SQL and Relational Algebra

T. M. Murali

August 31, 2009
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 RDBMSs.
- SQL has many standards and implementations:
  - ANSI SQL
  - SQL-92/SQL2 (null operations, outerjoins)
  - SQL-99/SQL3 (recursion, triggers, objects)
  - Vendor-specific variations.
What is Relational Algebra?

- Relational algebra is a notation for specifying queries about the contents of relations.
- Relational algebra eases the task of reasoning about queries.
- Operations in relational algebra have counterparts in SQL.
- To process a query, a DBMS translates SQL into a notation similar to relational algebra.
What is an Algebra?

- An algebra is a set of operators and operands.
What is an Algebra?

- An algebra is a set of operators and operands.
  - Arithmetic: operands are variables and constants, operators are $+,-,\times,\div,/$, etc.
  - Set algebra: operands are sets and operators are $\cup,\cap,\setminus$.
An algebra is a set of operators and operands.

- Arithmetic: operands are variables and constants, operators are $+, -, \times, \div, /$, etc.
- Set algebra: operands are sets and operators are $\cup, \cap, -$.

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

- $a^2 + 2 \times a \times b + b^2, (a + b)^2$.
- $R - (R - S), R \cap S$. 
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. (Later, we will define a relational algebra on bags.)
- Think of expressions in relational algebra as queries, which construct new relations from given relations.
- Four types of operators:
  - Remove 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.
The *projection* operator produces from a relation $R$ a new relation containing only some of $R$’s columns.

To obtain a relation containing only the columns $A_1, A_2, \ldots, A_n$ of $R$

$$\text{RA } \pi_{A_1,A_2,\ldots,A_n}(R)$$

$$\text{SQL } \text{SELECT } A_1, A_2, \ldots, A_n \text{ FROM } R;$$
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 $R$.

\[
\text{RA} \quad \sigma_C(R) \\
\text{SQL} \quad \text{SELECT} \ \star \\
\quad \text{FROM} \ R \\
\quad \text{WHERE} \ C;
\]

- The WHERE clause of an SQL command corresponds to $\sigma()$. 

Selection: Syntax of Conditional

- Syntax of 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 =, <>, <, >, <=, >=.
Set Operations: Union

- The union of two relations $R$ and $S$ is the set of tuples that are in $R$ or in $S$ or in both.
Set Operations: Union

- The *union* of two relations $R$ and $S$ is the set of tuples that are in $R$ or in $S$ or in both.
- $R$ 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

- The *union* of two relations $R$ and $S$ is the set of tuples that are in $R$ or in $S$ or in both.
- $R$ 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.

\[
\text{RA} \quad R \cup S \\
\text{SQL} \quad (\text{SELECT} \ast \text{FROM} \; R) \\
\quad \text{UNION} \\
\quad (\text{SELECT} \ast \text{FROM} \; S); \\
\]
Set Operations: Intersection

- The *intersection* of two relations $R$ and $S$ is the set of tuples that are in both $R$ and $S$.
- Same conditions hold on $R$ and $S$ as for the union operator.

```sql
RA  R ∩ S
SQL (SELECT * FROM R)
    INTERSECT
    (SELECT * FROM S);
```
Set Operations: Difference

- The difference of two relations $R$ and $S$ is the set of tuples that are in $R$ but not in $S$.
- Same conditions hold on $R$ and $S$ as for the union operator.

\[
\text{RA} \quad R - S \\
\text{SQL} \quad (\text{SELECT } * \text{ FROM } R) \text{ EXCEPT} (\text{SELECT } * \text{ FROM } S);\\n\]
Set Operations: Difference

- The *difference* of two relations $R$ and $S$ is the set of tuples that are in $R$ but not in $S$.
- Same conditions hold on $R$ and $S$ as for the union operator.
  
  **RA**  \[ R - S \]
  
  **SQL**  \[ (\text{SELECT } * \text{ FROM } R) \text{ EXCEPT} (\text{SELECT } * \text{ FROM } S) \];

- \[ R - (R - S) = \]
Set Operations: Difference

- The *difference* of two relations $R$ and $S$ is the set of tuples that are in $R$ but not in $S$.
- Same conditions hold on $R$ and $S$ as for the union operator.
  
  **RA**  
  $R - S$

  **SQL**  
  $(SELECT * FROM R)$

  `EXCEPT`

  $(SELECT * FROM S)$;

- $R - (R - S) = R \cap S$.
- Compare to
  
  $(SELECT * FROM R)$ `EXCEPT` $(SELECT * FROM S))$;  
  $(SELECT * FROM R)$ `INTERSECT` $(SELECT * FROM S)$;
Cartesian Product

- The Cartesian product (or cross-product or product) of two relations \( R \) and \( S \) is 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 \( S \).
  - We rename attributes to avoid ambiguity or we prefix attribute with the name of the relation it belongs to.

\[
\begin{align*}
\text{RA} & \quad R \times S \\
\text{SQL} & \quad \text{SELECT * FROM } R, S;
\end{align*}
\]
The \textit{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 $C$.

\begin{align*}
\text{RA} & \quad R \bowtie_{C} S \\
\text{SQL} & \quad \text{SELECT} \ast \\
& \text{FROM} \ R, S \\
& \text{WHERE} \ C;
\end{align*}
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 \( C \).

\[
\begin{align*}
\text{RA} & \quad R \bowtie_C S \\
\text{SQL} & \quad \text{SELECT} \ * \\
& \quad \text{FROM} \ R, \ S \\
& \quad \text{WHERE} \ C;
\end{align*}
\]

\( R \bowtie_C S = \)
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 $C$.

\[ RA \quad R \bowtie_C S \]

\[ SQL \quad SELECT * \]
\[ \quad FROM R, S \]
\[ \quad WHERE C; \]

\[ R \bowtie_C S = \sigma_C(R \times S). \]
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  \[ R \Join S \]
  
  SQL  \[ \text{SELECT R.A, R.B, R.C, S.D} \]
  FROM R,S
  WHERE R.B = S.B AND R.C = S.C;

- A dangling tuple is one that fails to pair with any tuple in the other relation.
Operators Covered So Far

- Remove parts of a single relation:
  - projection: $\pi_{A,B}(R)$ and SELECT A, B FROM R.
  - selection: $\sigma_C(R)$ and SELECT * FROM R WHERE C.

- 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).

- Combine the tuples of two relations:
  - Cartesian product: $R \times S$ and ...
  - Theta-join: $R \Join \sigma_C S$ and ...
  - Natural join: $R \Join S$; in SQL, list the conditions that the common attributes be equal in the WHERE clause.
Operators Covered So Far

- Remove parts of a single relation:
  - 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. Canonical SQL query.
Operators Covered So Far

- Remove parts of a single relation:
  - 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 \( C \). Canonical SQL query.

- Set operations (\( R \) and \( S \) must have the same attributes, same attribute types, and same order of attributes):
  - union: \( R \cup S \) and \((R) \cup (S)\).
  - intersection: \( R \cap S \) and \((R) \cap (S)\).
  - difference: \( R - S \) and \((R) \setminus (S)\).
Operators Covered So Far

- Remove parts of a single relation:
  - 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`. Canonical SQL query.

- 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)`.

- Combine the tuples of two relations:
  - Cartesian product: $R \times S$ and `... FROM R, S ...`.
  - Theta-join: $R \bowtie^C S$ and `... FROM R, S WHERE C`.
  - Natural join: $R \bowtie S$; in SQL, list the conditions that the common attributes be equal in the `WHERE` clause.
Other Details in SQL

- Read Chapters 6.1.3-6.1.8 of the textbook for strings 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: $\pi_{A,B}(R)$, $\sigma_C(R)$, $\rho_{S(A_1,A_2)}(R)$, $R \cup S$, $R \cap S$, $R - S$, $R \times S$, $R \Join S$, $R \Join_S S$.

- Do we need all these operators?
Independence of Operators

- The operators we have covered so far are: \( \pi_{A,B}(R) \), \( \sigma_C(R) \), \( \rho_{S(A_1, A_2)}(R) \), \( R \cup S \), \( R \cap S \), \( R - S \), \( R \times S \), \( R \bowtie S \), \( R \bowtie_C S \).

- Do we need all these operators? **NO!**

- \( R \cap S = R - (R - S) \).

- \( R \bowtie_C S = \sigma_C(R \times S) \).

- \( R \bowtie S = ?? \).
Independence of Operators

- The operators we have covered so far are: \( \pi_{A,B}(R) \), \( \sigma_C(R) \), \( \rho_{S(A_1,A_2)}(R) \), \( R \cup S \), \( R \cap S \), \( R - S \), \( R \times S \), \( R \bowtie S \), \( R \bowtie_C S \).

- Do we need all these operators? NO!

- \( R \cap S = R - (R - S) \).

- \( R \bowtie_C S = \sigma_C(R \times S) \).

- \( R \bowtie S = ?? \).
  - Suppose \( R \) and \( S \) share the attributes \( A_1, A_2, \ldots A_n \).
Independence of Operators

The operators we have covered so far are: $\pi_{A,B}(R)$, $\sigma_{C}(R)$, $\rho_{S(A_1,A_2)}(R)$, $R \cup S$, $R \cap S$, $R - S$, $R \times S$, $R \bowtie S$, $R \bowtie S$. 

Do we need all these operators? NO!

$R \cap S = R - (R - S)$.

$R \bowtie S = \sigma_{C}(R \times S)$.

$R \bowtie S = ??.$

Suppose $R$ and $S$ share the attributes $A_1, A_2, \ldots A_n$.

Let $L$ be the list of attributes in $R$’s schema followed by the list of attributes that are only in $S$’s schema.
Independence of Operators

- The operators we have covered so far are: \( \pi_{A,B}(R) \), \( \sigma_C(R) \), \( \rho_{S(A_1,A_2)}(R) \), \( R \cup S \), \( R \cap S \), \( R - S \), \( R \times S \), \( R \bowtie S \), \( R \bowtie_C S \).
- Do we need all these operators? NO!
- \( R \cap S = R - (R - S) \).
- \( R \bowtie S = \sigma_C(R \times S) \).
- \( R \bowtie_C S =?? \).
  - Suppose \( R \) and \( S \) share the attributes \( A_1, A_2, \ldots A_n \).
  - Let \( L \) be the list of attributes in \( R \)'s schema followed by the list of attributes that are only in \( S \)'s schema.
  - Let \( C \) be the condition
    \[ R.A_1 = S.A_1 \text{ AND } R.A_2 = S.A_2 \text{ AND } \ldots \text{ AND } R.A_n = S.A_n \]
Independence of Operators

- The operators we have covered so far are: \( \pi_{A,B}(R) \), \( \sigma_C(R) \), \( \rho_{S(A_1,A_2)}(R) \), \( R \cup S \), \( R \cap S \), \( R - S \), \( R \times S \), \( R \bowtie S \), \( R \bowtie_C S \).

- Do we need all these operators? NO!

- \( R \cap S = R - (R - S) \).

- \( R \bowtie_C S = \sigma_C(R \times S) \).

- \( R \bowtie_S S = ?? \).

  - Suppose \( R \) and \( S \) share the attributes \( A_1, A_2, \ldots, A_n \).
  - Let \( L \) be the list of attributes in \( R \)'s schema followed by the list of attributes that are only in \( S \)'s schema.
  - Let \( C \) be the condition
    \[ R.A_1 = S.A_1 \quad \text{AND} \quad R.A_2 = S.A_2 \quad \text{AND} \quad \ldots \quad \text{AND} \quad R.A_n = S.A_n \]
  - \( R \bowtie_S S = \pi_L(\sigma_C(R \times S)) \)
Introduction to RA and SQL Queries and Operations

Independence of Operators

- The operators we have covered so far are: $\pi_{A,B}(R)$, $\sigma_C(R)$, $\rho_{S(A_1A_2)}(R)$, $R \cup S$, $R \cap S$, $R - S$, $R \times S$, $R \bowtie S$, $R \bowtie_C S$.

- Do we need all these operators? NO!

- $R \cap S = R - (R - S)$.

- $R \bowtie S = \sigma_C(R \times S)$.

- $R \bowtie_C S = ??$.

- Suppose $R$ and $S$ share the attributes $A_1, A_2, \ldots A_n$.

- Let $L$ be the list of attributes in $R$’s schema followed by the list of attributes that are only in $S$’s schema.

- Let $C$ be the condition $R.A_1 = S.A_1 \text{ AND } R.A_2 = S.A_2 \text{ AND } \ldots \text{ AND } R.A_n = S.A_n$.

- $R \bowtie S = \pi_L(\sigma_C(R \times S))$.

- All other operators are independent, i.e., no operator can be written in terms of the others.

T. M. Murali August 31, 2009 CS4604: SQL and Relational Algebra
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.

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

- Use the AS keyword in the SELECT clause to rename attributes.
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”?

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.
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”?
  - We need to take the cross-product of Students with itself.
  - How do we refer to the two “copies” of Students?
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”?
  - 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.

\[
\rho_S(A_1, A_2, \ldots, A_n)(R): \text{give } R \text{ the name } S; \ R \text{ has } n \text{ attributes, which are called } A_1, A_2, \ldots, A_n \text{ in } S.
\]

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.
Example of Renaming

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

\[
\text{RA } \pi_{S1.\text{Name}, S2.\text{Name}}(\sigma_{S1.\text{Address} = S2.\text{Address}}(\rho_{S1}(\text{Students}) \times \rho_{S2}(\text{Students}))).
\]

- Are these correct?
- No, the result includes tuples where a student is paired with himself/herself.
- Add the condition \( S1.\text{Name} < S2.\text{Name} \).
Example of Renaming

Name pairs of students who live at the same address.

\[ \text{RA } \pi_{S1.Name, S2.Name} (\sigma_{S1.Address = S2.Address} (\rho_{S1(Students)} \times \rho_{S2(Students)})). \]

\[ \text{SQL } \text{SELECT S1.name, S2.name} \]
\[ \text{FROM Students AS S1, Students AS S2} \]
\[ \text{WHERE S1.address = S2.address}; \]

Are these correct?
Example of Renaming

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

\[
\begin{align*}
\text{RA} & \quad \pi_{S1.Name, S2.Name}( \\
& \quad \sigma_{S1.Address = S2.Address}(\rho_{S1}(\text{Students}) \times \rho_{S2}(\text{Students}))). \\
\text{SQL} & \quad \text{SELECT S1.name, S2.name} \\
& \quad \text{FROM Students AS S1, Students AS S2} \\
& \quad \text{WHERE S1.address = S2.address;} \\
\end{align*}
\]

- Are these correct?
- No, the result includes tuples where a student is paired with himself/herself.
Example of Renaming

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

  \[
  \text{RA } \pi_{S1.Name,S2.Name} (\sigma_{S1.Address = S2.Address} (\rho_{S1}(\text{Students}) \times \rho_{S2}(\text{Students}))).
  \]

  \[
  \text{SQL } \text{SELECT S1.name, S2.name}
  \text{ FROM Students AS S1, Students AS S2}
  \text{ WHERE S1.address = S2.address;}
  \]

- Are these correct?
- No, the result includes tuples where a student is paired with himself/herself.
- Add the condition \( S1.name < S2.name \).
Example RA/SQL Queries

Solve problems in Handout 1.