Introduction to Databases, Fall 2005
IT University of Copenhagen

Lecture 7: Relational algebra and SQL

October 24, 2005

Lecturer: Rasmus Pagh

Today's lecture

- Basics of relational algebra.
- Relational algebra on bags and commercial RDBMSs.
- More relational algebra (and SQL).
- Algebraic laws.

What you should remember from previously

In this lecture I will assume that you remember:

- The mathematical definition of a relation as a set of tuples.
- Projection and selection using `SELECT-FROM-WHERE`.
- Natural join.
- Nested SQL queries.

Next: Basics of relational algebra.

What is an algebra?

An **algebra** consists of a set of **atomic operands**, and a set of **operators**.

We can form algebraic **expressions** by applying operators to operands (which can be atomic or expressions themselves).

Example:

In the algebra of arithmetic, the atomic operands are constants and variables, and the operators are $+$, -, $/$, and $\cdot$.

Using these we can form expressions like $((x+7)/(y-3))+x$.

Relational algebra

Relational algebra, defined in its basic form by E. F. Codd in 1970, has relations as atomic operands, and various operations on relations (such as select and join) as operators.

It is the mathematical basis of SQL queries.

Example relational algebra expression:

$$\sigma_{a \geq 5}(R_1 \bowtie R_2) \cup R_3$$

using the operators $\sigma_{a \geq 5}$, $\bowtie$, and $\cup$ on operands R_1, R_2, and R_3.

Why is relational algebra useful?

Top reasons why relational algebra is covered in most database textbooks:

1. It gives another view of SQL queries, and thus a better *understanding*.
2. It is used in *query optimization* (to learn about this, enroll for Advanced Database Technology in spring 2006!)
3. It can be used for *reasoning* about relational queries and constraints.
4. It is the historical background of relational databases.

Recap of set notation

To describe the operators of relational algebra we will use mathematical notation for describing sets (recall that a relation is a set of tuples).

The notation $\{X \mid Y\}$ is used to describe "the set of all elements of the form X that satisfy the condition Y".

Examples:

- The set of negative integers: $\{x \mid x \in \mathbf{Z}$ (the set of integers), $x < 0\}$.
- The set of two-tuples of strings:
  $\{(x, y) \mid x$ is a string, and y is a string$\}$.

Selection in relational algebra

Recall that **selection** is the operation of choosing the tuples in a relation satisfying some condition.

In relational algebra, the operator σ_C is used for selection with condition C.

Formally, $\sigma_C(R) = \{t \in R \mid t \text{ satisfies } C\}$. Thus:

$$\sigma_C(R)$$

corresponds in SQL to

SELECT *
FROM R
WHERE C

Projection in relational algebra

Recall that **projection** is the operation of choosing certain attributes of a relation.

In relational algebra, the operator $\pi_{A_1,\ldots,A_n}$ is used for projection onto attributes $A_1, \ldots, A_n$.

Formally:

$$\pi_{A_1,\ldots,A_n}(R) = \{(a_1, a_2, \ldots, a_n) \mid \text{there exists } t \in R \text{ where for all } i, \\ a_i \text{ is the value of attribute } A_i \text{ of } t\}$$

Projection in relational algebra and SQL

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

corresponds in SQL to

SELECT $A_1, \ldots, A_n$
FROM R

Note that projection is the operator we use to compute relation instances in a decomposition.

Set operators in relational algebra

Since relations are sets, we can apply the standard set operators.

- **Union:** $R_1 \cup R_2 = \{x \mid x \in R_1 \text{ or } x \in R_2\}$.
- **Intersection:** $R_1 \cap R_2 = \{x \mid x \in R_1 \text{ and } x \in R_2\}$.
- **Difference:** $R_1 - R_2 = \{x \mid x \in R_1 \text{ and } x \notin R_2\}$.

In SQL, the above expressions correspond to, respectively:

- R_1 `UNION` R_2
- R_1 `INTERSECT` R_2
- R_1 `EXCEPT` R_2

Problem session (5 minutes)

Try to come up with a formal definition of the *natural join* operation, i.e., the join operation used to combine decomposed relation instances.

Suppose the relations to be joined are $R_1(A_1, \ldots, A_n, B_1, \ldots, B_m)$ and $R_2(A_1, \ldots, A_n, C_1, \ldots, C_k)$. You should express your answer in the form

$$\{X \mid Y\}$$

as in the examples you have seen.

Cheating can be done by looking at the next slide!

Natural join in relational algebra

The join operation used when recombining decomposed relations is called **natural join**, denoted by $\bowtie$ in relational algebra.

Suppose we have relations $R_1(A_1, \ldots, A_n, B_1, \ldots, B_m)$ and $R_2(A_1, \ldots, A_n, C_1, \ldots, C_k)$. Then formally:

$$\begin{aligned} R_1 \bowtie R_2 = \{(a_1, \ldots, a_n, b_1, \ldots, b_m, c_1, \ldots, c_k) \\ \mid \ & \text{there exists } t \in R_1 \text{ and } u \in R_2 \text{ with} \\ & \text{values } a_1, \ldots, a_n \text{ on attributes } A_1, \ldots, A_n \text{ of } t \text{ and } u, \\ & \text{values } b_1, \ldots, b_m \text{ on attributes } B_1, \ldots, B_m \text{ of } t, \\ & \text{and values } c_1, \ldots, c_k \text{ on attributes } C_1, \ldots, C_k \text{ of } u\} \end{aligned}$$

Natural join in relational algebra and SQL

$$R_1 \bowtie R_2$$

corresponds in SQL to

R_1 `NATURAL JOIN` R_2

Natural join is the operator we use to recover the original relation in a decomposition.

Note: `NATURAL JOIN` is not supported in Oracle.

Next: Relational algebra on bags and commercial RDBMSs.

Relational algebra vs SQL

In all the cases we saw, the correspondence between relational algebra and SQL queries is **not** what you might think!

Let's look at some examples from an Oracle session...

Relations in SQL are bags

What we have seen is that relations in SQL are **bags** (or **multisets**), i.e., tuples may appear more than once.

The fact that the same attribute may occur several times is a different (and less important) issue that we won't go into.

It is possible to define **relational algebra on bags**, whose operators are basically identical to those of SQL.

Features of relational algebra on bags

Relational algebra on bags is basically the same as relational algebra (on sets), without duplicate elimination.

- $\pi_{A_1,\ldots,A_n}(R)$ has one tuple for each tuple of R, even if the tuples become identical when some attributes are removed.
- $\sigma_C(R)$ contains all tuples of R satisfying C, including duplicates.
- A tuple occurs $x \cdot y$ times in $R_1 \bowtie R_2$ if it was formed by combining a tuple occurring x times in R_1 with a tuple occurring y times in R_2.
- $R_1 \cup R_2$ contains all tuples of R_1 and R_2, including duplicates. (This corresponds to `UNION ALL` in SQL.)
- $R_1 \cap R_2$ and $R_1 - R_2$ can also be defined – see book for details.
- A new **duplicate elimination** operator: $\delta(R)$ is the set of (different) tuples occurring in the bag R.

Why bags?

The reason for using bags (rather than sets, which are easier to handle) is database *efficiency*.

Since efficiency is crucial for commercial RDBMSs, SQL was carefully designed to allow efficient evaluation of queries.

The reason why bags are used is that *duplicate elimination* is relatively costly (requires time and memory), so it is generally an advantage to use it only when necessary.

Duplicate elimination in SQL

We can force duplicate elimination in a `SELECT-FROM-WHERE` by adding the keyword `DISTINCT`.

Example: To compute the relational algebra expression $\pi_{A_1,\ldots,A_n}(R)$ in SQL, use `SELECT DISTINCT` $A_1, \ldots, A_n$ `FROM` R.

Some SQL operators, like `UNION`, `INTERSECT`, and `EXCEPT`, automatically perform duplicate elimination.

If we always used `DISTINCT` etc., the semantics of SQL would match relational algebra. However, when efficiency is an issue this is a bad idea.

Problem session (5-10 minutes)

Suppose that R and S are relations with the same attributes, and consider the expression

$$((\sigma_{C_1}(R)) \cup S) \bowtie (\sigma_{C_2}(R))$$

Write SQL expressions that are equivalent to the above:

1. When interpreted as an expression in relational algebra (on sets).

2. When interpreted as an expression in relational algebra on bags.

Next: More relational algebra (and SQL).

Other kinds of join

- **Cartesian product.** $R_1 \times R_2$, corresponds in SQL to
  `SELECT * FROM` R_1`,` R_2.

- **Theta-join (Θ-join).** $R_1 \underset{C}{\bowtie} R_2$, corresponds in SQL to
  `SELECT * FROM` R_1`,` R_2 `WHERE` C.

- **Outerjoin.** $R_1 \overset{\circ}{\bowtie} R_2$, includes all tuples of $R_1 \bowtie R_2$, and further includes **dangling** tuples of R_1 and R_2 that are *not* matched with any tuple of the other relation, padded with `NULL` values.
  Corresponds in SQL to R_1 `FULL NATURAL OUTER JOIN` R_2.

[Figure 5.19 shown on slide]

Aggregation operators

Aggregation operators are used to compute facts about the values of some attribute in a relation.

The standard aggregation operators are: `SUM`, `AVG`, `MIN`, `MAX`, and `COUNT`, computing, respectively, the sum, average, minimum, maximum and number of the attribute values.

In relational algebra, the aggregation of attribute A in a relation R with operator `OP` is written: $\gamma_{\mathtt{OP}(A)}(R)$

Aggregation can be done in SQL by specifying the aggregation operator in the `SELECT` clause:

```
SELECT OP(A)
FROM R
```

Grouping and aggregation

Aggregation is most useful in connection with **grouping**, where the tuples of a relation are split into groups, for each of which the aggregate is computed.

The tuples in a relation are divided into groups based on the values of a specified set of **grouping attributes**, and the aggregate is computed for each group.

Aggregation of attribute A in a relation R with operator `OP` on grouping attributes $A_1, \ldots, A_n$ is written in relational algebra as $\gamma_{A_1,\ldots,A_n,\mathtt{OP}(A)}(R)$

The SQL equivalent is

SELECT $A_1, \ldots, A_n$, OP(A)
FROM R
GROUP BY $A_1, \ldots, A_n$

Semantics of aggregation

When computing an aggregate, we get one tuple for each list of values of the grouping attributes. In addition to the grouping attributes, the tuple contains the aggregate value(s) for the group.

The formal definition of the grouping and aggregation operators in relational algebra depends on the operator in question. For SUM we have:

$$\gamma_{A_1,\ldots,A_n,\texttt{SUM}(A)}(R) = \{(a_1, a_2, \ldots, a_n, s) \mid (a_1, a_2, \ldots, a_n) \in \pi_{A_1,\ldots,A_n}(R)$$
$$\text{and } s = \Sigma_{t \in \sigma_{A_1=a_1,\ldots,A_n=a_n}(R)} \pi_A(t)\}$$

Aggregate conditions on groups

Sometimes we wish to perform a selection of certain groups, based on an aggregate value of that group.

SQL supports a convenient way of doing this (with no direct equivalent in relational algebra):

```
SELECT <attributes and aggregates in the result>
FROM R
GROUP BY <grouping attributes>
HAVING <condition that may involve aggregates>
```

The `HAVING` clause may contain conditions like `MIN(year) < 1930`, where `MIN(year)` is the minimum value of the `year` attribute within the group.

Note: This would make no sense in a `WHERE` clause. (Why?)

Next: Algebraic laws.

Laws in relational algebra

An algebraic law is an equation (or other mathematical statement) which is always true in a particular algebra.

Using such laws we could, e.g., conclude that the following two relational algebra expressions are equivalent:

$$((\sigma_{C_1}(R_1)) \cup R_2) \bowtie (\sigma_{C_2}(R_1))$$

$$(\sigma_{C_1 \texttt{ AND } C_2}(R_1)) \cup (\sigma_{C_2}(R_1 \bowtie R_2))$$

The laws of relational algebra allow us to:

- Reason about relational expressions (and thus SQL expressions).
- Perform query optimization (ADBT, spring 2006).

Basic laws in relational algebra

Commutativity laws, examples

- $R \bowtie S = S \bowtie R$
- $R \cup S = S \cup R$
- $R \cap S = S \cap R$

Associativity laws, examples

- $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
- $(R \cup S) \cup T = R \cup (S \cup T)$
- $(R \cap S) \cap T = R \cap (S \cap T)$

Problem session (5 minutes)

Argue that the equation

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

is indeed a valid algebraic law.

In other words: Argue that the equality holds for any relations R, S, and T.

Relational algebraic law school, continued

Distributive laws, examples

- $(R \cap S) \cup T = (R \cup T) \cap (S \cup T)$
- $(R \cup S) \bowtie T = (R \bowtie T) \cup (S \bowtie T)$
- $\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$
- $\pi_L(R \bowtie S) = \pi_L(\pi_{L \cup J}(R) \bowtie \pi_{L \cup J}(S))$, where J is the set of common attributes of R and S.

Example of use: By the second law, the expression

$$((\sigma_{C_1}(R_1)) \cup R_2) \bowtie (\sigma_{C_2}(R_1))$$

is equivalent to $((\sigma_{C_1}(R_1)) \bowtie (\sigma_{C_2}(R_1))) \cup (R_2 \bowtie (\sigma_{C_2}(R_1)))$.

An algebraic criterion for decomposition

We can decompose a relation R into $R_1(A_1, \ldots, A_n, B_1, \ldots, B_m)$ and $R_2(A_1, \ldots, A_n, C_1, \ldots, C_k)$ exactly when we have the equality:

$$R = (\pi_{A_1,\ldots,A_n,B_1,\ldots,B_m}(R)) \bowtie (\pi_{A_1,\ldots,A_n,C_1,\ldots,C_k}(R))$$

This is the formal way of stating that the relation instances of R_1 and R_2, derived from R by projection, should always join to form R.

Most important points in this lecture

As a minimum, you should after this week:

- Know the meaning of the most common relational algebra operators: $\cup$, $\cap$, $-$, π, σ, $\times$, $\bowtie$, γ.
- Be able to translate simple SQL queries to relational algebra on bags, and vice versa.
- Recognize relational algebra laws.

Next lecture

Next time we will cover a mix of topics:

- *Constraints*, which are assertions that we want to be true at all times in the database.
- *Triggers*, which are database modifications that can be activated by other database modifications.
- Access privileges in SQL.
- . . . and if time allows, XML as a data interchange format.