Unit 2:
SQL

2.1. DML: Queries and Data Manipulation
2.2. SQL Exercises (Lab. sessions)

UD 2.1 DML: Queries and Data Manipulation

1. Introduction to SQL
2. Queries
   - 2.1. Simple queries using one table.
   - 2.2. Simple queries with several tables
   - 2.3. Subqueries
   - 2.4. Quantified comparison predicates
   - 2.5. Grouping
   - 2.6. Set operations
   - 2.7. Joins
3. Database updates
4. Commands for handling transactions

1. Introduction to SQL

SQL (**S**tructured **Q**uery **L**anguage) is a standard language for defining and manipulating a relational database.

Includes:

- Features from Relational Algebra (Algebraic Approach)
- Features from Tuple Relational Calculus (Logical Approach)
- Others

- SQL has evolved: SQL'92, SQL'99, SQL:2003, SQL:2008, SQL:2011
- We will use the basics of the language (present from SQL'92)

SQL sublanguages

DDL (Data Definition Language): Creation and modification of relational DB schemas.

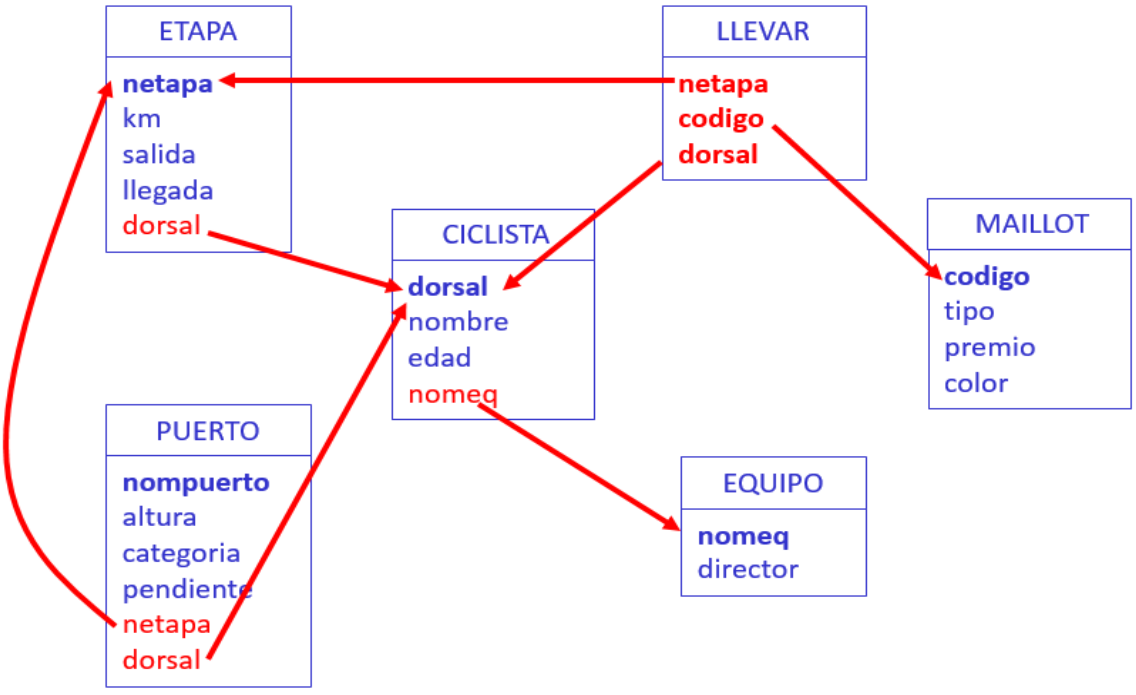
DML (Data Manipulation Language): Queries and database updates.

- SELECT: Allows the declaration of queries to retrieve the information from the database
- INSERT: Performs the insertion of one or more rows in a table
- DELETE: Allows the user to delete one o more rows from a table
- UPDATE: Modifies the values of one or more columns and/or one or more rows in a table

Control Language: Dynamically changes the database properties

```
EQUIPO(nomeq: char(25), director: char(30))
        CP:{nomeq}
CICLISTA(dorsal: entero, nombre: char(30), edad: entero, nomeq: char(25))
        CP:{dorsal}
        CAj:{nomeq}→ EQUIPO
        VNN:{nomeq}
        VNN:{nombre}
ETAPA(netapa: entero, km: entero, salida: char(35), llegada: char(35),
                        dorsal: entero)
        CP:{netapa}
        CAj:{dorsal}→ CICLISTA
MAILLOT(codigo: char(3), tipo: char(30), premio: entero, color: char(25))
        CP:{codigo}
PUERTO(nompuerto: char(30), altura: entero, categoria: char(1),
                        pendiente: real, netapa: entero, dorsal: entero)
        CP:{nompuerto}
        CAj:{netapa}→ ETAPA
        CAj:{dorsal}→ CICLISTA
        VNN:{netapa}
LLEVAR(dorsal: entero, netapa: entero, codigo: char(3))
        CP:{netapa,codigo}
        CAj:{netapa}→ ETAPA
        CAj:{dorsal}→ CICLISTA
        CAj:{codigo}→ MAILLOT
        VNN:{dorsal}
```


Ciclismo (Cycling race)



1. Introduction to SQL
2. Queries
2.1. Simple queries using one table.
2.2. Simple queries with several tables
2.3. Subqueries
2.4. Quantified comparison predicates
2.5. Grouping
2.6. Set operations
2.7. Joins
3. Database updates
4. Commands for handling transactions

EXAMPLE: List the name and the age of all the cyclists.

```
SELECT nombre, edad
FROM Ciclista;
```

Syntax

select [all | distinct] {*expression, expressicyclistsression*$_n$} | *

from table

[where *conditional_expression*]

[order by {*column*$_1$, *column*$_2$, … *column*$_m$}]

- all : Allows identical rows to appear in the result (default value)
- distinct: Doesn't allow repeated rows in the result
- *conditional_expression* expresses a condition that the recovered rows must fulfill. It can be composed of several predicates joined by AND, OR, or NOT

- Comparison predicates: =, <>, >, <, >=, <=.
- like: Allows the comparison of a string with a pattern
- between: Allows checking whether a number is within a range.
- in: Allows checking whether the value in within a set.
- is null: Allows checking whether the value is null.

Mathematics operations

+, –, *, /, etc.

EXAMPLE:

List for each maillot its type and the prize in euros (suppose that the prizes are in pesetas and 1€ = 166 ptas.) but only for those maillots which prize is greater than 100 euros.

```
SELECT tipo, premio / 166
FROM Maillot
WHERE premio / 166 > 100;
```

EXAMPLE:
List the name, the height and the category of all the mountain pass order by height and category.

```
SELECT nompuerto, altura, categoria
FROM Puerto
ORDER BY altura, categoria ;
```


EXAMPLE: List the name and the height of all the mountain passes in the1ª category in ascending order according to the height

1. Which tables contain the information?
2. What are the conditions for the tuples?
3. What information is going to be returned?
4. Is any order required?

```
SELECT nompuerto, altura
FROM Puerto
WHERE categoria = ´1´
ORDER BY altura ;
```

EXAMPLE:

List the teams of the ciclists (only the team).

```
SELECT DISTINCT team
FROM Ciclista;
```

EXAMPLE:

List all the information in the Equipos table.

```
SELECT  *
FROM Equipo;
```


EXAMPLE:

List the stage numbers where the arrival city name has, as second letter a “o”, or where the departure city name contains two or more ‘a’s

```
SELECT netapa
FROM Etapa
WHERE llegada LIKE ‘_o%’ OR salida LIKE ‘%a%a%’
```

LIKE: Is followed by a pattern which will be used with strings

EXAMPLE:
Obtain the name and age of the cyclists who belong to the teams whose name contains the string “100%”.

```
SELECT nombre, edad
FROM Ciclista
WHERE nomeq LIKE ‘%100\%%’ ESCAPE ‘\’
```


Wrong query (sintax error)

```
select nomeq
from Equipo
where  director = null
```

The right query is:

```
select nomeq
from Equipo
where  director IS NULL
```

Using the NULL VALUE

A α B (where α is a comparison operator) is evaluated as undefined if at least one A or B is null; otherwise it is evaluated to the certainty value of the comparison A α B

Example:

```
SELECT *
FROM T
WHERE atrib1 > atrib2
```

If a tuple has $atrib_1$ = 50 and $atrib_2$ is null, the comparison is undefined, and therefore that tuple will not be included in the query result.

Aggregated values (non-gruped queries)

{ avg | max | min | sum | count } ([all | distinct] *expresión_escalar*) | count(*)

- Distinct is used to remove the duplicate values before the aggregated function calculate the result.
- The NULL values are removed before the aggregated function calculate the result.
- If the number of selected rows is 0, COUNT returns 0, and the rest of the functions return the NULL value.

EXAMPLE:

SELECT 'Num. ciclysts=', COUNT(*), 'average age=', AVG(edad)

FROM Ciclista

WHERE nomeq = 'Banesto';

In non-grouped queries, the SELECT clause can only include references to aggregated functions or literals, since the functions will return just a single value.

WRONG EXAMPLE:

SELECT nombre, AVG(edad)

FROM Ciclista

WHERE nomeq = 'ONCE';

Alias and DESCending order

Name and age of the 'Banesto' cyclists ordered from oldest to youngest. The column name must be "Banesto"

```
SELECT nombre AS Banesto, edad
FROM ciclista
WHERE nomeq= 'Banesto'
ORDER BY edad DESC;
```

1. Introduction to SQL
2. Queries
    2.1. Simple queries using one table.
    2.2. Simple queries with several tables
    2.3. Subqueries
    2.4. Quantified comparison predicates
    2.5. Grouping
    2.6. Set operations
    2.7. Joins
3. Database updates
4. Commands for handling transactions

If the information required by the query is in several tables, the query will include those tables in the FROM clause.

A query over several tables corresponds to the Cartesian product:

- If we do not express several conditions to connect them, the number of rows will be very high.
- If there are foreign keys defined, it is usual that some conditions are formed by an equality between the foreign key and the corresponding attributes in the table to which it refers.
- With n tables, we will typically have (at least) n-1 connections. (The way in which tables are connected and the attributes that are used determine the meaning of the query.)

Example: SELECT * FROM T1, T2 WHERE T1.n = T2.n

T1

t1	n
a1	b1
a2	b2
a3	b3

T2

t2	n
c1	d1
c2	b2

T1 x T2

t1	n	t2	n
~~a1~~	~~b1~~	~~c1~~	~~d1~~
~~a1~~	~~b1~~	~~c2~~	~~b2~~
~~a2~~	~~b2~~	~~c1~~	~~d1~~
a2	b2	c2	b2
~~a3~~	~~b3~~	~~c1~~	~~d1~~
~~a3~~	~~b3~~	~~c2~~	~~b2~~

EXAMPLE: List pairs stage-mountain pass which have been won by the same cyclist.

1. Which tables contain the information?

   FROM Etapa, Puerto

2. Which rows must be selected?

   WHERE etapa.dorsal = puerto.dorsal;

3. What attributes are going to be returned?

   SELECT etapa.netapa, nompuerto

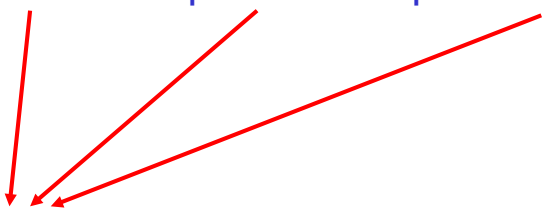
Note that the columns *dorsal* from *Etapa* and dorsal from *Puerto* is prefixed by the table name to avoid ambiguity:

```
SELECT etapa.netapa, nompuerto
FROM Etapa, Puerto
WHERE etapa.dorsal = puerto.dorsal ;
```

Alias

EXAMPLE: Obtain the names of the cyclists who belong to the team coached by 'Alvaro Pino'

```
SELECT C.nombre
FROM Ciclista C, Equipo E
WHERE C.nomeq = E.nomeq AND E.director = 'Alvaro Pino';
```



The alias can be used to refer any table:

Syntax: **FROM** table **[AS]** *alias*

EXAMPLE:

Obtain the name of the cyclists and the stage number such that the cyclist has won that stage. Additionally, the stage must be more than 150 km. long

```
SELECT C.nombre, E.netapa
FROM Ciclista C, Etapa E
WHERE C.dorsal = E.dorsal AND E.km > 150;
```

EXAMPLE: List the names of the cyclists who belongs to the same team as 'Miguel Induráin' but who are younger than he is

1. Which tables contain the information?

FROM Ciclista

But we need to compare tuples of Ciclista with tuples of the same table, so we use this **table twice**

FROM Ciclista C1, Ciclista C2

2. Which rows must be selected?

WHERE C2.nombre='Miguel Induráin' AND C1.nomeq = C2.nomeq AND C1.edad < C2.edad;

3. What attributes are going to be returned?

SELECT DISTINCT C1.nombre

==> **SELECT DISTINCT** C1.nombre **FROM** Ciclista C1, Ciclista C2
WHERE C2.nombre='Miguel Induráin'
AND C1.nomeq = C2.nomeq AND C1.edad < C2.edad;

29

UD 2.1 DML: Queries and Data Manipulation

1. Introduction to SQL
2. Queries
    - 2.1. Simple queries using one table.
    - 2.2. Simple queries with several tables
    - **2.3. Subqueries**
    - 2.4. Quantified comparison predicates
    - 2.5. Grouping
    - 2.6. Set operations
    - 2.7. Joins
3. Database updates
4. Commands for handling transactions

What is a subquery ?

A subquery is a query between parenthesis included inside other query.


```
SELECT DISTINCT E.netapa, km
FROM Etapa E, Puerto P
WHERE E.netapa = P.netapa
```

Using a subquery:

```
SELECT netapa, km
FROM Etapa
WHERE netapa IN (SELECT netapa
                 FROM Puerto)
```

Main query

The subquery returns the number of the stages

EXAMPLE: Obtain the names of the cyclists who belong to the team coached by ‘Alvaro Pino’.

Solved through equalities:

```
SELECT C.nombre
FROM Ciclista C, Equipo E
WHERE C.nomeq = E.nomeq AND E.director = ‘Alvaro Pino’;
```

Using a subquery:

```
SELECT C.nombre
FROM Ciclista C
WHERE C.nomeq = ( SELECT E.nomeq FROM Equipo E
                  WHERE E.director = ‘Alvaro Pino’ );
```

The name of the cyclist is not in the table used in the subquery (Equipo)

Name of the team directed by ‘Alvaro Pino’

This is possible only if the subquery returns one single value

Predicates that accept subqueries

Subqueries can appear in the search conditions, either in the “where” or in the “having” clauses, as arguments of some predicates:

- Comparison predicates: **=**, **<>**, **>**, **<**, **>=**, **<=.**
- **IN**: Cheks that a value belongs to the collection (table) returned by the subquery
- **EXISTS**: It is equivalent to the existential quantifier. It checks if a subquery returns some row.

Comparison Predicates: =, <>, >, <, >=, <=

SYNTAX:

row_constructor ***comparison predicate*** *row_constructor*

"row_constructor" is either a sequence of constants or a subquery.

```
('Álvaro Pino', 28) <=   (SELECT nombre , edad
                          FROM ciclista
                          WHERE dorsal= '666')
```

When row constructor returns more than one column, the lexicographic order will be used in the comparison of each operator.

For simplicity, we will only see queries with one column in the subquery.

Subqueries can be a parameter of a comparison if (and only if):

- Return only a single row, and
- the number of columns match (in number and type) with the other side of the comparison predicate.

If the result of the subquery is empty, the row is converted into a row with NULL values in all its columns and the result of the comparison will be undefined.

EXAMPLE: Obtain the name of the mountain passes whose height is greater than the mean of the height of all 2nd category mountain passes.

1. Which tables contain the information?

Puerto ==> FROM Puerto

2. Which rows must be selected?

altura > AVG(altura) of the second category mountain passes

It is 1 value = 1 row

==> WHERE altura > (SELECT AVG(altura) FROM Puerto
WHERE categoria = '2');

Check any height (altura) with the value returned by AVG(altura)

EXAMPLE: Obtain the name of the mountain passes whose height is greater than the mean of the height of all 2nd category mountain passes.

1. Which tables contain the information?

Puerto ==> FROM Puerto

2. Which rows must be selected?

altura > AVG(altura) of the second category mountain passes

==> WHERE altura > (SELECT AVG(altura) FROM Puerto
WHERE categoria = '2');

3. What attributes are going to be returned?

nompuerto ==> SELECT nompuerto *==> 1 column with n rows*

==> SELECT nompuerto FROM Puerto
WHERE altura > (SELECT AVG(altura) FROM Puerto
WHERE categoria = '2');

EXAMPLE: Obtain the name of the mountain passes whose height is greater than the mean of the height of all 2nd category mountain passes.

```
SELECT nompuerto FROM Puerto
WHERE altura >   (  SELECT AVG(altura)
                    FROM Puerto
                    WHERE categoria = '2' );
```


EXAMPLE: Obtain the name of the mountain passes whose height is greater than the mean of the height of all 2nd category mountain passes.

```
SELECT nompuerto FROM Puerto
WHERE altura > (SELECT altura FROM Puerto
                WHERE categoria = '2' );
```

1 value (1 row)

1 column with n rows

==> It can't be checked !

WRONG: (execution error).

EXAMPLE: Obtain the name of the mountain passes whose height is greater than the mean of the height of all 2nd category mountain passes.

WRONG: (Execution error):

```
SELECT nompuerto
FROM Puerto
WHERE altura > AVG (SELECT altura FROM Puerto
                    WHERE categoría = '2' );
```


EXAMPLE: List the name of the departure and the arrival cities of the stages where the steepest mountain passes ("puerto") are located.

```
SELECT DISTINCT E.salida, E.llegada
FROM Etapa E, Puerto P
WHERE E.netapa = P.netapa
    AND pendiente = (SELECT MAX(pendiente)
                     FROM Puerto) ;
```

IN Predicate

SYNTAX:

row_constructor [NOT] IN (*table_expression*)

EXAMPLE:
Obtain the "netapa" of the stages which have been won by cyclists whose age is greater than 30 years.

```
SELECT netapa
FROM Etapa
WHERE dorsal IN ( SELECT dorsal
                  FROM Ciclista
                  WHERE edad > 30);
```


Subqueries and DISTINCT

EXAMPLE: Obtain the coach of the teams which have one or more cyclists with a name beginning with 'A'.

```
SELECT director
FROM Equipo
WHERE nomeq IN
        (SELECT nomeq FROM Ciclista
         WHERE nombre LIKE 'A%'));
```

We don't need the DISTINCT (it's not harmful to include it)

But …

```
SELECT DISTINCT Q.director
FROM Ciclista C, Equipo Q
WHERE C.nomeq = Q.nomeq
    AND C.nombre LIKE 'A%'));
```

Nested Queries

EXAMPLE:

Obtain the stage number won by cyclists who belong to teams whose coach (director) has a name beginning with 'A'.

```
SELECT netapa FROM Etapa
WHERE dorsal IN
        (SELECT dorsal FROM Ciclista
         WHERE nomeq IN (SELECT nomeq FROM Equipo
                         WHERE director LIKE 'A%'));
```


EXISTS Predicate

Syntax:

EXISTS (*table_expression*)

- The exists predicate is evaluated to true if the expression SELECT returns at least one row.

- In general, IN and EXISTS are interchangeable and, when there is no negation, they can be eliminated (creating queries using multiple tables and adding comparison with the foreign keys)*

(*) That is also true for "NOT IN" and "NOT EXISTS"

EXAMPLE: Obtain the name of the cyclists who has worn a maillot with a prize lower than 8000 eur.

```
SELECT C.nombre FROM Ciclista C
WHERE EXISTS ( SELECT *
                FROM Maillot M, Llevar L
                WHERE M.premio < 8000 AND M.codigo = L.codigo
                    AND C.dorsal = L.dorsal ) ;
```

Also:

```
SELECT C.nombre FROM Ciclista C
WHERE 0 < ( SELECT COUNT(*)
                FROM Maillot M, Llevar L
                WHERE M.premio < 8000 AND M.codigo = L.codigo
                    AND C.dorsal = L.dorsal );
```


EXAMPLE: Obtain the name of the cyclists who has worn a maillot with a prize lower than 8000 eur.

```
SELECT C.nombre FROM Ciclista C
    WHERE  C.dorsal IN ( SELECT L.dorsal
                        FROM Llevar L, Maillot M
                        WHERE M.premio < 8000
                            AND L.codigo = M.codigo ) ;
```

Also:

```
SELECT DISTINCT C.nombre
FROM Ciclista C, Llevar L, Maillot M
WHERE  C.dorsal = L.dorsal
      AND L.codigo = M.codigo
      AND M.premio < 8000 );
```

EXAMPLE: Obtain the name of the cyclists who haven't won any stage.

```
SELECT nombre
FROM Ciclista C
WHERE NOT EXISTS (SELECT *
                  FROM Etapa E
                  WHERE E.dorsal = C.dorsal);
```

WHERE NOT EXISTS (SELECT * FROM …)

Is equivalent to: WHERE 0 = (SELECT COUNT(*) FROM …)

EXAMPLE: Obtain the name of the cyclists who haven't won any stage.

```
SELECT nombre
FROM Ciclista C
WHERE dorsal NOT IN (SELECT dorsal
                     FROM Etapa E
                     WHERE E.dorsal IS NOT NULL);
```

~~SELECT nombre~~
~~FROM Ciclista C, Etapa E~~
~~WHERE C.dorsal <> E.dorsal ;~~

No sense

Universal Quantification

SQL'92 and most DBMS nowadays do not provide the universal quantification (FORALL). We must transform the query to solve it with an EXISTS:

$$\forall X\ F(X) \equiv \neg\exists X \neg F(X)$$

EXAMPLE:

"Obtain the name of the cyclists (if any) who have won all the stages with more than 200 km.

The query is converted into:

"Obtain the name of the cyclists such that there does not exist a stage with more than 200 km which has not be won by that cyclist"

(*) Assuming that we know that there are some stage with more than 200 km

Obtain the name of the cyclists such that there does not exist a stage with more than 200 km which has not be won by that cyclist

```
SELECT nombre FROM Ciclista C
WHERE NOT EXISTS ( SELECT *
                   FROM Etapa E
                   WHERE km > 200 AND
                         C.dorsal <> E.dorsal );
```

Assuming that we know that there are some stage with more than 200 km

Problem: What happens if there is no stage with more than 200 km?

```
SELECT nombre FROM Ciclista C
WHERE NOT EXISTS ( SELECT *
                   FROM Etapa E
                   WHERE km > 200 AND C.dorsal <> E.dorsal );
```

In that case, this query returns the name of all the cyclists !!!

Solution:

Check that exists at least one stage with more tan 200 Km (ADD-ON).


```
SELECT nombre FROM Ciclista C
WHERE NOT EXISTS (SELECT * FROM Etapa E
                  WHERE km > 200 AND
                        C.dorsal <> E.dorsal )
      AND EXISTS ( SELECT *
                   FROM Etapa E
                   WHERE km > 200)        ;
```

EXAMPLE:
List the cyclists who have worn all the (kinds of) maillots.

```
SELECT C.nombre FROM Ciclista C
WHERE NOT EXISTS
                (SELECT * FROM Maillot M E
                 WHERE NOT EXISTS
                                (SELECT * FROM Llevar L
                                WHERE C.dorsal = L.dorsal )
                                    AND L.código = M.código )
        AND EXISTS ( SELECT *   FROM Maillot )
```


EXAMPLE: List the name of all the cyclist who have won all the mountain passes in some stage and have won that stage

```
SELECT C.nombre FROM Ciclista C, Etapa E
WHERE E.dorsal = C.dorsal
   AND NOT EXISTS ( SELECT *  FROM Puerto P
                              WHERE P.netapa = E.netapa
                                AND    C.dorsal <> P.dorsal );

 AND EXISTS ( SELECT *  FROM Puerto P
                        WHERE P.netapa = E.netapa );
```

Because there could be some stage with no mountain passes

1. Introduction to SQL
2. Queries
    2.1. Simple queries using one table.
    2.2. Simple queries with several tables
    2.3. Subqueries
    2.4. Quantified comparison predicates
    2.5. Grouping
    2.6. Set operations
    2.7. Joins
3. Database updates
4. Commands for handling transactions

SYNTAX:

row_constructor { *ALL* | *ANY* | *SOME* } (*table_expression*)

- The comparison predicate which is quantified with ALL is evaluated to true if it is true for all the rows in the table expression (if the table is empty it is also evaluated to true).
- The comparison predicate which is quantified with ANY or SOME is evaluated to true if it is true for some of the rows in the table expression (if the table is empty it is evaluated to false).

(*) The predicate “IN” is equivalent to the quantified comparison predicate “=any”.

EXAMPLE:

Obtain the name of the mountain passes (“puerto”) and the cyclists who won the passes with the highest slope (“pendiente”).

```
SELECT P.nompuerto, C.nombre
FROM Puerto P, Ciclista C
WHERE P.dorsal = C.dorsal
    AND P.pendiente >= ALL (SELECT P1.pendiente
                            FROM Puerto P1):
```


EXAMPLE:

Obtain the Mountain passes (“puerto”) and the cyclists who won them, such that the pass is not the one with the lowest slope.

```
SELECT P.nompuerto, C.nombre
FROM Puerto P, Ciclista C
WHERE P.dorsal = C.dorsal
    AND P.pendiente > ANY (SELECT P1.pendiente
                           FROM Puerto P1):
```

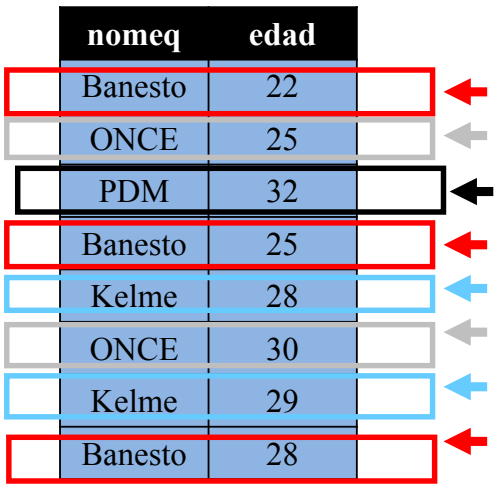
(*) ANY can always be converted into an ALL by negating the condition and adding a NOT, and vice versa.

1. Introduction to SQL
2. Queries
    2.1. Simple queries using one table.
    2.2. Simple queries with several tables
    2.3. Subqueries
    2.4. Quantified comparison predicates
    2.5. Grouping
    2.6. Set operations
    2.7. Joins
3. Database updates
4. Commands for handling transactions

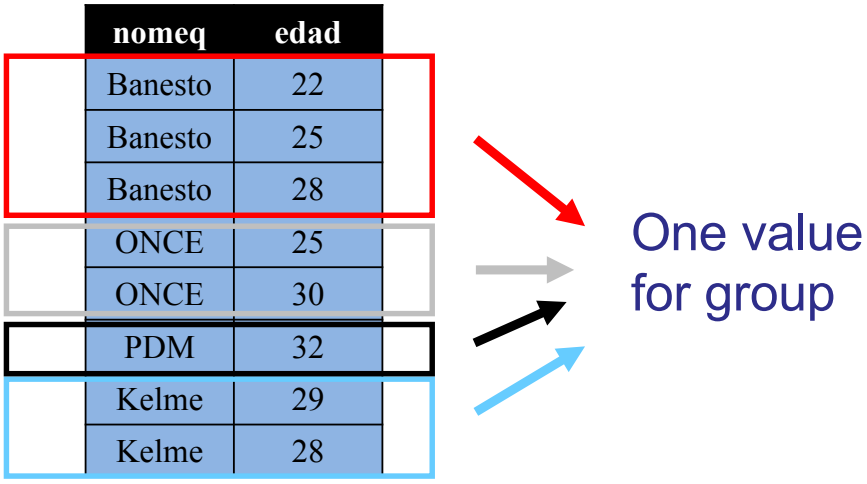
A group is a set of rows with the same value for the subset of columns used for grouping (used in the GROUP BY).

Example: Obtain the name of all the teams and the average age of the cyclist in each team.

```
SELECT nomeq, AVG(edad)
FROM Ciclista
GROUP BY nomeq;
```



The aggregated functions in the grouped queries do not work like the rest of queries. Here, they return a value for each group which is formed.



SELECT nomeq, AVG(edad)

FROM Ciclista

GROUP BY nomeq;

Returns:

nomeq	edad
Banesto	25
ONCE	27,5
PDM	32
Kelme	28,5

SYNTAX:

SELECT [ALL | DISTINCT] {expression1, $expression_2$,..., $expression_n$|*}
FROM *$table_1$, $table_2$ …, $table_n$*
[WHERE condition]
[GROUP BY *$column_1$, $column_2$,..., $column_n$*
[HAVING *conditional_expression*]]
[ORDER BY *$column_1$, $column_2$,..., $column_n$*]

Group, where and having

The HAVING clause can only appear in grouped queries, and it is an extra condition similar to WHERE, but applied to the groups:

1º) WHERE condition (used for the rows)

2º) Grouping and calculus of aggregated functions

3º) HAVING condition (used for the groups)

In the HAVING clause can only appear references to columns used in the group by, aggregated functions, or subqueries.

In the SELECT clause can only appear references to columns used in the group by, aggregated functions, or literals.

WRONG EXAMPLE:

SELECT nomeq, ~~nombre,~~ AVG(edad) FROM Ciclista

GROUP BY nomeq;

Group, where and having

The where clause is applied before grouping.

EXAMPLE: Obtain the name of the teams with a name starting by 'K', and the average age of their cyclists who are older than 25

5 ----> SELECT nomeq, AVG(edad)

1 ----> FROM Ciclista

2 ----> WHERE edad > 25

3 ----> GROUP BY nomeq;

4 ----> HAVING nomeq LIKE "K%"

In the grouped queries, it is possible to use nested aggregated functions.

Example:

Obtain the average age for the teams with the maximum average age (of their members).

```
SELECT  MAX(AVG(edad))
FROM Ciclista
GROUP BY nomeq;
```


Obtain the name of the teams and the average age of their cyclists who are older than 25, from those teams with more than 8 cyclists who are older than 25.

```
SELECT nomeq, AVG(edad)
FROM Ciclista
WHERE edad > 25
GROUP BY nomeq
HAVING COUNT(dorsal) > 8 ;
```

Obtain the name of the teams and the average age of their cyclists who are older than 25, from those teams with more than 8 cyclists ~~who are older than 25~~.

```
SELECT C.nomeq, AVG(C.edad)
FROM Ciclista C
WHERE C.edad > 25
GROUP BY C.nomeq
HAVING 8 < (SELECT COUNT(*)
            FROM Ciclista C2
            WHERE C.nomeq = C2.nomeq));
```

Example:
Obtain the name of the cyclist and the number of mountain passes he has won, but only if the mean of the slope (pendiente) of the won mountain passes (puertos) is greater than 10.

```
SELECT C.nombre, COUNT(P.nompuerto)
FROM Ciclista C, Puerto P
WHERE C.dorsal = P.dorsal
GROUP BY C.dorsal, C.nombre      /*Always group by the PK */
HAVING AVG (P.pendiente) >10;
```

Example:
Obtain the name of the cyclists who have won at least one stage and belong to a team with more than 5 cyclists, indicating how many stages has won that cyclist.

```
SELECT C.nombre, count(*)
FROM ciclista C, etapa E
WHERE C.dorsal = E.dorsal
    AND   5< ( SELECT count(*)
               FROM Ciclista C2
               WHERE C2.nomeq = C.nomeq )
GROUP BY C.nombre, C.dorsal;
```


1. Introduction to SQL
2. Queries
    2.1. Simple queries using one table.
    2.2. Simple queries with several tables
    2.3. Subqueries
    2.4. Quantified comparison predicates
    2.5. Grouping
    2.6. Set operations
    2.7. Joins
3. Database updates
4. Commands for handling transactions

Other table combinatios

There are other ways to combine several tables in the same query. All of them, along with the ways we have already seen, are what we have called "*table_expression*".

- Include several tables in the FROM clause.
- Use of subqueries in the conditions in the where or having clause .
- **Set table combinations**: use the set operators to combine the tables.
- **Table joins**: combine two tables by using different variants of the JOIN operator in Relational Algebra.

Set combinations

Correspond to the UNION, DIFFERENCE and INTERSECTION in the relational algebra.

- UNION
- EXCEPT (MINUS in Oracle 10)
- INTERSECT

They make possible to combine tables with compatible schemas.

UNION

table_expression union [all] *table_expression*

Performs a union of the rows of the tables expressed by the two “table_expression”.

Duplicates will be allowed if the option ALL is set.

Example:

Obtain the name of all the people participating in the cycling race.

```
(SELECT nombre FROM Ciclista)
        UNION
(SELECT director FROM Equipo)
```

Obtain the name of the cyclists who have worn some maillot or have won a mountain pass or a stage.

```
SELECT nombre
FROM Ciclista
WHERE dorsal IN
        (SELECT dorsal FROM Llevar
                UNION
        SELECT dorsal FROM Puerto
                UNION
        SELECT dorsal FROM Etapa)
```

INTERSECCION

table_expression intersect *table_expression*

Performs a intersection of the rows of the tables expressed by the two “table_expression”.

Example:

Obtain the name of the cyclists with the same name as a team director.

```
(SELECT nombre FROM Ciclista)
        INTERSECT
(SELECT director FROM Equipo)
```

DIFERENCE

table_expression$_1$ except[1] *table_expression*$_2$

Return the tupls in table_expression$_1$ which do not appear in table_expresion$_2$.

Example:

Name of cyclist which do not appear in the table of team directors

```
(SELECT nombre FROM Ciclista)
        EXCEPT
(SELECT director FROM Equipo)
```

[1]MINUS in Oracle 10

1. Introduction to SQL
2. Queries
    2.1. Simple queries using one table.
    2.2. Simple queries with several tables
    2.3. Subqueries
    2.4. Quantified comparison predicates
    2.5. Grouping
    2.6. Set operations
    2.7. Joins
3. Database updates
4. Commands for handling transactions

JOIN

3 Types of Joins (concatenation in the relational algebra).

1. Cross join
2. Inner Join
3. Outer join

1. Cross join

table_reference$_1$ cross join *table_reference*$_2$

≡

SELECT * FROM *table_reference*$_1$, *table_reference*$_2$

2. Inner Join

Syntax (3 different forms):

table_reference$_1$
[NATURAL] [INNER] JOIN
table_reference$_2$
[ON condition| USING (*column*$_1$, *column*$_2$,…,*column*$_n$)]

Form 1:

table$_1$* [inner] join *table$_2$* on *conditional expression

≡ SELECT * FROM *table1*, *table2*
WHERE *conditional_expression*

EJEMPLO: Obtain the names of the mountain passes (puertos) and the number of the stage (etapa=) in which the pass is, if the stage length is greater than 200km.

```
SELECT nompuerto, P.netapa
FROM Puerto P JOIN Etapa E ON P.netapa= E.netapa+1
WHERE E.km>200
```


Form 2:

***table1* [inner] join *table2* using (c1, c2,… cn)**

≡ SELECT * from *table1, table2*
WHERE *table1.c1 = table2.c1*
AND *table1.c2 = table2.c2*
AND..... and *table1.cn = table2.cn*

Example: Obtain the name of the mountain passes (puerto), the number of the stage (etapa) in which the mountain pass is and the length of the stage, but only if the mountain pass is higher than 800.

```
SELECT nompuerto, netapa, km
FROM Puerto JOIN Etapa USING (netapa)
WHERE altura>800
```

Form 3:

table1 **natural inner join** ***table2***

≡ *table1* join *table2* using (c1, c2, ..., cn)
where table1 has n attributes.

(It is a regular JOIN but using the common attributes of both tables)

EXAMPLE: Obtain the name of the cyclists of the team directed by 'Alvaro Pino'.

```
SELECT nombre
FROM Ciclista  NATURAL JOIN Equipo
WHERE director = 'Alvaro Pino';
```


EXAMPLE: Obtain the name of the mountain passes and their stage and km of the stage of those mountain passes higher than 800.

```
SELECT nompuerto, netapa, km
FROM Puerto JOIN Etapa USING (netapa)
WHERE altura>800
```

WRONG:

~~SELECT nompuerto, netapa, km~~
~~FROM Puerto NATURAL JOIN Etapa~~
~~WHERE P.altura>800~~

because dorsal appears in both tables

Obtain the number and name of the cyclists, and the amount of maillots worn by each of them

```
SELECT C.dorsal, C.nombre, COUNT (DISTINCT L.codigo)
FROM  Ciclista C, Llevar L
WHERE C.dorsal = L.dorsal
GROUP BY C.dorsal, C.nombre
```

```
SELECT dorsal, ciclista.nombre, COUNT (DISTINCT llevar.codigo)
FROM Ciclista NATURAL INNER JOIN Llevar
GROUP BY dorsal, ciclista.nombre
```

3. OuterJoin

Combine **all** the rows from one of the tables (even if there is no correspondence for some row in the other table)

table_expression
[NATURAL] {LEFT | RIGHT | FULL} [OUTER] JOIN
table_expression
[ON condition| USING ($column_1, column_2,..., column_n$)]

Table1 LEFT JOIN Table2 ON conditional_expression

Inner join of *Table1* and *Table2* UNION tuples from *Table1* that do not appear in the inner join, using NULL values in the rest of columns

FULL: Returns all the tuples from *table1* and *table2*

Example:
Obtain the name of all the cyclists and the total number of stages won by each of them.

```
SELECT nombre, COUNT(netapa)
FROM Ciclista NATURAL LEFT JOIN Etapa
GROUP BY dorsal, nombre
```


Example:
Obtain for all cyclist, his/her number (dorsal), name (nombre) and the code of every maillot worn by that cyclist (and the number of the stage in which that cyclist has worn that maillot).

```
SELECT C.dorsal, nombre, codigo, netapa
FROM Ciclista C LEFT JOIN Llevar L ON  C.dorsal = L.dorsal
```

List the name of all the teams, indicating how many cyclist there are in each of them

```
SELECT equipo.nomeq, COUNT(dorsal)
FROM Equipo LEFT JOIN Ciclista
                        ON equipo.nomeq= ciclista.nomeq
GROUP BY equipo.nomeq
```

```
( SELECT nomeq, count(*)
  FROM ciclista
  GROUP BY nomeq )
UNION
( SELECT nomeq, 0
  FROM equipo
  WHERE nomeq NOT IN (SELECT nomeq FROM ciclista) )
```


UD 2.1 DML: Queries and Data Manipulation

1. Introduction to SQL
2. Queries
    2.1. Simple queries using one table.
    2.2. Simple queries with several tables
    2.3. Subqueries
    2.4. Quantified comparison predicates
    2.5. Grouping
    2.6. Set operations
    2.7. Joins
3. Database updates
4. Commands for handling transactions

3. Database updates

DML (Data Manipulation Language): Queries and database updates.

- SELECT: Allows the declaration of queries to retrieve the information from the database
- INSERT: Performs the insertion of one or more rows in a table
- DELETE: Allows the user to delete one o more rows from a table
- UPDATE: Modifies the values of one or more columns and/or one or more rows in a table

The INSERT command

INSERT INTO table [(*column1, column2,…, column$_n$*)]

{ DEFAULT VALUES |

VALUES (atom$_1$, atom$_2$,… atom$_n$) |

table_expression}

- If we do not include the list of columns, we will have to specify all the attributes of the table.
- If we include the option "*default values*", we will insert a single row with all the default values which were defined in the definition of the table.
- In the option (*atom_commalist)*, the atoms are given by scalar expressions.
- In the option *table_expression*, we insert the rows which result from the execution of the expression (a SELECT).

Example (a complete tuple):

Add a cyclist with dorsal 101, name 'Joan Peris', age 27, and team 'Kelme'.

```
INSERT INTO Ciclista
VALUES (101, 'Joan Peris', 27,'Kelme');
```

Example (an incomplete tuple) :

Add a cyclist with dorsal 101, name 'Joan Peris', and team 'Kelme' (we don't know the age):

```
INSERT INTO Ciclista (dorsal, nombre, nomeq)
VALUES (101, 'Joan Peris', 'Kelme');
```


Example (inserting many tuples):

Add into the *Ganador* (winner) table (same schema as Ciclista) all the information of the cyclists who have won some stage (etapa).

```
INSERT INTO Ganador
( SELECT dorsal, nombre, edad, nomeq
FROM Ciclista
WHERE dorsal IN (SELECT dorsal FROM etapa) )
```

The DELETE command

DELETE FROM *table* [WHERE conditional_expression]

If we include the WHERE clause, then it will only delete the rows which make the condition true. In other case, all the tuples will be deleted.

Example:

Delete the information about the cyclist 'M. Indurain' because he is retired.

```
DELETE FROM Ciclista
WHERE nombre = 'M. Indurain'
```


The UPDATE command

```
UPDATE table
SET asignnment1, asignnment 2,..., asignnment n
[ WHERE conditional_expression ]
```

Where the assignements are of the form::

column = {DEFAULT | NULL | scalar_expression}

Example:

Increase the *premio* (prize) of the maillots by 10%

```
UPDATE Maillot
SET premio = premio * 1.10
```

Example:

Move all the Kelme cyclists to the *K10* team.

```
UPDATE Ciclista
        SET nomeq = 'K10'
        WHERE nomeq='Kelme' ;
```


1. Introduction to SQL
2. Queries
    2.1. Simple queries using one table.
    2.2. Simple queries with several tables
    2.3. Subqueries
    2.4. Quantified comparison predicates
    2.5. Grouping
    2.6. Set operations
    2.7. Joins
3. Database updates
4. Commands for handling transactions

4. Commands for handling transactions

A transaction is a logical unit of work consisting of one or more SQL statements that is guaranteed to be atomic with respect to recovery.

Transaction initiation::

It is implicit. A transaction begin with the first SQL statement in a session, or when the previous transaction ends.

Transaction completion:

- COMMIT: The transaction ends **successfully**, making the database changes permanent.
- ROLLBACK: The transaction **aborts**, backing out any changes made by the transactions.

Example:
The 'Banesto' team changes its name to 'BanQue'

/* Here we must write a special command to differ the evaluation of the foreign key *nomeq* in *Ciclista*. We will study this SQL command in the next chapters */

```
UPDATE Equipo SET nombre = 'BanQue'
WHERE nomeq = 'Banesto';

UPDATE Ciclista SET nomeq = 'BanQue'
WHERE nomeq = 'Banesto';

COMMIT;
```

Successful transaction

SQL Queries review

Operator	Relational Algebra	SQL
selection	R Donde F	SELECT ... FROM R WHERE F
projection	R [A_i , A_j ..., A_k]	SELECT A_i , A_j ..., A_k FROM R
Cartesian product	R_1 x R_2, ... x R_n	SELECT ... FROM R_1, R_2, ..., R_n, o SELECT...FROM R_1 CROSS JOIN R_2, ..., CROSS JOIN R_n
join	R_1 R_2	SELECT... FROM R_1 NATURAL JOIN R_2
union	$R_1 \cup R_2$	SELECT * FROM R_1 UNION SELECT * FROM R_2
difference	R_1 - R_2	SELECT * FROM R_1 EXCEPT SELECT * FROM R_2
intersection	$R_1 \cap R_2$	SELECT * FROM R_1 INTERSECT SELECT * FROM R_2

Some exercises

1. List the name of the youngest cyclist. (21)
2. List the value of the attribute netapa and the departure city for those stages with no mountain passes. (16)
3. List the name of the departure and the arrival of the stages where the steepest mountain passes (“puerto”) are located. (19)
4. List the name of the cyclists who have won all the mountain passes (“puerto”) in one stage and have also won the stage. (27)
5. List the code and the color of those jerseys (‘maillots’) which have only been worn by cyclists of the same team. (29)
6. List the name of the cyclists who belong to a team which has more than five cyclists and have also won one or more stages. Please also indicate how many stages he has won.
7. List the name of all the cyclists who belong to a team which has more than five cyclists indicating how many stages he has won. (35)
8. List the cyclist number (dorsal) and the name of the cyclists who have not worn all the jerseys (maillots) worn by the cyclist with number 20. (40)
9. List the name of the teams and the average age of the cyclists of those teams who have the highest average age of all the teams. (36)
10. List the name of those teams such that their cyclists have only won mountain passes (puerto) of category = 1. (30)