Álgebra relacional

CI-0127 Bases de Datos, Universidad de Costa Rica

Sivana Hamer

Importante

Este documento recopila contenidos de diversos de sitios web especializados, académicos y documentos compartidos por universidades. Toda la información es utilizada con fines estrictamente académicos. Para más información, puede ver las referencias.

1 Introduction

Relational algebra is a formal language that defines a set of operations to the relational model. A sequence of relational algebra operations is a *relational algebra expression*. The results of applying an expression is a new relation that represents the result of the query in the database. Relational algebra is important as: (1) it formally defines relational model operations; (2) is used in processing and optimizing queries in RDMSs; and, (3) some concepts are used in the query language for RDMSs. We can also represent relational algebra expressions as a *query tree* or *query graph*.

The operations can be classified into two groups:

Set operations . These operators are related to set theory. These include UNION, INTERSECTION, SET DIFFERENCE, and CARTESIAN PRODUCT.

Relation operations. These operators are developed for relational databases. These include SELECT, PROJECT, and JOIN.

Furthermore, depending on the number of operators, we can also classify the operations as *unary operations* if they operate on only one relation (e.g., SELECT, PROJECT) or *binary operations* if they operation if they operate on two relations (e.g., JOIN).

2 SELECT operation

The SELECT operation (σ) selects all the tuples that satisfy a condition (c) from a relation (R). This operation works as a filter to the tuples (horizontal partition) that satisfy a condition with those that don't. Thus, we can say that the select operation selects certain rows. The following notation is used.

$$\sigma_c(R)$$

- The number of tuples selected will always be less or equal than the number of tuples in R.
- The fraction of tuples selected by the conditions is the *selectivity* of the condition.
- The condition can have multiple clauses separated by boolean operators (**AND** or **OR**). We can also apply a **NOT** to reverse the boolean result.

- The clauses can be between attributes ($Fname == Lname$) or between an attribute and a constant value ($Fname ==$ '$Pedro$').
- The clauses can compare one of the comparison operators $\{=, <, \leq, >, \geq, \neq\}$. If the values compared are unordered (e.g. Strings that represent colors) only the operators can be used $\{=, \neq\}$.
- σ in SQL is the **WHERE** clause.

Example: Get all the students who are named Pedro and are born after 1990.

$$\sigma_{Fname='Pedro' AND Byear>1990}(STUDENT)$$

```
SELECT *
FROM STUDENT
WHERE Fname = 'Pedro' AND Byear > 1990
```

3 PROJECT operation

The PROJECT operation (π) selects a list of attributes (L) of a relation (R). It is represented by the symbol. This operation works as a vertical partition between the needed columnas and those not needed. Thus, we can say that the project operation selects certain columns. The following notation is used.

$$\pi_L(R)$$

- The columns are retrieved in the same order as they appear in the list.
- The number of tuples selected will always be less or equal than the number of tuples in R.
- The *degree* is equal to the number of attributes in L.
- The operation removes duplicated tuples using *duplicate elimination*.
- π in SQL is specified by the attribute list in the **SELECT** clause with a **DISTINCT**.

Example: Get the first and last name for all the students.

$$\pi_{Fname,Lname}(STUDENT)$$

```
SELECT DISTINCT Fname, Lname
FROM STUDENT
```

4 RENAME operation

The RENAME operator (ρ) can rename a relation or attributes names. To rename a relation R into a new relation S, the notation is the following:

$$\rho_S(R)$$

To rename attributes based on a list of attributes $L = \{A_1, \cdots, A_m\}$ of a relation R, the notation is the following.

$$\rho_{(A_1,\cdots,A_m)}(R)$$

To rename the relation and the attribute names, the notation is the following.

$$\rho_{S(A_1,\cdots,A_m)}(R)$$

- The rename is applied in order.
- ρ in SQL is the **AS** aliasing.

Example: Rename the table STUDENT, and the attributes of the first name and birth year.

$$\rho_{S(FirstName,BirthYear)}(STUDENT)$$

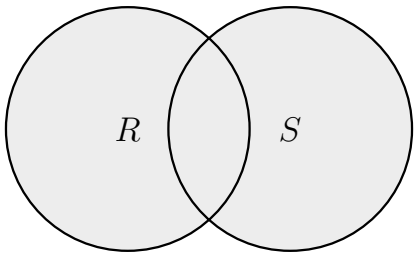
```
SELECT S.Fname AS FirstName, S.Byear AS BirthYear
FROM STUDENT AS S
```

5 UNION operation

The UNION operation ($\cup$) gets all the tuples in R or S, removing duplicate tuples. The following notation is used.

$$R \cup S$$

In a Venn Diagram, the UNION operation looks as following:



- To apply the operator, the *union compilability* or *type compatibility* condition must be followed. The attributes of the relations must have the same degree (number of attributes) and the corresponding pairs of attributes the same domain. Thus, the type of attributes must be the same.
- $\cup$ in SQL is **UNION**. **UNION ALL** does not eliminate duplicates.

Example: Get the list of all students or instructors.

$$STUDENT \cup INSTRUCTOR$$

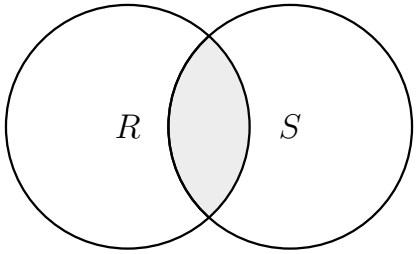
```
SELECT *
FROM STUDENT
UNION
SELECT *
FROM INSTRUCTOR
```

6 INTERSECTION operation

The INTERSECTION operation ($\cap$) gets all the tuples in R and S. The following notation is used.

$$R \cap S$$

In a Venn Diagram the operation is the following:



- The *type compatibility* condition applies for the operator.
- $\cap$ in SQL is **INTERSECTION**. **INTERSECTION ALL** does not eliminate duplicates.

Example: Get the list of people who are both students and instructors.

$$STUDENT \cap INSTRUCTOR$$

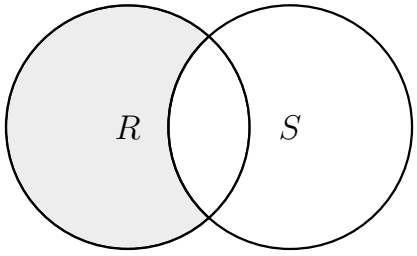
```
SELECT *
FROM STUDENT
INTERSECTION
SELECT *
FROM INSTRUCTOR
```

7 MINUS operation

The MINUS operation ($-$) gets all the tuples in R but not in S. The following notation is used.

$$R - S$$

In a Venn Diagram the operation is the following:



- It is also called *SET DIFFERENCE* or *EXCEPT*.
- The *type compatibility* condition applies for the operator.
- $-$ in SQL is **EXCEPT**. **EXCEPT ALL** does not eliminate duplicates.

Example: Get the list of students who are not instructors.

$$STUDENT - INSTRUCTOR$$

```
SELECT *
FROM STUDENT
EXCEPT
SELECT *
FROM INSTRUCTOR
```

8 CARTESIAN PRODUCT operation

The CARTESIAN PRODUCT operation ($\times$) combines every tuple from set $R(A_1, ..., A_n)$ with every tuple in set $S(B_1, ..., B_m)$. The following notation is used.

$$R \times S$$

- It is also called *CROSS PRODUCT* or *CROSS JOIN*.
- The number of tuples created will be $n_R * n_S$. n_R is the number of tuples in R, while n_S is the number of tuples in S.
- The degree of attributes of a tuple is $n + m$. n is the degree in R, while n_S m is the degree in S.
- The tuples for the new relation are generated in the order $(A_1, ..., A_n, B_1, ..., B_m)$.
- We mostly apply this operation, followed by a σ operation.
- $\times$ in SQL is **CROSS JOIN**. Also, if there are two tables in the **FROM** clause without a join condition in the **WHERE** clause.

Example: Get all the combinations of students enrolled to a course.

$$STUDENT \times ENROLLED$$

```
SELECT *
FROM STUDENT, ENROLLED
```

or

```
SELECT *
FROM STUDENT CROSS JOIN ENROLLED
```

9 JOIN operation

The JOIN operation ($\bowtie$) combines two relations (R and S) based on condition c. Only the combined tuples that satisfy the condition c are included in the resulting relation as a combined tuple. The following notation is used.

$$R \bowtie_c S$$

- We can have multiple conditions unified by boolean operators in c.
- A *THETA JOIN* has one of the comparison operators $\{=, <, \leq, >, \geq, \neq\}$ denominated θ.
- An *EQUIJOIN* uses only the $=$ comparison operator. The columns involved in the *EQUIJOIN* are repeated, as both have the same values in each column.
- A *NATURAL JOIN* $(*)$ is an *EQUIJOIN* that gets rid of the repeated column by combining both relations by a join attribute. The join attribute has the same name in both relations. If there is no column with the same name, it can be renamed using the ρ operation.
- Without a condition c, the *JOIN* becomes a *CARTESIAN PRODUCT*.
- The number of tuples created will be between 0 and $n_R * n_S$. n_R is the number of tuples in R, while n_S is the number of tuples in S.
- The degree and order of the resulting attributes is the same as the *CARTESIAN PRODUCT*.
- The *join selectivity* is the expected size of the join result divided by the maximum size of $n_R * n_S$.
- The *JOIN*s are *inner joins*.
- $\bowtie$ can be applied in SQL in various ways: (1) by specifying the join condition in the **WHERE** clause; (2) using a nested relation; and, (3) using join table instructions such as **JOIN**, **NATURAL JOIN** or **LEFT OUTER JOIN**.

Example: Get the courses students have enrolled.

$$(\rho_S(STUDENT)) \bowtie_{S.id=E.id} (\rho_E(ENROLLED))$$

```
SELECT *
FROM STUDENT AS S, ENROLLED AS E
WHERE S.id = E.id
```

or

```
SELECT *
FROM STUDENT AS S JOIN ENROLLED AS E ON S.id = E.id
```

$$STUDENT * ENROLLED$$

```
SELECT *
FROM STUDENT NATURAL JOIN ENROLLED
```

10 Sequences of operations

Generally, we apply several algebraic operations for a query. We could write an *in-line expression* that has all the operations in an expression. Alternatively, we can save in an intermediate relation the result using the assignment operator $(\leftarrow)$.

Example: Get the name of the students born after 1990.

The in-line expression is:

$$\pi_{Fname,Lname}(\sigma_{Byear>1990}(STUDENT))$$

Using intermediate relations is:

$$OVER1990 \leftarrow \sigma_{Byear>1990}(STUDENT)$$
$$RESULT \leftarrow \pi_{Fname,Lname}(OVER1990)$$

11 Complete set of operations

With the operations $\{\sigma, \pi, \cup, \rho, -, \times\}$ we can generate all the other algebra operations as a sequence of operations. Thus, these operations are a *complete set*. We can derive the following operations, though they are useful to express as they are frequently applied in DMS.

- $R \cap S \equiv (R \cup S) - ((R \cup S) \cup (S \cup R))$
- $\sigma_c(R \times S) \equiv (R \bowtie_c S)$
- $R \div S \equiv \pi_y(R) - \pi_y((S \times \pi_y(R)) - R)$

12 Equivalences

There are many rules that can transform a relational algebra expression into an equivalent expression. Two relational algebra expressions are *equivalent* if they have the same set of attributes in a different order but the expressions represent the same information. Thus, two expressions are equivalent if the same set of tuples is retrieved.

1. **Cascade of σ.** A selection condition with AND can be broken up into a sequence of individual σ operations.
$$\sigma_{c_1 AND c_2 AND \cdots AND c_n}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(\cdots(\sigma_{c_n}(R))\cdots))$$
Example: Get the students with the first name Pedro, born after 1990.
$$\sigma_{Fname=`Pedro' AND Byear>1990}(STUDENT) \equiv \sigma_{Fname=`Pedro'}(\sigma_{Byear>1990}(STUDENT))$$

2. **Commutativity of σ.** σ can be applied in any order.
$$\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$$
Example: Get the students with the first name Pedro, born after 1990.
$$\sigma_{Byear>1990}(\sigma_{Fname=`Pedro'}(STUDENT) \equiv \sigma_{Fname=`Pedro'}(\sigma_{Byear>1990}(STUDENT))$$

3. **Cascade of π.** When there is a sequence of π operations, all except the outermost one can be ignored.
$$\pi_{List_1}(\pi_{List_2}(\cdots(\pi_{List_n}(R))\cdots)) \equiv \pi_{List_1}(R)$$
Example: Get the if of the students.
$$\pi_{id}(\pi_{id,Fname}(\pi_{id,Fname,Lname}(STUDENT))) \equiv \pi_{id}(STUDENT)$$

4. **Commuting σ with π.** If the selection condition c involves only attributes $A_1, A_2, \cdots, A_N$ in the projection list, then the two operations can be applied in any order.

$$\pi_{A_1,A_2,\cdots,A_N}(\sigma_c(R)) \equiv \sigma_c(\pi_{A_1,A_2,\cdots,A_N}(R))$$

Example: Get the id with the full name of students with the first name Pedro.

$$\pi_{id,Fname,Lname}(\sigma_{Fname='Pedro'}(STUDENT)) \equiv \sigma_{Fname='Pedro'}(\pi_{id,Fname,Lname}(STUDENT))$$

5. **Commutativity $\bowtie$ and $\times$.** $\bowtie$ and $\times$ can be applied in any order.

$$R \bowtie_c S \equiv S \bowtie_c R$$

$$R \times S \equiv S \times R$$

Example: Get the courses students have enrolled.

$$STUDENT \bowtie_{S.id=E.id} ENROLLED \equiv ENROLLED \bowtie_{S.id=E.id} STUDENT$$

6. **Commutativity of σ with $\bowtie$ and $\times$.** If all the attributes of the selection condition c involve only the attributes of the joined relations, they can be applied in any order.

$$\sigma_c(R \bowtie S) \equiv (\sigma_c(R)) \bowtie S$$

Example: Get the students with the first name Pedro and born after 1990, with their course enrollment information.

$$\sigma_{Fname='Pedro' AND Byear>1990}(STUDENT \bowtie_{S.id=E.id} ENROLLED) \equiv$$
$$(\sigma_{Fname='Pedro' AND Byear>1990}(STUDENT)) \bowtie_{S.id=E.id} ENROLLED$$

Alternatively, if c can be written as c_1 and c_2 where c_1 has only the attributes of R and c_2 has only the attributes of S, they can be applied in any order.

$$\sigma_c(R \bowtie S) \equiv (\sigma_{c_1}(R)) \bowtie (\sigma_{c_2}(S))$$

Example: Get the students with the first name Pedro and with their enrolled courses where they got grades higher than 90.

$$\sigma_{Fname='Pedro' AND grade>90}(STUDENT \bowtie_{S.id=E.id} ENROLLED) \equiv$$
$$(\sigma_{Fname='Pedro'}(STUDENT)) \bowtie_{S.id=E.id} (\sigma_{grade>90}(ENROLLED))$$

7. **Commutativity of π with $\bowtie$ and $\times$.** For a list of attributes of the projection is called L, such that $L = A_1, \cdots, A_n, B_1, \cdots, B_m$ where $A_1, \cdots, A_n$ are attributes of R and $B_1, \cdots, B_n$ are attributes of S.

$$\pi_L(R \bowtie_c S) \equiv (\pi_{A_1,\cdots,A_n}(R)) \bowtie_c (\pi_{B_1,\cdots,B_m}(S))$$

If attributes $A_{n+1}, \cdots, A_{n+k}$ of R and $B_{m+1}, \cdots, B_{m+p}$ of S are involved for the join condition but are not in L, then these attributes must be added to the join condition. This does not apply for the $\times$ operator.

$$\pi_L(R \bowtie_c S) \equiv (\pi_{A_1,\cdots,A_n,A_{n+1},\cdots,A_{n+k}}(R)) \bowtie_c (\pi_{B_1,\cdots,B_m,B_{m+1},\cdots,B_{m+p}}(S))$$

Example: Get the name and course initials of students enrolled in courses.

$$\sigma_{Fname,Lname,Sigla}(STUDENT \bowtie_{S.id=E.id} ENROLLED) \equiv$$
$$(\pi_{Fname,Lname,id}(STUDENT)) \bowtie_{S.id=E.id} (\pi_{id,Sigla}(ENROLLED))$$

8. **Commutativity of $\cap$ and $\cup$.** $\cap$ and $\cup$ can be applied in any order.

$$R \cap S \equiv S \cap R$$

$$R \cup S \equiv S \cup R$$

Example: Get the list of people who are both students and instructors.

$$STUDENT \cap INSTRUCTOR \equiv INSTRUCTOR \cap STUDENT$$

9. **Associativity of $\bowtie$, $\times$, $\cap$ and $\cup$.** These four operations, represented by a θ, then can be associated in any order.

$$(R\theta S)\theta T \equiv R\theta(S\theta T)$$

Example: Get for all the students their enrolled courses with the course information.

$$(STUDENT \bowtie_{S.id=E.id} ENROLLED) \bowtie_{E.sigla=C.sigla} COURSE \equiv$$
$$STUDENT \bowtie_{S.id=E.id} (ENROLLED \bowtie_{E.sigla=C.sigla} COURSE)$$

10. **Commutativity of σ with set operations.** These set operations $\cap$, $\cup$ and $-$ (represented by θ can be applied in any order with σ

$$\sigma_c(R\theta S) \equiv (\sigma_c(R))\theta(\sigma_c(S))$$

Example: Get the list of people who are both students and instructors named Pedro.

$$\sigma_{Fname=\text{`}Pedro\text{'}}(STUDENT \cap TEACHER) \equiv$$
$$(\sigma_{Fname=\text{`}Pedro\text{'}}(STUDENT))\theta(\sigma_{Fname=\text{`}Pedro\text{'}}(TEACHER))$$

11. **Commutativity of π with $\cup$.** π and $\cup$ can be applied in any order.

$$\pi_L(R \cup S) \equiv (\pi_L(R)) \cup (\pi_L(S))$$

Example: Get the list of all students or instructors born after 1990.

$$\pi_{Byear>1990}(STUDENT \cup TEACHER) \equiv$$
$$(\pi_{Byear>1990}(STUDENT)) \cup (\pi_{Byear>1990}(TEACHER))$$

12. **Converting a $(\sigma, \times)$ sequence into $\bowtie$.** If a $\times$ is followed by a condition c of a σ, then it can be changed into a $\bowtie$.

$$\sigma_c(R \times S) \equiv (R \bowtie_c S)$$

Example: Get the courses students have enrolled.

$$\sigma_{s.id=e.id}(STUDENT \times ENROLLED) \equiv (STUDENT \bowtie_{s.id=e.id} ENROLLED)$$

There are other transformations such as pushing σ to set operators, removing trivial transformations and De Morgan's laws.

13 Query tree

A *query tree* is a tree data structure that represents a relational algebra expression. The *leaves* of the tree are the inputs. The *internal nodes* represent the relational algebra operations. When we execute a node when its operands are available and we replace the node with the executing results. The tree specified the order to execute the operations of the query. We start by executing the leaf nodes and finish when the root node is executed, producing the query result.

Example: Get the id of students who have gotten a 90 in an enrolled course.

$$\pi_{S.id}(\sigma_{E.grade=90}(STUDENT \bowtie_{S.id=E.id} ENROLLED))$$

```
SELECT S.id
FROM STUDENT as S, ENROLLED as E
WHERE S.id = E.id AND E.grade = 90
```

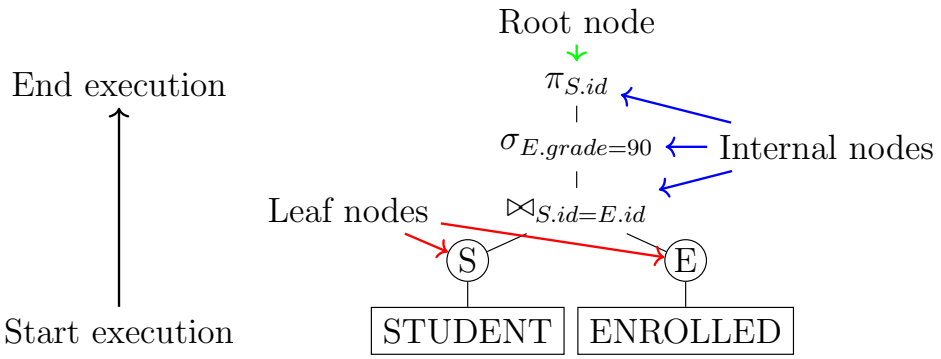


Figure 1: Parts of a query tree

14 Query graph

A *query graph* is a data structure that represents a relational calculus expression. Relationships are expressed as *relation nodes* with circles. Constant values from the query selection condition, represented by double circles or ovals, are represented by *constant nodes*. Selection or join operations are represented by *edges*. Finally, the *attributes* retrieved per relation are displayed in square brackets above each relation. Query graphs have no order, therefore there is only one query graph per query.

Example: Get the id of students who have gotten a 90 in an enrolled course.

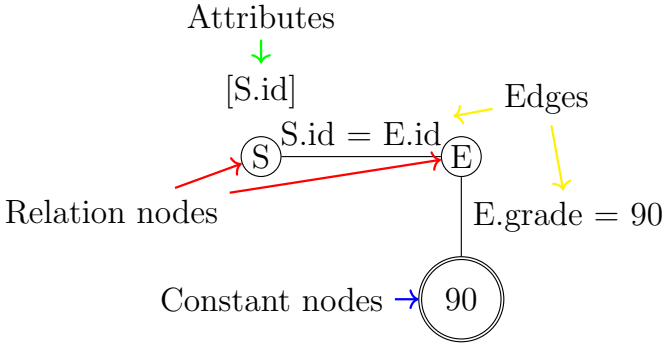


Figure 2: Parts of a query graph

References

[1] R. Elmasri and S. Navathe, *Fundamentals of database systems*. Pearson, 2016, vol. 7, chapter 6, 18.1 & 19.1.

[2] J. Widom. Relational algebra 1. [Video]. [Online]. Available: https://www.youtube.com/watch?v=tii7xcFilOA&list=PL6hGtHedy2Z4EkgY76QOcueU8lAC4o6c3&index=9

[3] ——. Relational algebra 2. [Video]. [Online]. Available: https://www.youtube.com/watch?v=GkBf2dZAES0&list=PL6hGtHedy2Z4EkgY76QOcueU8lAC4o6c3&index=10

