Uso de EXISTS para cuantificación universal (NO HAY EN SQL)

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

Obtener el nombre de los profesores que imparten todas las asignaturas.

```
SELECT P.nombre
FROM Profesor P
WHERE NOT EXISTS (SELECT * FROM Asignatura A
                  WHERE NOT EXISTS (SELECT *
                                    FROM Docencia D
                                    WHERE D.cod_pro=P.cod_pro ∧
                                          D.cod_asg=A.cod_asg))
```

El lenguaje SQL

Obtener el nombre de los profesores que imparten todas las asignaturas de su departamento de mas de 6 créditos.

¿qué pasa si en el departamento de un profesor no hay asignaturas de mas de 6 créditos?

El lenguaje SQL

¿qué pasa si en el departamento de un profesor PX no hay asignaturas de mas de 6 créditos?

FALSO para todo valor de AX

$$\{PX.nombre \mid Profesor(PX) \wedge \forall AX((Asignatura(AX) \wedge AX.cod\_dep = PX.cod\_dep \wedge (AX.teoría + AX.prac) > 6) \rightarrow \exists DX(Docencia(DX) \wedge DX.cod\_pro = PX.cod\_pro \wedge DX.cod\_asg = AX.cod\_asg))\}$$

¡El profesor PX aparecería en el resultado de la consulta!

CIERTO

El lenguaje SQL

¡Si estos profesores no deben salir en la consulta, entonces se debe **hacer un control** para comprobar que en el departamento del profesor existe alguna asignatura de mas de seis créditos!.

$$\{PX.nombre \mid Profesor(PX) \wedge \exists AX(Asignatura(AX) \wedge AX.cod\_dep = PX.cod\_dep \wedge (AX.teoría + AX.prac) > 6) \wedge \forall AX((Asignatura(AX) \wedge AX.cod\_dep = PX.cod\_dep \wedge (AX.teoría + AX.prac) > 6) \rightarrow \exists DX(Docencia(DX) \wedge DX.cod\_pro = PX.cod\_pro \wedge DX.cod\_asg = AX.cod\_asg))\}$$

El lenguaje SQL.

SQL

```
SELECT PX.nombre
FROM Profesor PX
WHERE  EXISTS (SELECT *
                FROM Asignatura AX
                WHERE AX.cod_dep=PX.cod_dep AND (AX.teoría+AX.prac)>6)
                AND
     NOT EXISTS (SELECT * FROM Asignatura AX
                WHERE AX.cod_dep= PX.cod_dep AND (AX.teoría+AX.prac)>6
                AND
                NOT EXISTS (SELECT * FROM Docencia DX
                            WHERE DX.cod_pro=PX.cod_pro AND
                            DX.cod_asg=AX.cod_asg) ) )
```

Uso de EXISTS para cuantificación universal

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

Obtener el nombre del ciclista que ha ganado todas las etapas de más de 200 km.

C.nombre| Ciclista(C)∧∀ X (Etapa(X)∧X.Km>200→C.dorsal=X.dorsal)

es equivalente a:

C.nombre| Ciclista(C)∧¬ ∃X(Etapa(X)∧X.Km>200∧C.dorsal<>X.dorsal)

Para poder expresar esta consulta en SQL se convertirá en: "Obtener el nombre del ciclista tal que *no* existe una etapa de más de 200 km. que él *no* haya ganado"

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

Uso de "Coletillas" en consultas con cuantificación universal

¿Qué pasa si no hay etapas de más de 200 km?

Solución:

```
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);
```

Ejercicios:
Práctica 3: El lenguaje SQL (1a Parte)
Hacer el bloque de consultas con cuantificación universal de las BDs Ciclismo y Música

Biblioteca

AUTOR(autor_id: tira(4), nombre: tira(35), nacionalidad: tira(20))
Clave Primaria: {autor_id}

LIBRO(id_lib: tira(10), titulo: tira(80), año: entero, num_obras: entero)
Clave Primaria: {id_lib} VNN: {titulo}

TEMA(tematica: tira(20), descripcion: tira(50))
Clave Primaria: {tematica}

OBRA(cod_ob: entero, titulo: tira(80), año: d_cat, tematica: tira(20))
Clave Primaria: {cod_ob}
Clave Ajena: {tematica}→ TEMA
VNN: {titulo}

AMIGO(num: entero, nombre: tira(60), telefono: tira(10))
Clave Primaria: {num}
VNN: {nombre}

PRESTAMO(num: entero, id_lib:tira(10))
Clave Primaria: {num,id_lib}
Clave Ajena: {num} → AMIGO
Clave Ajena: {id_lib} → LIBRO

ESTA_EN(cod_ob: entero, id_lib:tira(10))
Clave Primaria: {cod_ob,id_lib}
Clave Ajena: {cod_ob} → OBRA
Clave Ajena: {id_lib} → LIBRO

ESCRIBIR(cod_ob: entero, autor_id:tira(4))
Clave Primaria: {cod_ob,autor_id}
Clave Ajena: {cod_ob}→ OBRA
Clave Ajena: {autor_id}→ AUTOR

Consultas Agrupadas

SELECT [**ALL** | **DISTINCT**] A_{1i}, A_{2j},..., A_{nk}| *
FROM R_1, R_2, ..., R_n
[**WHERE** *condición*]
[**GROUP BY** B_1, B_2,..., B_m]
[**HAVING** *condición*]

GROUP BY: define grupos de tuplas en el conjunto de tuplas seleccionadas por la condición WHERE. Los grupos se definen por la igualdad de valor en los atributos de agrupación (B_1, B_2,..., B_m).

HAVING: de los grupos definidos se seleccionan aquellos que cumplen la condición expresada.

Consultas Complejas
Relación Selección-Agrupamiento

Un grupo se puede entender como un conjunto de filas con el mismo valor para el conjunto de columnas por las que se agrupa (las incluidas en la cláusula GROUP BY).

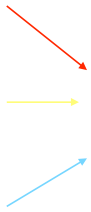
EJEMPLO: Obtener el nombre de cada equipo y la edad media de los ciclistas de dicho equipo:

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

Nomeq	Edad
Banesto	22
ONCE	25
PDM	32
Banesto	25
Kelme	28
ONCE	30
Kelme	29
Banesto	28

Las funciones agregadas en las consultas agrupadas funcionan de forma diferente que en las consultas normales, devolviendo un valor por cada grupo formado.

Nomeq	Edad
Banesto	22
Banesto	25
Banesto	28
ONCE	25
ONCE	30
PDM	32
Kelme	29
Kelme	28



Un Valor por Grupo

Entonces, para

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

La solución, es:

Nomeq	Edad
Banesto	25
ONCE	27,5
PDM	32
Kelme	28,5

Consultas Agrupadas

EJEMPLO: Obtener el número total de profesores de cada departamento

cod_pro	nombre	telOfono	cod_dep
JCC	Juan C. Casamayor R4denas	7796	DSIC
RFC	Robert Fuster i Capilla	6789	MAT
JBD	JosOV. Benlloch Dualde	5760	DISCA
MAF	MarTa Alpuente Frasnedo	3560	DSIC
CPG	Cristina POrez Guillot	7439	IDM
JTM	JosOM. Torralba MartTnez	4590	OEM
IGP	Ignacio Gil Pechu3n	3423	OEM
DGT	Daniel Gil Tom3s	5679	DISCA
MCG	Matilde Celma GimOnez	7756	DSIC

cod_dep	
DSIC	3
MAT	1
DISCA	2
IDM	1
OEM	2

```
SELECT cod_dep, COUNT (*)
 FROM Profesor
 GROUP BY cod_dep
```

Consultas Agrupadas

Obtener el número total de profesores de los departamentos que tienen mas de 2 profesores.

cod_pro	nombre	telOfono	cod_dep
JCC	Juan C. Casamayor R4denas	7796	DSIC
RFC	Robert Fuster i Capilla	6789	MAT
JBD	JosOV. Benlloch Dualde	5760	DISCA
MAF	MarTa Alpuente Frasnedo	3560	DSIC
CPG	Cristina POrez Guillot	7439	IDM
JTM	JosOM. Torralba MartTnez	4590	OEM
IGP	Ignacio Gil Pechu3n	3423	OEM
DGT	Daniel Gil Tom3s	5679	DISCA
MCG	Matilde Celma GimOnez	7756	DSIC

cod_dep	
DSIC	3

```
SELECT cod_dep, COUNT (*)
 FROM Profesor
 GROUP BY cod_dep
 HAVING COUNT (*) > 2
```

EJEMPLO INCORRECTO:

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

La regla sintáctica que aplican los sistemas relacionales para asegurar el buen funcionamiento de las consultas agrupadas es la siguiente:

"En la selección de una consulta agrupada, sólo pueden aparecer referencias a columnas por las cuales se agrupa, referencias a funciones agregadas o literales".

GROUP y WHERE

Si se incluye la cláusula **where,** la aplicación de esta cláusula se produce previamente a la agrupación.

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

Evaluación:

1) Se seleccionan n tuplas de las relaciones que cumplan la condición de la cláusula WHERE.
2) En el conjunto de tuplas seleccionadas se definen grupos basados en el valor de los atributos de agrupación.
3) De los grupos definidos se seleccionan los que cumplen la condición de la cláusula HAVING.

La cláusula **HAVING** sólo puede ir en consultas agrupadas y es similar a **WHERE,** pero en un orden diferente:

1º) Condición **WHERE** (se usa para las filas)

2º) Agrupamiento y cálculo de valores agregados

3º) Condición **HAVING** (se usa para los grupos)

En la cláusula **HAVING** sólo podrán aparecer directamente referencias a columnas por las cuales se agrupan o a funciones agregadas.

EJEMPLO: Obtener el nombre de cada equipo y la edad media de sus ciclistas con más de 25 años, de aquellos equipos con más de 3 corredores mayores de 25 años.

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

EJEMPLO: Obtener el nombre del ciclista y el número de puertos que ha ganado, siendo la media de la pendiente de éstos superior a 10.

```
SELECT C.nombre, COUNT(P.nompuerto)
FROM Ciclista C, Puerto P
WHERE C.dorsal = P.dorsal
GROUP BY C.dorsal, C.nombre   /* Agrupar siempre por CP */
HAVING AVG (P.pendiente) >10;
```

Ejercicios:
Práctica 3: El lenguaje SQL (1a Parte)
Hacer el bloque de “consultas agrudapas” de las BDs Ciclismo y Música

COMBINACIONES DE TABLAS

Existen otras formas de combinar varias tablas en consultas y todas ellas, junto con las ya vistas, dan lugar a una “expresión de tabla”.

Existen, en definitiva, varias formas de combinar dos tablas en el lenguaje SQL:

- → Incluir varias tablas en la cláusula **from**.
- → Uso de **subconsultas** en las condiciones de las cláusulas **where** o **having**.
- → **Combinaciones conjuntistas de tablas:** utilizando operadores de la teoría de conjuntos para combinar las tablas.
- → **Concatenaciones de tablas:** utilizando diferentes formas variantes del operador concatenación del Álgebra Relacional.

El Lenguaje Estándar SQL

Operador	Álgebra Relacional	SQL
Selección	R Donde F	SELECT ... FROM R WHERE F
Proyección	R [A_i , A_j ..., A_k]	SELECT A_i , A_j ..., A_k FROM R
Producto Cartesiano	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
Concatenación	R_1 R_2	SELECT... FROM R_1 **NATURAL JOIN** R_2
Unión	$R_1 \cup R_2$	SELECT * FROM R_1 **UNION** SELECT * FROM R_2
Diferencia	R_1 - R_2	SELECT * FROM R_1 **EXCEPT** SELECT * FROM R_2
Intersección	$R_1 \cap R_2$	SELECT * FROM R_1 **INTERSECT** SELECT * FROM R_2

COMBINACIONES CONJUNTISTAS DE TABLAS

Corresponden a los operadores unión, intersección y diferencia del Álgebra Relacional. Dadas dos tablas A y B:

- → UNION: la tabla resultado tendrá las filas de A y B
- → INTERSECT: la tabla resultado tendrá las filas que se encuentren a la vez en A y en B.
- → EXCEPT: la tabla resultado tendrá las filas de A que no se encuentren en B.

Permiten combinar tablas que tengan esquemas compatibles (mismo número de elementos seleccionados, mismo orden, mismos tipos y nombres).

UNION

expresión_tabla* union [ALL] *término_tabla

Realiza la unión de las filas de las tablas provenientes de las dos expresiones.

Se permitirán o no duplicados según se incluya o no la opción ALL.

EJEMPLO: Obtener el nombre de todo el personal de la vuelta.

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

UNION

Ejemplo 2. Obtener el nombre de todo el personal (profesores y directores de departamento).

```
SELECT director
 FROM  Departamento
        UNION
SELECT nombre
 FROM  Profesor
```

INTERSECT

expresión_tabla* intersect *término_tabla

Realiza la intersección de las filas de las tablas provenientes de las dos expresiones.

EJEMPLO: Obtener los nombres de las personas que son tanto ciclistas como directores de equipo .

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

INTERSECT

Ejemplo 2. Obtener los departamentos que tienen adscritas asignaturas y profesores.

```
SELECT DISTINCT cod_dep
 FROM  Profesor
       INTERSECT
SELECT DISTINCT cod_dep
FROM  Asignatura
```

EXCEPT

expresión_tabla* except *término_tabla

En Oracle es Minus

Realiza la diferencia de las filas de las tablas provenientes de las dos expresiones.

EJEMPLO: Obtener los nombres que aparecen en la tabla de ciclistas y no en la de directores.

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

EXCEPT

Ejemplo 2. Obtener los departamentos que no tienen adscritas asignaturas.

```
SELECT cod_dep
 FROM  Departamento
        EXCEPT
SELECT DISTINCT cod_dep
FROM  Asignatura
```

En ORACLE, el operador EXCEPT se denomina MINUS.

Concatenación de tablas

SELECT [ALL | DISTINCT] $A_{1i},...,A_{2j},...,A_{nk}$ | *
FROM concatenación de tablas
[WHERE *condición*]
[GROUP BY B_1, B_2,..., B_m]
[HAVING *condición*]

- ✓concatenación interna: INNER JOIN
- ✓concatenación externa: OUTER JOIN

Concatenación de tablas

Corresponden a variantes del operador concatenación del Álgebra Relacional.

- Producto cartesiano – CROSS JOIN
- Concatenación interna – NATURAL JOIN
- Concatenación externa – LEFT, RIGHT, FULL
- Concatenación unión – UNION JOIN

Producto Cartesiano

(CROSS JOIN)

referencia_tabla1 **cross join** *referencia_tabla2*

≡

SELECT * from *referencia_tabla1*, *referencia_tabla2*

✓La tabla resultado de la operación CROSS JOIN es el producto cartesiano de las dos tablas operandos.

Concatenación Interna

referencia_tabla1 **[natural] [inner] join** *referencia_tabla2*

[**on** *expresión_condicional* | **using** (*comalista_columna*)]

tabla1 **join** *tabla2* **on** *expresión_condicional*

≡

SELECT * FROM *tabla1*, *tabla2*
WHERE *expresión_condicional*

✓Natural Join: se concatenan las tuplas de tabla1 y tabla2 que tienen el mismo valor en todos los atributos del mismo nombre

✓Join...ON: combina una fila de cada operando cuando la condición expresada se evalúe a cierta.

✓Inner Join... USING: combina una fila de cada operando cuando el valor en las columnas comunes es idéntico.

Ejemplo- Natural Join

```
SELECT PX.cod_pro, PX.nombre, COUNT(DX.cod_asg)
 FROM Profesor PX, Docencia DX
 WHERE PX.cod_pro = DX.cod_pro
 GROUP BY cod_pro
```

concatenación de Profesor y Docencia

```
SELECT cod_pro, nombre, COUNT (cod_asg)
 FROM Profesor NATURAL JOIN Docencia
 GROUP BY cod_pro
```

✓Natural Join: se concatenan las tuplas de Profesor y Docencia que tienen el mismo valor en los atributos del mismo nombre (cod_pro)

Concatenación JOIN...USING

SELECT [**ALL** | **DISTINCT**] $A_1, A_2,..., A_n$ | *
FROM *tabla1* JOIN *tabla2* USING ($C_1, C_2,..., C_n$) (1)
[**WHERE** *condición*] (2)
[**GROUP BY** $B_1, B_2,..., B_m$] (3)
[**HAVING** *condición*] (4)

se concatenan las tuplas de tabla1 y tabla2 que tienen el mismo valor en los atributos comunes $C_1, C_2,..., C_n$

Es útil cuando no interesa que las relaciones se concatenen por todos los atributos del mismo nombre (NATURAL JOIN).

Concatenación JOIN...ON

SELECT [**ALL** | **DISTINCT**] $A_1, A_2, \ldots, A_n$ | *
FROM *tabla1* JOIN *tabla2* ON condición1 (1)
[**WHERE** *condición2*] (2)
[**GROUP BY** $B_1, B_2, \ldots, B_m$] (3)
[**HAVING** *condición*] (4)

se concatenan las tuplas de tabla1 y tabla2 que cumplen condición1

Es útil cuando:

✓ interesa concatenar tuplas de tabla1 y tabla2 por condiciones distintas de la igualdad.

✓ los atributos por los que se desea concatenar no tienen el mismo nombre en ambas relaciones.

Ejemplo: Obtener nombre y director de todos los equipos que tengan ciclistas.

Forma ya conocida:

```
SELECT e.nomeq, e.director
FROM Equipo E, Ciclista C
WHERE E.nomeq=C.nomeq
```

Empleo del JOIN:

```
SELECT e.nomeq, e.director
FROM  Equipo E  JOIN Ciclista C  ON E.nomeq=C.nomeq
```

Concatenación Externa

referencia_tabla [natural]
{left [outer] |
right [outer] |
full [outer] } JOIN ***referencia_tabla***
[on ***expresión_condicional*** | using
(***comalista_columna***)]

FULL, se muestran las tuplas no concatenadas de *tabla1* y *tabla2*

Cont. Concatenación externa sobre la izquierda:

tabla1 **left join** *tabla2* **on** *expresión_condicional*

(Concat. interna de *tabla1* y *tabla2*)

union

(tuplas de la *tabla1* que no están en la concatenación interna
con valores nulos en el resto de columnas

RIGHT ≡ **LEFT** con la diferencia de que las tuplas que se muestran son las de *tabla2*.

Concatenación Unión

referencia_tabla **union join** *referencia_tabla*

Crea una tabla donde el esquema es la unión de los esquemas de las dos tablas, que pueden ser distintos.

tabla1 **union join** *tabla2*

tuplas de *tabla1* con valores nulos en las columnas de *tabla2*

union

tuplas de *tabla2* con valores nulos en las columnas de *tabla1*

EJERCICIO 33: Obtener nombre de todos los equipos indicando cuantos ciclistas tiene cada uno

1ª Opción (errónea) – 21 filas

```
SELECT e.nomeq, count(c.dorsal)
FROM Equipo E, Ciclista C
WHERE E.nomeq=C.nomeq
GROUP BY e.nomeq
```

El equipo PDM con 0 ciclistas no aparece

2ª Opción Correcta – 22 filas

```
SELECT E.nomeq, count(c.dorsal)
FROM (EQUIPO E left join ciclista c on E.NOMEQ=C.NOMEQ)
GROUP BY E.nomeq;
```

EJERCICIO 33: Obtener nombre de todos los equipos indicando cuantos ciclistas tiene cada uno

3ª Opción Correcta – 22 filas

```
SELECT e.nomeq, count(c.dorsal)
FROM Equipo E, Ciclista C
WHERE E.nomeq=C.nomeq  (+)
GROUP BY e.nomeq
```

Se pone (+) en la parte de la ICA de la tabla de la que no se quieren mantener las tuplas