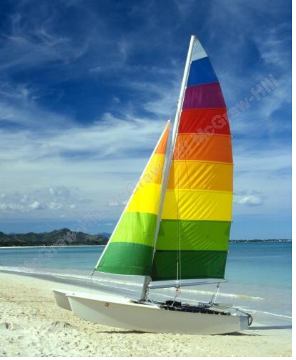


Chapter 3: Formal Relational Query Languages

Database System Concepts, 6^{th} Ed.

Chapter 3: Formal Relational Query Languages

- Relational Algebra - Extensions
- Tuple Relational Calculus
- Domain Relational Calculus

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.

Formal Definition

- A basic expression in 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 also 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

Additional Operations

We define additional operations that do not add any power to the relational algebra, but they simplify common queries.

- Set intersection
- Natural join
- Assignment
- Outer join

Set-Intersection Operation

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

Set-Intersection Operation – Example

- Relation *r, s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cap s$

A	B
α	2

Natural-Join Operation

- Notation: $r \bowtie s$
- Let r and s be relations on schemas R and S respectively. Then, $r \bowtie s$ is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from r and t_s from s.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple t to the result, where
    - t has the same value as t_r on r
    - t has the same value as t_s on s
- Example:

  $R = (A, B, C, D)$

  $S = (E, B, D)$
  - Result schema = (A, B, C, D, E)
  - $r \bowtie s$ is defined as:

$$\Pi_{r.A,\ r.B,\ r.C,\ r.D,\ s.E}\ (\sigma_{r.B = s.B\ \wedge\ r.D = s.D}\ (r \times s))$$

Natural Join Example

- Relations r, s:

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ε

s

- r ⋈ s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Natural Join and Theta Join

- Find the names of all instructors in the Comp. Sci. department together with the course titles of all the courses that the instructors teach
  - $\Pi_{name,\ title}\ (\sigma_{dept\_name = \text{"Comp. Sci."}}\ (instructor \bowtie teaches \bowtie course))$
- Natural join is associative
  - $(instructor \bowtie teaches) \bowtie course$ is equivalent to $instructor \bowtie (teaches \bowtie course)$
- Natural join is commutative
  - $instruct \bowtie teaches$ is equivalent to $teaches \bowtie instructor$
- The **theta join** operation $r \bowtie_\theta s$ is defined as
  - $r \bowtie_\theta s = \sigma_\theta (r \times s)$

Assignment Operation

- The assignment operation (←) 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.

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples from one relation that does 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.
    - We shall study the precise meaning of comparisons with nulls later

Outer Join – Example

- Relation *instructor1*

ID	*name*	*dept_name*
10101	Srinivasan	Comp. Sci.
12121	Wu	Finance
15151	Mozart	Music

- Relation *teaches1*

ID	*course_id*
10101	CS-101
12121	FIN-201
76766	BIO-101

Outer Join – Example

- Join

instructor $\bowtie$ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201

- Left Outer Join

instructor ⟕ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*

Outer Join – Example

- Right Outer Join

  instructor ⟖ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
76766	null	null	BIO-101

- Full Outer Join

  instructor ⟗ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*
76766	null	null	BIO-101

Outer Join using Joins

- Outer join can be expressed using basic operations
  - e.g. $r ⟕ s$ can be written as

    $(r \bowtie s) \cup (r - \prod_R(r \bowtie s)) \times \{(null, \ldots, null)\}$

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null.*
- Aggregate functions simply ignore null values
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same

Null Values

- Comparisons with null values return the special truth value: *unknown*
  - If *false* was used instead of *unknown*, then *not (A < 5)* would not be equivalent to *A >= 5*
- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*,
(*unknown* **or** *false*) = *unknown*
(*unknown* **or** *unknown*) = *unknown*
  - AND: (*true* **and** *unknown*) = *unknown,*
(*false* **and** *unknown*) = *false,*
(*unknown* **and** *unknown*) = *unknown*
  - NOT*:* (**not** *unknown*) = *unknown*
  - In SQL "*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*
- Result of a select predicate is treated as *false* if it evaluates to *unknown*

Division Operator

- Given relations r(R) and s(S), such that $S \subset R$, $r \div s$ is the largest relation t(R-S) such that

  $$t \times s \subseteq r$$

- E.g. let $r(ID, course\_id) = \Pi_{ID,\ course\_id}\ (takes)$ and

  $$s(\text{course\_id}) = \Pi_{course\_id}\ (\sigma_{\text{dept\_name=“Biology”}}(course))$$

  then $r \div s$ gives us students who have taken all courses in the Biology department
- Can write $r \div s$ as

  $$temp1 \leftarrow \Pi_{R-S}\ (r)$$

  $$temp2 \leftarrow \Pi_{R-S}\ ((temp1 \times s) - \Pi_{R-S,S}\ (r))$$

  $$result = temp1 - temp2$$

  - The result to the right of the $\leftarrow$ is assigned to the relation variable on the left of the $\leftarrow$.
  - If $u = r \times s$ than $u \div r = s$ division can be seen as invers of cart. prod.

Extended Relational-Algebra-Operations

- Generalized Projection
- Aggregate Functions

Generalized Projection

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

$$\prod_{F_1, F_2, \ldots, F_n}(E)$$

- *E* is any relational-algebra expression
- Each of F_1, F_2, …, F_n is an arithmetic expression involving constants and attributes in the schema of *E*.
- Given relation *instructor(ID, name, dept_name,* salary) where salary is annual salary, get the same information but with monthly salary

$$\prod_{ID,\ name,\ dept\_name,\ salary/12}(instructor)$$

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

$$_{G_1, G_2, \ldots, G_n}\, \mathcal{G}_{\, F_1(A_1), F_2(A_2), \ldots, F_n(A_n)}(E)$$

  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

- Note: Some books/articles use γ instead of $\mathcal{G}$ (Calligraphic G)

Aggregate Operation – Example

- Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

- $\mathcal{G}_{\mathbf{sum(c)}}\,(\mathrm{r})$

sum(*c*)
27

Aggregate Operation – Example

- Find the average salary in each department

$$_{dept\_name}\mathcal{G}_{\textbf{avg}(salary)}\,(instructor)$$

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

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 the aggregate operation

$$_{dept\_name}\,\mathcal{G}_{\,\mathbf{avg}(salary)\ \mathbf{as}\ avg\_sal}\,(instructor)$$

Modification of the Database

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating
- All these operations can be expressed using the assignment operator

Deletion

- A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.
- Can delete only whole tuples; cannot delete values on only particular attributes
- A deletion is expressed in relational algebra by:

$$r \leftarrow r - E$$

where r is a relation and E is a relational algebra query.

Deletion Examples

- Delete all account records in the Perryridge branch.

$$account \leftarrow account - \sigma_{branch\_name = \text{“Perryridge”}}(account)$$

- Delete all loan records with amount in the range of 0 to 50

$$loan \leftarrow loan - \sigma_{amount \geq 0 \text{ and } amount \leq 50}(loan)$$

- Delete all accounts at branches located in Needham.

$$r_1 \leftarrow \sigma_{branch\_city = \text{“Needham”}}(account \bowtie branch)$$

$$r_2 \leftarrow \Pi_{account\_number,\ branch\_name,\ balance}(r_1)$$

$$r_3 \leftarrow \Pi_{customer\_name,\ account\_number}(r_2 \bowtie \text{depositor})$$

$$account \leftarrow \text{account} - r_2$$

$$depositor \leftarrow \text{depositor} - r_3$$

Insertion

- To insert data into a relation, we either:
  - specify a tuple to be inserted
  - write a query whose result is a set of tuples to be inserted
- in relational algebra, an insertion is expressed by:

  $$r \leftarrow r \cup E$$

  where *r* is a relation and *E* is a relational algebra expression.
- The insertion of a single tuple is expressed by letting *E* be a constant relation containing one tuple.

Insertion Examples

- Insert information in the database specifying that Smith has $1200 in account A-973 at the Perryridge branch.

$$account \leftarrow account \cup \{(\text{“A-973”}, \text{“Perryridge”}, 1200)\}$$

$$\text{depositor} \leftarrow depositor \cup \{(\text{“Smith”}, \text{“A-973”})\}$$

- Provide as a gift for all loan customers in the Perryridge branch, a $200 savings account. Let the loan number serve as the account number for the new savings account.

$$r_1 \leftarrow (\sigma_{branch\_name\ =\ \text{“Perryridge”}}(borrower \bowtie \text{loan}))$$

$$account \leftarrow account \cup \prod_{loan\_number,\ branch\_name,\ 200} (r_1)$$

$$\text{depositor} \leftarrow depositor \cup \prod_{customer\_name,\ loan\_number} (r_1)$$

Updating

- A mechanism to change a value in a tuple without charging *all* values in the tuple
- Use the generalized projection operator to do this task

$$r \leftarrow \prod_{F_1, F_2, \ldots, F_l,} (r)$$

- Each F_i is either
  - the I^{th} attribute of r, if the I^{th} attribute is not updated, or,
  - if the attribute is to be updated F_i is an expression, involving only constants and the attributes of r, which gives the new value for the attribute

Update Examples

- Make interest payments by increasing all balances by 5 percent.

$$account \leftarrow \Pi_{\,account\_number,\ branch\_name,\ balance\ *\ 1.05}\ (account)$$

- Pay all accounts with balances over $10,000 6 percent interest and pay all others 5 percent

$$account \leftarrow \Pi_{\,account\_number,\ branch\_name,\ balance\ *\ 1.06}\ (\sigma_{\,BAL\ >\ 10000}\ (account\ )) \cup \Pi_{\,account\_number,\ branch\_name,\ balance\ *\ 1.05}\ (\sigma_{BAL\ \leq\ 10000}\ (account))$$

Example Queries

- Find the names of all customers who have a loan and an account at bank.

$$\Pi_{customer\_name}\ (borrower) \cap \Pi_{customer\_name}\ (depositor)$$

- Find the name of all customers who have a loan at the bank and the loan amount

$$\Pi_{customer\_name,\ loan\_number,\ amount}\ (borrower \bowtie loan)$$

Example Queries

- Find all customers who have an account from at least the "Downtown" and the Uptown" branches.
  - Query 1

$$\Pi_{customer\_name}\,(\sigma_{branch\_name\,=\,\text{"Downtown"}}\,(depositor \bowtie account\,)) \cap$$

$$\Pi_{customer\_name}\,(\sigma_{branch\_name\,=\,\text{"Uptown"}}\,(depositor \bowtie account))$$

  - Query 2

$$\Pi_{customer\_name,\ branch\_name}\,(depositor \bowtie account)$$
$$\div\ \rho_{temp(branch\_name)}\,(\{(\text{"Downtown"}),\ (\text{"Uptown"})\})$$

Note that Query 2 uses a constant relation.

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 the relation *r*
- *P* is a *formula* similar to that of the predicate calculus

Predicate 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:
   - $\exists\, t \in r\, (Q\, (t\,)) \equiv$ "there exists" a tuple *t* in the relation *r* such that predicate *Q* (*t*) is true
   - $\forall t \in r\, (Q\, (t\,)) \equiv Q$ is true "for all" tuples *t* in the relation *r*

Example Queries

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

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

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

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

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

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

Safety of Expressions

- It is possible to write tuple calculus expressions that generate infinite relations.
- For example, $\{ t \mid \neg\, t \in r \}$ results in an infinite relation if the domain of any attribute of relation *r* is infinite
- To guard against the problem, we restrict the set of allowable expressions to **safe expressions**.

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
  - $\{< i, n, d, s> \mid \ < i, n, d, s> \in instructor \land s > 80000\}$
- As in the previous query, but output only the *ID* attribute value
  - $\{< i> \ \mid < i, n, d, s> \in instructor \land s > 80000\}$
- Find the names of all instructors whose department is in the Watson building

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

Example Queries

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

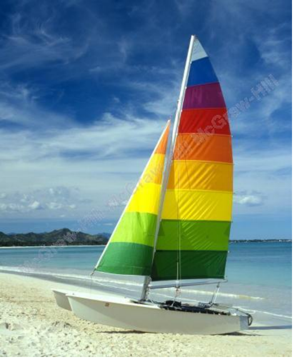
$$\{<c> \mid \exists\, a, s, y, b, r, t\ (<c, a, s, y, b, r, t> \in section\ \land$$
$$s = \text{“Fall”} \land y = 2009\ )$$
$$\lor\ \exists\, a, s, y, b, r, t\ (<c, a, s, y, b, r, t> \in section\ ] \land$$
$$s = \text{“Spring”} \land y = 2010)\}$$

This case can also be written as

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

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

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



End of Chapter 3