table whose RollNo is greater than the RollNo of Student2 table.

**Student1⋈cStudent1.RollNo>Student2.RollNoStudent2**

**Syntax:**

SELECT \* FROM Student1, Student2 WHERE Student1.RollNo > Student2.RollNo;

**b. Equi Join**

It is a special case of Condition Join. When you want to join two relations based on the equality condition, it is termed as Equi Join. It is denoted by the symbol **⋈**.

**For e.g. -** Select the students from Student1 table whose RollNo is equalto the RollNo of Student2 table.

**Student1⋈Student1.RollNo=Student2.RollNoStudent2**

**Syntax:**

SELECT \* FROM Student1, Student2 WHERE Student1.RollNo=Student2.RollNo;

**c. Natural Join**

Natural Join is that type of join in which equijoin is by default applied to all the attributes in two or more relation. Its specialty is if you want to consider the equality between two relations, you don’t need to define the equality; it is predefined for all the attributes if you use Natural Join. It is denoted by the symbol **⋈**.

**For e.g. -** Select the students from Student1 table whose RollNo is equal to the RollNo of Student2 table.

**Student1⋈Student2**

**Syntax:**

SELECT \* FROM Student1 NATURAL JOIN Student2;

# **Relational Calculus**

Contrary to Relational Algebra which is a procedural query language to fetch data and which also explains how it is done, **Relational Calculus** in non-procedural query language and has no description about how the query will work or the data will b fetched. It only focusses on what to do, and not on how to do it.

Relational Calculus exists in two forms:

1. Tuple Relational Calculus (TRC)
2. Domain Relational Calculus (DRC)

## ***Tuple Relational Calculus (TRC)***

In tuple relational calculus, we work on filtering tuples based on the given condition.

**Syntax:** { T | Condition }

In this form of relational calculus, we define a tuple variable, specify the table(relation) name in which the tuple is to be searched for, along with a condition.We can also specify column name using a . dot operator, with the tuple variable to only get a certain attribute(column) in result. A tuple variable is nothing but a name, can be anything, generally we use a single alphabet for this, so let's say T is a tuple variable.

To specify the name of the relation(table) in which we want to look for data, we do the following: Relation(T), where T is our tuple variable.

For example if our table is **Student**, we would put it as Student(T)

Then comes the condition part, to specify a condition applicable for a particluar attribute(column), we can use the . dot variable with the tuple variable to specify it, like in table **Student**, if we want to get data for students with age greater than 17, then, we can write it as,

T.age > 17, where T is our tuple variable.

Putting it all together, if we want to use Tuple Relational Calculus to fetch names of students, from table **Student**, with age greater than **17**, then, for T being our tuple variable,

T.name | Student(T) AND T.age > 17

## ***Domain Relational Calculus (DRC)***

In domain relational calculus, filtering is done based on the domain of the attributes and not based on the tuple values.

**Syntax:** { c1, c2, c3, ..., cn | F(c1, c2, c3, ... ,cn)}

where, c1, c2... etc represents domain of attributes(columns) and F defines the formula including the condition for fetching the data.

For example,

{< name, age > | ∈ Student ∧ age > 17}

Again, the above query will return the names and ages of the students in the table **Student** who are older than 17.

**Difference between Relational Algebra and Relational Calculus:**

|  |  |  |
| --- | --- | --- |
|  | **Relational Algebra** | **Relational Calculus** |
| 1. | It is a Procedural language. | While Relational Calculus is Declarative language. |
| 2. | Relational Algebra means how to obtain the result. | While Relational Calculus means what result we have to obtain. |
| 3. | In Relational Algebra, The order is specified in which the operations have to be performed. | While in Relational Calculus, The order is not specified. |
| 4. | Relational Algebra is independent on domain. | While Relation Calculus can be a domain dependent. |
| 5. | Relational Algebra is nearer to a programming language. | While Relational Calculus is not nearer to programming language. |
|  |  |  |