Relational calculus & relational algebra

Many high-level relational data manipulation languages (such as SQL) are based on either relational algebra or relational calculus. They are theoretical languages and were defined by E F Codd. Both these formal languages, relational calculus and relational algebra, have influenced commercial relational query languages in use today, for example SQL and visual query languages such as Query-By-Example (QBE). They illustrate the basic operations required by any such data manipulation language.

A query written in relational calculus describes the desired result without specifying how the result is achieved – languages based on relational calculus are called nonprocedural or **declarative.**. A query written in relational algebra, on the other hand, consists of a collection of step-by-step 'procedures' for computing the desired result and is said to be **procedural**. Relational algebra is examined more fully later in this section.

SQL

Structured Query Language (SQL), which is based on both relational algebra and relational calculus, was developed in the 1970s by IBM to support its relational products and has become the most important query language for relational databases. SQL was adopted by the American National Standards Institute (ANSI) and the International Standards Organisation (ISO) in 1986 and 1987 respectively. Since then it has continued to evolve. SQL is examined in detail in Chapters 5 and 6.

Relational algebra (RA)

Relational algebra refers to a collection of operations that act on relations and produce a single relation as a result. The reasons for studying RA include:

- Relational algebra is used as the basis for other, higher-level, data manipulation languages including SQL. We will use relational algebra to explain, in Chapters 5 and 6, how SQL queries are evaluated
- The algebra serves as a convenient basis for optimisation. An optimiser is an important component of a DBMS. When implementing a query, there is usually more than one way of performing the data access. It is up to the optimiser to decide which strategy to adopt. We will see in Chapter 7 that DBMSs often use the relational algebra as a high-level intermediate language into which SQL code is translated during the query optimisation process
- The algebra is also used as a measure to compare high-level relational languages such as SQL, and to test if the language is **relationally complete** – for example, to check if the query language is as powerful as the algebra.

There are a number of syntaxes available to illustrate the relational algebra operations. Here we use a fairly user-friendly syntax.

Relational algebra operators

Each RA operator takes one or more relations as its input and produces a new relation as output. Codd originally defined eight operators, in two classes:

The special relational operators:	RESTRICT	PROJECT
	JOIN	DIVIDE
Set operators:	UNION	INTERSECTION
	DIFFERENCE	Cartesian PRODUCT

The operators Restrict and Project act on single relations; Join, Divide, Union, Intersection, Difference, and Cartesian Product act on two relations at a time. The result of the each of the operations on the relation or relations is to produce one single relation. The relational algebra operators are described below.

RESTRICT (originally called SELECT)

This extracts tuples which satisfy a given condition from a single relation. For example, in the Film relation in figure 2.4, the condition would be 'all films directed by Tarantino'. The result is a new relation of the shaded area (2 tuples in this case).

Figure 2.4: Example of Restrict

Film

filmNo	title	director	country	year	genre
005	Reservoir Dogs	Tarantino	US	1992	Crime
006	Pulp Fiction	Tarantino	US	1994	Crime
008	Trainspotting	Boyle	UK	1996	
109	Infernal Affairs	Wai-Keung	China	2002	Crime
111	Snakes on a Plane	Ellis	US	2006	Disaster

Relational algebra operation:

Restrict from Film where director = 'Tarantino'

PROJECT

This extracts specified attributes from a specified relation. For example, in figure 2.5, we extract from the relation Film the attributes or columns 'title' and 'year'. The result is a new relation of the shaded area (2 columns in this case).

Figure 2.5: Example of Project

Film

filmNo	title	director	country	year	genre
005	Reservoir Dogs	Tarantino	US	1992	Crime
006	Pulp Fiction	Tarantino	US	1994	Crime
008	Trainspotting	Boyle	UK	1996	
109	Infernal Affairs	Wai-Keung	China	2002	Crime
111	Snakes on a Plane	Ellis	US	2006	Disaster

Relational algebra operation:

Project over Film [title, year]

JOIN

The Join operator is used when we require data from more than one table. The operation involves the linking of relations together by combining rows which are related in some way. Various forms exist, such as natural join, equi-join, theta-join and outer join. The natural join is a special case where the link is through some shared characteristic (attribute) usually a foreign key. In the example below, the Film relation has been altered to include a director number and a Director relation has been included. When we join the two tables together, we use the attribute directorNo as the common value.

Figure 2.6: Example of Join

Film

filmNo	title	directorNo	country	year	genre
005	Reservoir Dogs	101	US	1992	Crime
006	Pulp Fiction	101	US	1994	Crime
008	Trainspotting	322	UK	1996	
109	Infernal Affairs	166	China	2002	Crime
111	Snakes on a Plane	753	US	2006	Disaster

Director

directorNo	dName
101	Tarantino
322	Boyle
166	Wai-Keung
753	Ellis

The resulting relation is:

filmNo	title	directorNo	country	year	genre	dName
005	Reservoir Dogs	101	US	1992	Crime	Tarantino
006	Pulp Fiction	101	US	1994	Crime	Tarantino
008	Trainspotting	322	UK	1996		Boyle
109	Infernal Affairs	166	China	2002	Crime	Wai-Keung
111	Snakes on a Plane	753	US	2006	Disaster	Ellis

Relational algebra operation:

Film Join Director where Film directorNo = Director directorNo

The result of a Join operation is a new wider relation: each row is formed by joining two tuples (rows) in the original relations such that they have the same values in the common domain. In figure 2.6, the Film and Director relations are joined to produce a new relation. Note that when a natural join is performed the resulting relation only includes one copy of the 'join attributes' (in the example one copy of 'directorNo').

Other types of joins will be examined in Chapter 5.

Note that in the above example it is likely that filmNo would be chosen as the primary key of the Film relation; also directorNo for the Director relation. Note also that directorNo is a foreign key in the Film relation. **When joining relations together, they are usually (but not always) joined using a primary and foreign key 'link'.**

UNION

This builds a relation consisting of all tuples appearing in either or both of two specified relations. The two relations must be **union-compatible**, which means that they must have the same number of attributes and these attributes must come from the same domain, i.e. the corresponding attributes must be from the same pool of values and of the same type (e.g. numerical, character).

To demonstrate the Union operation the following two film relations will be used: one for English language films; the other for Canadian films.

Figure 2.7: Example relations which are union-compatible

EnglishLanguageFilm

filmNo	title	director	year
005	Reservoir Dogs	Tarantino	1992
006	Pulp Fiction	Tarantino	1994
008	Trainspotting	Boyle	1996
110	Videodrome	Cronenberg	1983
111	Snakes on a Plane	Ellis	2006
115	eXistenZ	Cronenberg	1999

CanadianFilm

filmNo	title	director	year
110	Videodrome	Cronenberg	1983
112	Mon Oncle Antoine	Jutra	1971
115	eXistenZ	Cronenberg	1999

The union of the two relations produces a relation of **either** English language films **or** Canadian films **or both**.

Relational algebra operation:

Union EnglishLanguageFilm **with** CanadianFilm

The resulting relation is shown in the next figure.

Figure 2.8: Example of a union

filmNo	title	director	year
005	Reservoir Dogs	Tarantino	1992
006	Pulp Fiction	Tarantino	1994
008	Trainspotting	Boyle	1996
110	Videodrome	Cronenberg	1983
111	Snakes on a Plane	Ellis	2006
112	Mon Oncle Antoine	Jutra	1971
115	eXistenZ	Cronenberg	1999

INTERSECTION

This builds a relation consisting of all tuples appearing in both of two specified relations. Again the two relations must be union-compatible. For example, if we have the two film relations as above, then the intersection of the two relations produces a relations of films which are English language **and (at the same time)** Canadian.

Relational algebra operation:

Intersection EnglishLanguageFilm **and** CanadianFilm

The resulting relation is shown in figure 2.9:

Figure 2.9: Example of Intersection

filmNo	title	director	year
110	Videodrome	Cronenberg	1983
115	eXistenZ	Cronenberg	1999

With the Intersection operation the order of the relations (or operands) can be reversed without affecting the result. In the example above the following operation would give the same result:

Intersection CanadianFilm and EnglishLanguageFilm

Note that the operators union, intersection and join all display this characteristic that their operands can be reversed and produce the same result – they are said to be **commutative**. Other operators involving two operands (difference and divide) are not commutative.

DIFFERENCE

This operation builds a relation consisting of all tuples appearing in the first but not the second of two specified relations. The two relations must be union-compatible. For example, if we have the two film relations as above, then the difference of the two relations is: 'all English language films, which are not Canadian'.

Relational algebra operation:

Difference EnglishLanguageFilm **and** CanadianFilm

The resulting relation is shown in figure 2.10:

Figure 2.10: Example of Difference

filmNo	title	director	year
005	Reservoir Dogs	Tarantino	1992
006	Pulp Fiction	Tarantino	1994
008	Trainspotting	Boyle	1996
111	Snakes on a Plane	Ellis	2006