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 RDMBSs.
- SQL has many standards and implementations:
  - ANSI SQL
  - SQL-92/SQL2 (null operations, outerjoins)
  - SQL-99/SQL3 (recusion, 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 $\cup, \cap, -$.

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 $\cup, \cap, -$.
- 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.

Projection

- 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

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

SQL
```
SELECT A1, A2, ..., An
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.

RA $\sigma_C(R)$

SQL
```
SELECT *
FROM R
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.

RA $R \cup S$

SQL
```
(SELECT * FROM R)
UNION
(SELECT * 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.

RA $R \cap S$

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.

RA $R - S$

SQL
```
(SELECT * FROM R)
EXCEPT
(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.

RA $R - S$

SQL
```
(SELECT * FROM R)
EXCEPT
(SELECT * 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 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 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.
  - We rename attributes to avoid ambiguity or we prefix attribute with the name of the relation it belongs to.

RA $R \times S$

SQL
```
SELECT *
FROM R, S;
```

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 $R \bowtie_{C} S$

SQL
```
SELECT *
FROM R, S
WHERE C;
```

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 $R \underset{C}{\bowtie} S$

SQL
```
SELECT *
FROM R, S
WHERE C;
```

- $R \underset{C}{\bowtie} S =$

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 $R \underset{C}{\bowtie} S$

  SQL
  ```
  SELECT *
  FROM R, S
  WHERE C;
  ```
- $R \underset{C}{\bowtie} S = \sigma_C(R \times S)$.

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
  ```
  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`.

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 tyes, 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)`.

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 tyes, 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: $R \underset{C}{\bowtie} S$ 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: $\pi_{A,B}(R)$, $\sigma_C(R)$, $\rho_{S(A_1,A_2)}(R)$, $R \cup S$, $R \cap S$, $R - S$, $R \times S$, $R \bowtie S$, $R \underset{C}{\bowtie} S$.
- Do we need all these operators?

Independence of Operators

- The operators we have covered so far are: $\pi_{A,B}(R)$, $\sigma_C(R)$, $\rho_{S(A_1,A_2)}(R)$, $R \cup S, R \cap S, R - S, R \times S, R \bowtie S, R \underset{C}{\bowtie} S$.
- Do we need all these operators? NO!
- $R \cap S = R - (R - S)$.
- $R \underset{C}{\bowtie} S = \sigma_C(R \times S)$.
- $R \bowtie S =??$.

Independence of Operators

- The operators we have covered so far are: $\pi_{A,B}(R)$, $\sigma_C(R)$, $\rho_{S(A_1,A_2)}(R)$, $R \cup S, R \cap S, R - S, R \times S, R \bowtie S, R \underset{C}{\bowtie} S$.
- Do we need all these operators? NO!
- $R \cap S = R - (R - S)$.
- $R \underset{C}{\bowtie} S = \sigma_C(R \times S)$.
- $R \bowtie S = ??$.
  - 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: $\pi_{A,B}(R)$, $\sigma_C(R)$, $\rho_{S(A_1,A_2)}(R)$, $R \cup S, R \cap S, R - S, R \times S, R \bowtie S, R \underset{C}{\bowtie} S$.
- Do we need all these operators? NO!
- $R \cap S = R - (R - S)$.
- $R \underset{C}{\bowtie} S = \sigma_C(R \times S)$.
- $R \bowtie S =??$.
  - 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: $\pi_{A,B}(R)$, $\sigma_C(R)$, $\rho_{S(A_1,A_2)}(R)$, $R \cup S, R \cap S, R - S, R \times S, R \bowtie S, R \underset{C}{\bowtie} S$.
- Do we need all these operators? NO!
- $R \cap S = R - (R - S)$.
- $R \underset{C}{\bowtie} S = \sigma_C(R \times S)$.
- $R \bowtie S =??$.
  - 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.
  - Let C be the condition
    $R.A_1 = S.A_1$ `AND` $R.A_2 = S.A_2$ `AND` $\ldots$ `AND` $R.A_n = S.A_n$

Independence of Operators

- The operators we have covered so far are: $\pi_{A,B}(R)$, $\sigma_C(R)$, $\rho_{S(A_1,A_2)}(R)$, $R \cup S, R \cap S, R - S, R \times S, R \bowtie S, R \underset{C}{\bowtie} S$.
- Do we need all these operators? NO!
- $R \cap S = R - (R - S)$.
- $R \underset{C}{\bowtie} S = \sigma_C(R \times S)$.
- $R \bowtie S =??$.
  - 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.
  - Let C be the condition $R.A_1 = S.A_1$ AND $R.A_2 = S.A_2$ AND ... AND $R.A_n = S.A_n$
  - $R \bowtie S = \pi_L(\sigma_C(R \times S))$

Independence of Operators

- The operators we have covered so far are: $\pi_{A,B}(R)$, $\sigma_C(R)$, $\rho_{S(A_1,A_2)}(R)$, $R \cup S$, $R \cap S$, $R - S$, $R \times S$, $R \bowtie S$, $R \underset{C}{\bowtie} S$.
- Do we need all these operators? NO!
- $R \cap S = R - (R - S)$.
- $R \underset{C}{\bowtie} S = \sigma_C(R \times S)$.
- $R \bowtie S =??$.
  - 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.
  - Let C be the condition
    $R.A_1 = S.A_1$ AND $R.A_2 = S.A_2$ AND $\ldots$ AND $R.A_n = S.A_n$
  - $R \bowtie S = \pi_L(\sigma_C(R \times S))$
- All other operators are independent, i.e., no operator can be written in terms of the others.

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.

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"?

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.

RA $\rho_{S(A_1,A_2,\ldots,A_n)}(R)$: give R the name S; R has n attributes, which are called $A_1, A_2, \ldots, A_n$ 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.

RA $\pi_{\texttt{S1.Name},\texttt{S2.Name}}($
$\sigma_{\texttt{S1.Address = S2.Address}}(\rho_{\texttt{S1}}(\texttt{Students}) \times \rho_{\texttt{S2}}(\texttt{Students})))$.

Example of Renaming

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

  RA $\pi_{\texttt{S1.Name},\texttt{S2.Name}}($
  $\sigma_{\texttt{S1.Address = S2.Address}}(\rho_{\texttt{S1}}(\texttt{Students}) \times \rho_{\texttt{S2}}(\texttt{Students}))).$

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

- Are these correct?

Example of Renaming

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

RA $\pi_{\texttt{S1.Name},\texttt{S2.Name}}($
$\sigma_{\texttt{S1.Address = S2.Address}}(\rho_{\texttt{S1}}(\texttt{Students}) \times \rho_{\texttt{S2}}(\texttt{Students})))$.

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.

Example of Renaming

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

RA $\pi_{\texttt{S1.Name},\texttt{S2.Name}}($
$\sigma_{\texttt{S1.Address = S2.Address}}(\rho_{\texttt{S1}}(\texttt{Students}) \times \rho_{\texttt{S2}}(\texttt{Students})))$.

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.
- Add the condition `S1.name < S2.name`.

Example RA/SQL Queries

Solve problems in Handout 1.