

Quantified comparison predicates (ANY and ALL)

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).

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

Quantified comparison predicates (ANY and ALL)

Example 8: 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 9: Get the name of 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 );
```

(*) The word ANY can always be converted into an ALL by changing its condition to the negated condition and adding a NOT, and vice versa.

EXISTS Predicate

exists (expression_table)

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

Example: Obtain the name of those cyclists who have worn a jersey with a prize lower than 50,000 ptas.

```
SELECT C.nombre FROM Ciclista C, Llevar L
WHERE C.dorsal = L.dorsal AND
EXISTS (SELECT * FROM Maillot M
        WHERE M.premio < 50000 AND M.codigo = L.codigo);
```

(*) In the example, we can only see that there is an external reference to a table L which is external to the subquery, which is frequent (although not exclusive) to the subqueries with EXISTS.

(*) In general, IN and EXISTS (without negation) are interchangeable and can be replaced by queries to several tables and introduce equalities which connect the tables.

EXISTS Predicate

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

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

or

```
SELECT nombre FROM Ciclista
WHERE dorsal NOT IN (SELECT dorsal FROM Etapa);
```

Equivalencies:

```
WHERE EXISTS (SELECT * FROM ...)
```

Is equivalent to: WHERE 0 < (SELECT COUNT(*) FROM ...)

```
WHERE NOT EXISTS (SELECT * FROM ...)
```

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

Use of EXISTS for Universal Quantification

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

Example: Obtain the name of the cyclist (if any) who has won **all** the stages 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);
```

The query has been converted into: “Obtain the name of the cyclist such that there does **not** exist a stage with more than 200 km. which has **not** been won by him”

Equivalence between logic and SQL

Logic expressions (e.g. CRT) can be converted into SQL in the following way:

Logic Exp.	Intermediate step	SQL
$p \rightarrow q$	$\Rightarrow \neg p \vee q$	(NOT p) OR q
$\forall x p$	$\Rightarrow \neg \exists x \neg p$	NOT EXISTS (...negate....)
$\forall x (p \rightarrow q)$	$\Rightarrow \neg \exists x \neg (p \rightarrow q) \equiv \neg \exists x \neg (\neg p \vee q)$ $\equiv \neg \exists x (p \wedge \neg q)$	

“Add-on”s for universally quantified queries

Example: Obtain the name of the cyclist (if any) who has won **all** the stages with more than 200 km.

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

What happens if there are no stages with more than 200 km?

WE WOULD GET ALL THE CYCLISTS!!!

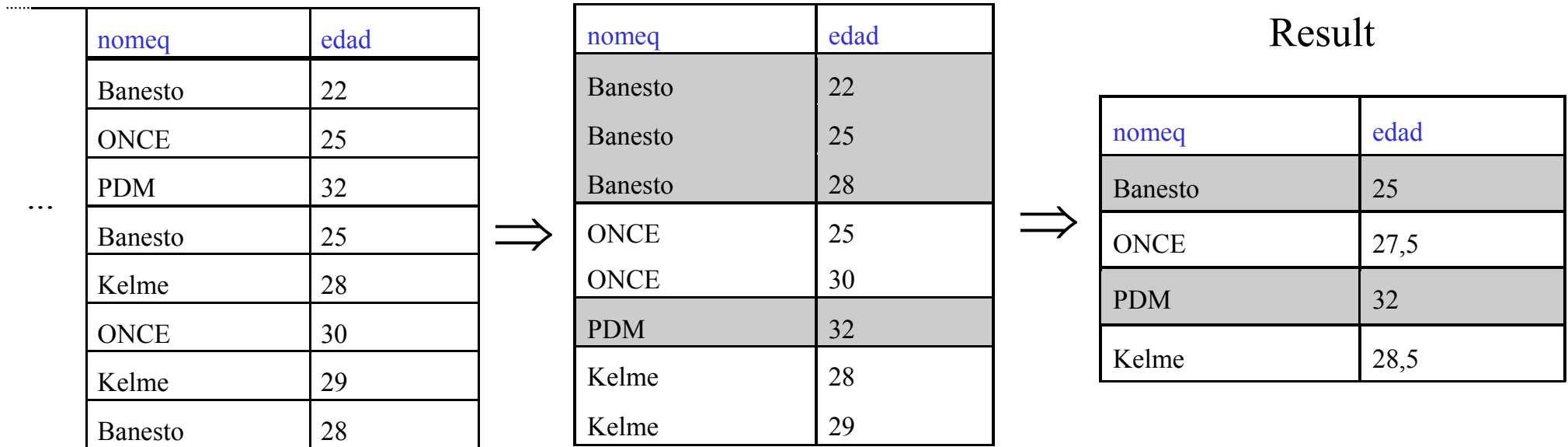
Solution:

```
SELECT C.nombre FROM Ciclista C
WHERE NOT EXISTS (SELECT * FROM Etapa E
                  WHERE E.km > 200 AND C.dorsal <> E.dorsal )
AND EXISTS (SELECT * FROM ETAPA E2 WHERE E2.km > 200);
```

Complex Queries: GROUP BY

Example: Obtain the name of **each** team and the average age of the cyclists of that team:

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



Relationship between SELECT and GROUP BY

A group can be understood as a set of rows with the same value for the set of columns which is used for grouping (included in the clause GROUP BY).

The aggregated functions in the grouped queries work in a different way than in the rest of queries. Here, they return a value for each group which is formed.

The idea is that we can only SELECT in such a way that we have a single value (direct or aggregated) for each formed group.

INCORRECT EXAMPLE:

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

The syntactic rule which is applied by the DBMS and fixed by SQL:

“in the SELECT clause of an aggregated query, we can only use columns which appear in the GROUP BY, references to aggregated functions or literals (constants)”.

GROUP BY, WHERE and HAVING

GROUP and WHERE

If we include the WHERE clause, the execution of this clause is applied before the grouping.

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

GROUP, WHERE and HAVING

The HAVING clause, which can only be included with grouped queries, is similar to the WHERE but the order is as follows:

- 1º) WHERE condition (used for the rows)
- 2º) Grouping and calculus of aggregated values
- 3º) HAVING condition (used for the groups)

In the HAVING clause, we will only be able to include references to columns which appear in the GROUP BY, aggregated functions or subqueries.

Examples with HAVING

Example: Obtain the name of **each** team and the average age of their cyclists who are older than 25, from those teams with more than 3 cyclists who are older than 25.

```
SELECT nomeq, AVG(edad) FROM Ciclista
WHERE edad > 25
GROUP BY nomeq
HAVING COUNT(dorsal) > 3;
```

Example: Obtain the name of the cyclist and the number of mountain passes he has won, the mean of the slope being 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;
```

Other table combinations

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”.

There are, hence, several ways to combine two tables in SQL:

- 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 Table Combinations

Correspond to the UNION, DIFFERENCE and INTERSECTION in the R.A.

- UNION
- EXCEPT
- INTERSECT

Make it possible to combine tables which have 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 the cyclists from ‘Banesto’ and from ‘ONCE’.

```
(SELECT nombre FROM Ciclista WHERE nomeq = ‘Banesto’)
```

```
UNION
```

```
(SELECT nombre FROM Ciclista WHERE nomeq = ‘ONCE’)
```

Table joins

They are variants of the JOIN operator in R.A.

- Cartesian product (cross join)

- Inner joins

table_reference [natural] [inner] join *table_reference*
[on *conditional_expression* | using (*column_comma*list)]

- Outer joins

table_reference [natural]
{left [outer] |
right [outer] |
full [outer] } JOIN *table_reference*
[on *conditional_expression* | using (*column_comma*list)]

- Union join

... FROM T1 UNION JOIN T2 ≡ ... FROM $\left(\begin{array}{l} \text{select t1.*, null, null, ..., null from t1} \\ \text{union all} \\ \text{select null, null, ..., null, t2.* from t2} \end{array} \right)$ 14

Relational Algebra and SQL

R.A.	STANDARD SQL	SQL in ORACLE'10
• \cup	• UNION	• UNION
• $-$	• EXCEPT	• Ok in Oracle 11g
• \cap	• INTERSECT	• INTERSECT
• \times	• CROSS JOIN	• Ok in Oracle 11g
• \bowtie	• JOIN	• Ok in Oracle 11g
Other		
• \cup (dups.)	• UNION ALL	• UNION ALL
• \bowtie	• Left/Right/Full JOIN	• Ok in Oracle 11g. No in some versions of Access
	• SELECT COUNT(DISTINCT a)	• Ok in Oracle 11g. Not in Access

Adding new information: INSERT

insert into table [*column_commalist*]

{ default values | values (*atom_commalist*) | *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: Add a cyclist with dorsal 101, name ‘Tom’, age 20 and team ‘Kelme’.

```
insert into Ciclista
```

```
values (101, ‘Tom’, 20, ‘Kelme’);
```

Modifying existing information: UPDATE

update table

set *assignments_commlist*

[where *conditional_expression*]

Where the assignments are of the form:

column = {default | null | *scalar_expression*}

If the WHERE clause is included, then the UPDATE instruction will only be applied to the rows which make the condition true.

Example: Increment 10% the slope of the mountain pass (puerto) called 'Aitana'.

```
UPDATE Puerto SET pendiente = pendiente * 1.10
```

```
WHERE nompuerto = 'Aitana';
```

Removing existing information: DELETE

delete from table [where *conditional_expression*]

If we include the WHERE clause, then it will only delete those rows which make the condition true.

Example: Erase the information about the cyclist 'M. Indurain', since he retired long time ago.

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

More examples for the CYCLISM schema

1) Obtain the number of stages and the departure city for those stages with no mountain passes.

```
SELECT netapa, salida
```

```
FROM etapa
```

```
WHERE not exists ( SELECT * FROM puerto
```

```
WHERE puerto.netapa=etapa.netapa );
```

More examples for the CYCLISM schema

2) Obtain the name of the departure and arrival cities for the stage with the steepest mountain pass.

```
SELECT e.salida, e.llegada
```

```
FROM etapa e, puerto p
```

```
WHERE e.netapa=p.netapa AND
```

```
    p.pendiente=(select MAX(pendiente) from puerto );
```

More examples for the CYCLISM schema

3) Who is the youngest cyclist?

```
SELECT nombre
```

```
FROM ciclista c
```

```
WHERE edad = ( SELECT MIN(edad) FROM ciclista );
```

More examples for the CYCLISM schema

4) Obtain the name of the cyclists who have won all the mountain passes in one stage and also have won the same 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.dorsal=c.dorsal
              AND p.netapa=e.netapa );
```

More examples for the CYCLISM schema

5) Obtain the colour of the jerseys which have only been worn by cyclists at the same time.

```
SELECT DISTINCT color FROM maillot m, llevar l, ciclista c
WHERE c.dorsal=l.dorsal AND m.codigo=l.codigo
AND NOT EXISTS ( SELECT * FROM llevar l2, ciclista c2
                  WHERE c2.dorsal=l2.dorsal AND
                  c2.nomeq<>c.nomeq AND l2.codigo=l.codigo);
```

More examples for the CYCLISM schema

6) Obtain the name of the cyclists who belong to a team which has more than five racers, also indicating the number of stages which have been won one by each of them.

```
SELECT c.nombre, COUNT(*) FROM ciclista c, etapa e
WHERE c.dorsal=e.dorsal AND
      5<( SELECT COUNT(*) FROM ciclista c2
          WHERE c2.nombre=c.nombre )
GROUP BY c.nombre, c.dorsal;
```

More examples for the CYCLISM schema

7) Obtain the name of the teams with the highest average cyclist age.

```
SELECT C.nomeq, AVG(C.edad)
```

```
FROM ciclista C
```

```
GROUP BY C.nomeq
```

```
HAVING AVG(C.edad) >= ALL
```

```
(SELECT AVG(D.edad)
```

```
FROM ciclista D
```

```
GROUP BY D.nomeq);
```

More examples for the CYCLISM schema

8) Name of the cyclists who have not worn all the jerseys which have been worn by the cyclist with “dorsal” 1.

```
SELECT c.nombre FROM ciclista c
WHERE EXISTS( SELECT * FROM l1var l
              WHERE l.dorsal=1 AND
              NOT EXISTS(SELECT * FROM l2var l2
                        WHERE l2.dorsal=c.dorsal AND
                        l2.codigo=l.codigo) );
```