Queries and Query Languages

- A *query* is a question which is posed to a database.
- Formally, queries are considered to be *passive*. That is, they do not modify the state of the database.
- Often, *active* queries, or *updates*, are also included in the general context of database queries.
- In any case, most practical query languages also provide facilities for specifying updates.

Query Languages to be Covered in the Course

- Ideal Query Languages
  - The Relational Algebra
    - The relational algebra is based upon a set of operations on relations. Each operation takes one or more relations as its arguments, and produces a new relation as the result. Since the operations may be composed with one another, they form an *algebra.*
    - A query is then a functional composition of operations which yields the (derived) relations which comprise the answer to the query.
    - From a logic point of view, the relational algebra is a set of operations on the models of the theory.

- The Relational Calculus

  - In the relational calculus, a query is specified as a logical formula in a first-order logic based upon the database schema. The set of tuples for which the formula is true constitutes the answer to the query.

- Equivalence result: Under very reasonable assumptions of which operators and formulas are allowed, the relational algebra and relational calculus may be shown to be equivalent. Thus, it is a matter of ease of use, rather than power, which suggests that one should be chosen over the other.

- A Practical Query Language

  - SQL (Structured Query Language)

    - SQL has become an industry standard.

    - It incorporates aspects of both the relational algebra and the relational calculus.

The Relational Algebra

We start with the most basic class: SPJ-queries

- SPJ queries are constructed from three basic operations,
    - *S*elect: horizontal restriction
    - *P*roject: vertical restriction
    - *J*oin: vertical reconstruction

The Select Operation:

- The basic syntax is

$$\sigma_{<cond>}(<rel\_arg>)$$

Here:

- σ is the symbol used to denote a select operation.
- <cond> is the condition of the selection.
- <rel_arg> is the relation to which the selection is applied.

Example: Call the following instance r.

R

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2
a_1	b_3	c_3
a_2	b_4	c_4

Then the answer to the query

$$\sigma_{(A=a_1)}(r)$$

is

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2
a_1	b_3	c_3

- The same notation may be used to represent the query on a schema. For example,

$$\sigma_{(A=a_1)}(R)$$

  may be used to represent the query to be applied to schema R.

- The most basic operation just allows equality selection on domain values, but more general operations may include arithmetic comparison.

Example: Call the following instance s.

S		
A	B	C
1	b_1	c_1
3	b_2	c_2
3	b_3	c_3
4	b_4	c_4

The solution to the query $\sigma_{(A \geq 3)}(s)$ is

A	B	C
3	b_2	c_2
3	b_3	c_3
4	b_4	c_4

- Disjunctive formulas are also possible. Construct the answer to the following query.

$$\sigma_{((A=1)\vee(A=4))}(S)$$

- When working with selections, it is critical to specify exactly the allowable selection conditions.

The Project Operation:

- Just as the select operation is a “horizontal” restriction which retains selected rows of a relation, so too is the projection operation a “vertical” restriction, which retains selected columns.

The general syntax is $\pi_{<cols>}$(<rel_arg>).

Example: Call the following instance r.

R

A	B	C
a_1	b_1	c_1
a_1	b_2	c_1
a_2	b_3	c_3
a_2	b_4	c_3

The solution to the query $\pi_{\{A,C\}}(r)$, or just $\pi_{AC}(r)$, is

A	C
a_1	c_1
a_2	c_3

Notice that duplicate cells are omitted.

The associated schema query is $\pi_{AC}(R)$.

Join:

The join operation is much more complex than the previous two. We begin with the simplest, yet most widely used version.

Natural Join:

- It is easiest to explain with an example. Let r and s be the instances associated with the following database.

R

A	B
a_1	b_1
a_2	b_1
a_2	b_3
a_3	b_3

S

B	C
b_1	c_1
b_1	c_2
b_3	c_3
b_4	c_3

Then the natural join of r and s, denoted $r \bowtie s$, is

T

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_1	c_1
a_2	b_1	c_2
a_2	b_3	c_3
a_3	b_3	c_3

The idea is to match the tables on their common attributes.

- A join is said to be *lossless* if the original relations may be recovered from the appropriate projections on the join. In this example, this means that

$$\pi_{AB}(r \bowtie s) = r \quad \text{and} \quad \pi_{BC}(r \bowtie s) = s.$$

- The join of this example is not lossless.
- The natural join of the following two instances is lossless.

R

A	B
a_1	b_1
a_2	b_1
a_2	b_3
a_3	b_3

S

B	C
b_1	c_1
b_1	c_2
b_3	c_3

- In general, for a join to be lossless, the projections on the matching columns must match.

Other Forms of Join:

- *General equality join:* Even if attribute names do not match, a join may be constructed by specifying which columns to match. In this case, the duplicate columns are often retained.
- Example: $R \bowtie_{\{B,D\}} S$

R

A	B
a_1	b_1
a_2	b_1
a_2	b_3
a_3	b_3

S

D	C
b_1	c_1
b_1	c_2
b_3	c_3

Result:

T

A	B	C	D
a_1	b_1	c_1	b_1
a_1	b_1	c_2	b_1
a_2	b_1	c_1	b_1
a_2	b_1	c_2	b_1
a_2	b_3	c_3	b_3
a_3	b_3	c_3	b_3

- *Cartesian product:* In this context, the join on an empty set of columns (*i.e.,* R $\bowtie_{\emptyset}$S) yields the Cartesian product of the two relations.

- *Theta joins:* More general matching operators may be applied in situations in which they make sense. These are called theta joins.

- Example: R $\bowtie_{\{B \geq D\}}$S (Here the values for attributes B and D must be amenable to an ordering relation. The operator ≥ is called the *theta operator*.

Join Conventions and Notation

The textbook introduces some relatively nonstandard notation for joins. In these slides, the more common notation, which is used in this course, is presented. This notation should be used on obligatory exercises, and on the examination.

Conventions and notation for the natural join:

- The “bowtie” symbol $\bowtie$ without any subscript always denotes the natural join. This means that:

  - the two relations are joined on their common attributes, and

  - the common attributes are not repeated. Thus, the R[ABCD] $\bowtie$ S[BDE] is a relation on attributes ABCDE.

- Note that the natural join is a Cartesian product iff the attribute sets of the two relations are disjoint. For example R[ABCD] $\bowtie$ S[EFG] is a Cartesian product, with attributes ABCDEFG.

Conventions and notation for other types of join:

- With subscripts, the bowtie symbol may only be applied to relations which do not have any attribute names in common. (Rename if necessary.) In this case, all attributes of both relations are present in the resulting relation.

- Equality joins are represented as in the following example:

  $$R[ABC] \bowtie_{\{B=D\}} S[DEF]$$

- A Cartesian product is indicated by a “$\emptyset$” as subscript:

  $$R[ABC] \bowtie_{\emptyset} S[DEF]$$

- Other joins are indicated by the appropriate operations in the subscript:

  $$R[ABC] \bowtie_{\{B \leq D\}} S[DEF]$$

Other Relational Algebra Operators:

- In addition to the “SPJ” trio, it is often the case that set-based operations are allowed in the relational algebra. The most common operations are the following:
  - Union ($\cup$)
  - Set difference ($\setminus$)
  - Intersection ($\cap$), which can in fact be constructed from union and difference.

Note that, for queries to make sense, these set operations may only be performed on relations over identical attribute sets, or at least over relations with a bijective correspondence between the domains of their attribute sets.

Format of queries:

- Officially, the correct format is to write queries in functional composition format.
- In a less elegant representation intermediate variables are introduced and queries are written in an imperative program style.
- Examples will be given later.

The Relational Calculus

- The relational calculus uses logical formulas to specify queries.
- This is natural, since the relational model is closely based upon first-order predicate logic.
- There are two versions of the relational calculus, domain calculus and tuple calculus.
- In each case, the formulas must be controlled (limited to *safe* formulas) to avoid meaningless queries.

The Domain Calculus

- The domain calculus shows clearly the connection between the relational model and first-order predicate logic.
- Each relation symbol becomes a relation name in a first-order logic.
- For each attribute A, there is a set of variables whose values may range over the domain assigned to that attribute. Such variables will be represented with a superscript matching the attribute name. (Example: x^A, y^A, for domain $\mathcal{A}$.)
- Specific domain values may be represented with constant symbols.
- There are no non-nullary function symbols.

Example: Consider the problem of computing the *AC*-projection of the join of the following two relational instances (*i.e.,* $\pi_{AC}(R \bowtie S)$):

R	
A	*B*
a_1	b_1
a_2	b_1
a_2	b_3
a_3	b_3

S	
B	*C*
b_1	c_1
b_1	c_2
b_3	c_3
b_4	c_3

To make things work, we need to assign an order to the attributes. We use the ordering indicated in the picture. The domain calculus query to compute the projection of the natural join is as follows.

$$\{(x^A, x^C) \mid (\exists x^B)(R(x^A, x^B) \wedge S(x^B, x^C))\}$$

Equality and even arithmetic comparisons may also be used in a limited fashion within the logic. Here is an example of a selection query revisited:

S

A	B	C
1	b_1	c_1
3	b_2	c_2
3	b_3	c_3
4	b_4	c_4

The solution to the query $\sigma_{(A \geq 3)}(S)$ is

A	B	C
3	b_2	c_2
3	b_3	c_3
4	b_4	c_4

And this query is represented in the domain calculus by

$$\{(x^A, x^B, x^C) \mid S(x^A, x^B, x^C) \wedge (x^A \geq 3)\}$$

- These queries are *safe,* in the sense that the answer only involves values extracted from the relations in the database. In formulating a precise notion of safe queries, we must be careful to ensure this condition. An example of an unsafe query is:

$$\{ x^A \mid \neg(\exists x^B)(R(x^A, x^B))\}$$

- The formalization of safety is rather technical, and will not be presented here. For a query designed by hand, it is usually obvious whether or not it is safe. Of course, a query processor must be able to detect safety, or lack thereof.

Another example: Here is an informal example of an unsafe query. Consider the database of all people living in Sweden, and suppose that the Last_Name field may contain any string of up to 64 characters. Then the following is an unsafe query which is easily represented in the domain calculus.

“Give me the set of strings of length at most 64 which do not represent the last name of someone living in Sweden.”

The Tuple Calculus

- The tuple calculus differs from the domain calculus in that the variables range over entire tuples from relations, rather than over single domain values.
- The idea is best illustrated by example: Suppose that we have the following simple relational database schema.

R	
A	*B*

S	
B	*C*

The composition $\pi_{AC}(R \bowtie S)$ is reconstructed via the following formula in the tuple calculus:

$$\{(x.A, y.C) \mid R(x) \wedge S(y) \wedge (x.B = y.B)\}$$

- Here x and y are *tuple variables,* and their values range over (binary in this case) tuples for the associated relations.
- x.*A*, for example, denotes the *A*-indexed component of tuple x.
- Note that x.*A* and y.*C* are *free variables;* an existential quantifier is not used.
- As in the case of the domain calculus, a notion of safe query may be formulated.

Division – a Special Operator in the Relational Algebra

Suppose that we have the following simple relational database schema.

Employee	
Emp_Num	Name

Project	
Emp_Num	Proj_Num

Consider the query “Provide the names of those employees who work on every project.”

This is easy in the tuple calculus.

$$
\begin{aligned}
&\{x.\mathrm{Name} \mid \mathrm{Employee}(x) \wedge \\
&\quad (\forall y)(\exists z)(\mathrm{Project}(y) \Rightarrow (\mathrm{Project}(z) \wedge \\
&\qquad (y.\mathrm{Proj\_Num} = z.\mathrm{Proj\_Num}) \wedge \\
&\qquad (x.\mathrm{Emp\_Num} = z.\mathrm{Emp\_Num})))\}
\end{aligned}
$$

At first glance it seems impossible within the relational algebra. However, there is an operation, called division, which makes it possible.

Definition: Let R[$\mathcal{A}$] and S[$\mathcal{B}$] be relation schemata, and assume that $\mathcal{B} \subseteq \mathcal{A}$. Then R ÷ S, called R *divided by* S, is the relation schema on attribute set $\mathcal{A} \setminus \mathcal{B}$ defined by the following formula.

$$R \div S \;=\; \pi_{\mathcal{A}\setminus\mathcal{B}}(R) \setminus \pi_{\mathcal{A}\setminus\mathcal{B}}((\pi_{\mathcal{A}\setminus\mathcal{B}}(R) \bowtie S) \setminus R)$$

This quantity is also called the *quotient* of R by S.

Despite its apparently indecipherable definition, it has a relatively straightforward characterization. Let r be an instance of R[$\mathcal{A}$], and let s be an instance of S[$\mathcal{B}$]. Then

$$r \div s \;=\; \{\, t[\mathcal{A}\setminus\mathcal{B}] \mid (\forall\, t_S \in s)(\exists\, t_R \in r)((t_R[\mathcal{B}] = t_S) \wedge (t_R[\mathcal{A}\setminus\mathcal{B}] = t))\}$$

Here is the solution to the query:

$$\pi_{\{\mathrm{Name}\}}((\text{Employee} \bowtie \text{Project}) \div \pi_{\mathrm{Proj\_Num}}(\text{Project}))$$

- Such “double negation” constructs occur frequently when writing queries with universal quantification in SQL, as will be seen later.

C'est simple comme bonjour, n’est-ce pas?

Renaming – Another Useful Operator in the Relational Algebra

Suppose that we again have the following simple relational database schema.

Employee	
Emp_Num	Name

Project	
Emp_Num	Proj_Num

Consider the query “Provide the names of those employees who work on exactly one project.”

This is easy in the tuple calculus.

$$
\begin{aligned}
\{&x.\text{Name} \mid \text{Employee}(x) \wedge \\
&(\exists y)(\text{Project}(y) \wedge (x.\text{Emp\_Num} = y.\text{Emp\_Num})) \wedge \\
&(\forall z)(\forall w)((\text{Project}(z) \wedge \text{Project}(w) \wedge \\
&\qquad\qquad (x.\text{Emp\_Num} = z.\text{Emp\_Num}) \wedge \\
&\qquad\qquad (x.\text{Emp\_Num} = w.\text{Emp\_Num})) \Rightarrow \\
&\qquad\qquad\quad (z.\text{Proj\_Num} = w.\text{Proj\_Num}))\}
\end{aligned}
$$

Again, at first glance it seems impossible with the relational algebra. However, it is easily realized with the aid of *renaming.* In renaming, a copy of a relation schema is made, with the name of one or more attributes changed.

Copy-with-Rename(
        Project, Project',{(Proj_Num, Proj_Num')})

creates the following relation schema:

Project'	
Emp_Num	Proj_Num'

To solve the query, let us proceed step-by-step.

First, join Project with Project':

$R_1 \leftarrow$ Project $\bowtie$ Project'.

R_1		
Emp_Num	Proj_Num	Proj_Num'

Now, restrict this relation to those tuples with *different* values for Proj_Num and Proj_Num':

$R_2 \leftarrow \sigma_{(\text{Proj\_Num} \neq \text{Proj\_Num'})}(R_1)$

The following formula identifies the employees who work on more than one project.

$R_3 \leftarrow \pi_{\text{Emp\_Num}}(R_2)$

To identify the employees who work on no projects, use

$R_4 \leftarrow \pi_{\text{Emp\_Num}}(\text{Employees}) \setminus \pi_{\text{Emp\_Num}}(\text{Project})$.

Then the difference below identifies the employees who work on exactly one project.

$R_5 \leftarrow \pi_{\text{Emp\_Num}}(\text{Employees}) \setminus (R_3 \cup R_4)$.

The query asks for names, however. Join the above result with the Employee relation:

$R_6 \leftarrow R_5 \bowtie \text{Employee}$.

Finally, project the names:

$R_7 \leftarrow \pi_{\text{Name}}(R_6)$.

As a single expression in the relational algebra, this becomes:

$$
\begin{aligned}
&\pi_{\text{Name}}(\\
&\quad (\pi_{\text{Emp\_Num}}(\text{Employee}) \setminus \\
&\quad\ \pi_{\text{Emp\_Num}}(\\
&\qquad \sigma_{(\text{Proj\_Num} \neq \text{Proj\_Num}')}(\text{Project} \bowtie \text{Project}'))\\
&\quad\ \cup\\
&\quad\ (\pi_{\text{Emp\_Num}}(\text{Employee}) \setminus \pi_{\text{Emp\_Num}}(\text{Project}))\\
&\quad\ \bowtie \text{Employee})
\end{aligned}
$$

Exercise: Build a query in the relational algebra which gives the names of those employees who work on exactly two projects.