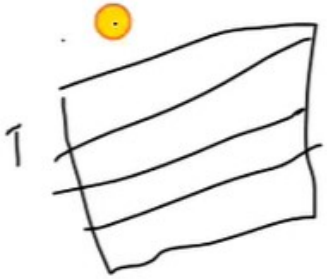
Relational Calculus

- **Relational Calculus** is a **non-procedural query language** (or **declarative language**). It uses mathematical **predicate calculus** (**or first-order logic**) instead of algebra.
- **Relational Calculus** tells **what to do** but never explains how to do.
- **Relational Calculus** provides **description about the query** to get the result *where as* ***Relational Algebra*** **gives the method** to get the result.

- When applied to database, it comes in **two flavors**:

1. **Tuple Relational Calculus (TRC):**
   - Proposed by Codd in the year 1972
   - **Works on tuples** (or rows)
2. **Domain Relational Calculus (DRC):**
   - Proposed by Lacroix & pirotte in the year 1977
   - **Works on domain of attributes** (or Columns)



Relational Calculus

- **Calculus** has *variables, constants, comparison operator, logical connectives* and *quantifiers.*
  - ✓ ***TRC***: **Variables range over *tuples*.**
    - ✓ Like SQL
  - ✓ ***DRC***: **Variables range over *domain elements*.**
    - ✓ Like Query-By-Example (QBE)
- Expressions in the calculus are called ***formulas***.
- **Resulting tuple** is an assignment of constants to variables that make the **formula** evaluate to ***true***.

Tuple Relational Calculus

- **Tuple relational calculus** is a **non-procedural query language**
- **Tuple relational calculus** is used for **selecting the tuples** in a relation that satisfy the given **condition** (or **predicate**). The **result** of the relation can have one or more tuples.
- A **query** in **TRC** is expressed as:

$$\{t \mid P(t)\}$$

Where t denotes *resulting tuple* and $P(t)$ denotes *predicate (or condition) used to fetch tuple* t

- **Result of Query: It is the set of all tuples t such that predicate P is true for t**
- **Notations** used:
  - t is a *tuple variable*,
  - $t[A]$ denotes the value of tuple t on attribute A
  - $t \in r$ denotes that tuple t is in relation r
  - P is a **formula** similar to that of the ***predicate calculus***

Predicate Calculus Formula

1. **Set of attributes and constants**
2. **Set of comparison operators:** e.g., $<, \leq, =, \neq, >, \geq$
3. **Set of connectives:** and ($\wedge$), or ($\vee$), not ($\neg$)
4. **Implication** ($\Rightarrow$): $x \Rightarrow y$, if x is true, then y is true $\quad x \Rightarrow y \equiv \neg x \vee y$
5. **Quantifiers:** **Existential Quantifiers** ($\exists$) **and Universal Quantifier**($\forall$).
   - $\exists t \in r\,(Q(t)) \equiv$ "there exists" a tuple in t in relation r such that predicate $Q(t)$ is true
   - $\forall t \in r\,(Q(t)) \equiv$ Q is true "for all" tuples t in relation r

Free and Bound variables:

- The use of **quantifiers** $\exists X$ and $\forall X$ in a formula is said to ***bind*** X in the formula.
- A variable that is **not bound** is **free**.
- Let us revisit the definition of a **query**: $\{t \mid P(t)\}$
- ***There is an important restriction***
  - the variable t that appears to the left of `|' must be the ***only free variable*** in the formula $P(t)$.
  - in other words, **all other tuple variables** must be **bound** using a **quantifier**

Example TRC

- **Find the *loan-number*, *branch-name*, and *amount* for all loans of over $1200.**

$$\{ t \mid t \in loan \wedge t[amount] > 1200 \}$$ **(Selection)**

It selects ***all tuples*** *t* from *relation* ***loan*** such that the resulting ***loan*** tuples will have ***amount*** greater than $1200

- **Find the *loan number* for each loan of an amount greater than $1200**

$$\{ t \mid \exists s \in loan \ (t[loan\text{-}number] = s[loan\text{-}number] \wedge s[amount] > 1200) \}$$ **(Projection)**

It selects the ***set of tuples*** *t* such that there exists a ***tuple*** *s* in relation ***loan*** for which the values of *t* & *s* for the *loan-number* attribute are equal and the value of *s* for the *amount* attribute is greater than $1200

branch (branch-name, branch-city, assets)
customer (customer-name, customer-street, customer-city)
account (account-number, branch-name, balance)
loan (loan-number, branch-name, amount)
depositor (customer-name, account-number)
borrower (customer-name, loan-number)

Example Queries TRC

Example Queries: TRC...

branch (branch-name, branch-city, assets)
customer (customer-name, customer-street, customer-city)
account (account-number, branch-name, balance)
loan (loan-number, branch-name, amount)
depositor (customer-name, account-number)
borrower (customer-name, loan-number)

- **Find the *names of all customers* having a *loan* at the *Perryridge* branch**

$$\{ t \mid \exists s \in borrower\ (t[customer\text{-}name] = s[customer\text{-}name] \wedge \exists u \in loan\ (u[branch\text{-}name] = \text{“Perryridge”} \wedge u[loan\text{-}number] = s[loan\text{-}number])) \}$$

Join

- **Find the *names of all customers* having a loan, an account, or both at the bank**

$$\{ t \mid \exists s \in borrower\ (t[customer\text{-}name] = s[customer\text{-}name]) \vee \exists u \in depositor\ (t[customer\text{-}name] = u[customer\text{-}name]) \}$$

OR Union

- **Find the names of all customers who have a loan and an account at the bank**

$$\{ t \mid \exists s \in borrower\ (t[customer\text{-}name] = s[customer\text{-}name]) \wedge \exists u \in depositor\ (t[customer\text{-}name] = u[customer\text{-}name]) \}$$

AND ∩

Domain Relational Calculus DRC

- **Domain Relational Calculus** is a **non-procedural query language.**
- In **Domain Relational Calculus** the records are filtered based on the **domains.**
- **DRC** uses **list of attributes** to be selected from relation based on the **condition (or predicate).**
- **DRC** is same as **TRC** but differs by **selecting the attributes** rather than **selecting whole tuples**
- In **DRC**, each **query** is an expression of the form:

$$\{ < a_1, a_2, \ldots, a_n > \mid P(a_1, a_2, \ldots, a_n) \}$$

$a_1, a_2, \ldots, a_n$ represent *domain variables*

P represents a ***predicate*** similar to that of the *predicate calculus*

- **Result of Query:** **It is the set of all tuples $a_1, a_2, \ldots, a_n$ such that predicate P is true for $a_1, a_2, \ldots, a_n$ tuples**

Examples Queries DRC

- Find the *loan-number, branch-name,* and *amount* for loans of over $1200:

$$\{< l, b, a > \mid < l, b, a > \in loan \wedge a > 1200\}$$

(Selection)

- Find the *loan number* for each loan of an amount greater than $1200:

$$\{< l > \mid \exists\, b, a\ (< l, b, a > \in loan \wedge a > 1200)\}$$

(Selection then Projection)

branch (branch-name, branch-city, assets)
customer (customer-name, customer-street, customer-city)
account (account-number, branch-name, balance)
loan (loan-number, branch-name, amount)
depositor (customer-name, account-number)
borrower (customer-name, loan-number)

Examples Queries DRC

- Find the *names of all customers* who have a loan of over \$1200:

$$\{< c > \mid \exists\, l, b, a\, (< c, l > \in borrower \wedge (< l, b, a > \in loan \wedge a > 1200))\}$$

- Find the *names of all customers* having a *loan* at the *Perryridge* branch and find the *loan amount*:

$$\{< c, a > \mid \exists\, l\, (< c, l > \in borrower \wedge \exists\, b\ (< l, b, a > \in loan \wedge b = \text{“Perryridge”}))\}$$

branch (branch-name, branch-city, assets)
customer (customer-name, customer-street, customer-city)
account (account-number, branch-name, balance)
loan (loan-number, branch-name, amount)
depositor (customer-name, account-number)
borrower (customer-name, loan-number)