Week 8 Tutoring

CSE 180

Relational Algebra

Relational Algebra

A query language for manipulating data in the relational data model

- Based on sets
- Each RA operator is either a unary or binary operator

Operators

- Selection (σ)
- Projection (π)
- Set-theoretic operations:
  - Union (∪)
  - Set-difference (-)
  - Cross-product (x)
  - Intersection (∩)
- Renaming (ρ) and Assignment (¬)
- Natural Join ( ), Theta-Join ( )
- Division (/ or ÷)

Codd's Theorem

Theorem: The Relational Algebra query language has the same expressive power as the Relational Calculus query language

Theorem: All operators (σ , π , x , U , -) are independent of each other. In other words, for each relational operator o, there is no relational algebra expression that is built from the rest that defines o

- You can use these operators to express other important operators (i.e. joins, set intersection, division, outer join)

Selection: $\sigma_{condition}(R)$

Takes a relation R and extracts only the rows from R that satisfy the condition

- A combination of the form <expr> <op> <expr>

Example:

$\sigma_{rating > 6}$ (Hotels)

Can also do it with AND statements: $\sigma_{rating > 6 \text{ AND } capacity > 50}$ (Hotels)

Projection $\pi_{\text{<attribute list>}}$ (R)

For every tuple in relation R, output only the attributes appearing in attribute list

- May be duplicates; for Codd's RA, duplicates are always eliminated

Example:

$\pi_{\text{name, address}}$ (Hotels)

Set Union R ∪ S

The output consists of the set of all tuples in either R or S (or both)

- R and S must be union-compatible

Properties:

- Commutativity - Yes
- Associativity - Yes
- Distributive - No

Set Difference: R - S

Output consists of all tuples in R but not in S

- R and S must be union-compatible

Properties:

- Commutativity - No
- Associativity - No
- Distributive - No

Product: R x S

R x S = { (a1, …, am, b1, …, bn) | (a1, …, am) ∈ R and (b1, …, bn) ∈ S) }.

R

A	B	C
a1	b1	c1
a2	b1	c2

S

D	E
d1	e1
d2	e2
d3	e3

R x S

A	B	C	D	E
a1	b1	c1	d1	e1
a1	b1	c1	d2	e2
a1	b1	c1	d3	e3
a2	b1	c2	d1	e1
a2	b1	c2	d2	e2
a2	b1	c2	d3	e3

Properties:

- Commutativity - No
- Associativity - No
- Distributive - No

Derived Operations

Theta-Join: $R \bowtie_{\theta} S$

Equivalent to writing $\sigma_{\theta}(R \times S)$

1. Compute R x S, then keep only those tuples in R x S that satisfy θ

Natural Join: $R \bowtie S$

Equivalent to $\pi_{(attr(R) \cup attr(S))} (\sigma_{R.A1=S.A1 \text{ AND } R.A2 = S.A2 \text{ AND } \ldots \text{ AND } R.Ak=S.Ak} (R \times S))$

1. Compute R x S
2. Keep only those tuples in R x S satisfying: R.A1=S.A1 AND R.A2 = S.A2 AND … AND R.Ak=S.Ak 3.
3. Output is projection on the set of attributes in R U S (without repeats of the attributes that appear in both)

Derived Operations cont.

Semi-Join: $R \ltimes S$

Is equivalent to $\pi_{attr(R)} (R \bowtie S)$

1. Computer natural join of R and S
2. Output the projection of that on the attributes of R

Set Intersection $R \cap S$

Is equal to $R - (R - S) = S - (S - R)$

Properties:

- Commutativity - Yes
- Associativity - Yes
- Distributive - No

Derived Operations cont.

Division $R \div S$

- Input: $attr(S) \subset attr(R)$ and $attr(S)$ is non-empty
- Example: $R(A,B,C,D)$, $S(B,D)$.
    - Meaning: $R \div S = \{ (a, c) \mid \text{for all } (b,d) \in S, \text{ we have } (a,b,c,d) \in R \}$
- Example: Find the names of drinkers who like all beers
    - $Likes(drinker, beer) \div \pi_{beer} (BeersInfo(beer, maker))$

Housekeeping Operations

Assignment ←

- It may be convenient to write a relational-algebra expression by assigning parts of it to temporary relation variables.
- I.e. R3Sailors ← $\sigma_{\text{rating = 3}}$ (Sailors)

Rename: ρ

- rename relation R to S with attributes A1, ..., An.
- I.e. $\rho_{\text{BeersInfo(beer,maker)}}$ Beers(name, manuf)
    - name is now *beer* and manuf is now *maker*

Application Programming

Approaches

1. Stored Procedure language
   - Persistent Stored Modules (PSM) allows us to store procedures as database schema elements
   - I.e. PL/pgSQL in Lab 4
2. Embedded in a host language
3. Connection tools/libraries are used to allow conventional language to access a database
4. RESTful Web Services using HTTP

Basic PL/pgSQL form

```
CREATE PROCEDURE <name>
    AS $$
        <optional_local_declarations>
        <body>
    $$ LANGUAGE plpgsql;
CREATE FUNCTION <name> (<parameter list>) RETURNS <type>
    AS $$
        <optional_local_declarations>
        <body>
    $$ LANGUAGE plpgsql;
```

Modes

Mode	Can be used	Can be changed
IN	Yes	No
OUT	No	Yes
INOUT	Yes	Yes

- IN is the default for PL/pgSQL
- Function parameters can have any mode
- Procedure parameters can have mode IN or INOUT, but not OUT

Invoking

- Use the statement CALL with the name of your Stored Procedure and the arguments for that procedure

  Example: CALL JoeMenu('Moosedrool', 5.00);

Kinds of Statements

- RETURN; returns from a Stored Procedure
- RETURN <expression>; returns from a Stored Function
- DECLARE <name> <type> [:= <initial value>]; declare and initialize local variables
- BEGIN… END is used to group statements
- <variable := <expression>; assign variables
- IF <condition> THEN <statement(s)> [ELSIF <condition> THEN <statements>… ELSE <statements>] END IF;
- CASE statements
- Loops: <loop name>: LOOP <statements > END LOOP;
  - EXIT [<loop name>] [WHEN condition]
  - Can also do while loops and for loops (and continue and exit)

Queries

- SELECT-FROM-WHERE queries are not permitted
- Can only do:
    1. Queries which produce one scalar value can be an expression in an assignment
    2. Single row SELECT...INTO
    3. Cursors

Cursors

- A tuple-variable that ranges over all tuples in the result of some query
- Declare a cursor: DECLARE c CURSOR FOR <query>;
- Use the cursor: OPEN c;
- Finished using the cursor: CLOSE c;
- Fetching from a Cursor: FETCH FROM c INTO x1, x2, … , xn;
    - The x’s are variables that have now been assigned
    - After the FETCH, the cursor is now moved to point at the next tuple
- Breaking cursor loops:
    - FOUND tells you if FETCH returned a tuple
    - You can use EXIT WHEN <condition> i.e. condition can be NOT FOUND