CPSC 421
Database Management Systems

Lecture 3:
Relational Algebra, More SQL,
… Repeat

* Some material adapted from R. Ramakrishnan, L. Delcambre, and B. Ludaescher

Today’s Agenda

• Quiz
• Finish up MySQL
• More on SQL queries
• Relational Algebra
Why are mathematical definitions useful?

Can you think of examples when a mathematical definition improved a practical problem?

Query Languages for Relational DBs

<table>
<thead>
<tr>
<th>SQL</th>
<th>Relational Algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Practical definition of relational database</td>
<td>Mathematical definition of relational database</td>
</tr>
<tr>
<td>Operates on tables (with duplicates -- bags)</td>
<td>Operates on relations (i.e., sets)</td>
</tr>
</tbody>
</table>
| Various keywords, statements
  SELECT, FROM, WHERE, ... | Set-based operations
  Intersection, Union, ... |

“cross-fertilization”
Commercial DBMSs – Sets or Bags?

• The default is to produce a bag (or multiset) of rows as a query answer
• If you want a set, use DISTINCT
• Why do you think they do this?

• Note that even though relational algebra was originally defined as set-based, SQL queries are represented internally using relational algebra (w/ extra operators)
• There are also versions of relational algebra defined using bag semantics

The Plan …

• Present mathematical definition of a relational database (relational algebra)

• Intermix relational algebra and SQL
  – We'll use relational algebra again when we talk about evaluating and optimizing queries
Mathematically Describing a Relational DB

• A relation is a set of tuples
  – See the original definition of the model ... optional reading (Codd 1970)

• Define query operators as set-theoretic functions

Together these form the relational algebra

Cross Products

• Let \( A = \{a, b, c\} \) and \( B = \{1, 2\} \)

• In set theory, the cross product is defined as

\[
A \times B = \{(a, 1), (b, 1), (c, 1), (a, 2), (b, 2), (c, 2)\}
\]

• \( A \times B \) is a set consisting of ordered pairs (2-tuples) where each pair consists of an element from \( A \) and an element from \( B \)
Practice Question

• Suppose A = \{a, b, c\} and B = \{1, 2\}

• What is B \times B?

B \times B = \{(1, 1), (2, 1), (1, 2), (2, 2)\}

Defining Relations

• Suppose we have the relation

  – Person(name, salary, num, status) with domains

    NameValues = \{all possible strings of 30 characters\}
    SalValues = \{real numbers between 0 and 100,000\}
    StatusValues = \{“f”, “p”\}
    NumValues = \{integers between 0 and 9999\}

• Any instance of the relation is always a subset (\subseteq) of

  – NameValues \times SalValues \times NumValues \times StatusValues

* Note that a “domain” is a set of simple, atomic values
Defining Relations

- Each relation instance is a subset of the cross product of its domains (see the Codd paper)
- One element of a relation is called a tuple
  - If n domains, then n-tuples

- A relation is **always** a set by definition
  - Recall: If you add the element 2 to the set \{1, 2, 3, 4\}, then you still have the set \{1, 2, 3, 4\}
  - If you add the tuple \{101, "J. Smith", 1000, "checking"\} to the account relation instance you don’t change the result

Set Theory Refresher

- Consider two sets
  - \(S_1 = \{1, 3, 5, 7\}\)
  - \(S_2 = \{1, 2, 3, 4\}\)

- What do these return?
  - \(S_1 \cap S_2\)
  - \(S_1 \cup S_2\)
  - \(S_1 - S_2\)
  - \(S_1 \times S_2\)
Relational Algebra has Additional Operations

- Consider two sets
  - $S_1 = \{1, 3, 5, 7\}$
  - $S_2 = \{1, 2, 3, 4\}$

- We will talk about these a bit later
  - $S_1 \bowtie_{\text{condition}} S_2$
  - $S_1 \div S_2$
  - $\sigma_{\text{condition}}(S_1)$
  - $\pi_{\text{attribute-list}}(S_1)$
  - $\rho_{\text{renaming-specification}}(S_1)$

Relational Algebra as a Query Language

- We don’t normally use relational algebra directly
  - Products don’t allow you to write relational algebra queries

- But, it is used internally in a DBMS to represent a query plan

- It is also often used in theoretical work on databases
  - (although fragments of first order logic are frequently used as well …)
Relational Algebra Queries w/out Operators

- What does the following SQL query return?

  ```sql
  SELECT *
  FROM Student;
  ```

  **Answer:** Student
  (i.e., this query is the identity function)

- A relation name by itself is a valid relational algebra query
- Listing the relation name just returns the tuples in the relation

Relational Algebra: **Selection** operator (\(\sigma\))

<table>
<thead>
<tr>
<th>Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
</tr>
<tr>
<td>101</td>
</tr>
<tr>
<td>102</td>
</tr>
<tr>
<td>103</td>
</tr>
<tr>
<td>104</td>
</tr>
<tr>
<td>105</td>
</tr>
</tbody>
</table>

The relational algebra query:

\[
\sigma_{\text{Balance}<3000}(\text{Account})
\]

is similar to the SQL query:

```sql
SELECT *
FROM Account
WHERE Balance < 3000;
```
Relational Algebra: Selection operator (σ)

Select (σ) is a unary operator:

σ : R → R

- i.e., it is always applied to a single relation

σ_{Balance < 3000}(Account)

select operator

the predicate (condition)

Attribute Comparator (≥, >, =, ≠, ≤, <) Attribute | Constant

Examples

σ_{Balance < 3000}(Account)

σ_{Number = 103}(Account)

σ_{Balance = Number}(Account)

σ_{Type = "checking"}(σ_{Balance < 3000}(Account))

<table>
<thead>
<tr>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>
Relational Algebra: **Projection operator (π)**

The relational algebra query:

$$\pi_{\text{Number, Owner}}(\text{Account})$$

is similar to the SQL query:

```sql
SELECT Number, Owner
FROM Account;
```

<table>
<thead>
<tr>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>

Relational Algebra: **Projection operator (π)**

Project (π) is also a unary operator:

$$\pi : R \rightarrow R$$

– i.e., it is always applied to a single relation

$$\pi_{\text{Number, Owner}}(\text{Account})$$
**Example**

\[ \pi_{\text{Owner}}(\text{Account}) \quad \text{vs.} \quad \text{SELECT Owner FROM Account;} \]

<table>
<thead>
<tr>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>

- Relations are always sets
- So the query answer is a *set* of names ...
- and J. Smith appears just once in the answer

**Combining Select and Project**

\[ \pi_{\text{Owner}}(\sigma_{\text{Balance} < 3000}(\text{Account})) \]
\[ \sigma_{\text{Balance} < 3000}(\pi_{\text{Owner, Balance}}(\text{Account})) \]
\[ \pi_{\text{Owner}}(\sigma_{\text{Balance} < 3000}(\pi_{\text{Owner, Balance}}(\text{Account}))) \]
\[ \sigma_{\text{Type} = \text{"checking"}}(\sigma_{\text{Balance} < 3000}(\pi_{\text{Owner, Balance}}(\text{Account}))) \]

Do you seen any problems with these queries?
Are any of these equivalent?
Equivalence implies for any DB instance

<table>
<thead>
<tr>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>
Relational Algebra Cross Products

• Given \( A = \{a, b, c\}, B = \{1, 2\}, \) and \( C = \{x, y\}, \) the cross product \((A \times B) \times C\) in set theory is

\[
\{((a, 1), x), ((b, 1), x), (c, 1, x), \ldots, ((c, 2), y)\}
\]

• Codd simplified cross products in relational algebra by eliminating parentheses … “flattening” the tuples

\[
\{(a, 1, x), (b, 1, x), (c, 1, x), \ldots, (c, 2, y)\}
\]

Relational Algebra: Cross Product operator (X)

• Used in the basic definition of a relation

  “An instance of a relation is a subset of the cross product of its domains”

• Is also an operator in the relational algebra

  So, we can use it in queries
Example

• Suppose we have these relations in a university DB

   Teacher(tnum, tname)
   Course(cnum, cname)

• This is just a simple example!
• If we developed an actual university schema, it would be much more detailed than this

<table>
<thead>
<tr>
<th>tnum</th>
<th>tname</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Smith</td>
</tr>
<tr>
<td>105</td>
<td>Crowley</td>
</tr>
<tr>
<td>110</td>
<td>Yerion</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cnum</th>
<th>tname</th>
</tr>
</thead>
<tbody>
<tr>
<td>346</td>
<td>Operating Systems</td>
</tr>
<tr>
<td>491</td>
<td>Senior Design</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tnum</th>
<th>tname</th>
<th>cnum</th>
<th>tname</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Smith</td>
<td>346</td>
<td>Operating Systems</td>
</tr>
<tr>
<td>105</td>
<td>Crowley</td>
<td>346</td>
<td>Operating Systems</td>
</tr>
<tr>
<td>110</td>
<td>Yerion</td>
<td>346</td>
<td>Operating Systems</td>
</tr>
<tr>
<td>101</td>
<td>Smith</td>
<td>491</td>
<td>Senior Design</td>
</tr>
<tr>
<td>105</td>
<td>Crowley</td>
<td>491</td>
<td>Senior Design</td>
</tr>
<tr>
<td>110</td>
<td>Yerion</td>
<td>491</td>
<td>Senior Design</td>
</tr>
</tbody>
</table>

• The cross product produces every possible combination of teacher and course
**Relational Algebra: Join operator (⋈)**

The relational algebra query

\[
\text{Account} \bowtie_{\text{Number} = \text{Account}} \text{Deposit}
\]

is equivalent to …

\[
\sigma_{\text{Number} = \text{Account}} (\text{Account} \times \text{Deposit})
\]
Relational Algebra: Join operator (⋈)

The join operator is defined for convenience

$$R_1 \bowtie_{a_1 = a_2} R_2 \equiv \sigma_{a_1 = a_2}(R_1 \times R_2)$$

- Thus, we don’t really “need” the join operator
- Any query with a join can always be rewritten into cross product followed by selection
- And vice-versa … it is also possible to define cross-product based on join

A Few Details about Join

- Each simple Boolean predicate in the join condition must compare an attribute from one relation to an attribute in the other relation
  - In the query
    $$\text{Account} \bowtie_{\text{Number} = \text{Account} \land \text{type} = "checking"} \text{Deposit}$$
    - type="checking" is not a join condition
- If you have a join with NO condition, then it is just a cross product
Examples

- Student $\bowtie_{\text{advisor}=\text{number}}$ Faculty
- Student $S \bowtie_{S\text{.age} < F\text{.age}}$ Faculty $F$
- Student $S \bowtie_{S\text{.salary} \geq F\text{.age}}$ Faculty $F$

- Join is sometimes called “theta-join” or “$\Theta$-join” where $\Theta$ represents any of the 6 comparators

- The most common join is called a equi-join (for equality condition)

  $$R_1 \bowtie_{a_1 = a_2} R_2$$

Challenge Question

How can you in general convert a basic SQL statement (SELECT-FROM-WHERE) to an equivalent relational algebra expression

- SELECT DISTINCT attributes
- FROM $T_1, T_2, \ldots$
- WHERE conditions
Basic SQL = SPJ (except tables vs. relations)

A basic SQL query of the form

```sql
SELECT DISTINCT attributes 
FROM T1, T2, ...
WHERE conditions
```

is the same as

```
\pi_\text{attributes} (\sigma_\text{conditions} (T_1 \times T_2 \times \ldots))
```

SELECT-FROM-WHERE queries are sometimes described
as equivalent to the Select-Project-Join (SPJ) subset of
relational algebra

– Confusingly, SQL SELECT == RA Project (not RA Select!)