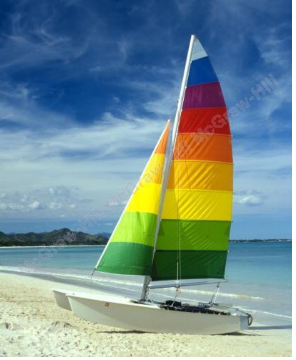


Chapter 6: Formal Relational Query Languages

Database System Concepts, 6^{th} Ed.

©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use

Chapter 6: Formal Relational Query Languages

- Relational Algebra
- Tuple Relational Calculus
- Domain Relational Calculus

Formal Languages

- A **formal language** is a set of strings of symbols that may be constrained by rules that are specific to it.
- E.g.
  - Relational Algebra
  - Tuple Relational Calculus
  - Domain Relational Calculus

Relational Algebra

- Relational algebra received little attention outside of pure mathematics until the publication of E.F. Codd's relational model of data in 1970. Codd proposed such an algebra as a basis for database query languages.
- 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.

Precedence of Relational Operators

- Precedence of relational operators

1. [σ, $\prod$, ρ] (highest).
2. [x ,$\bowtie$].
3. $\cap$.
4. [$\cup$, –]

Division Operator ***

- Inverse of Cartesian product
- 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. Practical exercise 6.4
  - let $r(ID, course\_id) = \prod_{ID,\ course\_id} (takes)$ and

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

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

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

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

$$result = temp1 - temp2$$

Extended Relational-Algebra-Operations

- Generalized Projection
- Aggregate Functions

Aggregate Functions and Operations

- **Aggregate 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)

Result of Aggregate Operation

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

- For each group (g_1, g_2, . . . , g_n), the result has a tuple
  - (g_1, g_2, . . . , g_n, a_1, a_2, . . . , a_n) where,
    - for each i, a_i is the result of applying the aggregate function F_i on the multiset of values for attribute A_i in the group.

Aggregate Operation – Example 1

- Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

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

sum(*c*)
27

Aggregate Operation – Example 2

- Find the total number of instructors who teach a course in the Spring 2010 semester
  - to eliminate duplicates, we use the hyphenated string "**distinct**" appended to the end of the function name (for example, **count-distinct**).

$$\mathcal{G}_{count-distinct(ID)}(\sigma_{semester=\text{"Spring"} \wedge year=2010}(teaches))$$

Aggregate Operation – Example 3

- Find the average salary in each department

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

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 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
  - See Appendix in the end of this Chapter

Multiset Relational Algebra

- Pure relational algebra removes all duplicates
  - e.g. after projection
- Multiset relational algebra retains duplicates, to match SQL semantics
  - SQL duplicate retention was initially for efficiency, but is now a feature
- Multiset relational algebra defined as follows
  - selection: has as many duplicates of a tuple as in the input, if the tuple satisfies the selection
  - projection: one tuple per input tuple, even if it is a duplicate
  - cross product: If there are *m* copies of *t1* in *r*, and *n* copies of *t2* in *s*, there are *m* x *n* copies of *t1.t2* in *r* x *s*
  - Other operators similarly defined
    - E.g. union: *m* + *n copies,* intersection: min(*m, n*) copies difference: max(0, *m* – *n*) copies

SQL and Relational Algebra

- **select** $A_1, A_2, .. A_n$
  from $r_1, r_2, \ldots, r_m$
  where P

  is equivalent to the following expression in multiset relational algebra

  $$\Pi_{A1, .., An} (\sigma_P (r_1 \times r_2 \times .. \times r_m))$$

- **select** A_1, A_2, **sum**(A_3)
  from $r_1, r_2, \ldots, r_m$
  where P
  group by A_1, A_2

  is equivalent to the following expression in multiset relational algebra

  $$_{A1, A2}\mathcal{G}_{\mathbf{sum}(A3)} (\sigma_P (r_1 \times r_2 \times .. \times r_m)))$$

SQL and Relational Algebra

- More generally, the non-aggregated attributes in the **select** clause may be a subset of the **group by** attributes, in which case the equivalence is as follows:

  select A_1, **sum**(A_3)
  from $r_1, r_2, \ldots, r_m$
  where P
  group by A_1, A_2

  is equivalent to the following expression in multiset relational algebra

$$\Pi_{A1, sumA3}({}_{A1,A2}\mathcal{G}_{\mathbf{sum}(A3)\ \mathbf{as}\ sumA3}(\sigma_P(r_1 \times r_2 \times .. \times r_m)))$$

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
- Nonprocedural query language:
  - It describes the desired information without giving a specific procedure for obtaining that information.

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:
   - Existential: $\exists\, t \in r\, (Q\,(t\,)) \equiv$ "there exists" a tuple in t in relation r such that predicate $Q\,(t\,)$ is true
   - Universal: $\forall t \in r\, (Q\,(t\,)) \equiv Q$ is true "for all" tuples t in relation r
     - $Q\,(t_1\,) \wedge Q\,(t_2) \wedge \ldots \wedge Q\,(t_n)$

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

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

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

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

Expressive Power of Languages

- For every relational-algebra expression using only the basic operations, there is an equivalent expression in the tuple relational calculus
  - The basic operations:
    - $\cup, -, \times, \sigma$, and ρ,
    - without the extended relational operations such as generalized projection and aggregation
- For every tuple-relational-calculus expression, there is an equivalent relational algebra expression.

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:

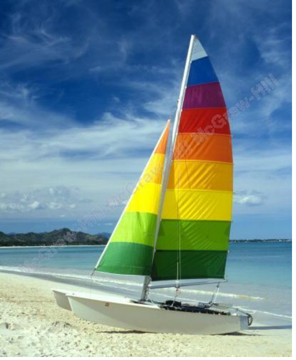
$$\{ < x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n)\}$$

  - $x_1, x_2, \ldots, x_n$ 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
  - $\{< i, n, d, s> \mid \; < i, n, d, s> \in \mathit{instructor} \land s > 80000\}$
- As in the previous query, but output only the *ID* attribute value
  - $\{< i> \mid \; < i, n, d, s> \in \mathit{instructor} \land s > 80000\}$
- Find the names of all instructors whose department is in the Watson building

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



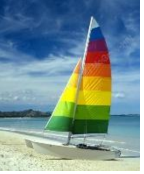
End of Chapter 6

©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use

Practical Exercises

Textbook-6th edition:

- 6.1
- 6.11



Appendix ***

- Deletion in relational algebra ***
- Insertion in relational algebra ***
- Update in relational algebra ***

Deletion in Relational Algebra

- 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.

 ©Silberschatz, Korth and Sudarshan

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 in Relational Algebra

- 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.

 ©Silberschatz, Korth and Sudarshan

Insertion Examples

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

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

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

- 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.

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

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

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

Updating in Relational Algebra

- 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 l^{th} attribute of r, if the l^{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

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

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.

Bank Example Queries

- Find all customers who have an account at all branches located in Brooklyn city.

$$\Pi_{customer\_name,\ branch\_name}(depositor \bowtie account)$$
$$\div\ \Pi_{branch\_name}(\sigma_{branch\_city\ =\ \text{“Brooklyn”}}(branch))$$