EI1020/MT1020. Databases Academic Year 2020-21

Session 6. Queries over multiple tables: JOIN

Document version: 2021-04-25

1 Objectives

The objectives to be covered in this session are:

- To be able to identify the need for join operations (`JOIN`) and carry them out.
- To be able to properly execute joins when one or both tables involved in the operation do not have related rows.

2 Some advice for this session

As usual, first create a working directory for this session within your directory `ei1020`. Change to this directory and save each SQL query in a different file with the `.sql` extension. Use a naming system that can be employed in all sessions. For example, the file containing the first exercise of today might be named `s06_ej01.sql`.

Before writing each query, you can check in the course website the results your query should obtain, so that you can find out if you have understood the problem statement correctly.

3 In the previous session...

We practised some operators and functions that can be used in the `SELECT` and `WHERE` clauses. By using these operators and functions, expressions *about rows* were created. For example:

```
1  SELECT DISTINCT EXTRACT( month FROM fecha ) AS meses
2  FROM   facturas
3  WHERE  codcli IN ( 45, 54, 87, 102 )
4  AND    EXTRACT( year FROM CURRENT_DATE ) - 1 =
5         EXTRACT( year FROM fecha );
```

the above sentence starts from the `FACTURAS` table and then it selects *the rows that satisfy the condition in the WHERE clause*. Next, *for each selected row*, the month is extracted from the fecha value.

Describe what the above query does.

. . . . . .

. . . . . .

4 In this session . . .

We will see how to design queries involving data from multiple tables by using the `JOIN` operation.

4.1 Inner Join

Joining relations is one of the most useful operations of relational algebra and, consequently, also of the SQL language. This operation allows us to combine information from two or more tables. As you know, the join operation is not a primitive one, since it can be defined as a Cartesian product followed by a restriction. However, the DBMS implements it in a different way because it is neither necessary nor efficient, to compute the full Cartesian product and then make the restriction on it.

Let us do a short exercise:

How many rows and columns has the `CLIENTES` table? . . . . . .

How many rows and columns has the `FACTURAS` table? . . . . . .

Execute the following query:

```
SELECT *
FROM   facturas, clientes;
```

How many rows and columns are there in the result? . . . . . .

What relational algebra operation has been applied in the above sentence? . . . . . .

Although there are columns with the same name in both tables, there is no naming conflict with these columns because in SQL the column name is composed by the name of the table, followed by a period and by the name of the column (`facturas.codcli` and `clientes.codcli`). For convenience, when there is no ambiguity to refer to a column, we can omit its table name.

As we have mentioned before, a join is defined as a Cartesian product plus a constraint. If the `FROM` clause computes the Cartesian product, how do you have to modify the previous query to join the `CLIENTES` and `FACTURAS` tables?

```
SELECT *
FROM   facturas, clientes
WHERE  . . . . . .
```

Does each row in the result represent a customer or an invoice? . . . . . .

Have you got as many rows as invoices are in the database or as many as customers are in the database? . . . . . .

Why? . . . . . .

Execute the following query, which retrieves invoice data and customer name of invoices with

VAT (IVA) equal to 16 % and without any discount:

```
SELECT facturas.codfac, facturas.fecha, facturas.codcli,
       clientes.nombre, facturas.codven
FROM   facturas, clientes                    -\
WHERE  facturas.codcli = clientes.codcli     --\_ join
AND    facturas.iva = 16                     ---- restriction
AND    COALESCE( facturas.dto, 0 ) = 0;      ---- restriction
```

The query you have just executed uses a syntax that is deprecated in the SQL standard. The syntax of the new standard is more desirable because it allows us to identify more clearly what are restrictions (they appear in the `WHERE` clause) and what are join conditions (they appear in the `FROM` clause in the `JOIN` operator). The following query reformulates the previous one using this syntax:

```
SELECT codfac, fecha, codcli, nombre, codven
FROM   facturas NATURAL JOIN clientes      -- join
WHERE  iva = 16                            -- restriction
AND    COALESCE( dto, 0 ) = 0;             -- restriction
```

Note that the column names in the `WHERE` and `SELECT` clauses are not preceded by the name of the table to which they belong because no ambiguity is possible in this case. The `NATURAL JOIN` operator is the `JOIN` operator of relational algebra: It joins rows with equal values in the columns that have the same name in both tables (in this example the `codcli` attribute). In the result, the columns used to join the tables appear only once (and hence there is no ambiguity in referring to `codcli` in the `SELECT` clause).

The use of `NATURAL JOIN` makes the query dependent on the data structure (as the join is done based on the attributes with the same name). If you change the table schema (attributes are added, removed or the name of some of them are changed) the query may malfunction or may not show the desired result. For this reason, instead of the SQL `NATURAL JOIN` operator, it is recommended to use better alternatives which are described below.

When several tables that have columns with the same name are joined, and we do not want all these columns to be taken into account in the join, the `INNER JOIN` operator can be used. This operator indicates the join condition with the `USING` or `ON` keywords. Let us see an example:

```
SELECT facturas.codfac, facturas.fecha, facturas.codcli,
       clientes.nombre, facturas.codven, vendedores.nombre
FROM   facturas
       INNER JOIN clientes USING ( codcli )
       INNER JOIN vendedores USING ( codven )
WHERE  facturas.iva = 16
AND    COALESCE( facturas.dto, 0 ) = 0;
```

Since this form of join is the most common, the `INNER` keyword can be omitted. When the columns used to join both tables do not have the same name, the `ON` keyword must be used to specify the join condition, as shown in the following example. This example also introduces the use of table aliases, which allow us to abbreviate the full name of a table when referring to its columns:

```
SELECT f.codfac, f.fecha, f.codcli, c.nombre, f.codven, v.nombre
FROM   facturas AS f
       JOIN clientes AS c ON ( f.codcli = c.codcli )
       JOIN vendedores AS v ON ( f.codven = v.codven )
WHERE  f.iva = 16
```

```
6   AND     COALESCE( f.dto, 0 ) = 0;
```

With the ON keyword it is allowed to use any kind of condition using columns from both tables with any of the relational operators: <, >, <=, >=, <> and =. With the USING keyword, only the name of a column in common in both tables is used.

It is strongly recommended that, when building joins, you specify the tables in the same order as they appear in the reference diagram:

In this way, it will be easier to debug statements and identify what each one of them does: in the result of a query written in this way, **each row will represent the same thing that represents each row of the table that appears first in the FROM clause** and the result will have, at most, as many rows as rows are in that table.

Note that the CLIENTES and the VENDEDORES tables have a foreign key with the same name, both referring to the PUEBLOS table. Therefore, if the three tables take part in a JOIN, there might be ambiguities if the USING keyword is used. If this is the case, the ON keyword must be used exactly to indicate what attributes should be used in the join condition and thus avoid ambiguity.

To better understand the problem, try to write a SELECT query to display the code of invoices, the customer name and her/his town, and the salesperson name and her/his town.

.....

.....

.....

4.2 Outer Join

Usually we will join tables using foreign keys (though it does not always have to be the case). Then, each row of the table with a foreign key will be joined with the row of the referenced table that has in the primary key the same value as the foreign key.

However, in some situations foreign keys accept null values. The question is, what rows will be joined with the rows with null value in the foreign key?. And, in general, what will happen when two tables are joined and there are null values in the column used to do the join? The answer is that these rows will be lost. The following example shows this situation: when we join the FACTURAS and CLIENTES tables using the foreign key codcli, the invoices with null value in the codcli column will not appear in the result.

```
1   SELECT f.codfac, f.fecha, f.codcli, c.nombre
2   FROM   facturas AS f JOIN clientes AS c USING ( codcli );
```

When a JOIN is done, the rows without any correspondence in the other table will also disappear. In the previous example, when we join the FACTURAS and CLIENTES tables using the foreign key,

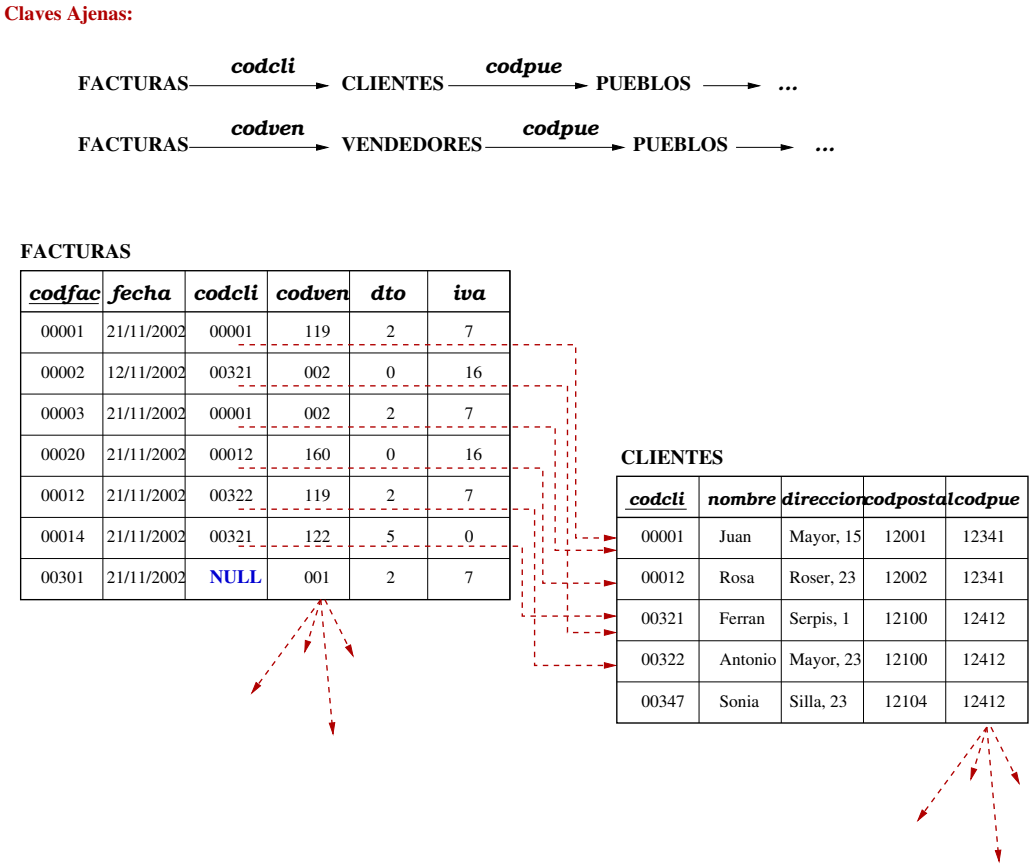

Figure 1: Foreign keys for the `FACTURAS` and `CLIENTES` tables

customers without any invoice are lost. The result of executing the above join, shown in figure 1, does not include invoice `00301` nor customer `00347`.

Now, suppose we want to obtain a list containing all invoices for the month of December last year. For each invoice, the names of the customer and the salesperson have to be shown. We write the following query:

```
SELECT f.codfac, f.fecha, f.codcli, c.nombre, f.codven, v.nombre
FROM   facturas AS f
       JOIN clientes AS c USING ( codcli )
       JOIN vendedores AS v USING ( codven )
WHERE  EXTRACT( month FROM f.fecha ) = 12
AND    EXTRACT( year FROM f.fecha ) =
       EXTRACT( year FROM CURRENT_DATE ) - 1 ;
```

In the result there are some invoices of December last year that do not appear in the result. Why?

.....

To avoid the loss of rows when joining tables, we could use `OUTER JOIN` clause, which has the following three variants: `LEFT`, `RIGHT` and `FULL`. Using `LEFT/RIGHT OUTER JOIN` the result shows all table rows of LEFT/RIGHT table, that is, no rows are lost. Those rows on the left/right table that can be paired with a row of the other table will be joined using (`INNER`) `JOIN`; the rest of the rows will appear in the results joined with a null value row. Using `FULL OUTER JOIN` both operations

are done: `LEFT OUTER JOIN` and `RIGHT OUTER JOIN`. So, no rows are lost in any of the tables used in the join.

Since in the example presented above, we have lost December last year invoices because the foreign keys `codcli` and `codven` accept null values, the correct query will be:

```
SELECT f.codfac, f.fecha, f.codcli, c.nombre, f.codven, v.nombre
FROM   facturas AS f
       LEFT OUTER JOIN clientes AS c USING ( codcli )
       LEFT OUTER JOIN vendedores AS v USING ( codven )
WHERE  EXTRACT( month FROM f.fecha ) = 12
AND    EXTRACT( year FROM f.fecha ) =
       EXTRACT( year FROM CURRENT_DATE ) - 1 ;
```

Look at the following query:

```
SELECT f.codfac, f.fecha, c.codcli, c.nombre
FROM   facturas AS f
       RIGHT OUTER JOIN clientes AS c USING ( codcli )
       JOIN pueblos AS pu USING ( codpue )
WHERE  pu.codpro = '12'
ORDER  BY 3 DESC;
```

What does this query do?

.....

Should the second `JOIN` be a `RIGHT OUTER JOIN`?

.....

Let us look at the last example:

```
SELECT f.codfac, f.dto, v.codven, v.nombre
FROM   facturas AS f
       FULL OUTER JOIN vendedores AS v USING (codven)
WHERE  COALESCE( f.dto, 0 ) <> 0
ORDER  BY 3 DESC;
```

What does the above query do?

.....

5 Examples

1. Full data of the customer which appears in the 5886 invoice.

   The query using `JOIN` is the following:

   ```
   SELECT c.*
   ```

```
FROM   facturas AS f JOIN clientes AS c USING ( codcli )
WHERE  f.codfac = 5886;
```

2. Code of invoices in which the article that currently has the highest price has been sold.

   In the following statement a subquery has been used to do the restriction:

```
SELECT DISTINCT l.codfac
FROM   lineas_fac AS l JOIN articulos AS a USING ( codart )
WHERE  a.precio =  ( SELECT MAX( precio )
                     FROM articulos ) ;
```

3. For each salesperson in the Castellón province, show her/his name and the name of her/his manager.

```
SELECT emp.codven, emp.nombre AS empleado, jef.nombre AS jefe
FROM   vendedores AS emp
       LEFT OUTER JOIN vendedores AS jef ON ( emp.codjefe = jef.
          codven )
       JOIN pueblos AS pue ON ( emp.codpue = pue.codpue )
WHERE  pue.codpro = '12';
```

   Explain if it is necessary to use `LEFT OUTER JOIN`

   .....

   .....

   Is it possible to write the `JOIN pueblos` part with `USING` instead of `ON`?

   .....

   .....

4. We want to show the number of invoices issued for each customer. The query result must show the customer name and must include the invoices without a customer.

   The following considerations should be made:

   - First of all, we must think about the tables to be used in the query, which have to appear in the `FROM` clause. Note that the required data are included in the `FACTURAS` and `CLIENTES` tables.
   - Each invoice has to be joined with the customer data. The join has a row for each invoice and the result must show a row for each customer that matches with the required restrictions. In order to keep invoices without a customer, an `OUTER JOIN` must be used.
   - There are no restrictions.
   - To be able to count, we must use the `GROUP BY` clause. Because the count has to be done for each customer, and because we want to show columns `codcli` and `nombre` in the result, the grouping will be done using these columns.
   - Because in the table used in the `FROM` clause we have a row for each invoice, and because we want to count invoices, we could use `COUNT(*)`.
   - Finally, in the `SELECT` clause we choose the code of each customer, her/his name and the number of invoices.

```
SELECT codcli, nombre,
       COUNT( * ) AS facturas
FROM   facturas LEFT OUTER JOIN clientes USING ( codcli )
GROUP  BY codcli, nombre;
```

Given that the name could have a null value (invoices without customers in the join are joined with a null row), the function `COALESCE` could be used to show a message indicating it.

```
SELECT codcli, COALESCE( nombre, '-Sin Cliente-' ) AS nombre,
       COUNT( * ) AS facturas
FROM   facturas LEFT OUTER JOIN clientes USING ( codcli )
GROUP BY codcli, nombre;
```

6 Session exercices

The exercises in this section have to be solved individually. On the course website, you can check if the statements that you have written obtain the expected results.

1. Show data from customers' invoices in the Castellón province showing also the customer name. Order the result by customer code and, if a customer has several invoices, order them by date.

2. Show the invoice lines that have been made by sellers from Viver along with the code and the name of the salesperson. Order the invoices and their lines.

3. For each customer of the *Comunidad Valenciana*, show her/his name and full address (with the name of the town and the province). Order the result by customer. The first letter of each word of the full address has to be capitalized, but all the province name must be capitalized. Check the information in the course website to see the required format for the results.

4. Show data from the invoices in which a switch (*interruptor*) has been sold. Order the invoices.

5. Show the name of the customers who live in towns from Madrid [1]. You have to do the query using the name of the province, not using its code.

6. Show the name of the towns from Castellón in which there are customers. Order the results by town name.

7. Show the invoice code, date, and code and name of the customers whose name begins with "A". Customers without invoices must also be shown. Order the result by invoice code.

8. Modify the previous exercise to also show the invoices without any customer.

9. For each article, show the customers' codes who have bought it. Order the results by the article description in descending order. Note that there are several articles that were never sold and that these articles must also be shown.

10. Show, for all towns in Castellón, the customers that are in each one. Towns without customers must also be shown. Order the result by town name and customer code.

[1] Though not explicitly stated, you must also show the customer code, since it is the primary key. Otherwise, what will happen if two customers from Madrid have the same name? Thereafter, you will see that in the query results the primary keys will be always shown, although not explicitly requested in the exercise statement.

11. Show, for each salesperson, the date of the invoices done in the past year with a discount of 20 %. Sort the result by salesperson code and invoice date.

12. Modify the previous exercise to also show the information of invoices without any salesperson. For these invoices, the salesperson name has to be '-Sin vendedor-'.

13. Given the following SQL query, describe in natural language what it does.

```
SELECT l1.codart, l1.precio, f1.fecha, l2.precio, f2.fecha
FROM   lineas_fac AS l1
       JOIN facturas AS f1 ON ( l1.codfac = f1.codfac )
       JOIN lineas_fac AS l2 USING ( codart )
       JOIN facturas AS f2 ON ( l2.codfac = f2.codfac )
WHERE  ( l1.precio * 1.1 ) < l2.precio
AND    f1.fecha <= f2.fecha;
```

6.1 Supplementary Exercises

Show the name of the towns from Castellón whose name is the same as other towns from another province.

Suppose that we have the following towns (the name of the town and the code of the province are shown):

```
Pueblos:
Castellon 12
Villareal 12
Castellon 15
Sagrillas 19
Almazora  12
Almazora  23
```

To be able to compare the name of each town with the rest of the towns we should use the `JOIN` operator between `pueblos` table (p1) and a "copy" of itself (p2), joining those towns that have the same name.

```
p1:               p2:
Castellon 12      Castellon 12
Villareal 12      Villareal 12
Castellon 15      Castellon 15
Sagrillas 19      Sagrillas 19
Almazora  12      Almazora  12
Almazora  23      Almazora  23
```

To do that, in the `FROM` clause we should write:

```
FROM   pueblos AS p1 JOIN pueblos AS p2 USING ( nombre )
```

The result would be:

```
Castellon 12        Castellon 12
Castellon 12        Castellon 15
Villareal 12        Villareal 12
Castellon 15        Castellon 15
Castellon 15        Castellon 12
Sagrillas 19        Sagrillas 19
Almazora  12        Almazora  12
Almazora  12        Almazora  23
Almazora  23        Almazora  23
Almazora  23        Almazora  12
```

Note that each town is always joined with itself (a town has the same name as itself), and possibly with others of the same name from other provinces.

Complete the query to obtain what is required in this exercise.

7 What you do not have to forget

- Ordering the tables in the `FROM` clause as they appear in the reference diagram helps us to verify the behaviour of the query every time:
  - We can get to know if we have forgotten to include any intermediate table.
  - We can get to know what each row represents in the result of the join.
  - It will be easier to decide what we have to include in the `COUNT()` function, when needed.
  - It will be easier to determine if it is necessary to use `DISTINCT` in the final projection, `SELECT`
- Throughout the life-cycle of a database, it is possible that several columns are added to a table to store more information. If this table has been used in a `NATURAL JOIN` in any query, we must be careful when choosing the column name because, if the new column has the same name as another column in another table, the join operation will no longer have the same results. You can avoid these problems by using always (`INNER`) `JOIN` (with `ON` or `USING`), which requires to specify the columns to perform the join. Then, although new columns are added to the tables, the operation performed will not change even if there exist columns with the same names in both tables.
- With the `ON` keyword it is allowed to use any kind of condition using columns from both tables with any of the relational operators: <, >, <=, >=, <> and =. With `USING` keyword only the name of a column in common in both tables is used, the condition to join rows in this case is the equality.
- To join tables using the `JOIN` clause, you have to keep in mind what the purpose of the join is, and if, when we do the join, some rows could be lost and we want to show these rows, we have to use the `OUTER JOIN`