

Query Languages

- A query language is a language in which a user requests information from a database. These are typically higher-level than programming languages.
- They could be:
 - Procedural, where the user instructs the system to perform a sequence of operations on the database. This will compute the desired information.
 - Nonprocedural, where the user specifies the information desired without giving a procedure for obtaining the information.
- A complete query language also contains facilities to insert and delete tuples as well as to modify parts of existing tuples.

1. A basic expression consists of either
 - A relation in the database.
 - A constant relation.
2. General expressions are formed out of smaller subexpressions using
 - $\sigma_p(E_1)$ select (p a predicate)
 - $\Pi_s(E_1)$ project (s a list of attributes)
 - $\rho_x(E_1)$ rename (x a relation name)
 - $E_1 \cup E_2$ union
 - $E_1 - E_2$ set difference
 - $E_1 \times E_2$ cartesian product

Relational Algebra

Borrow

bname	loan#	cname	amount
Downtown	17	Jones	1000
Lougheed_Mall	23	Smith	2000
SFU	15	Hayes	1500

Branch

bname	assets	bcity
Downtown	9,000,000	Vancouver
Lougheed_Mall	21,000,000	Burnaby
SFU	17,000,000	Burnaby

The relational algebra is a procedural query language

- Six fundamental operations:
 - select (unary)
 - project (unary)
 - rename (unary)
 - cartesian product (binary)
 - union (binary)
 - set-difference (binary)
- Several other operations, defined in terms of the fundamental operations:
 - set-intersection
 - natural join
 - division
 - assignment

Operations produce a new relation as a result.

Relational Algebra – Fundamental Operations

Borrow

bname	loan#	cname	amount
Downtown	17	Jones	1000
Lougheed_Mall	23	Smith	2000
SFU	15	Hayes	1500

Branch

bname	assets	bcity
Downtown	9,000,000	Vancouver
Lougheed_Mall	21,000,000	Burnaby
SFU	17,000,000	Burnaby

The SELECT operation

Select selects tuples that satisfy a given predicate. Select is denoted by a lowercase Greek sigma (σ), with the predicate appearing as a subscript. The argument relation is given in parentheses following the σ .

For example, to select tuples (rows) of the *borrow* relation where the branch is “SFU”, we would write

$$\sigma_{bname="SFU"}(borrow)$$

The new relation created as the result of this operation consists of one tuple: (*SFU*, 15, *Hayes*, 1500).

We allow comparisons using $=$, \neq , \langle , \leq , $>$ and \geq in the selection predicate.

We also allow the logical connectives \vee (or) and \wedge (and). For example:

$$\sigma_{bname="Downtown" \wedge amount > 1200}(borrow)$$

Relational Algebra – Fundamental Operations

cname	banker
Hayes	Jones
Johnson	Johnson

The PROJECT operation

Project copies its argument relation for the specified attributes only. Since a relation is a **set**, duplicate rows are eliminated. Projection is denoted by the Greek capital letter pi (Π). The attributes to be copied appear as subscripts.

For example, to obtain a relation showing customers and branches, but ignoring amount and loan#, we write

$$\Pi_{bname, cname}(borrow)$$

We can perform these operations on the relations resulting from other operations. To get the names of customers having the same name as their bankers,

$$\Pi_{cname}(\sigma_{cname=banker}(client))$$

You can think of **SELECT** as taking **rows** of a relation, and **PROJECT** as taking **columns** of a relation.

Relational Algebra – Fundamental Operations

cname	banker
Hayes	Jones
Johnson	Johnson

The CARTESIAN PRODUCT operation

The **cartesian product** of two relations is denoted by a cross (\times), written

$$r_1 \times r_2 \text{ for relations } r_1 \text{ and } r_2$$

The result of $r_1 \times r_2$ is a new relation with a tuple for each possible **pairing** of tuples from r_1 and r_2 . In order to avoid ambiguity, the attribute names have attached to them the name of the relation from which they came. If no ambiguity will result, we drop the relation name.

The result $client \times customer$ is a very large relation. If r_1 has n_1 tuples, and r_2 has n_2 tuples, then $r = r_1 \times r_2$ will have $n_1 n_2$ tuples.

The resulting scheme is the concatenation of the schemes of r_1 and r_2 , with relation names added as mentioned.

Relational Algebra – Fundamental Operations

cname	banker
Hayes	Jones
Johnson	Johnson

The RENAME operation

The **rename** operation solves the problems that occurs with naming when performing the cartesian product of a relation with itself. Suppose we want to find the names of all the customers who live on the same street and in the same city as Smith. We can get the street and city of Smith by writing

$$\Pi_{street,ccity}(\sigma_{cname="Smith"}(customer))$$

To find other customers with the same information, we need to reference the *customer* relation again:

$$\sigma_P(customer \times (\Pi_{street,ccity}(\sigma_{cname="Smith"}(customer))))$$

where P is a selection predicate requiring *street* and *ccity* values to be equal.

Problem: how do we distinguish between the two street values appearing in the Cartesian product, as both come from a *customer* relation?

Solution: use the rename operator, denoted by the Greek letter rho (ρ). We write

$$\rho_x(r)$$

to get the relation r under the name of x .

Relational Algebra – Fundamental Operations

(a)

cname
Hayes
Adams




Fig. 3.5

The UNION operation

The **union** operation is denoted \cup as in set theory. It returns the union (set union) of two compatible relations. For a union operation $r \cup s$ to be legal, we require that

- r and s must have the same number of attributes.
- The domains of the corresponding attributes must be the same.

To find all customers of the SFU branch, we must find everyone who has a loan or an account or both at the branch. We need both *borrow* and *deposit* relations for this:

$$\Pi_{cname}(\sigma_{bname="SFU"}(borrow)) \cup \Pi_{cname}(\sigma_{bname="SFU"}(deposit))$$

As in all set operations, duplicates are eliminated, giving the relation of Figure 3.5(a).

Borrow

bname	loan#	cname	amount
Downtown	17	Jones	1000
Lougheed_Mall	23	Smith	2000
SFU	15	Hayes	1500

Deposit

bname	account#	cname	balance
Downtown	101	Johnson	500
Lougheed_Mall	215	Smith	700
SFU	102	Hayes	400
SFU	304	Adams	1300

Relational Algebra – Fundamental Operations

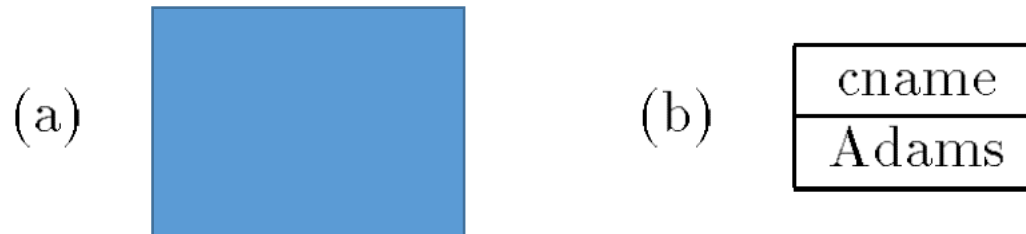


Fig. 3.5

The DIFFERENCE operation

Set difference is denoted by the minus sign ($-$). It finds tuples that are in one relation, but not in another. Thus $r - s$ results in a relation containing tuples that are in r but not in s .

To find customers of the SFU branch who have an account there but no loan, we write

$$\Pi_{cname}(\sigma_{bname="SFU"}(deposit)) - \Pi_{cname}(\sigma_{bname="SFU"}(borrow))$$

The result is shown in Figure 3.5(b).

bname	loan#	cname	amount
Downtown	17	Jones	1000
Lougheed_Mall	23	Smith	2000
SFU	15	Hayes	1500

Borrow

bname	account#	cname	balance
Downtown	101	Johnson	500
Lougheed_Mall	215	Smith	700
SFU	102	Hayes	400
SFU	304	Adams	1300

Deposit

Relational Algebra – Additional Operations

Additional operations are defined in terms of the fundamental operations. They do not add power to the algebra, but are useful to simplify common queries.

The SET INTERSECTION operation

Set intersection is denoted by \cap , and returns a relation that contains tuples that are in **both** of its argument relations. It does not add any power as

$$r \cap s = r - (r - s)$$

To find all customers having both a loan and an account at the SFU branch, we write

$$\Pi_{cname}(\sigma_{bname="SFU"}(borrow)) \cap \Pi_{cname}(\sigma_{bname="SFU"}(deposit))$$

bname	loan#	cname	amount
Downtown	17	Jones	1000
Lougheed_Mall	23	Smith	2000
SFU	15	Hayes	1500

Borrow

bname	account#	cname	balance
Downtown	101	Johnson	500
Lougheed_Mall	215	Smith	700
SFU	102	Hayes	400
SFU	304	Adams	1300

Deposit

Relational Algebra – Additional Operations

cname	ccity
Smith	Burnaby
Hayes	Burnaby
Jones	Vancouver

The NATURAL JOIN operation

Often we want to simplify queries on a cartesian product. For example, to find all customers having a loan at the bank and the cities in which they live, we need *borrow* and *customer* relations:

$$\Pi_{borrow.cname,ccity} (\sigma_{borrow.cname=customer.cname} (borrow \times customer))$$

Our selection predicate obtains only those tuples pertaining to only one *cname*.

This type of operation is very common, so we have the **natural join**, denoted by a \bowtie sign. Natural join combines a cartesian product and a selection into one operation. It performs a selection forcing equality on those attributes that appear in both relation schemes. Duplicates are removed as in all relation operations.

To illustrate, we can rewrite the previous query as

$$\Pi_{cname,ccity}(borrow \bowtie customer)$$

cname	street	ccity
Johnson	Pender	Vancouver
Smith	North	Burnaby
Hayes	Curtis	Burnaby
Adams	No.3 Road	Richmond
Jones	Oak	Vancouver

Customer

bname	loan#	cname	amount
Downtown	17	Jones	1000
Lougheed_Mall	23	Smith	2000
SFU	15	Hayes	1500

Borrow

Relational Algebra – Additional Operations

Division, denoted \div , is suited to queries that include the phrase “for all”.

The DIVISION operation

Suppose we want to find all the customers who have an account at **all** branches located in Brooklyn. Strategy: think of it as three steps.

We can obtain the names of all branches located in Brooklyn by

$$r_1 = \Pi_{bname}(\sigma_{bcity = \text{“Brooklyn”}}(branch))$$

Figure 3.19 in the textbook shows the result.

We can also find all *cname*, *bname* pairs for which the customer has an account by

$$r_2 = \Pi_{cname, bname}(deposit)$$

Figure 3.20 in the textbook shows the result.

Now we need to find all customers who appear in r_2 with **every** branch name in r_1 . The divide operation provides exactly those customers:

$$\Pi_{cname, bname}(deposit) \div \Pi_{bname}(\sigma_{bcity = \text{“Brooklyn”}}(branch))$$

which is simply $r_2 \div r_1$.

Relational Algebra – Additional Operations

The DIVISION operation (in **formal terms**)

- Let $r(R)$ and $s(S)$ be relations.
- Let $S \subseteq R$.
- The relation $r \div s$ is a relation on scheme $R - S$.
- A tuple t is in $r \div s$ if for every tuple t_s in s there is a tuple t_r in r satisfying both of the following:

$$t_r[S] = t_s[S] \quad (3.2.1)$$

$$t_r[R - S] = t[R - S] \quad (3.2.2)$$

- These conditions say that the $R - S$ portion of a tuple t is in $r \div s$ if and only if there are tuples with the $r - s$ portion **and** the S portion in r for **every** value of the S portion in relation S .

The division operation can be defined in terms of the fundamental operations.

$$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - r)$$

Relational Algebra – Additional Operations

The Assignment operation

Sometimes it is useful to be able to write a relational algebra expression in parts using a temporary relation variable (as we did with r_1 and r_2 in the division example).

The assignment operation, denoted \leftarrow , works like assignment in a programming language. We could rewrite our division definition as

$$\begin{aligned}temp1 &\leftarrow \Pi_{R-S}(r) \\temp2 &\leftarrow \Pi_{R-S}((temp1 \times s) - r) \\result &= temp1 - temp2\end{aligned}$$

No extra relation is added to the database, but the relation variable created can be used in subsequent expressions. Assignment to a permanent relation would constitute a modification to the database.

Tuple Relational Calculus

1. The tuple relational calculus is a nonprocedural language. (The relational algebra was procedural.) We must provide a formal description of the information desired.
2. A query in the tuple relational calculus is expressed as

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

i.e. the set of tuples t for which predicate P is true.

3. We also use the notation

- $t[a]$ to indicate the value of tuple t on attribute a .
- $t \in r$ to show that tuple t is in relation r .

For example, to find the branch-name, loan number, customer name and amount for loans over \$1200:

$$\{t \mid t \in borrow \wedge t[amount] > 1200\}$$

This gives us all attributes, but suppose we only want the customer names. (We would use **project** in the algebra.) We need to write an expression for a relation on scheme $(cname)$.

$$\{t \mid \exists s \in borrow (t[cname] = s[cname] \wedge s[amount] > 1200)\}$$

Domain Relational Calculus

1. Domain variables take on values from an attribute's domain, rather than values for an entire tuple.

1. An expression is of the form

$$\{\langle x_1, x_2, \dots, x_n \rangle \mid P(x_1, x_2, \dots, x_n)\}$$

where the $x_i, 1 \leq i \leq n$, represent domain variables, and P is a **formula**.

2. An atom in the domain relational calculus is of the following forms

- $\langle x_1, \dots, x_n \rangle \in r$ where r is a relation on n attributes, and $x_i, 1 \leq i \leq n$, are domain variables or constants.
- $x \Theta y$, where x and y are domain variables, and Θ is a comparison operator.
- $x \Theta c$, where c is a constant.

3. **Formulae** are built up from atoms using the following rules:

- An atom is a formula.
- If P is a formula, then so are $\neg P$ and (P) .
- If P_1 and P_2 are formulae, then so are $P_1 \vee P_2$, $P_1 \wedge P_2$ and $P_1 \Rightarrow P_2$.
- If $P(x)$ is a formula where x is a domain variable, then so are $\exists x(P(x))$ and $\forall x(P(x))$.

Find branch name, loan number, customer name and amount for loans of over \$1200.

$$\{\langle b, l, c, a \rangle \mid \langle b, l, c, a \rangle \in \text{borrow} \wedge a > 1200\}$$

Find all customers who have a loan for an amount > than \$1200.

$$\{\langle c \rangle \mid \exists b, l, a (\langle b, l, c, a \rangle \in \text{borrow} \wedge a > 1200)\}$$

Safety of Expressions & Expressive Power of Languages

Safety of Expressions

- A Tuple Relational Calculus expression may generate an infinite expression
i.e. $\{t \mid \neg(t \in borrow)\}$
- A Domain Relational Calculus expression may generate infinite expression
- Solution: restrict it to safe expressions ONLY

Expressive Power

All three of the following are equivalent:

- The relational algebra.
- The tuple relational calculus restricted to safe expressions.
- The domain relational calculus restricted to safe expressions.