Relational db: the origins

Frege: FO logic  Tarski: algebra for FO

Codd: relational databases

Relational Calculus (aka FO)

• Models data manipulation core of SQL
  – idea: specify “what” not “how”

• General form defines the set of tuples t in the answer:
  \{t \mid \text{property (t)}\}

• property (t) is described by a language based on predicate calculus (first-order logic)
Reminder (CSE 20):
some predicate calculus examples on natural numbers

• The set of even numbers:
  \{ x | \exists y ( x = 2 * y) \}

• ? The set of prime numbers
  \{ x | x \neq 1 \land \forall y \forall z [ x = y * z \rightarrow ((y = 1) \lor (z = 1))] \}

\exists : “there exists” existential quantification
\forall : “for all” universal quantification

Relational calculus speaks about tuples

Display the movie table

SELECT  *
FROM movie

In words (making answer tuple explicit):

“The answer consists of tuples m such that
m is a tuple in movie

Need to say:
“tuple m is in relation R”:  m \in R
Examples

Find the directors and actors of currently playing movies

SELECT m.Director, m.Actor
FROM movie m, schedule s
WHERE m.Title = s.Title

In words (making answer tuple explicit):

“The answer consists of tuples t such that there exist tuples m in movie and s in schedule for which t.Director = m.Director and t.Actor = m.Actor and m.Title = s.Title”

Need to say:
“there exists a tuple x in relation R”:  x  R
Refer to the value of attribute A of tuple x: x(A)
Boolean combinations

Examples (cont’d)

Find the directors and actors of currently playing movies

“In logic notation (tuple relational calculus):

{ t: Director, Actor |  m  movie  s  schedule
[ t(Director) = m(Director)  t(Actor) = m(Actor)
 m(Title) = s(Title) ] }
∃ m ∈ R : existential quantification
“there exists some tuple m in relation R ....”

Sometimes need to say
“for every tuple m ....”

Example: “every director is also an actor”

Need to say:
“for every tuple m in movie there exists a tuple t in movie
such that m.Director = t.Actor”

Logic notation: universal quantification ∀ m ∈ R

∀ m ∈ movie ∃ t ∈ movie [ m(Director) = t(Actor) ]

(The answer to this query is true or false)

Tuple Relational Calculus

• In the style of SQL: language talks about tuples
• What you can say:
  – refer to tuples: tuple variables t, s, ...
  – a tuple t belongs to a relation R: t ∈ R
  – conditions on attributes of a tuple t and s:
    • t(A) = (≠)(≥) constant
    • t(A) = s(B)
    • t(A) ≠ s(B)
    • etc.
• Simple expressions above: atoms
Tuple Relational Calculus (2)

• Combine properties using Boolean operators
  – \( \land, \lor, \neg \)
  – (abbreviation: \( p \rightarrow q \equiv \neg p \lor q \))

• Quantifiers
  – there exists: \( \exists t \in R \varphi(t) \)
  – for every: \( \forall t \in R \varphi(t) \)

  similar to local variable declarations

More on quantifiers

• scope of quantifier:
  – scope of \( \exists t \in R \varphi(t) \) is \( \varphi \)
  – scope of \( \forall t \in R \varphi(t) \) is \( \varphi \)

• free variable:
  – not in scope of any quantifier
  – free variables are the “parameters” of the formula
Examples

{ t: Director, Actor | ∃ m ∈ movie ∃ s ∈ schedule
[ t(Director) = m(Director) ∧ t(Actor) = m(Actor) ∧ m(Title) = s(Title) ] }

[ t(Director) = m(Director) ∧ t(Actor) = m(Actor) ∧ m(Title) = s(Title) ]
free: t, m, s

∃ s ∈ schedule
[ t(Director) = m(Director) ∧ t(Actor) = m(Actor) ∧ m(Title) = s(Title) ]
free: t, m

∃ m ∈ movie ∃ s ∈ schedule
[ t(Director) = m(Director) ∧ t(Actor) = m(Actor) ∧ m(Title) = s(Title) ]
free: t

Tuple Calculus Query

• {t: <att> | φ(t)}

  – where φ is a calculus formula
    with only one free variable t
  – produces as answer a table with attributes <att>
    consisting of all tuples v which make φ(v) true
  – Note: φ(v) has no free variables so it has no parameters
    and it evaluates to true or false
  – Range of answer tuple: usually specified in the query
    Otherwise, it is by default the active domain:
    set of values in database, or mentioned in query
Examples (Movie Database)

- Find the titles of currently playing movies
  
  \[ \{ t: \text{title} \mid \exists s \in \text{schedule} \ [ s(\text{title}) = t(\text{title})] \} \]

- Find the titles of movies by Berto
  
  \[ \{ t: \text{title} \mid \exists m \in \text{movie} \ [ m(\text{director}) = \text{“Berto”} \land t(\text{title}) = m(\text{title})] \} \]

- Find the title and director of currently playing movies
  
  \[ \{ t: \text{title}, \text{director} \mid \exists s \in \text{schedule} \ \exists m \in \text{movie} \ [ s(\text{title}) = m(\text{title}) \land t(\text{title}) = m(\text{title}) \land t(\text{director}) = m(\text{director})] \} \]

Examples (max salary)

- Find employees with the highest salary:

<table>
<thead>
<tr>
<th>employee</th>
<th>name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \{ x: \text{name} \mid \exists y \in \text{employee} \ [ x(\text{name}) = y(\text{name}) \land \forall z \in \text{employee} \ [ y(\text{salary}) \geq z(\text{salary}) ] \} \]
Examples (Movie Database)

• Find actors playing in every movie by Berto

\{a: \text{actor} | \exists y \in \text{movie} \ [a(\text{actor}) = y(\text{actor}) \land \\
\forall m \in \text{movie} \ [m(\text{director}) = \text{“Berto”} \Rightarrow \exists t \in \text{movie} \ (m(\text{title}) = \\
t(\text{title}) \land t(\text{actor}) = y(\text{actor}))]]\}

Is the following correct?
\{a: \text{actor} | \exists y \in \text{movie} \ [a(\text{actor}) = y(\text{actor}) \land \\
\forall m \in \text{movie} \ [m(\text{director}) = \text{“Berto”} \land \exists t \in \text{movie} \ (m(\text{title}) = \\
t(\text{title}) \land t(\text{actor}) = y(\text{actor}))]]\}

A: YES     B: NO

Examples (Movie Database)

• Find actors playing in every movie by Berto

\{a: \text{actor} | \exists y \in \text{movie} \ [a(\text{actor}) = y(\text{actor}) \land \\
\forall m \in \text{movie} \ [m(\text{director}) = \text{“Berto”} \Rightarrow \exists t \in \text{movie} \ (m(\text{title}) = \\
t(\text{title}) \land t(\text{actor}) = y(\text{actor}))]]\}

Typical use of \(\forall\):

\(\forall m \in R \ [ \text{filter}(m) \rightarrow \text{property}(m)]\)

Intuition: check \text{property}(m) for those \text{m} that satisfy \text{filter}(m)

\text{we don’t care about the m’s that do not satisfy filter(m)}
∀ m ∈ R [ filter(m) → property(m)]

```
R

T filter(m₁) m₁    property(m₁)
F filter(m₂) m₂    don’t care
F filter(m₃) m₃    don’t care
T filter(m₄) m₄    property(m₄)
T filter(m₅) m₅    property(m₅)
F filter(m₆) m₆    don’t care
```

∀ m ∈ movie [ filter(m) → property(m)]

```
Movie         title      director   actor
m.Dir = Berto

T filter(m₁) m₁    --- Berto ---  property(m₁)
F filter(m₂) m₂    --- Hitchcock ---  don’t care
F filter(m₃) m₃    --- Hitchcock ---  don’t care
T filter(m₄) m₄    --- Berto ---  property(m₄)
T filter(m₅) m₅    --- Berto ---  property(m₅)
F filter(m₆) m₆    --- Fellini ---  don’t care
```
Tuple Calculus and SQL

- Example: “Find theaters showing movies by Bertolucci”:
  - SQL:
    
    ```sql
    SELECT s.theater
    FROM schedule s, movie m
    WHERE s.title = m.title AND m.director = "Bertolucci"
    ```
  
  - tuple calculus:
    
    ```
    \{ t: theater | \exists s \in schedule \exists m \in movie [ t(theater) = s(theater) \land s(title) = m(title) \land m(director) = Bertolucci ] \}
    ```

Basic SQL Query

```sql
SELECT A_1, ..., A_n
FROM R_1, ..., R_k
WHERE cond(R_1, ..., R_k)
```  

Tuple Calculus

\{ t: A_1, ..., A_n | \exists r_1 \in R_1 \ldots \exists r_k \in R_k \{ \land A_j = r_j(A_j) \land cond(r_1, \ldots, r_k) \}\}

- Note: basic SQL query uses only \( \exists \);
- no explicit construct for \( \forall \)
Using Tuple Calculus to Formulate SQL Queries

- Example: “Find actors playing in every movie by Berto”
- Tuple calculus
  - \{a: actor | \exists y \in \text{movie} [a(\text{actor}) = y(\text{actor}) \land
    \forall m \in \text{movie} [m(\text{director}) = "Berto" \rightarrow \exists t \in \text{movie} (m(\text{title}) =
    t(\text{title}) \land t(\text{actor}) = y(\text{actor}))]]\}
- Eliminate \forall:\n  - \{a: actor | \exists y \in \text{movie} [a(\text{actor}) = y(\text{actor}) \land
    \neg \exists m \in \text{movie} [m(\text{dir}) = "Berto" \land \neg \exists t \in \text{movie} (m(\text{title}) = t(\text{title})
    \land t(\text{actor}) = y(\text{actor}))]]\}
- Rule: \( \forall x \in R \varphi(x) \equiv \neg \exists x \in R \neg \varphi(x) \)
Convert to SQL query

- Basic rule: one level of nesting for each “¬∃”

\[
\{a: \text{actor} | \exists y \in \text{movie} \ [a(\text{actor}) = y(\text{actor}) \land \\
\neg \exists m \in \text{movie} \ [m(\text{dir}) = \text{"Berto"} \land \neg \exists t \in \text{movie} \ (m(\text{title}) = t(\text{title}) \land t(\text{actor}) = y(\text{actor}))])}
\]

```
SELECT y.actor FROM movie y
WHERE NOT EXISTS
(SELECT * FROM movie m
WHERE m.dir = 'Berto' AND
NOT EXISTS
(SELECT *
FROM movie t
WHERE m.title = t.title AND t.actor = y.actor ))
```

Another possibility (with similar nesting structure)

```
SELECT actor FROM movie
WHERE actor NOT IN
(SELECT s.actor
FROM movie s, movie m
WHERE m.dir = 'Berto'
AND s.actor NOT IN
(SELECT t.actor
FROM movie t
WHERE m.title = t.title ))
```

- Note: Calculus is more flexible than SQL because of the ability to mix ∃ and ∀ quantifiers