WEEK 4

Relational Calculus
Chapter 7
Query Languages

Ease of Comprehension

QBE

SQL

Power of Language

FOPC

Relational Calculus

- Comes in two flavors: **Tuple relational calculus** (TRC) and **Domain relational calculus** (DRC).
- Calculus has variables, constants, comparison ops, logical connectives and quantifiers.
  - **TRC**: Variables range over (i.e., get bound to) tuples.
  - **DRC**: Variables range over domain elements (= field values).
  - Both TRC and DRC are simple subsets of first-order logic.
- Expressions in the calculus are called **formulas**. An answer tuple is essentially an assignment of constants to variables that make the formula evaluate to true.
Relational Calculus

- Propositional logic
- Predicate Calculus
  → Relational Calculus

X: It is raining today
Y: Picnic is cancelled
- X \lor Y
- X \rightarrow Y      IF X then Y      Logical implication

Relational Calculus

- X, Y             Symbols (propositional)
- \neg \land \lor \rightarrow Connectives (according to precedence rule)

<exp>::= <prop symbol>  X\mid Y\mid Z\mid R
::= <constant>  T\mid F
::= \neg(exp)
::= (exp < binary op > exp)

\lor \mid \land \rightarrow \leftrightarrow
Some Rules of Inference

1) Modus Ponens Rule
   If X and \( X \rightarrow Y \) then Y
   \((X \text{ and } (X \rightarrow Y)) \rightarrow Y\)

2) Disjunctive Syllogism
   \( X \rightarrow X \lor Y \)

3) Resolution (can be proved using truth table)
   \( \neg X \lor Y \equiv X \rightarrow Y \)

Predicate with multiple variables

Relations are predicates

AGE (name, number)
SUP (S#, Sname, Status, City)
Quantifiers

1) Universal Quantifier (\(\forall\))
   - (a) Every supplier has a unique S#
   - (b) Every supplier has a unique name.

2) Individual (Existential) \(\exists\)
   there exists a member with some property.

\((\forall S\#)(\exists S\text{name})\text{ so that }S\# \rightarrow S\text{name}\)

Tuple Relational Calculus

- **Query** has the form: \(\{ t \mid P(t) \}\)
  where \(t\) is a tuple variable, it ranges over some relation
- **Answer** includes all tuples that make the formula (predicate) \(P\) be true.

Example \(\{ t \mid t \in S \land t.\text{city} = \text{‘London’}\}\)
Answer: Find all tuples of \(S\) satisfying the above predicate.
Tuple Relational Calculus

- **Formula** is recursively defined, starting with simple **atomic formulas** (getting tuples from relations or making comparisons of values), and building bigger and better formulas using the **logical connectives**.

---

TRC Formulas

- **Atomic formula**:  
  - \( t \in R \) where \( t \) is a tuple variable and \( R \) is a relation  
  - \( X \text{ op } Y \), or \( X \text{ op constant} \), here \( \text{op} \) is one of \(<, >, \leq, \geq, \neq\)

- **Formula**:  
  - an atomic formula, or  
  - \( \neg p, p \land q, p \lor q, p \rightarrow q \) where \( p \) and \( q \) are formulas  
  - \( \exists X (p(X)) \), where variable \( X \) is tuple variable  
  - \( \forall X (p(X)) \)

- The use of quantifiers \( \exists X \) and \( \forall X \) is said to **bind** \( X \).  
  - A variable that is **not bound** is **free**.
TRC Formulas

1) \{ t \mid P(t) \}  Set of all tuples for which predicate P is true
   t = tuple variable, it ranges over some relation

2) \exists t \ Q(t)  There exists a tuple t such that predicate Q(t) is true.

3) \forall t \ Q(t)  Q(t) is true for all tuple t

Range definition

Range of Sx is S
(Sx is variable for tuple of S)
Example:
   Give S# of suppliers who are located in LA
   and have status = 3
   Range of Sx is S
   (Sx.S#) WHERE Sx.city = CITY('LA') AND Sx.Status = Status('3')
Some valid Formulas

- \( \text{NOT ( Sx.CITY} = \text{‘London’)} \)

- \( Sx.S# = SPy.S# \)
  \( \quad \text{AND SPy.P#} \neq Pz.P# \)

- \( \exists SPx ( SPx.S# = Sx.S# \)
  \( \quad \text{AND SPx.P#} = \text{‘P2’}) \)

Equivalence Rules

Let \( F(x) \) and \( G(x) \) be the formulas (predicates)

1) \( \forall x F = \neg \exists x \neg F \)
2) \( \forall x \neg F = \neg \exists x F \)
3) \( \neg \forall x F = \exists x \neg F \)
4) \( \forall x F \lor G = \forall x [ F \lor G ] \)
5) \( \forall x F \land G = \forall x [ F \land G ] \)
6) \( \exists x F \lor G = \exists x [ F \lor G ] \)
7) \( \exists x F \land G = \exists x [ F \land G ] \)
8) \( \forall x [ F \land G ] = \forall x F \land \forall x G \)
9) \( \exists x [ F \lor G ] = \exists x F \lor \exists x G \)
10) \( \neg ( \forall x F(x) ) = \exists x \neg F(x) \)
11) \( \neg ( \exists x F(x) ) = \forall x \neg F(x) \)
Quantifiers and Operators

A = \{ x^{(1)}, x^{(2)} \}  \quad \text{Domain}
B = \{ y^{(1)}, y^{(2)} \}  \quad \text{Domain}

1) \forall x \ H(x) = ??
   \forall x \ H(x) = H( x^{(1)} ) \land H( x^{(2)} )

2) \exists y \ G(y) = G( y^{(1)} ) \lor G( y^{(2)} )

3) \exists y \ \forall x \ F(x,y) \equiv \forall x \ \exists y \ F(x,y)

Multiple Operators

Ex: \ F(x,y) \text{ is true if item x is sold by dept y.}

\exists y ( \forall x \ F(x,y) ) \equiv \text{there exists a department that sells all items}
   \equiv \text{For all items, there exists a dept. which sells all of them.}

\forall x ( \exists y \ F(x,y) ) \equiv \text{For every item, there exists a dept. y so that F(x,y)
Multiple Operators

Example: \( \text{SUP}(x,y,u,v) \)

Example: Every class held on M at time ‘t’ hour is also held on W and F at the same hour.
\[ \forall x \forall y \text{ Stu-Sch}(x,y,d,t) \quad \text{where } x = \text{name} \]
\[ y = \text{class} \]
\[ d = \text{day} \]
\[ t = \text{time} \]

\[ \Rightarrow \text{Stu-Sch}(x,y,W,t) \land \text{Stu-Sch}(x,y,M,t) \land \text{Stu-Sch}(x,y,F,t) \]

Example Query with two Relations

Give S# for suppliers who supply red parts

Range of Sx is S
Range of SPx is SP
Range of Px is P
(SPx.S#) where
\[ \exists \text{Px.P#} = \text{SPx.P#} \land \text{Px.COLOR} = \text{COLOR(‘red’)} \]
Example Query with two Relations

Give S# for suppliers who supply at least one red part.

\[(Sx.SNAME) \text{ WHERE } \exists SPx \ (Sx.S# = SPx.S# \text{ AND } \exists Px \ (SPx.P# = Px.P# \text{ AND } Px.COLOR = \text{COLOR ('red')})]\]

Example Query with two Relations

Prefix normal form in which all quantifiers appear at the front of the formula.

Alternate:-

\[Sx.SNAME \text{ where } \exists SPx \ \exists Px \ (Sx.S# = SPx.S# \text{ AND } SPx.P# = Px.P# \text{ AND } Px.COLOR = \text{COLOR ('red')})\]
Example Query

Give supplier names who supply all parts.

How can we use ∃ and ∀ quantifiers?
Example Query

Give S# for suppliers who supply at least those parts supplied by supplier S2.
Use: IF f THEN g (¬f OR g)

Domain Relational Calculus

- Query has the form:
  \[ \left\{ \langle x_1, x_2, \ldots, x_n \rangle \mid p(x_1, x_2, \ldots, x_n) \right\} \]

- Answer includes all tuples \( \langle x_1, x_2, \ldots, x_n \rangle \) that make the formula (predicate) \( p(x_1, x_2, \ldots, x_n) \) be true.

- Formula is recursively defined, starting with simple atomic formulas (getting tuples from relations or making comparisons of values), and building bigger and better formulas using the logical connectives.
**DRC Formulas**

- **Atomic formula:**
  - \((x_1, x_2, \ldots, x_n) \in Rname\), or \(X \text{ op } Y\), or \(X \text{ op } \text{ constant}\)
  - \(\text{op}\) is one of \(<,>,=,\leq,\geq,\neq\)

- **Formula:**
  - an atomic formula, or
  - \(\neg p, p \land q, p \lor q\), where \(p\) and \(q\) are formulas, or
  - \(\exists X (p(X))\), where variable \(X\) is free in \(p(X)\), or
  - \(\forall X (p(X))\), where variable \(X\) is free in \(p(X)\)
  - membership condition. \(R\) (pair, pair, . . .)

**Example of membership condition**

- \(SP (S\#:\text{S#}(\text{'S1'}), P\#:P\#)\text{P1})\)
  - condition is true iff there exists a shipment tuple with S# value S1 and P# value P1.

- \(SP (S\#:SX, P\#:PX)\)
  - condition is true iff there exists a shipment tuple with S# value equal to the current range variable SX and P# value equal to the current value of range variable PX.
Unsafe Queries, Expressive Power

- It is possible to write syntactically correct calculus queries that have an infinite number of answers! Such queries are called unsafe.
  - e.g., \{t | \neg (t \in S)\}

- It is known that every query that can be expressed in relational algebra can be expressed as a safe query in DRC/TRC; the converse is also true.

- Relational Completeness: Query language (e.g., SQL) can express every query that is expressible in relational algebra/calculus.

Summary

- The relational model has rigorously defined query languages that are simple and powerful.
- Relational algebra is more operational; useful as internal representation for query evaluation plans.
- Relational calculus is non-operational, and users define queries in terms of what they want, not in terms of how to compute it. (Declarativeness.)
- Several ways of expressing a given query; a query optimizer should choose the most efficient version.
- Algebra and safe calculus have same expressive power, leading to the notion of relational completeness.