UNIT-2

Relational Algebra and Calculus

PRELIMINARIES

In defining relational algebra and calculus, the alternative of referring to fields by position is more convenient than referring to fields by name: Queries often involve the computation of intermediate results, which are themselves relation instances, and if we use field names to refer to fields, the definition of query language constructs must specify the names of fields for all intermediate relation instances.

We present a number of sample queries using the following schema:

Sailors (*sid:* integer, *sname:* string, *rating:* integer, *age:* real)
Boats (*bid:* integer, *bname:* string, *color:* string)
Reserves (*sid:* integer, *bid:* integer, *day:* date)

The key fields are underlined, and the domain of each field is listed after the field name. Thus *sid* is the key for Sailors, *bid* is the key for Boats, and all three fields together form the key for Reserves. Fields in an instance of one of these relations will be referred to by name, or positionally, using the order in which they are listed above.

RELATIONAL ALGEBRA

Relational algebra is one of the two formal query languages associated with the re-lational model. Queries in algebra are composed using a collection of operators. A fundamental property is that every operator in the algebra accepts (one or two) rela-tion instances as arguments and returns a relation instance as the result.

Each relational query describes a step-by-step procedure for computing the desired answer, based on the order in which operators are applied in the query.

Selection and Projection

Relational algebra includes operators to *select* rows from a relation (σ) and to *project* columns (π). These operations allow us to manipulate data in a single relation. Con - sider the instance

of the Sailors relation shown in Figure 4.2, denoted as *S2*. We can retrieve rows corresponding to expert sailors by using the σ operator. The expression,

$$\sigma_{rating>8}(S2)$$

The selection operator σ specifies the tuples to retain through a *selection condition*. In general, the selection condition is a boolean combination (i.e., an expression using the logical connectives $\wedge$ and $\vee$) of *terms* that have the form *attribute* op *constant* or *attribute1* op *attribute2*, where op is one of the comparison operators <, <=, =, =, >=, or >.

The projection operator π allows us to extract columns from a relation; for example, we can find out all sailor names and ratings by using π. The expression $\pi_{sname,rating}(S2)$

Suppose that we wanted to find out only the ages of sailors. The expression

$$\pi_{age}(S2)$$

a single tuple with *age=35.0* appears in the result of the projection. This follows from the definition of a relation as a *set* of tuples. However, our discussion of relational algebra and calculus assumes that duplicate elimination is always done so that relations are always sets of tuples.

Set Operations

The following standard operations on sets are also available in relational algebra: *union* (U), *intersection* ($\cap$), *set-difference* ($-$), and *cross-product* ($\times$).

- Union: $R \cup S$ returns a relation instance containing all tuples that occur in *either* relation instance R or relation instance S (or both). R and S must be *union-compatible*, and the schema of the result is defined to be identical to the schema of R.

- Intersection: $R \cap S$ returns a relation instance containing all tuples that occur in *both* R and S. The relations R and S must be union-compatible, and the schema of the result is defined to be identical to the schema of R.

- Set-difference: $R - S$ returns a relation instance containing all tuples that occur in R but not in S. The relations R and S must be union-compatible, and the schema of the result is defined to be identical to the schema of R.

- Cross-product: $R \times S$ returns a relation instance whose schema contains all the fields of R (in the same order as they appear in R) followed by all the fields of S

(in the same order as they appear in *S*). The result of $R \times S$ contains one tuple $\langle r, s \rangle$ (the concatenation of tuples *r* and *s*) for each pair of tuples $r \in R$, $s \in S$. The cross-product opertion is sometimes called Cartesian product.

sid	*sname*	*rating*	*age*
31	*Lubbe*	*8*	*55.5*
58	*Rusty*	*10*	*35.0*

Figure 4.9 $S1 \cap S2$

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0

Figure 4.10 $S1 - S2$

The result of the cross-product $S1 \times R1$ is shown in Figure 4.11 The fields in $S1 \times R1$ have the same domains as the corresponding fields in *R*1 and *S*1. In Figure 4.11 *sid* is listed in parentheses to

emphasize that it is not an inherited field name; only the corresponding domain is inherited.

(*sid*)	*sname*	*rating*	*age*	(*sid*)	*bid*	*day*
22	*Dustin*	*7*	*45.0*	22	*101*	*10/10/96*
22	*Dustin*	*7*	*45.0*	58	*103*	*11/12/96*
31	*Lubber*	*8*	*55.5*	22	*101*	*10/10/96*
31	*Lubber*	*8*	*55.5*	58	*103*	*11/12/96*
58	*Rusty*	*10*	*35.0*	22	*101*	*10/10/96*
58	*Rusty*	*10*	*35.0*	58	*103*	*11/12/96*

Figure 4.11 $S1 \times R1$

Renaming

We introduce a renaming operator ρ for this purpose. The expression $\rho(R(F), E)$ takes an arbitrary relational algebra expression *E* and returns an instance of a (new) relation called *R*. *R* contains the same tuples as the result of *E*, and has the same schema as *E*, but some fields are renamed. The field names in relation *R* are the same as in *E*, except for fields renamed in the *renaming list F*.

For example, the expression $\rho(C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1)$ returns a relation that contains the tuples shown in Figure 4.11 and has the followi ng schema: C(*sid1:* integer, *sname:* string, *rating:* integer, *age:* real, *sid2:* integer, *bid:* integer,*day:* dates).

Joins

The *join* operation is one of the most useful operations in relational algebra and is the most commonly used way to combine information from two or more relations. Although a join can be defined as a cross-product followed by selections and projections, joins arise much more frequently in practice than plain cross-products.joins have received a lot of attention, and there are several variants of the join operation.

Condition Joins

The most general version of the join operation accepts a *join condition c* and a pair of relation instances as arguments, and returns a relation instance. The *join condition* is identical to a *selection condition* in form. The operation is defined as follows:

$$R \lhd\rhd c\ S = \sigma c(R \times S)$$

Thus $\lhd\rhd$ is defined to be a cross-product followed by a selection. Note that the condition *c* can (and typically *does*) refer to attributes of both *R* and *S*.

(*sid*)	*sname*	*rating*	*age*	(*sid*)	*bid*	*day*
22	*Dustin*	*7*	*45.0*	58	*103*	*11/12/96*
31	*Lubber*	*8*	*55.5*	58	*103*	*11/12/96*

Figure 4.12 *S*1 $\lhd\rhd$ *S*1.*sid*<*R*1.*sid R*1

Equijoin

A common special case of the join operation *R* $\lhd\rhd$ *S* is when the *join condition* con-sists solely of equalities (connected by $\wedge$) of the form *R.name*1 = *S.name*2, that is, equalities between two fields in *R* and *S*. In this case, obviously, there is some redun-dancy in retaining both attributes in the result.

Natural Join

A further special case of the join operation *R* $\lhd\rhd$ *S* is an equijoin in which equalities are specified on *all* fields having the same name in *R* and *S*. In this case, we can simply omit the join condition; the default is that the join condition is a collection of equalities on all common fields.

Division

The division operator is useful for expressing certain kinds of queries, for example: "Find the names of sailors who have reserved all boats." Understanding how to use the basic operators of the algebra to define division is a useful exercise.

(Q1) Find the names of sailors who have reserved boat 103.

This query can be written as follows:
$\pi sname((\sigma bid=103 Reserves) \lhd\rhd Sailors)$

We first compute the set of tuples in Reserves with *bid* = 103 and then take the natural join of this set with Sailors. This expression can be evaluated on instances of Reserves and Sailors. Evaluated on the instances *R*2 and *S*3, it yields a relation

(Q2) Find the names of sailors who have reserved a red boat.

$\pi sname((\sigma color='red' Boats) \lhd\rhd Reserves \lhd\rhd Sailors$

This query involves a series of two joins. First we choose (tuples describing) red boats.

(Q3) Find the colors of boats reserved by Lubber.

$\pi color((\sigma sname='Lubber' Sailors) \lhd\rhd Reserves \lhd\rhd Boats)$

This query is very similar to the query we used to compute sailors who reserved red boats. On instances *B*1, *R*2, and *S*3, the query will return the colors green and red.

(Q4) Find the names of sailors who have reserved at least one boat.

$\pi sname(Sailors \lhd\rhd Reserves)$

(Q5) Find the names of sailors who have reserved a red or a green boat.

$\rho(T empboats, (\sigma color='red' Boats) \cup (\sigma color='green' Boats))$
$\pi sname(Tempboats \lhd\rhd Reserves \lhd\rhd Sailors)$

(Q6) Find the names of sailors who have reserved a red and a green boat

$\rho(T empboats2, (\sigma color='red' Boats) \cap (\sigma color='green' Boats))$
$\pi sname(Tempboats2 \lhd\rhd Reserves \lhd\rhd Sailors)$

However, this solution is incorrect —it instead tries to compute sailors who have re-served a boat that is both red and green.

$\rho(T empred, \pi sid((\sigma color='red' Boats) \lhd\rhd Reserves))$
$\rho(T empgreen, \pi sid((\sigma color='green' Boats) \lhd\rhd Reserves))$
$\pi sname((Tempred \cap Tempgreen) \lhd\rhd Sailors)$

(Q7) Find the names of sailors who have reserved at least two boats.

$\rho(Reservations,\ \pi sid,sname,bid(Sailors \lhd\rhd Reserves))$

$\rho(Reservationpairs(1 \rightarrow sid1,\ 2 \rightarrow sname1,\ 3 \rightarrow bid1,\ 4 \rightarrow sid2,$

$5 \rightarrow sname2, 6 \rightarrow bid2), Reservations \times Reservations)$

$\pi sname1\sigma(sid1=sid2) \cap (bid1=bid2)Reservationpairs$

(Q8) Find the sids of sailors with age over 20 who have not reserved a red boat.

$\pi sid(\sigma age>20 Sailors) - \pi sid((\sigma color='red' Boats) \lhd\rhd Reserves \lhd\rhd Sailors)$

This query illustrates the use of the set-difference operator. Again, we use the fact that *sid* is the key for Sailors.

(Q9) Find the names of sailors who have reserved all boats.

The use of the word *all* (or *every*) is a good indication that the division operation might be applicable:

$\rho(T\ empsids,\ (\pi sid,bidReserves)/(\pi bidBoats))$
$\pi sname(Tempsids \lhd\rhd Sailors)$

(Q10) Find the names of sailors who have reserved all boats called Interlake.

$\rho(T\ empsids,\ (\pi sid,bidReserves)/(\pi bid(\sigma bname='Interlake'\ Boats)))$
$\pi sname(Tempsids \lhd\rhd Sailors)$

RELATIONAL CALCULUS

Relational calculus is an alternative to relational algebra. In contrast to the algebra, which is procedural, the calculus is nonprocedural, or *declarative*, in that it allows us to describe the set of answers without being explicit about how they should be computed.

Tuple Relational Calculus

A tuple variable is a variable that takes on tuples of a particular relation schema as values. That is, every value assigned to a given tuple variable has the same number and type of fields.

(Q11) Find all sailors with a rating above 7.

{S I S E *Sailors* ^ *S. rating* > 7*}*

with respect to the given database instance, *F* evaluates to (or simply 'is') true if one ofthe followingholds:

- *F* is an atomic formula *R* ☐ *Rel,* and *R* is assigned a tuple in the instance of relation*Rel.*

- *F* is a comparison *R.a* op *S.b*, *R.a* op *constant*, or *constant* op *R.a*, and the tuples assigned to *R* and *S* have field values *R.a* and *S.b* that make the comparison true.

- *F* is of the form ¬*p*, and *p* is not true; or of the form *p* ^ *q*, and both *p* and *q* are true; or of the form *p V q*, and one of them is true, or of the form □*p* *q* and *q* is true whenever[4] *p* is true.

- *F* is of the form □*R*(*p*(*R*)), and there is some assignment of tuples to the free variables in *p*(*R*), including the variable *R*,[5] that makes the formula *p*(*R*) true.

- *F* is of the form ◇*R*(*p*(*R*)), and there is some assignment of tuples to the free variables in *p*(*R*) that makes the formula *p*(*R*) true no matter what tuple is assigned to *R*.

(Q12) Find the names and ages of sailors with a rating above 7 .

$$\{P \mid \exists S \in Sailors(S.rating > 7 \wedge P.name = S.sname \wedge P.age = S.age)\}$$

This query illustrates a useful convention: *P* is considered to be a tuple variable with exactly two fields, which are called *name* and *age*, because these are the only fields of *P* that are mentioned and *P* does not range over any of the relations in the query; that is, there is no subformula of the form *P* ∈ *Relname*.

(Q13) Find the sailor name, boat id, and reservation date for each reservation

$$\{P \mid \exists R \in Reserves \quad \exists S \in Sailors$$
$$(R.sid = S.sid \wedge P.bid = R.bid \wedge P.day = R.day \wedge P.sname = S.sname)\}$$

(Q1) Find the names of sailors who have reserved boat 103.

$$\{P \mid \exists S \in Sailors \quad Reserves(R.sid = S.sid \quad R.bid = 103 \wedge P.sname = S.sname)\}$$

$\exists R \in$ $\wedge$

This query can be read as follows: "Retrieve all sailor tuples for which there exists a tuple in Reserves, having the same value in the *sid* field, and with *bid* = 103."

(Q2) Find the names of sailors who have reserved a red boat.

$$\{P \mid \exists S \in Sailors \; \exists R \in Reserves(R.sid = S.sid \wedge P.sname = S.sname \wedge \exists B \in Boats(B.bid = R.bid \wedge B.color = 'red'))\}$$

This query can be read as follows: "Retrieve all sailor tuples S for which there exist tuples R in Reserves and B in Boats such that $S.sid = R.sid$, $R.bid = B.bid$, and $B.color = 'red'$."

(Q7) Find the names of sailors who have reserved at least two boats. $\{P \mid$

$$S \; Sailors \; R1 \in Reserves \wedge R2 \in Reserves \; (S.sid = R1.sid \wedge R1.sid = R2.sid \wedge R1.bid \neq R2.bid \wedge P.sname = S.sname)\}$$

(Q9) Find the names of sailors who have reserved all boats.

$$\{P \mid \exists S \in Sailors \; \forall B \in Boats \; (\exists R \in Reserves(S.sid = R.sid \wedge R.bid = B.bid \wedge P.sname = S.sname))\}$$

(Q14) Find sailors who have reserved all red boats.

$$\{S \mid S \in Sailors \wedge \forall B \in Boats \; (B.color = 'red' \rightarrow (\exists R \in Reserves(S.sid = R.sid \wedge R.bid = B.bid)))\}$$

Domain Relational Calculus

A domain variable is a variable that ranges over the values in the domain of some attribute (e.g., the variable can be assigned an integer if it appears in an attribute

whose domain is the set of integers). A DRC query has the form $\{ \langle x_1, x_2, \ldots, x_n \rangle \mid p(\langle x_1, x_2, \ldots, x_n \rangle)\}$, where each x_i is either a *domain variable* or a constant and $p(\langle x_1, x_2, \ldots, x_n \rangle)$ denotes a DRC formula whose only free variables are thevari-ables among the x_i, $1 \leq i \leq n$. The result of this query is the set of all tuples $\langle x_1, x_2, \ldots, x_n \rangle$ for which the formula evaluates to true.