Relational Calculus

Relational calculus is based on **predicate calculus**. This is a way of defining a set using a variable and a predicate (something that works out to either true or false). Any value of the variable for which the predicate works out as true is a member of the set. **Relational calculus** is a way of defining tables in terms of other tables also by means of a variable and a predicate.

In **tuple oriented** relational calculus a relation (table) is defined using a variable which stands for a tuple (row) of an existing table and a predicate. Any value of the variable (i.e. tuple of the original relation/ row of the original table) for which the predicate is true is a tuple of the new relation.

Tuple variables are declared as belonging to a particular relation. Declarations have the form:

RANGE OF <variable name> **IS** <relation name>

Relational calculus expressions take the form:

<tuple variable> **WHERE** <predicate>

The predicate can contain all the usual comparison operators and the Boolean operators AND, OR and NOT.

For example

RANGE OF m IS musicians
m WHERE m.home = "Exland"

defines a table which is equivalent to the Relational Algebra expression

SELECT HOME = EXLAND (MUSICIANS).

The effect of a PROJECT is achieved by qualifying the tuple variable before WHERE with an attribute (column) name. Because the result is always a set it will never contain any duplicates whatever the result.

The names of musicians born after 1899 can be found by

m.surname WHERE m.dob > 31/12/1899

which is equivalent to

PROJECT SURNAME (SELECT DOB > 31/12/1899 (MUSICIANS))

in Relational Algebra.

It is possible to use more than one tuple variable in defining a new relation and that allows us to achieve the same effect as a JOIN in Relational Algebra by using a predicate that depends on two values from two different columns being the same.

For example

RANGE OF m IS musicians
RANGE OF c IS compositions
m.surname, c.title WHERE c.mno = m.mno

defines a table which contains the names of all compositions associated with the names of their composers.

Tuple variables can normally only be used in the predicate if they appear before the WHERE. Other tuple variables can be used in the special context of the **existential quantifier**. This is itself a predicate but has its own tuple variable and sub-predicate. It takes the general form

EXISTS <tuple variable> **(** <predicate> **)**

and delivers the value true if the predicate is true for at least one value of its tuple variable. The tuple variable of the EXISTS can only be used within its own predicate, the one surrounded by the brackets.

For example the names of all composers are given by

```
m.surname  WHERE    EXISTS  c  ( c.mno = m.mno )
```

The equivalent of the relational algebra operators INTERSECTION, UNION and DIFFERENCE can be achieved in relational calculus using AND, OR and AND NOT respectively within the predicate.

To find the names of composers who are not performers the expression is

```
RANGE  OF  m  IS  musicians
RANGE  OF  c  IS  compositions
RANGE  OF  p  IS  performers
m.surname  WHERE
                EXISTS  c ( c.mno = m.mno )
                AND  NOT
                EXISTS  p ( p.mno = m.mno )
```

This is equivalent to

```
PROJECT  SURNAME  (MUSICIANS  JOIN  COMPOSERS)
            DIFFERENCE
PROJECT  SURNAME  (MUSICIANS  JOIN  PERFORMERS)
```

Finally we have one more predicate: the **universal quantifier**. The syntax of this is

FORALL <tuple variable> (<predicate>)

It also introduces a new variable within the predicate but it delivers the value true only if the predicate is true for every value its tuple variable can take. As with the existential quantifier, the new tuple variable can only be used within the pair of brackets surrounding its predicate.

For example

```
RANGE  OF  m  IS  musicians
RANGE  OF  p1  IS  performers
RANGE  OF  p2  IS  performers
m.surname WHERE
      FORALL p1 (
            EXISTS  p2  (
                  (p1.instrument = p2.instrument)   AND   (p2.mno = m.mno)
            )
      )
```

list the names of musicians who play all the instruments.

We can achieve the same effect as all the operators of relational algebra in relational calculus. This means that relational calculus is a **relationally complete** language. Relational calculus is the basis of most commonly used database query languages. In particular SQL is based on relational calculus.