Chapter 6: Formal Relational Query Languages

Database System Concepts, 6th Ed.

Relational Algebra

- Procedural language
- Six basic operators
  - select: σ
  - project: Π
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ
- The operators take one or two relations as inputs and produce a new relation as a result.

Select Operation

- Notation: $\sigma_p(r)$
- p is called the **selection predicate**
- Defined as:

$$\sigma_p(\mathbf{r}) = \{t \mid t \in r \textbf{ and } p(t)\}$$

Where p is a formula in propositional calculus consisting of **terms** connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
Each **term** is one of:

<attribute> *op* <attribute> or <constant>

where *op* is one of: $=, \neq, >, \geq. <. \leq$

- Example of selection:

$$\sigma_{dept\_name=\text{“Physics”}}(instructor)$$

Project Operation

- Notation: $$\Pi_{A_1, A_2, \ldots, A_k}(r)$$

  where A_1, A_2 are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets
- Example: To eliminate the *dept_name* attribute of *instructor*

$$\Pi_{ID,\ name,\ salary}\ (instructor)$$

Union Operation

- Notation: $r \cup s$
- Defined as:

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

- For $r \cup s$ to be valid.
  1. *r, s* must have the *same* **arity** (same number of attributes)
  2. The attribute domains must be **compatible** (example: 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)
- Example: to find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both

$$\Pi_{course\_id}\,(\sigma_{\,semester=\text{"Fall"}\ \land\ year=2009}\,(section))\ \cup$$
$$\Pi_{course\_id}\,(\sigma_{\,semester=\text{"Spring"}\ \land\ year=2010}\,(section))$$

Set Difference Operation

- Notation $r - s$
- Defined as:

  $$r - s = \{t \mid t \in r \textbf{ and } t \notin s\}$$

- Set differences must be taken between **compatible** relations.
  - *r* and *s* must have the same arity
  - attribute domains of *r* and *s* must be compatible
- Example: to find all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

$$\Pi_{course\_id}\,(\sigma_{semester=\text{“Fall”}\ \land\ year=2009}\,(section))\ -$$
$$\Pi_{course\_id}\,(\sigma_{semester=\text{“Spring”}\ \land\ year=2010}\,(section))$$

Set-Intersection Operation

- Notation: $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r \textbf{ and } t \in s \}$
- Assume:
  - r, s have the *same arity*
  - attributes of r and s are compatible
- Note: $r \cap s = r - (r - s)$

Cartesian-Product Operation

- Notation *r* x *s*
- Defined as:

$$r \times s = \{t\, q \mid t \in r \textbf{ and } q \in s\}$$

- Assume that attributes of r(R) and s(S) are disjoint. (That is, $R \cap S = \emptyset$).
- If attributes of *r(R)* and *s(S*) are not disjoint, then renaming must be used.

Rename Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.
- Example:

$$\rho_X(E)$$

  returns the expression E under the name X
- If a relational-algebra expression E has arity n, then

$$\rho_{x(A_1, A_2, \ldots, A_n)}(E)$$

  returns the result of expression E under the name X, and with the attributes renamed to $A_1, A_2, \ldots, A_n$.

Formal Definition

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation
- Let E_1 and E_2 be relational-algebra expressions; the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_p (E_1)$, P is a predicate on attributes in E_1
  - $\prod_S(E_1)$, S is a list consisting of some of the attributes in E_1
  - $\rho_{x} (E_1)$, x is the new name for the result of E_1

Assignment Operation

- The assignment operation ($\leftarrow$) provides a convenient way to express complex queries.
  - Write query as a sequential program consisting of
    - a series of assignments
    - followed by an expression whose value is displayed as a result of the query.
  - Assignment must always be made to a temporary relation variable.
- Example: Write $r \bowtie s$ as

$$temp1 \leftarrow r \times s$$

$$temp2 \leftarrow \sigma_{r.A1=s.A1 \wedge r.A2=s.A2 \ldots \wedge r.An=s.An}(temp1)$$

$$result = \Pi_{R \cup S}\ (temp2)$$

  - The result to the right of the $\leftarrow$ is assigned to the relation variable on the left of the $\leftarrow$.
  - May use variable in subsequent expressions.

Extended Relational-Algebra-Operations

- Generalized Projection
- Outer Join
- Aggregate Functions

Generalized Projection

- Extends the projection operation by allowing arithmetic functions to be used in the projection list.

$$\Pi_{F1, F2, \ldots, Fn}(E)$$

- *E* is any relational-algebra expression
- Each of F_1, F_2, …, F_n are are arithmetic expressions involving constants and attributes in the schema of *E*.
- Given relation *credit-info(customer-name, limit, credit-balance),* find how much more each person can spend:

$$\Pi_{\textit{customer-name, limit – credit-balance}}\ \textit{(credit-info)}$$

Aggregate Functions and Operations

- **Aggregation function** takes a collection of values and returns a single value as a result.

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- **Aggregate operation** in relational algebra

$$_{G1,\, G2,\, \ldots,\, Gn}\,\mathcal{G}_{\,F1(A1),\, F2(A2),\ldots,\, Fn(An)}\,(E)$$

  - *E* is any relational-algebra expression
  - $G_1, G_2 \ldots, G_n$ is a list of attributes on which to group (can be empty)
  - Each F_i is an aggregate function
  - Each A_i is an attribute name

Aggregate Operation – Example

- Relation *account* grouped by *branch-name*:

branch-name	account-number	balance
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$$_{\text{branch-name}}\, g\, _{\text{sum(balance)}}\ (\text{account})$$

branch-name	balance
Perryridge	1300
Brighton	1500
Redwood	700

Aggregate Functions (Cont.)

- Result of aggregation does not have a name
  - Can use rename operation to give it a name
  - For convenience, we permit renaming as part of aggregate operation

$$\text{branch-name}\ \mathcal{g}\ \textbf{sum}(\text{balance})\ \textbf{as}\ \text{sum-balance}\ (\text{account})$$

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that do not match tuples in the other relation to the result of the join.
- Uses *null* values:
  - *null* signifies that the value is unknown or does not exist
  - All comparisons involving *null* are (roughly speaking) **false** by definition.
    - Will study precise meaning of comparisons with nulls later

Outer Join – Example

- Relation *loan*

loan-number	branch-name	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

- Relation borrower

customer-name	loan-number
Jones	L-170
Smith	L-230
Hayes	L-155

Outer Join – Example

- **Inner Join**

loan $\bowtie$ *Borrower*

loan-number	branch-name	amount	customer-name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

- **Left Outer Join**

loan ⟕ Borrower

loan-number	branch-name	amount	customer-name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	null

Outer Join – Example

- **Right Outer Join**

  loan ⟖ *borrower*

loan-number	branch-name	amount	customer-name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	null	null	Hayes

- **Full Outer Join**

  loan ⟗ borrower

loan-number	branch-name	amount	customer-name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	null
L-155	null	null	Hayes

Tuple Relational Calculus

Tuple Relational Calculus

- A nonprocedural query language, where each query is of the form

$$\{t \mid P(t)\}$$

- It is the set of all tuples *t* such that predicate *P* is true for *t*
- *t* is a *tuple variable*, $t[A]$ denotes the value of tuple *t* on attribute *A*
- $t \in r$ denotes that tuple *t* is in relation *r*
- *P* is a *formula* similar to that of the predicate calculus

Tuple Relational Calculus Formula

1. Set of attributes and constants
2. Set of comparison operators: (e.g., $<$, $\leq$, $=$, $\neq$, $>$, $\geq$)
3. Set of connectives: and ($\wedge$), or (v), not ($\neg$)
4. Implication ($\Rightarrow$): x $\Rightarrow$ y, if x if true, then y is true

$$x \Rightarrow y \equiv \neg x \vee y$$

5. Set of quantifiers:
   - $\exists\, t \in r\, (Q\,(t\,))$ ≡ ”there exists” a tuple in *t* in relation *r* such that predicate *Q* (*t*) is true
   - $\forall t \in r\, (Q\,(t\,))$ ≡ *Q* is true “for all” tuples *t* in relation *r*

Example Queries

- Find the *ID, name, dept_name, salary* for instructors whose salary is greater than $80,000

$$\{t \mid t \in instructor \wedge t\,[salary\,] > 80000\}$$

Notice that a relation on schema (*ID, name, dept_name, salary*) is implicitly defined by the query

- As in the previous query, but output only the *ID* attribute value

$$\{t \mid \exists\, s \in \text{instructor}\ (t\,[ID\,] = s\,[ID\,] \wedge s\,[salary\,] > 80000)\}$$

Notice that a relation on schema (*ID*) is implicitly defined by the query

Example Queries

- Find the names of all instructors whose department is in the Watson building

$$\{t \mid \exists s \in instructor\ (t[name] = s[name] \wedge \exists u \in department\ (u[dept\_name] = s[dept\_name] \text{ “} \wedge u[building] = \text{“Watson”}))\}$$

- Find the set of all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or both

$$\{t \mid \exists s \in section\ (t[course\_id] = s[course\_id] \wedge s[semester] = \text{“Fall”} \wedge s[\text{year}] = 2009) \vee \exists u \in section\ (t[course\_id] = u[course\_id] \wedge u[semester] = \text{“Spring”} \wedge u[\text{year}] = 2010)\}$$

Example Queries

- Find the set of all courses taught in the Fall 2009 semester, and in the Spring 2010 semester

$$\{t \mid \exists s \in section\ (t[course\_id] = s[course\_id] \land s[semester] = \text{“Fall”} \land s[\text{year}] = 2009) \land \exists u \in section\ (t[course\_id] = u[course\_id] \land u[semester] = \text{“Spring”} \land u[\text{year}] = 2010)\}$$

- Find the set of all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

$$\{t \mid \exists s \in section\ (t[course\_id] = s[course\_id] \land s[semester] = \text{“Fall”} \land s[\text{year}] = 2009) \land \neg \exists u \in section\ (t[course\_id] = u[course\_id] \land u[semester] = \text{“Spring”} \land u[\text{year}] = 2010)\}$$

Universal Quantification

- Find all students who have taken all courses offered in the Biology department
  - $\{t \mid \exists\, r \in student\ (t[ID] = r[ID]) \land$
    $(\forall\, u \in course\ (u[dept\_name] = \text{“Biology”} \Rightarrow$
    $\exists\, s \in takes\ (t[ID] = s[ID] \land$
    $s[course\_id] = u[course\_id]))\}$

Domain Relational Calculus

Domain Relational Calculus

- A nonprocedural query language equivalent in power to the tuple relational calculus
- Each query is an expression of the form:

$$\{ < x_1, x_2, \ldots, x_n > \mid P (x_1, x_2, \ldots, x_n)\}$$

  - $x_1, x_2, \ldots, x_n$ represent domain variables
  - P represents a formula similar to that of the predicate calculus

Example Queries

- Find the *ID, name, dept_name, salary* for instructors whose salary is greater than \$80,000
  - $\{< i, n, d, s> \mid \ < i, n, d, s> \in instructor \wedge s > 80000\}$
- As in the previous query, but output only the *ID* attribute value
  - $\{< i> \ \mid < i, n, d, s> \in instructor \wedge s > 80000\}$
- Find the names of all instructors whose department is in the Watson building

$$\{< n > \mid \ \exists\, i, d, s\, (< i, n, d, s > \in instructor \\ \wedge \exists\, \text{b, a}\, (< d, b, a> \in department \ \wedge \ b = \text{“Watson”}\,))\}$$

Example Queries

- Find the set of all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or both

$$
\begin{aligned}
\{<c> \mid\ \exists\, a, s, y, b, r, t\ (<c, a, s, y, b, r, t> \in section\ \wedge \\
s = \text{“Fall”} \wedge y = 2009\ ) \\
\vee\ \exists\, a, s, y, b, r, t\ (<c, a, s, y, b, r, t> \in section\ ] \wedge \\
s = \text{“Spring”} \wedge y = 2010)\}
\end{aligned}
$$

This case can also be written as

$$
\begin{aligned}
\{<c> \mid\ \exists\, a, s, y, b, r, t\ (<c, a, s, y, b, r, t> \in section\ \wedge \\
(\ (s = \text{“Fall”} \wedge y = 2009\ )\ \vee (s = \text{“Spring”} \wedge y = 2010))\}
\end{aligned}
$$

- Find the set of all courses taught in the Fall 2009 semester, and in the Spring 2010 semester

$$
\begin{aligned}
\{<c> \mid\ \exists\, a, s, y, b, r, t\ (<c, a, s, y, b, r, t> \in section\ \wedge \\
s = \text{“Fall”} \wedge y = 2009\ ) \\
\wedge\ \exists\, a, s, y, b, r, t\ (<c, a, s, y, b, r, t> \in section\ ] \wedge \\
s = \text{“Spring”} \wedge y = 2010)\}
\end{aligned}
$$

Universal Quantification

- Find all students who have taken all courses offered in the Biology department
  - $\{< i > \mid \exists\, n, d, tc\ (< i, n, d, tc > \in student\ ) \land$
    $\forall\, ci, ti, dn, cr\ (< ci, ti, dn, cr > \in course \land dn = \text{“Biology”}$
    $\Rightarrow\ \exists\, si, se, y, g\ (<i, ci, si, se, y, g> \in takes\ ))\}$
  - Note that without the existential quantification on student, the above query would be unsafe if the Biology department has not offered any courses.

Important Instructions

- Read Chapter 6 from the book except for the following sections/sub-sections/topics:
  - 6.2.3, 6.2.4
  - 6.3.3, 6.3.4

End of Chapter 6