

# Relational Algebra and SQL

**Relational Algebra.** This algebra is an important form of query language for the relational model.

**The operators of the relational algebra:** divided into the following classes:

1. *Set operators:* union ( $\cup$ ), intersection ( $\cap$ ), and set difference ( $-$ )
2. *Operators that remove part of the relation:* projection or selection ( $\pi$  or  $\sigma$ )
3. *Operators that combine part of the relation:* cartesian product or joint ( $\times$  or  $\bowtie$ )
4. *Renaming operators:* rename the schema of the relation ( $\rho$ )

## Set operators

**Union ( $\cup$ ):**  $R \cup S$  represents a new relation whose tuples are either tuples belonging to  $R$ , to  $S$ , or both.

**Set difference ( $-$ ):**  $R - S$  represents a new relation whose tuples are tuples belonging to  $R$  but not to  $S$ .

**Intersection ( $\cap$ ):**  $R \cap S$  represents a new relation whose tuples are tuples belonging to  $R$  and  $S$ .

**Note:** Set operators only work on relations with the same schema.

**Example 1** Let  $R$  be the following relation instance:

$A$	$B$	$C$
1	2	3
2	1	1
3	2	2
4	1	3

and  $S$  be the following relation instance:

$A$	$B$	$C$
2	3	4
2	1	1
1	3	4
4	3	4

Then:  $R \cup S$  is

$A$	$B$	$C$
1	2	3
2	1	1
3	2	2
4	1	3
2	3	4
1	3	4
4	3	4

$R \cap S$  is

$A$	$B$	$C$
2	1	1

and  $R - S$  is

A	B	C
1	2	3
3	2	2
4	1	3

**NOTE:** If the schema of  $S$  changes, say  $S(A1, B1, C1)$  then all of the three operations ( $\cup$ ,  $\cap$ ,  $\setminus$ ) will yield the empty relation.

### Operators that remove part of the relation

**Projection ( $\pi$ )** This operation removes some columns from a relation. We write  $\pi_{A_1, \dots, A_k}(R)$  and the result is a new relation that has only the columns  $A_1, \dots, A_k$  of  $R$  and whose schema is the set of attributes  $\{A_1, \dots, A_k\}$  of the relation  $R$ . For the relation instance  $R$  in Example 1,

$$\pi_{A,B}(R)$$

is the relation instance:

A	B
1	2
2	1
3	2
4	1

**Selection ( $\sigma$ )** The operation selects some tuples from the current relation and defines a new relation.  $\sigma_C(R)$  represents a new relation that has tuples belonging to  $R$  and each of them satisfying the conditional expression (or condition)  $C$ .

$C$  is a conditional expression constructed from attributes of  $R$  or constants. For example,

$$A < B + C$$

would be a valid conditional expression with respect to the relation  $R$  in Example 1.

For the relation  $R$  in Example 1,  $\sigma_{A < B + C}(R)$  is the following relation instance:

A	B	C
3	2	2

### Operators that combine relations

**Cartesian Product ( $\times$ )** Given two relations  $R$  and  $S$ . The Cartesian product  $R \times S$  is the set of pairs that can be formed by choosing the first element of the pair to be any element of  $R$  and the second an element of  $S$ .

**Example 2** Let  $R$  be the following relation instance:

A	B	C
1	2	3
2	1	1
3	2	2
4	1	3

and  $S$  be the following relation instance:

<i>E</i>	<i>D</i>
2	3
2	1

Then:  $R \times S$  is

<i>A</i>	<i>B</i>	<i>C</i>	<i>E</i>	<i>D</i>
1	2	3	2	3
2	1	1	2	3
3	2	2	2	3
4	1	3	2	3
1	2	3	2	1
2	1	1	2	1
3	2	2	2	1
4	1	3	2	1

**NOTE:** When  $R$  and  $S$  share some attributes, the name of the shared attributes *need* to be redefined. Two common ways: (i) attaching the relation name to the attributes of the result; (ii) using the renaming operator.

Suppose  $S$  is given by

<i>A</i>	<i>D</i>
2	3
2	1

Then:  $R \times S$  is

<i>R.A</i>	<i>B</i>	<i>C</i>	<i>S.A</i>	<i>D</i>
1	2	3	2	3
2	1	1	2	3
3	2	2	2	3
4	1	3	2	3
1	2	3	2	1
2	1	1	2	1
3	2	2	2	1
4	1	3	2	1

Alternatively, we can write  $R \times S[A, B, C, D, E]$  and the result is

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
1	2	3	2	3
2	1	1	2	3
3	2	2	2	3
4	1	3	2	3
1	2	3	2	1
2	1	1	2	1
3	2	2	2	1
4	1	3	2	1

**Join** (also called *theta* join) The join operator combines two relations  $R$ , and  $S$  but using only tuples that are matching in some way. The general join operator has the form:

$$R \bowtie_{join\_cond} S$$

where *join\_cond* is an expression of the form

$$R.A_1 \ op_1 \ S.B_1 \ AND \ R.A_2 \ op_2 \ S.B_2 \ \dots \ AND \ R.A_n \ op_n \ S.B_n$$

$op_i$ 's are either  $=, \neq, >, \leq, <, \geq$ ,  $R.A_i$ 's are attributes of  $R$ , and  $S.B_j$ 's are attributes of  $S$ . (a conjunction of comparisons between attributes of  $R$  and  $S$ )

**Example 3** Let  $R$  be the following relation instance:

A	B	C
1	2	3
2	1	1
3	2	2
4	1	3

and  $S$  be the following relation instance:

B	D
2	3
3	1

Then:  $R \bowtie_{R.B=S.B} S$  is

A	B	C	S.B	D
1	2	3	2	3
3	2	2	2	3

The first tuple of  $R \bowtie_{R.B=S.B} S$  is combined from the 1<sup>st</sup> tuple of  $R$  and the 1<sup>st</sup> tuple of  $S$ . The second tuple of  $R \bowtie_{R.B=S.B} S$  is combined from the 3<sup>rd</sup> tuple of  $R$  and the 1<sup>st</sup> tuple of  $S$ . For the second tuple of  $S$ , the value of  $B$  is 3 and none of the tuples in  $R$  has  $B = 3$ . So, no new tuple for  $R \bowtie_{R.B=S.B} S$  can be formed using the second tuple of  $S$ .

**Special cases:** *Equi-join* is a join between  $R$  and  $S$  whose join condition is a conjunction of equality comparisons; *natural join* is an equi-join whose join condition is the conjunction of all equality comparisons between shared attributes of  $R$  and  $S$ . Often, a join without join-condition is understood as a natural join.

**Join is a derived operator** :  $R \bowtie_C S$  is given by

$$R \bowtie_C S = \sigma_C(R \times S).$$

**Renaming ( $\rho$ ):** This is another way for renaming  $\rho_{S(A_1, \dots, A_k)}(R)$  creates a new relation, named  $S$ , and attributes  $\{A_1, \dots, A_k\}$  with all the tuples of  $R$ .

**Example 4** Let  $R$  be the following relation instance:

A	B	C
1	2	3
2	1	1
3	2	2
4	1	3

Then,  $\rho_{T(M,N,P)}(R)$  is a new relation  $T$  with the following instance

M	N	P
1	2	3
2	1	1
3	2	2
4	1	3

NOTE:  $\rho_S(R)$  only changes the name of the relation.

**Outer-, Left Outer-, and Right Outer-join** The join operator might remove some tuples of the relations involved in the join. To retain the 'dangling' tuples (for different purposes), outer-, left outer-, or right outer-join can be used. They are defined as follows:

*Outer join*  $R \bowtie_C^{outer} S$  consists of (i) (the tuples in)  $R \bowtie_C S$ , (ii) the tuples in  $R$  that do not join with any tuple in  $S$  (NULL valued padded to the part of  $S$ ), and (iii) the tuples in  $S$  that do not join with any tuple in  $R$  (NULL valued padded to the part of  $R$ ).

*Left outer join*  $R \bowtie_C^{left} S$  consists of tuples in (i) and (iii) (of outer join)

*Right outer join*  $R \bowtie_C^{right} S$  consists of tuples in (i) and (ii) (of outer join)

**Example 5** Let  $R$  be the following relation instance:

A	B	C
1	2	3
2	1	1
3	2	2
4	1	3

and  $S$  be the following relation instance:

B	D
2	3
3	1

Then:  $R \bowtie_{R.B=S.B} S$  is

A	B	C	S.B	D
1	2	3	2	3
3	2	2	2	3

$R \bowtie_{R.B=S.B}^{outer} S$  is

A	B	C	S.B	D
1	2	3	2	3
3	2	2	2	3
2	1	1	NULL	NULL
4	1	3	NULL	NULL
NULL	NULL	NULL	3	1

$R \bowtie_{R.B=S.B}^{right} S$  is

A	B	C	S.B	D
1	2	3	2	3
3	2	2	2	3
2	1	1	NULL	NULL
4	1	3	NULL	NULL

$R \bowtie_{R.B=S.B}^{left} S$  is

A	B	C	S.B	D
1	2	3	2	3
3	2	2	2	3
NULL	NULL	NULL	3	1

**Division operator** For  $R$  with the set of attributes  $\{A_1, \dots, A_n, B_1, \dots, B_m\}$  and  $S$  with the set of attributes  $\{B_1, \dots, B_m\}$   $R/S$  is a relation over the set of attributes  $\{A_1, \dots, A_n\}$  with the set of tuples  $\langle a \rangle$  where  $\{\langle a \rangle\} \times S \subseteq R$ .

**Example 6** Let  $R$  be the following relation instance:

A	B	C
1	2	3
2	1	1
3	2	2
1	3	1

and  $S$  be the following relation instance:

B	C
2	3
3	1

Then:  $R/S$  is

A
1

**Combining Operations to Form Queries** The operators of relational algebra allows us to form expressions of arbitrary complexity by applying operators to either a given relation or to the relations that are the result of applying operators to relations.

Example like the theta-join operator, or something like  $\pi_{A,B}(\sigma_{C>D}(R \times S))$  etc.

**Simple SQL queries.** A simple SQL query has the following form

```
SELECT targetList
FROM   relationList
WHERE  condition
```

with

- `relationList` is a list of relations. It is often written as `Table1 T1, Table2 T2, ..., Tablek Tk` where `Tablei` is a relation;
- `targetList` is a list of attributes (of relations appearing in `relationList`) or expressions constructed from attributes (of relations appearing in `relationList`), constants, and the well-known arithmetic operations (+, −, /, \*, etc.) as well as Boolean operators AND, OR, NOT;
- `condition` is a Boolean formula formed from attributes (of relations appearing in `relationList`), constants, and the well-known arithmetic operations (+, −, /, \*, etc.), AND, OR, NOT, etc.

Roughly, a simple query of the above form is ‘equivalent’ to the relational algebraic expression

$$\pi_{\text{targetList}}(\sigma_{\text{condition}}(T1 \times T2 \times \dots \times Tk))$$

with the exception that without explicitly mention ‘DISTINCT’ before `targetList`, the SQL query will return duplicates in the answer while the evaluation of relational algebra expression will remove duplicates.

**What is important?** the steps involving in the evaluation of a SQL query, especially when the query contains subqueries; the use of Boolean operators in forming the condition in the WHERE clause.

**Aggregations in SQL.** There are five aggregates operators in SQL `COUNT`, `AVG`, `SUM`, `MIN`, `MAX` that operated on *numeric* attributes (except `COUNT`). When specified, these operators yield a *single* number, which can be used — like a constant — in the condition of the `WHERE` clause. Often, these operators are used in conjunction with `GROUP BY`. SQL queries with these operators and `GROUP BY` are of the following form:

```
SELECT targetList, aggList
FROM   relationList
WHERE  condition
GROUP BY groupList
```

with `aggList` and `groupList` are two new components of the query and

- `aggList` is a list of aggregations;
- `groupList` is a list of attributes, used to divide tuples into group before the aggregations are computed.

**What is important?** the steps involving in the evaluation of a SQL query when `GROUP BY` and aggregations are present.

**General SQL queries.** with `HAVING groupCondition` and `ORDER BY attList` clauses;  
`HAVING groupCondition` is used to eliminate the group that does not satisfy the `groupCondition` from the final result of the query; `ORDER BY attList` is used to order the final result. Again, it is important to know the steps (step-by-step) of the query evaluation.

**View.** helps improving the readability, maintainability, and security of SQL queries and data; can be materialized; difficult in maintaining; important to remember that views are not stored (different from *base relations*) and rewriting is used when evaluation of queries containing views;

**Modifying relation instance.** SQL can be used to modify relation instances; `INSERT`, `UPDATE`, and `DELETE` can be used to insert, update, or delete tuples from relations; all three operations can be done on a single tuple or a set of tuples; the syntax is really simple:

```
INSERT INTO Table(attList) VALUES(valueList)
INSERT INTO Table(attList) AS (SQL query)
UPDATE Table SET (att-valList) WHERE condition
DELETE FROM Table WHERE condition
```