CS 4604: Introduction to Database Management Systems

B. Aditya Prakash

Lecture #3: SQL and Relational Algebra
Reminder

- NO office hours today!
- Extended (10am-12noon) hours on 1/30 and 2/4
Formal Query Languages

- How do we collect information?
- E.g. Find SSNs of people in 4604 (recall everything is a set!)
What is SQL

- SQL = Structured Query Language (pronounced “sequel”).
- Language for defining as well as querying data in an RDBMS.
- Primary mechanism for querying and modifying the data in an RDBMS.
- SQL is declarative:
  - Say what you want to accomplish, without specifying how.
  - One of the main reasons for the commercial success of RDMBSs.
- SQL has many standards and implementations:
  - ANSI SQL
  - SQL-92/SQIL2 (null operations, outerjoins)
  - SQL-99/SQIL3 (recursion, triggers, objects)
  - Vendor-specific variations.
Relational Algebra

• Relational algebra is a notation for specifying queries about the contents of relations

• Notation of relational algebra eases the task of reasoning about queries

• Operations in relational algebra have counterparts in SQL
What is an Algebra?

- An algebra is a set of operators and operands
  - Arithmetic: operands are variables and constants, operators are +,-,×,÷, /, etc.
  - Set algebra: operands are sets and operators are ∩, U, -

- An algebra allows us to
  - construct expressions by combining operands and expression using operators
  - has rules for reasoning about expressions

\[ a^2 + 2 \times a \times b + 2b, \quad (a + b)^2 \]
\[ R - (R - S), \quad R \cap S \]
Basics of Relational Algebra

- Operands are relations, thought of as sets of tuples.
- Think of operands as variables, whose tuples are unknown.
- Results of operations are also sets of tuples.
- Think of expressions in relational algebra as queries, which construct new relations from given relations.

Four types of operators:
- Select/Show parts of a single relation: projection and selection.
- Usual set operations (union, intersection, difference).
- Combine the tuples of two relations, such as cartesian product and joins.
- Renaming.
Projection

- The projection operator produces from a relation R a new relation containing only some of R’s columns.

- “Delete” (i.e. not show) attributes not in projection list.

- Duplicates eliminated (sets vs multisets).

- To obtain a relation containing only the columns A1, A2, ..., An of R.

  **RA:** \( \pi \ A_1, A_2, \ldots, A_n \ (R) \)

  **SQL:** SELECT A1, A2, ..., An FROM R;
# Projection Example

\( S_2 \)

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

\[ \pi_{\text{sname}, \text{rating}}(S_2) \]

<table>
<thead>
<tr>
<th>sname</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>yuppy</td>
<td>9</td>
</tr>
<tr>
<td>lubber</td>
<td>8</td>
</tr>
<tr>
<td>guppy</td>
<td>5</td>
</tr>
<tr>
<td>rusty</td>
<td>10</td>
</tr>
</tbody>
</table>

\[ \pi_{\text{age}}(S_2) \]

<table>
<thead>
<tr>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>35.0</td>
</tr>
<tr>
<td>55.5</td>
</tr>
</tbody>
</table>
Selection

- The selection operator applied to a relation R produces a new relation with a **subset of R’s tuples**
- The tuples in the resulting relation satisfy some condition C that involves the attributes of R
  - with duplicate removal

\[
\text{RA: } \sigma_C(R)
\]

**SQL:**  \( \text{SELECT } * \text{FROM } R \text{ WHERE } C; \)

- The WHERE clause of a SQL command corresponds to \( \sigma( ) \)
Selection: Syntax of Conditional

- Syntax of conditional (C): similar to conditionals in programming languages.

- Values compared are constants and attributes of the relations mentioned in the FROM clause.

- We may apply usual arithmetic operators to numeric values before comparing them.

**RA**  Compare values using standard arithmetic operators.

**SQL**  Compare values using =, <>, <, >, <=, >=.
### Selection Example

#### $S_2$

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>uppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

#### $\sigma_{rating > 8}(S_2)$

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>uppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

#### $\pi_{sname, rating}(\sigma_{rating > 8}(S_2))$

<table>
<thead>
<tr>
<th>sname</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>uppy</td>
<td>9</td>
</tr>
</tbody>
</table>
rusty |10      |
Set Operations: Union

- Standard definition: The union of two relations R and S is the set of tuples that are in R, or S or in both.

- When is it valid?
  - R and S must have identical sets of attributes and the types of the attributes must be the same.
  - The attributes of R and S must occur in the same order.
Set Operations: Union

- RA  R U S
- SQL (SELECT * FROM R)
  UNION
  (SELECT * FROM S);
Set Operations: Intersection

- The intersection of $R$ and $S$ is the set of tuples in both $R$ and $S$
- Same conditions hold on $R$ and $S$ as for the union operator
  - $RA \quad R \cap S$
  - $SQL \quad (SELECT * FROM R) \quad INTERSECT \quad (SELECT * FROM S)$
Set Operations: Difference

- Set of tuples in R but NOT in S
- Same conditions on R and S as union
- $R \cap S$
- SQL: $(\text{SELECT * FROM R}) \text{ EXCEPT } (\text{SELECT * FROM S})$
- $R - (R - S) = R \cap S$
## Difference

### $S1$ vs. $S2$

### $S1$ - $S2$

<table>
<thead>
<tr>
<th>sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
</tbody>
</table>
Cartesian Product

- The Cartesian product (or cross-product or product) of two relations R and S is a the set of pairs that can be formed by pairing each tuple of R with each tuple of S.
  - The result is a relation whose schema is the schema for R followed by the schema for S.

RA: \( R \times S \)

SQL: \( \text{SELECT * FROM } R, S \);
### Cartesian Product

#### S1

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

#### R1

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

#### S1 X R1

<table>
<thead>
<tr>
<th>(sid)</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>(sid)</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>31</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

We rename attributes to avoid ambiguity or we prefix attribute with the name of the relation it belongs to.
Theta-Join

- The theta-join of two relations $R$ and $S$ is the set of tuples in the Cartesian product of $R$ and $S$ that satisfy some condition $C$.

$$RA: \quad R \bowtie_C S$$

$$SQL: \quad SELECT * \quad FROM \quad R, S \quad WHERE \quad C;$$

$$R \bowtie_C S = \sigma_C (R \times S)$$
**Theta-Join**

### $S1$ 

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

### $R1$ 

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

$S1 \bowtie_{S1.sid < R1.sid} R1$

<table>
<thead>
<tr>
<th>(sid)</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>(sid)</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

$R \bowtie_{C} S = \sigma_{C} (R \times S)$
Natural Join

- The natural join of two relations R and S is a set of pairs of tuples, one from R and one from S, that agree on whatever attributes are common to the schemas of R and S.
- The schema for the result contains the union of the attributes of R and S.
- Assume the schemas R(A,B, C) and S(B, C,D)

RA: \[ R \bowtie S \]

SQL: 
\[
\text{SELECT * FROM R, S}
\text{WHERE R.B = S.B AND R.C = S.C;}
\]
Operators so far

- **Remove parts of single relations**
  - Projection: $\pi_{(A,B)}(R)$ and SELECT A, B FROM R
  - Selection: $\sigma_C(R)$ and SELECT * FROM R WHERE C
  - Combining Projection and Selection:
    - $\pi_{(A,B)}(\sigma_C(R))$
    - SELECT A, B FROM R WHERE C
Operations so far

- Set operations
  - R and S must have the same attributes, same attribute types, and same order of attributes
  - Union: $R \cup S$ and (R) UNION (S)
  - Intersection: $R \cap S$ and (R) INTERSECT (S)
  - Difference: $R - S$ and (R) EXCEPT (S)
Operations so far

- Combine the tuples of two relations
  - Cartesian Product: $R \times S$, .... FROM $R$, $S$ ..... 
  - Theta Join: $R \bowtie_C S$, .... FROM $R$, $S$ WHERE $C$
  - Natural Join: $R \bowtie S$
Disambiguation and Renaming

- If two relations have the same attribute, **disambiguate** the attributes by prefixing the attribute with the name of the relation it belongs to.

- How do we answer the query “Name pairs of students who live at the same address”? 

  Students(Name, Address)
  
  – We need to take the cross-product of Students with itself?
  
  – How do we refer to the two “copies” of Students?
  
  – Use the rename operator.
Disambiguation and Renaming

**RA:** give R the name S;
R has n attributes,
which are $\rho_S (A_1,A_2, \ldots A_n) (R)$
called A1, A2, \ldots , An in S

**SQL:** Use the **AS** keyword in the **FROM** clause:
Students AS Students1 renames Students to Students1.

**SQL:** Use the **AS** keyword in the **SELECT** clause to rename attributes.
Name pairs of students who live at the same address:

RA: \[ \pi S1.\text{Name}, S2.\text{Name} \left( \sigma S1.\text{Address} = S2.\text{Address} \left( \rho_{S1}(\text{Students}) \times \rho_{S2}(\text{Students}) \right) \right) \]

SQL: \[ \text{SELECT S1.name, S2.name} \]
\[ \text{FROM Students AS S1, Students AS S2} \]
\[ \text{WHERE S1.address = S2.address} \]
Disambiguation and Renaming

- Name pairs of students who live at the same address:

  SQL: SELECT S1.name, S2.name
       FROM Students AS S1, Students AS S2
       WHERE S1.address = S2.address

- Are these correct?
- No !!! the result includes tuples where a student is paired with himself/herself

- Solution: Add the condition S1.name <> S2.name.
Other Details in SQL

- Read Chapters 6.1.3-6.1.8 of the textbook for string comparison, pattern matching, NULL and UNKNOWN values, dates and times, and ordering the output.
Independence of Operators

- The operators we have covered so far are:

\[ \pi_{A,B}(R), \sigma_C(R), \rho_S(A_1, A_2)(R), R \cup S, R \cap S, R - S, R \times S, R \Join S, R \Join_C S \]

- Are all of them needed?

- NO!
Independence of Operators

\[ R \cap S = R - (R - S) \]

\[ R \bowtie_C = \sigma_C (R \times S) \]

\[ R \bowtie S = ?? \]
Independence of Operators

\[ R \bowtie S \]

- Suppose \( R \) and \( S \) share the attributes \( A_1, A_2, \ldots, A_n \)
- Let \( L \) be the list of attributes in \( R \) followed by the list of attributes in \( S \)
- Let \( C \) be the condition

\[
R.A_1 = S.A_1 \ AND \ R.A_2 = S.A_2 \ AND \ \ldots \ \AND \ R.A_n = S.A_n
\]

\[
R \bowtie S = \pi_L(\sigma_C(R \times S))
\]