Predicate Logic and DBs

Relational query languages 1

Archetypal query in Quel constructs new relation from relations R_1, R_2, ..., R_k.

range of t_1 is R_1
range of t_2 is R_2
....
range of t_k is R_k
retrieve ($t_{i(1)}.A_{j(1)}$, ..., $t_{i(r)}.A_{j(r)}$)
where $Y(t_1, t_2, ..., t_k)$

$Y(t_1, t_2, ..., t_k)$ is a (quantifier-free) logical constraint on the tuples selected by the range variables t_1, t_2, ..., t_k in the construction process

Relational query languages 2

Illustrative example of use of QUEL:

parts	pnum, pname, colour, weight, qoh
supply	snum, pnum, jnum, shipdate, quantity

Display supplier, partname , shipdate for all parts shipped since 1994

range of p is parts
range of s is supply
retrieve (s.snum, p.pname, s.shipdate)
where (s.pnum = p.pnum) and (s.shipdate >= 1994)

Relational query languages 3

Linking example to the abstract formalism:

k=2 ..... two relations used in construction

Index the relations by integers, so that
R_1 is parts, R_2 is supply, t_1 is p, t_2 is s

Index the attributes of parts and supply by integers:
e.g. $t_1.A_3$ is p.colour, $t_2.A_3$ is s.jnum etc

Relational query languages 4

… linking example to abstract formalism …

i() and j() functions with domain set {1,2,3} constructing a relation with 3-tuples

i maps onto the set {1,2}
… from which relation are new fields derived?

j maps onto the set {1, 2, ..., 5}

… from which fields are new fields derived?

$Y(t_1, t_2)$ =Y(p, s) is a logical constraint on the tuples selected from parts and supply viz. "s and p must designate tuples with the same part number, and the shipdate for the supply tuple must be 1994 or later"

Relational query languages 5

In general, can translate this into a logical specification for a new relation constructed from a set of source relations $R_1, R_2, ..., R_k$ - express this in the form:

[The required relation is] the set of tuples of the form
u(r) = (u[1], u[2], ..., u[r])
where
t_i is a tuple in the relation R_i,
u is made up of particular components of the t_i's,
and
the t_i's used to construct u
satisfy some additional constraint.

Relational query languages 6

A suitable logical expression for the required relation is

$$\{ u(r) \mid (\exists t_1) \ldots (\exists t_k) \; (R_1(t_1) \wedge R_2(t_2) \wedge \ldots \wedge R_k(t_k) \wedge u[1] = t_{i(1)} [j(1)] \wedge u[2] = t_{i(2)} [j(2)] \wedge \ldots \wedge u[r] = t_{i(r)} [j(r)], \wedge Y(t_1, t_2, ..., t_k)) \}$$

Relational query languages 7

This uses a **Relational Calculus** formalism to define the set of tuples that make up the new relation by a predicate. Here $R_j(t_j)$ is a basic predicate asserting that t_j is a tuple in the relation R_j.

Need a "predicate calculus over relations" to do this.
There are two variants of this:
tuple relational calculus
domain relational calculus.

QUEL is *tuple* relational calculus based.

Relational Calculus 1

Predicate Calculus query languages ...

a query = finding values satisfying predicate

Two kinds of predicate calculus language

… terms (primitive objects of discourse)
"tuples" ⇒ **tuple** relational calculus
"domain values" ⇒ **domain** relational calculus

Relational Calculus 2

Expressions in the tuple relational calculus

Basic form of such an expression:
$\{t \mid \psi(t)\}$ where t is a **tuple variable**

Here t [or $t^{(r)}$] denotes a tuple of some fixed arity
(NB not *denoting a tuple of fixed type*)
and ψ is a formula built according to the conventional **first order predicate calculus** ("FOPC") rules

Recall the logical expression for a QUEL query

Archetypal query in QUEL constructs new relation from relations $R_1, R_2, ..., R_k$.

range of t_1 is R_1
range of t_2 is R_2
....
range of t_k is R_k
retrieve ($t_{i(1)}.A_{j(1)}, ..., t_{i(r)}.A_{j(r)}$)
where $Y(t_1, t_2, ..., t_k)$

$Y(t_1, t_2, ..., t_k)$ is a (quantifier-free) logical constraint on the tuples selected by the range variables $t_1, t_2, ..., t_k$ in the construction process

Recall the logical expression for a QUEL query

A suitable logical expression for the required relation is

$$
\{ u^{(r)} \mid (\exists t_1) ... (\exists t_k) \\
(\quad R_1(t_1) \wedge R_2(t_2) \wedge ... \wedge R_k(t_k) \\
\wedge u[1] = t_{i(1)} [j(1)] \\
\wedge u[2] = t_{i(2)} [j(2)] \\
\wedge \quad ... \\
\wedge u[r] = t_{i(r)} [j(r)], \\
\wedge Y(t_1, t_2, ..., t_k) \\
) \\
\}
$$

Safety of relational expressions 1

Without any restriction on a logical expression can define an infinite collection of tuples.

Need to restrict to sets of tuples that are finite to take account of storage and computation.

For example:

what is $\{ t \mid \neg\psi(t) \}$?

... very ill-defined collection of tuples

how do we compute $\{ t \mid (\exists s)(\psi(s,t)) \}$?

... when have we considered every possible s?

Safety of relational expressions 2

Essential to know when it is *safe* to evaluate expression

… can't have non-terminating behaviour in a database

Solution : need to set limits on the values for tuples under consideration to eliminate endless searches

... motivates *safety rules* for expressions

Safety of relational expressions 3

Safe relational calculus expressions

When can we evaluate $\{ t \mid \psi(t) \}$?

In computational terms, want to be able to evaluate truth or falsehood of expression after making a finite set of substitutions

Logical expression means *context-independent* interpretation

For context-independence: basis for restricting a search is what can be inferred about domain of values of interest *from the expression to be evaluated*.

Safety of relational expressions 4

For context-independence: basis for restricting a search is what can be inferred about domain of values of interest *from the expression to be evaluated*.

Motivates **definition** of Dom(ψ), viz: the set of components of tuples in relations mentioned in ψ together with all constants referenced by ψ

Note that Dom(ψ) is always a finite set