Relational Algebra

Databases

2018-2019

Jesús Correas – jcorreas@ucm.es

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

Bibliografía

- Basic Bibliography:
  - R. Elmasri, S.B. Navathe. **Fundamentals of Database Systems** (6th ed). Addison-Wesley, 2010. (in Spanish: **Fundamentos de Sistemas de Bases de Datos** (5a ed). Addison-Wesley, 2007). Chapter 6 (5th, 6th eds).
- Additional Bibliography:
  - A. Silberschatz , H. F. Korth, S. Sudarshan. **Database Systems Concepts** (5th ed) McGraw-Hill, 2006. (in Spanish: **Fundamentos de Bases de Datos** (5a ed) McGraw-Hill, 2006). Chapter 2.

Relational DB query languages

- **Query languages** are designed to request information from relational databases.
  - In principle they do not allow to modify data in the DB. We will see data modification operations later in SQL.
- There are two kinds of query languages:
  - **Procedural languages:** The language tells the DB system a series of operations that must be done to obtain the result.
  - **Non-procedural languages:** The information that must be retrieved is described, but no specific procedure is given to obtain the information.
- Several **formal languages** have been proposed to perform queries on a relational DB:
  - Procedural: **relational algebra.**
  - Non-procedural: **tuple relational calculus** and **domain relational calculus.**
- We will see **relational algebra,** since it establishes the foundations of the query subset of SQL.

Relational Algebra

- It is a **formal language** for **queries** based on **algebra of sets** in mathematical set theory:
  - It does not include data modification operations.
  - Each operation of Relational Algebra produces **a new relation** in the RM that can be further manipulated using operations of the algebra.
- Queries are defined as the application of a series of **operations** on relations in a relational DB:
  - Operations from set theory: **union $\cup$, intersection $\cap$, set difference $\setminus$ and cartesian product $\times$.**
  - **Rename ρ.**
  - **Selection σ.**
  - **Proyection π.**
  - **Join $\bowtie$.**
  - **Division $\div$.**
- We also use the **assignment ($\leftarrow$) operation** on temporary relations to give a name to intermediate results.

Operations from set theory

- **Union, intersection, set difference, and cartesian product.**
- All are **binary** operations.
- They are more restrictive than usual set theoretic operations: Operands in the first three operations **($\cup, \cap$ and $\setminus$)** must contain **tuples of the same type:** instances of relation schemas **with the same number of attributes, defined in the same domains and in the same order:**

$$R(A_1, \ldots, A_n), S(B_1, \ldots, B_n), Dom(A_i) = Dom(B_i)(\forall i \leq n)$$

- **Union:** $R \cup S$ produces a relation that includes all valid tuples that are **either in** R **or in** S**.** Duplicate tuples are eliminated.
- **Intersection:** $(R \cap S)$ produces a relation containing the valid tuples that are **in both** R **and** S**.**
- **Set difference:** $R \setminus S$ produces a relation that contains the valid tuples that are **in** R **but not in** S**.**
- We will see the **cartesian product** later.

Rename operation

- There are some operations in which the **names of attributes and relations are important.** It is thus useful to have a renaming operation.
- Given $R(A_1, \ldots, A_n)$, the **rename operation** $\rho_{S(B_1,\ldots,B_n)}(R)$ produces the following relation:
  - The **name** of the schema is S.
  - **Schema attributes:** $B_1, \ldots, B_n$.
  - **Relation instance:** The valid tuples in R.
- **Examples:** Given `EMPL(Id,Name,LastName,Salary)`:
  - **Rename attribute Id:**

    $\rho_{\texttt{EMPL(COD,Name,LastName,Salary)}}(\texttt{EMPL})$
  - **Rename all attributes:**

    $\rho_{\texttt{EMPL(COD,N,L,S)}}(\texttt{EMPL})$
  - **Rename schema name to *employees*, keeping attribute names:**

    $\rho_{\texttt{employees}}(\texttt{EMPL})$

Selection operation

- The **selection** operation selects the tuples of a relation that satisfy a **selection condition** given as a **Boolean expression.**
- Given a relation schema $R(A_1, \ldots, A_n)$ and a Boolean expression C specified on the attributes of R, $\sigma_{\mathrm{C}}(\mathrm{R})$ produces a relation with the following characteristics:
  - **Schema attributes:** the same attributes as R.
  - **Relation instance:** contains the tuples of R that satify C.
- Valid conditions are Boolean expressions of the form:

$C \rightarrow$ *attribute* **OP** *attribute*

$C \rightarrow$ *attribute* **OP** *constant*

$C \rightarrow C \wedge C$

$C \rightarrow C \vee C$

$C \rightarrow \neg C$

  - *attribute* is an attribute of R.
  - *constant* is a constant value.
  - **OP** is a relational operator: $\{<, >, \leq, \geq, =, \neq\}$.

- The selection condition is evaluated **for each tuple contained in the input relation.**

Selection operation

- If we see a relation as a table of values, the selection operation performs a **"horizontal partition"** of the input relation into two sets of tuples, discarding the tuples that do not satisfy the condition.
- **Examples:** Given the schema `EMPL(Id,Name,LastName,Salary)`:
  - Select the **employee with Id '27347234T':**

    $\sigma_{\texttt{Id='27347234T'}}(\texttt{EMPL})$
  - Select the **employees with a salary between 1200 and 1500 €:**

    $\sigma_{\texttt{(salary}\geq\texttt{1200)} \wedge \texttt{(salary}\leq\texttt{1500)}}(\texttt{EMPL})$
  - **Employees whose family name is 'García' or have a salary of less than 1000 €:**

    $\sigma_{\texttt{(LastName} = \texttt{'García')} \vee \texttt{(salary}<\texttt{1000)}}(\texttt{EMPL})$
  - **The input relation can be another expression of relational algebra. What is the meaning of the following expression?**

    $\sigma_{\texttt{LastName} = \texttt{'García'}}(\sigma_{\texttt{salary}<\texttt{1000}}(\texttt{EMPL}))$

Projection operation

- The **projection** operation **extracts columns** (values of attributes) from a relation.
- Given a relation schema $R(A_1, \ldots, A_n)$ and a subset of attributes $\{B_1, \ldots, B_k\} \subseteq \{A_1, \ldots, A_n\}$, the operation $\pi_{(B_1, \ldots, B_k)}(R)$ produces a relation with the following characteristics:
  - **Schema attributes:** $\{B_1, \ldots, B_k\}$.
  - **Relation instance:** the set of tuples composed of the values of attributes $B_1, \ldots, B_k$ in the set of valid tuples of R.
- Observe that the relation obtained **may contain less tuples than the ones contained in** R if the primary key is not contained in $\{B_1, \ldots, B_k\}$.
  - Relation instances are **sets:** the result of a projection is a **set** of tuples and **cannot contain duplicate tuples.**

Projection operation

- If we see a relation as a table of values, the projection operation performs a **"vertical partition"** of the input relation, keeping a subset of columns (attributes).
- **Examples:** Given the schema `EMPL(Id,Name,LastName,Salary)`:
  - Obtain the **Id and salary of employees:**

    $$\pi_{\texttt{(Id,Salary)}}(\texttt{EMPL})$$

  - **Id of employees with salary greater than 1200 euros:**

    $$\pi_{\texttt{(Id)}}(\sigma_{\texttt{(salario}>\texttt{1200)}}(\texttt{EMPL}))$$

  - We can use **temporary relation names** for intermediate results:

    $$\texttt{Emp1} \leftarrow \sigma_{\texttt{(salary}>\texttt{1200)}}(\texttt{EMPL})$$
    $$\texttt{Emp2} \leftarrow \pi_{\texttt{(Id)}}(\texttt{Emp1})$$

Cartesian product operation

- The **cartesian product** **combines the values contained in two relations.**
- The relational cartesian product **slightly differs from the standard set-theoretic cartesian product.**
- Given $R(A_1, \ldots, A_m)$ and $S(B_1, \ldots, B_n)$, the cartesian product produces a relation schema $R \times S$ with the following characteristics:
  - **Schema attributes:** The output schema has $m + n$ attributes: $A_1, \ldots, A_m, B_1, \ldots, B_n$
  - If two attributes A_i and B_j have the same name, they are renamed using the relation name as prefix: $R.A_i$ y $S.B_j$.
  - **Relation instance:** The tuples of the result are those coming from the set-theoretic cartesian product of the sets of tuples R and S.
- We can avoid the automatic renaming of attribute names if we apply a rename operation to the input relations before applying the cartesian product.
  - Otherwise, we can always use **attribute names prefixed by relation names in conditions.** Example: `EMPL.Id = DPTO.Id`

Cartesian product operation

- **Example:** Given the following relation schemas:

  `EMPL(Id,Name,LastName,Salary)`

  `PROJECT(PrjCode,Description,ManagerId)`

- **Name of the employees that are managers of (at least) one project:**

Cartesian product operation

- **Example:** Given the following relation schemas:

  `EMPL(Id,Name,LastName,Salary)`

  `PROJECT(PrjCode,Description,ManagerId)`
- **Name of the employees that are managers of (at least) one project:**
  - We can combine both tables using the **cartesian product:**

    Managers1 $\longleftarrow$ **EMPL** $\times$ **PROJECT**

    This obtains **all possible combinations of employees with projects.**

Cartesian product operation

- **Example:** Given the following relation schemas:

  EMPL(<u>Id</u>,Name,LastName,Salary)

  PROJECT(<u>PrjCode</u>,Description,ManagerId)

- **Name of the employees that are managers of (at least) one project:**
  - We can combine both tables using the **cartesian product:**

    $\mathbf{Managers1} \leftarrow \mathbf{EMPL} \times \mathbf{PROJECT}$

    This obtains **all possible combinations of employees with projects.**
  - We can then **select** those tuples in which the employee Id matches the manager of the project (in the same tuple):

    $\mathbf{Managers2} \leftarrow \sigma_{\mathbf{(Id=ManagerId)}}(\mathbf{Managers1})$

Cartesian product operation

- **Example:** Given the following relation schemas:

  `EMPL(Id,Name,LastName,Salary)`

  `PROJECT(PrjCode,Description,ManagerId)`
- **Name of the employees that are managers of (at least) one project:**
  - We can combine both tables using the **cartesian product:**

    $\texttt{Managers1} \leftarrow \texttt{EMPL} \times \texttt{PROJECT}$

    This obtains **all possible combinations of employees with projects.**
  - We can then **select** those tuples in which the employee Id matches the manager of the project (in the same tuple):

    $\texttt{Managers2} \leftarrow \sigma_{\texttt{(Id=ManagerId)}}(\texttt{Managers1})$
  - Finally, we **project** the name of the managers to obtain the final result:

    $\texttt{Managers} \leftarrow \pi_{\texttt{(Name,LastName)}}(\texttt{Managers2})$
- Observe that each manager appears only once **even though that manager possibly manages several projects.**
- In a compositional way:

  $\pi_{\texttt{(Name,LastName)}}(\sigma_{\texttt{(Id=ManagerId)}}(\texttt{EMPL} \times \texttt{PROJECT}))$

Cartesian product operation

- **Another example:** Given the following relation schemas:

  EMPL(Id,Name,LastName,Salary)

  PROJECT(PrjCode,Description,ManagerId)

  ALLOCATION(PrjCode,EmplId,Hours)

- **Name of the employees that work on a project more than 10 hours:**

Cartesian product operation

- **Another example:** Given the following relation schemas:

  `EMPL(Id,Name,LastName,Salary)`

  `PROJECT(PrjCode,Description,ManagerId)`

  `ALLOCATION(PrjCode,EmplId,Hours)`

- **Name of the employees that work on a project more than 10 hours:**
  - We first **select** the allocations of employees to projects of more than 10 hours:

    $\texttt{Distr10} \leftarrow \sigma_{\texttt{Hours}>\texttt{10}}(\texttt{ALLOCATION})$

Cartesian product operation

- **Another example:** Given the following relation schemas:

  `EMPL(Id,Name,LastName,Salary)`

  `PROJECT(PrjCode,Description,ManagerId)`

  `ALLOCATION(PrjCode,EmplId,Hours)`

- **Name of the employees that work on a project more than 10 hours:**
  - We first **select** the allocations of employees to projects of more than 10 hours:

    $\texttt{Distr10} \leftarrow \sigma_{\texttt{Hours}>\texttt{10}}(\texttt{ALLOCATION})$
  - We then **combine** this result with the EMPL relation to get the information of those employees:

    $\texttt{Empl10} \leftarrow \sigma_{(\texttt{Id=EmplId})}(\texttt{Distr10} \times \texttt{EMPL})$

Cartesian product operation

- **Another example:** Given the following relation schemas:

  EMPL(Id,Name,LastName,Salary)

  PROJECT(PrjCode,Description,ManagerId)

  ALLOCATION(PrjCode,EmplId,Hours)

- **Name of the employees that work on a project more than 10 hours:**
  - We first **select** the allocations of employees to projects of more than 10 hours:

    $\texttt{Distr10} \leftarrow \sigma_{\texttt{Hours}>10}(\texttt{ALLOCATION})$
  - We then **combine** this result with the EMPL relation to get the information of those employees:

    $\texttt{Empl10} \leftarrow \sigma_{(\texttt{Id=EmplId})}(\texttt{Distr10} \times \texttt{EMPL})$
  - Finally, we **project** the name of the selected employees:

    $\texttt{Result} \leftarrow \pi_{(\texttt{Name,LastName})}(\texttt{Empl10})$

- In a single expression:

$$\pi_{(\texttt{Name,LastName})}(\sigma_{(\texttt{Id=EmplId})}(\sigma_{\texttt{Hours}>10}(\texttt{ALLOCATION}) \times \texttt{EMPL}))$$

Cartesian product operation

- **Yet another example:** Given the same relation schemas:

  `EMPL(Id,Name,LastName,Salary)`

  `PROJECT(PrjCode,Description,ManagerId)`

  `ALLOCATION(PrjCode,EmplId,Hours)`

- **Id of the employees that are allocated to at least two projects:**

Cartesian product operation

- **Yet another example:** Given the same relation schemas:

  EMPL(Id,Name,LastName,Salary)

  PROJECT(PrjCode,Description,ManagerId)

  ALLOCATION(PrjCode,EmplId,Hours)

- **Id of the employees that are allocated to at least two projects:**
  - We first **rename** the ALLOCATION relation:

    $\texttt{ALLOC\_R} \leftarrow \rho_{\texttt{ALLOC\_R(CodR,IdR,HR)}}(\texttt{ALLOCATION})$

Cartesian product operation

- **Yet another example:** Given the same relation schemas:

  `EMPL(`<u>`Id`</u>`,Name,LastName,Salary)`

  `PROJECT(`<u>`PrjCode`</u>`,Description,ManagerId)`

  `ALLOCATION(`<u>`PrjCode`</u>`,`<u>`EmplId`</u>`,Hours)`

- **Id of the employees that are allocated to at least two projects:**
  - We first **rename** the `ALLOCATION` relation:

    $$\texttt{ALLOC\_R} \leftarrow \rho_{\texttt{ALLOC\_R(CodR, IdR, HR)}}(\texttt{ALLOCATION})$$

  - We then **combine** this intermediate result with `ALLOCATION` to get all combinations of ALLOCATION pairs:

    $$\texttt{En2} \leftarrow \sigma_{(\texttt{EmplId=IdR} \wedge \texttt{PrjCode}\neq\texttt{CodR})}(\texttt{ALLOC\_R} \times \texttt{ALLOCATION})$$

Cartesian product operation

- **Yet another example:** Given the same relation schemas:

  EMPL(<u>Id</u>,Name,LastName,Salary)

  PROJECT(<u>PrjCode</u>,Description,ManagerId)

  ALLOCATION(<u>PrjCode</u>,<u>EmplId</u>,Hours)

- **Id of the employees that are allocated to at least two projects:**
  - We first **rename** the ALLOCATION relation:

    $$\mathbf{ALLOC\_R} \leftarrow \rho_{\mathbf{ALLOC\_R(CodR,\,IdR,\,HR)}}(\mathbf{ALLOCATION})$$

  - We then **combine** this intermediate result with ALLOCATION to get all combinations of ALLOCATION pairs:

    $$\mathbf{En2} \leftarrow \sigma_{\mathbf{(EmplId=IdR} \ \wedge \ \mathbf{PrjCode \neq CodR)}}(\mathbf{ALLOC\_R} \times \mathbf{ALLOCATION})$$

  - Finally, we **project** the Id of employees:

    $$\mathbf{Result} \leftarrow \pi_{\mathbf{EmplId}}(\mathbf{En2})$$

  - **Note:** This kind of queries is usually solved differently in SQL.

Join operation

- The **conditional join** is the combination of two operations: **a cartesian product and a selection.**
- Given $R(A_1, \ldots, A_m)$ and $S(B_1, \ldots, B_n)$, the **conditional join** of R and S is defined as:

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

- Although it is a derived operation, it has its own operator symbol because it is **one of the most used operations.**
  - It simplifies the queries, that can be very complex.
- If the selection condition C does not hold for any tuple, $\bowtie_C$ **produces an empty set.**
- If a tuple contains **NULL** in an attribute used in the condition, **the tuple is not included in the result.**

Join operation

- **Examples:** Given the same relation schemas:

  EMPL(Id,Name,LastName,Salary)

  PROJECT(PrjCode,Description,ManagerId)

  ALLOCATION(PrjCode,EmplId,Hours)

- **Name of the employees that are managers of projects:**

  $\pi_{\text{(Name,LastName)}}(\texttt{EMPL} \bowtie_{\texttt{(Id=ManagerId)}} \texttt{PROJECT})$

- **Name of the employees that work on a project more than 10 hours:**

  $\pi_{\texttt{(Name,LastName)}}(\texttt{ALLOCATION} \bowtie_{\texttt{(Id=EmplId} \wedge \texttt{Hours} > \texttt{10)}} \texttt{EMPL})$

- **Id of the employees that work on at least two projects:**

  $\texttt{ALLOC\_R} \leftarrow \rho_{\texttt{ALLOC\_R(CodR,IdR,HR)}}(\texttt{ALLOCATION})$

  $\texttt{Res} \leftarrow \pi_{\texttt{EmplId}}(\texttt{ALLOC\_R} \bowtie_{\texttt{(EmplId=IdR} \wedge \texttt{PrjCode} \neq \texttt{CodR)}} \texttt{ALLOCATION})$

  (This query is usually solved differently in SQL, using an *aggregation function*.)

Natural Join operation

- A very common case of conditional join is the one in which the condition is a **conjunction of equalities of those attributes with the same name in both relations.**
- **Example:** Given the relation schemas:

  EMPL(**Id**,Name,LastName,Salary)

  PROJECT(PrjCode,Description,ManagerId)

  ALLOCATION(PrjCode,**Id**,Hours)

  - **Name of the employees that work on at least one project**

    $\pi_{\textbf{(Name,LastName)}}(\textbf{ALLOCATION} \bowtie_{\textbf{(ALLOCATION.Id=EMPL.Id)}} \textbf{EMPL})$

- The result of this expression will repeat the attributes included in the condition with **exactly same value and name (from different relations).**
- We can avoid duplicate columns and condition using the **Natural Join operation.**

Natural Join operation

- A natural join is like a conditional join in which the condition is composed of **all attributes with the same name in both operands.**
- The condition in $\bowtie$ **is omitted** as it is not necessary.
- Moreover, **duplicate attributes are removed** in the result.
- Given $R(A_1, \ldots, A_m)$ and $S(B_1, \ldots, B_n)$ where the attributes with the same name in R and S are $C_1, \ldots, C_j$, the **natural join** $R \bowtie S$ produces a schema with:
  - **Schema attributes:** $\{A_1, \ldots, A_m\} \cup \{B_1, \ldots, B_n\}$ (no duplicate attributes).
  - **Relation instance:** Valid tuples are the combination of valid tuples of R and S that have the same value in $C_1, \ldots, C_j$.
- If there are no attributes with the same name in both relations, **the natural join works just like the cartesian product.**

Natural Join operation

- **Examples:** Given the following relation schemas:

  `EMPL(Id,Name,LastName,Salary)`

  `PROJECT(PrjCode,Description,ManagerId,DptId)`

  `DEPARTMENT(DptId,Name)`

- List of **all projects with the name of the department they are assigned to:**

Natural Join operation

- **Examples:** Given the following relation schemas:

  EMPL(<u>Id</u>,Name,LastName,Salary)

  PROJECT(<u>PrjCode</u>,Description,ManagerId,DptId)

  DEPARTMENT(<u>DptId</u>,Name)

- List of **all projects with the name of the department they are assigned to:**

  $\pi_{\textbf{(Description,Name)}}(\texttt{\textbf{PROJECT}} \bowtie \texttt{\textbf{DEPARTMENT}})$

- **¿What is the result of the following query?:** $\texttt{\textbf{EMPL}} \bowtie \texttt{\textbf{DEPARTMENT}}$

Natural Join operation

- **Examples:** Given the following relation schemas:

  EMPL(Id,Name,LastName,Salary)

  PROJECT(PrjCode,Description,ManagerId,DptId)

  DEPARTMENT(DptId,Name)

- List of **all projects with the name of the department they are assigned to:**

  $\pi_{\textbf{(Description,Name)}}(\texttt{\textbf{PROJECT}} \bowtie \texttt{\textbf{DEPARTMENT}})$

- **¿What is the result of the following query?:** $\texttt{\textbf{EMPL}} \bowtie \texttt{\textbf{DEPARTMENT}}$
- List of **employees that manage projects and the departments that the projects are assigned to:**

Natural Join operation

- **Examples:** Given the following relation schemas:

  `EMPL(Id,Name,LastName,Salary)`

  `PROJECT(PrjCode,Description,ManagerId,DptId)`

  `DEPARTMENT(DptId,Name)`

- List of **all projects with the name of the department they are assigned to:**

  $\pi_{\textbf{(Description,Name)}}(\texttt{PROJECT} \bowtie \texttt{DEPARTMENT})$

- **¿What is the result of the following query?:** $\texttt{EMPL} \bowtie \texttt{DEPARTMENT}$

- List of **employees that manage projects and the departments that the projects are assigned to:**

  $\texttt{Mngrs} \leftarrow \rho_{\texttt{EMPL(ManagerId,ManagerName,LastName,Salary)}}(\texttt{EMPL})$

  $\texttt{Result} \leftarrow \pi_{\texttt{(ManagerName,Name)}}((\texttt{Mngrs} \bowtie \texttt{PROJECT}) \bowtie \texttt{DEPARTMENT})$

Extensions to the Relational Algebra

- A number of additional operators have been proposed to **extend the Relational Algebra** in order to **simplify complex queries.**
- Most operations can be implemented with a basic set: $\{\cup, \setminus, \times, \sigma, \pi, \rho\}$, but the resulting queries can be **very complex.**
- For example, **intersection** could be implemented as follows:

$$R \cap S = R \setminus (R \setminus S).$$

- On the other hand, **join operators** $R \bowtie_{[C]} S$ only produce the combinations of tuples that **actually match in both** R **and** S**.**
  - Tuples in R that do not match with any tuple in S **are not included in the result,** and vice versa.
  - There are specific operators that preserve all tuples from one of the operands: they are called **outer joins.**

Outer Join operations

- The **left outer join** $R ⟕ S$ is a natural join that produces **all tuples in** R combined with those in S that match with a tuple in R.
  - Those tuples from R that do not match any tuple in S **are filled with NULL values in the attributes from** S**.**
- The **right outer join** $R ⟖ S$ is defined the same way, but producing all tuples from its right hand operand, combined with the matching tuples from its left hand operand.
- Finally, a **full outer join** $R ⟗ S$ produces all tuples in both relations, combined with the matching tuples of the other relation, or NULL if there are no matching tuples.

Outer Join operations

Example:

EMPL

EmpId	Name	LastName		Salary
37X	Juan	Sánchez	Martín	1500
24Y	Adela	García	Sanz	2300

PROJECT

PrjId	Descr.	EmpId
4	Accounting	24Y
7	Marketing	55Z

Outer Join operations

Example:

EMPL

EmpId	Name	LastName		Salary
37X	Juan	Sánchez	Martín	1500
24Y	Adela	García	Sanz	2300

PROJECT

PrjId	Descr.	EmpId
4	Accounting	24Y
7	Marketing	55Z

EMPL $\bowtie$ PROJECT

EmpId	Name	LastName		Salary	PrjId	Descr.
24Y	Adela	García	Sanz	2300	4	Accounting

Outer Join operations

Example:

EMPL

EmpId	Name	LastName		Salary
37X	Juan	Sánchez	Martín	1500
24Y	Adela	García	Sanz	2300

PROJECT

PrjId	Descr.	EmpId
4	Accounting	24Y
7	Marketing	55Z

EMPL $\bowtie$ PROJECT

EmpId	Name	LastName		Salary	PrjId	Descr.
24Y	Adela	García	Sanz	2300	4	Accounting

EMPL $⟕$ PROJECT

EmpId	Name	LastName		Salary	PrjId	Descr.
37X	Juan	Sánchez	Martín	1500	**NULL**	**NULL**
24Y	Adela	García	Sanz	2300	4	Accounting

Outer Join operations

Example:

EMPL

EmpId	**Name**	**LastName**		**Salary**
37X	Juan	Sánchez	Martín	1500
24Y	Adela	García	Sanz	2300

PROJECT

PrjId	**Descr.**	**EmpId**
4	Accounting	24Y
7	Marketing	55Z

EMPL $\bowtie$ PROJECT

EmpId	**Name**	**LastName**		**Salary**	**PrjId**	**Descr.**
24Y	Adela	García	Sanz	2300	4	Accounting

EMPL ⟕ PROJECT

EmpId	**Name**	**LastName**		**Salary**	**PrjId**	**Descr.**
37X	Juan	Sánchez	Martín	1500	**NULL**	**NULL**
24Y	Adela	García	Sanz	2300	4	Accounting

EMPL ⟖ PROJECT

EmpId	**Name**	**LastName**		**Salary**	**PrjId**	**Descr.**
24Y	Adela	García	Sanz	2300	4	Accounting
55Z	**NULL**	**NULL**	**NULL**	**NULL**	7	Marketing

Outer Join operations

Example (cont'd):

EMPL

EmpId	**Name**	**LastName**		**Salary**
37X	Juan	Sánchez	Martín	1500
24Y	Adela	García	Sanz	2300

PROJECT

PrjId	**Descr.**	**EmpId**
4	Accounting	24Y
7	Marketing	55Z

EMPL $⟗$ PROJECT

EmpId	**Name**	**LastName**		**Salary**	**PrjId**	**Descr.**
37X	Juan	Sánchez	Martín	1500	**NULL**	**NULL**
24Y	Adela	García	Sanz	2300	4	Accounting
55Z	**NULL**	**NULL**	**NULL**	**NULL**	7	Marketing

Division operation

- The **division** operation may be helpful for some DB queries.
- Produces the tuples in a relation such that some attributes take **all values** that are contained in another relation.
- Given $R(A_1, \ldots, A_m)$ and $S(B_1, \ldots, B_n)$ such that $\{B_1, \ldots, B_n\} \subset \{A_1, \ldots, A_m\}$, the operation $R \div S$ produces a relation schema with the following characteristics:
  - **Schema attributes:** $\{C_1, \ldots, C_j\} = \{A_1, \ldots, A_m\} \setminus \{B_1, \ldots, B_n\}$.
  - **Relation instance:** A tuple t is in $T = R \div S$ if $\{t\} \times S$ is in R.
  - Informally, a tuple t is in T if t appears in R with **all values contained in** S.
- This operation requires the use of **universal quantification.** DBMS do not implement such kind of operations usually.
- We will see how it works with an example:

Division operation

- **Example:** Given the following relation schemas:

  `EMPL(EmpId,Name,LastName,Salary)`

  `ALLOCATION(PrjId,EmpId,Hours)`

- List the personal info of all employees that work **in all projects that work** the employee with EmpId 8967866R:
  - We select first the projects in which that employee works:

    $\mathbf{PrjEmp} \leftarrow \pi_{\mathbf{PrjId}}(\sigma_{\mathbf{(EmpId='8667866R')}}(\mathbf{ALLOCATION}))$
  - Next, we select the EmpIds of the employees that work in every project:

    $\mathbf{EmpIdPrj} \leftarrow \pi_{\mathbf{(PrjId,EmpId)}}(\mathbf{ALLOCATION})$
  - We then obtain the EmpIds of the sought employees using the division operation:

    $\mathbf{SoughtEmpId} \leftarrow \mathbf{EmpIdPrj} \div \mathbf{PrjEmp}$
  - And finally, we obtain the personal info of those employees:

    $\mathbf{Result} \leftarrow \mathbf{SoughtEmpId} \bowtie \mathbf{EMPL}$

Handling of null values

- Any **relational operation** $(<, >, \leq, \geq, =, \neq)$ applied to a NULL value produces an **unknown result.**
- Any **Boolean operation** with an **unknown value** produces an **unknown result.**
- Therefore, Boolean conditions may return three possible values: **true**, **false**, or **unknown.**
- Algebra operations behave as follows:
- **Selection $\sigma_C(R)$:** The result contains the tuples in R for which the condition C is **true** (they are **not** included if the result is unknown).
- **Projection, union, intersection, set difference:** Null values are handles just like any other value for removing duplicate tuples.

Handling of null values

- **Conditional join, natural join:** The following equivalence is used: $R \bowtie_C S = \sigma_C(R \times S)$.
  - Tuples with a NULL value in the connection attribute (or the attributes with the same name in natural join) $\boxed{\textbf{do not match}}$ and therefore **do not appear in the result.**
- **Outer joins:**
  - Work like inner joins for tuples that **match the join condition.**
  - The tuples that **do not match the join condition** are included in the result (depending on the type of outer join) filling with NULL values.