**UD3 BASES DE DATOS: INTRODUCCIÓN A SQL**

**SELECT** \*

**FROM** climatologia;

**SELECT** sirve para designar a las columnas de una tabla. El \* sirve para designar a todas las columnas de esa tabla.

**FROM** sirve para designar la tabla.

**SELECT** fecha, estacion, temperatura\_media

**FROM** climatologia;

Tras SELECT, en lugar de escribir \* se pueden escribir el nombre de las columnas que queremos que se muestren en pantalla.

**SELECT** estacion **AS** "ciudad", temperatura\_media **AS** "media", fecha

**FROM** climatologia;

**SELECT … AS “...”** sirve para poder nombrar una columna de la forma que queramos. ES COMO UN ALIAS, NO SE CAMBIA EL NOMBRE DE LA COLUMNA, SOLAMENTE LA PODEMOS NOMBRAR DE OTRA FORMA.

**SELECT DISTINCT** fecha

**FROM** climatologia;

**SELECT** \*

**FROM** climatologia

**ORDER BY** fecha;

**ORDER BY** sirve para que las columnas que vayan a ser mostradas se ordenen por una específica.

**SELECT** \*

**FROM** climatologia

**ORDER BY** fecha **DESC**, provincia, estacion

**ORDER BY … DESC, …, …** sirve para que (si algún valor de la columna elegida en el ORDER BY es repetido) se ordenen en lugar de por la columna del ORDER BY, por la que se haya escrito tras el DESC, y si algún valor de esta también se repite, se ordenen por la que se encuentre escrita tras esa. (en este caso “estacion”).

**SELECT** \*

**FROM** productos

**WHERE** precio < 10;

**WHERE** significa “donde…”, y sirve para aplicar un requisito. En este caso, mostrar todas las columnas de la tabla “productos” donde el precio sea menor a 10.

**SELECT** \*, salario\*14 **AS** “salario anual”

**FROM** empleados;

Si tras seleccionar todas las columnas con el \* y seguido escribimos una coma, se puede crear una columna inventada siempre y cuando escribamos de dónde salen sus datos. En este caso, la columna “salario” de la tabla “empleados” se ha multiplicado por 14, y se le ha dado el nombre de “salario anual”, dando como resultado esa nueva columna.

**SELECT** estacion || provincia

**FROM** climatologia;

El **operador ||** sirve para unir 2 columnas en una sola. Seguido de un AS y entre comillas dobles, se le puede llamar a esta nueva columna como uno quiera.

Los datos saldrían impresos de esta forma: estacionprovincia.

**SELECT** fecha, estacion || '(' || provincia || ')' **AS** "Estacion (provincia)", temperatura\_media

**FROM** climatologia;

Con respecto a los paréntesis y las comillas simples, si se concatenan con ||, sirven para que los datos se impriman de forma correcta, simplemente. En este caso, los datos saldrían impresos con el siguiente formato: estacion (provincia)**.**

**SELECT** provincia, anio,

(hombres**::numeric**/(hombres+mujeres))\*100 **AS** "porcentaje hombres",

(mujeres**::numeric**/(hombres+mujeres))\*100 **AS** "porcentaje mujeres"

**FROM** demografia\_basica;

El comando **::numeric** convierte un número entero en decimal. Por lo demás, esta sería la forma de usar operadores matemáticos, SI QUEREMOS CAMBIAR ALGÚN DATO O CREAR NUEVOS, VAN EN DENTRO DEL SELECT.

**SELECT COUNT** (\*)

**FROM** climatologia;

El comando **COUNT** sirve para que se cuenten el número de filas de las columnas que se pidan y almacenar esa cuenta en una nueva columna llamada “count”. En este caso, se cuenta el número de filas de todas las columnas de la tabla “climatologia”.

**TEMA 4**

**SELECT** \*

**FROM** employees, departments;

Producto Cartesiano entre las tablas “employees” y “departments”.

El **producto cartesiano** da como resultado otra tabla cuyo esquema es la concatenación de las tablas que se han multiplicado.

**NO SE PUEDE HACER EL “JOIN” DE 2 TABLAS QUE NO TENGAN UNA REFERENCIA FK → PK**

**SELECT** nombre, apellido, departamento

**FROM** departamentos

**CROSS JOIN** empleados

**ORDER BY** nombre, apellido, departamento;

**SELECT** cod\_curso, titulo, n\_curso

**FROM** tipos\_curso t

**NATURAL JOIN** cursos c;

**SELECT** e.nombre, a.nombre

**FROM** empresas e

**JOIN** alumnos a **USING**(cif);

**JOIN** **USING** si **nombre** **FK = nombre PK**.

**JOIN ON** si **nombre** **FK ≠ nombre PK**.

**TEMA 4**

LIMIT, OFFSET, GROUP BY

SELECT \*

FROM reserva

WHERE EXTRACT (year FROM fecha\_reserva) = 2024

ORDER BY fecha\_reserva DESC

LIMIT 50 OFFSET 2\*50;

SELECT COUNT(COALESCE(descuento, 0))

FROM vuelo;

SELECT AVG(descuento), AVG(COALESCE(descuento, 0))

FROM vuelo;

SELECT COUNT (DISTINCT descuento)

FROM vuelo;

SELECT ciudad, COUNT(\*)

FROM vuelo JOIN aeropuerto ON (desde = id\_aeropuerto)

GROUP BY ciudad;

SELECT ciudad, COUNT(\*)

FROM vuelo JOIN aeropuerto ON (desde = id\_aeropuerto)

GROUP BY ciudad;

/\*¿Cuántas personas han viajado en cada trayecto? Si hay más de un

vuelo por trayecto, el total de todos los vuelos.\*/

SELECT origen.ciudad, destino.ciudad, COUNT(\*)

FROM vuelo v JOIN aeropuerto origen ON (desde=origen.id\_aeropuerto)

JOIN aeropuerto destino ON (hasta=origen.id\_aeropuerto)

JOIN reserva r USING (id\_vuelo)

SELECT TO\_CHAR(salida, 'ID') AS "dia\_semana", COUNT(\*) AS "num\_vuelos"

FROM vuelo

GROUP BY dia\_semana

**HAVING COUNT(\*) >= 10**

ORDER BY dia\_semana;

HAVING sirve para ejecutar condiciones sobre aquellas columnas que dispongan de una función u operación. (Las operaciones como MIN, SUM, MAX, COUNT, AVG…). En este caso, se pide en el SELECT contar el número de vuelos por día de la semana, pero sólo queremos que se muestren por pantalla aquellos días en los que hayan salido más de diez. Pues para ello utilizamos HAVING.

CON HAVING LOS ALIAS IMPUESTOS EN EL SELECT NO FUNCIONAN.

SELECT EXTRACT(MONTH from salida) AS "mes", COUNT(\*)

FROM vuelo

GROUP BY mes

ORDER BY mes;

SELECT EXTRACT(MONTH from salida) AS "mes", EXTRACT(DAY from salida) AS "dia",

COUNT(\*)

FROM vuelo

GROUP BY mes, dia

ORDER BY mes, dia;

SELECT TO\_CHAR(salida, 'DD/MM') AS "dia\_y\_mes", COUNT(\*)

FROM vuelo

GROUP BY dia\_y\_mes

ORDER BY dia\_y\_mes;

SELECT TO\_CHAR(salida, 'dy') AS "dia\_semana", COUNT(\*)

FROM vuelo

GROUP BY dia\_semana

ORDER BY dia\_semana;

SELECT TO\_CHAR(salida, 'ID') AS "dia\_semana", COUNT(\*) AS "num\_vuelos"

FROM vuelo

GROUP BY dia\_semana

HAVING COUNT(\*) >= 10

ORDER BY dia\_semana;

**SUBCONSULTAS**

**1-** Selecciona los datos de aquellos empleados que tengan un salario mayor que John Chen y menor que Nancy Greenberg.

**SELECT** \*

**FROM** employees

**WHERE** salary > (**SELECT** salary

**FROM** employees

**WHERE** first\_name = 'John'

**AND** last\_name = 'Chen'

)

**AND** salary < (**SELECT** salary

**FROM** employees

**WHERE** first\_name = 'Nancy'

**AND** last\_name = 'Greenberg'

);

/\*Seleccionar la media de vuelos que sale cada día, independientemente

del aeropuerto del que salga el vuelo.\*/

SELECT dia\_semana, AVG(cantidad)

FROM (SELECT COUNT(\*) AS "cantidad", TO\_CHAR(salida,'Day') AS "dia\_semana",

salida::date, EXTRACT(isodow from salida) as "ndia"

FROM vuelo

GROUP BY TO\_CHAR(salida,'Day'), salida::date,

EXTRACT(isodow from salida) AS "ndia"

)

GROUP BY dia\_semana, ndia

ORDER BY ndia;

/\*Seleccionar aquellos empleados que cobran más que todos los

empleados del departamento "Purchasing".\*/

SELECT \*

FROM employees

WHERE salary > ALL (SELECT salary

FROM employees JOIN departments

USING (department\_id)

WHERE department\_name = 'Purchasing'

);

/\*Seleccionar, para cada provincia, el año en el que han

tenido más habitantes (hombres + mujeres.)\*/

SELECT \*, hombres + mujeres AS "n\_habitantes"

FROM demografia\_basica db1

WHERE hombres + mujeres

>= ALL (SELECT hombres + mujeres

FROM demografia\_basica db2

WHERE db1.provincia = db2.provincia

)

ORDER BY provincia;

-- Seleccionar el producto de cada

-- categoría del que más unidades

-- se han vendido.

-- Debe aparecer

-- Nombre categoría, nombre producto, nº total

-- de unidades.

SELECT category\_name, product\_name,

SUM(quantity)

FROM categories c JOIN products USING (category\_id)

JOIN order\_details USING (product\_id)

GROUP BY category\_name, product\_name

HAVING SUM(quantity) >= ALL (

SELECT SUM(quantity)

FROM categories c2 JOIN

products USING (category\_id)

JOIN order\_details USING (product\_id)

WHERE c.category\_name =

c2.category\_name

GROUP BY category\_name, product\_name

)

/\*Seleccionar, para cada estación meteorológica, la fecha en la que ha tenido

una temperatura\_máxima menor. Debe aparecer la provincia, el nombre de la

estación, la fecha y la temperatura\_máxima.\*/

SELECT provincia, estacion, fecha, temperatura\_maxima

FROM climatologia c1

WHERE temperatura\_maxima

<= ALL (SELECT temperatura\_maxima

FROM climatologia c2

WHERE c2.estacion = c1.estacion

);

**PÁGINA EXPLICACIÓN SUBCONSULTAS ¡MUY BUENA! :P**

<https://josejuansanchez.org/bd/unidad-09-teoria/index.html#operadores-que-podemos-usar-en-las-subconsultas>

**SI EN UNA CONSULTA CON UNA SUBCONSULTA NOS DA UN ERROR QUE DICE QUE ESTA ÚLTIMA ESTÁ UTILIZANDO UNA COLUMNA QUE NO ESTÁ AGRUPADA (QUE NO ESTÁ EN EL GROUP BY DE LA CONSULTA, no de la subconsulta, DE LA CONSULTA), ENTONCES ESCRIBIMOS EL NOMBRE DE DICHA COLUMNA EN EL GROUP BY DE LA CONSULTA AUNQUE NO APAREZCA EN EL SELECT.**

**EJEMPLO:**

SELECT c.nombre, c.apellido1, c.apellido2, a.nombre,

a.ciudad,

SUM(precio\*(1-(COALESCE(descuento, 0)/100))) AS "cuantia"

FROM vuelo v JOIN reserva r USING (id\_vuelo)

JOIN cliente c USING (id\_cliente)

JOIN aeropuerto a ON (desde=id\_aeropuerto)

GROUP BY c.nombre, c.apellido1, c.apellido2, a.nombre,

a.ciudad, **v.desde**

HAVING SUM(precio\*(1-(COALESCE(descuento, 0)/100)))

>= ALL (SELECT SUM(precio\*(1-(COALESCE(descuento, 0)/100)))

FROM vuelo v2 JOIN reserva r USING (id\_vuelo)

WHERE **v.desde** = v2.desde

GROUP BY id\_cliente);

La columna **v.desde** de la consulta principal es utilizada en la subconsulta, pero como no se utiliza para nada en la consulta principal.

Para que v.desde pueda ser utilizada en la subconsulta necesita aparecer en la consulta principal en el GROUP BY, por lo que, para ello, escribimos su nombre allí.

**EJEMPLOS CUTROSOS (UD4)**

CREATE TABLE alumno (

cod\_alumno SERIAL,

nombre VARCHAR(200),

apellido1 VARCHAR(200),

apellido2 VARCHAR(200),

nombre\_completo VARCHAR(600)

GENERATED ALWAYS AS (nombre || ' ' || apellido1 || ' ' || apellido2) STORED,

fecha\_nacimiento DATE,

edad\_31\_diciembre SMALLINT,

email VARCHAR(320)

);

**BASE DE DATOS (ALUMNO)**

DROP TABLE IF EXISTS alumno CASCADE;

CREATE TABLE alumno (

id\_alumno SERIAL,

nombre TEXT,

CONSTRAINT pk\_alumno PRIMARY KEY (id\_alumno)

);

DROP TABLE IF EXISTS asignatura CASCADE;

CREATE TABLE asignatura (

id\_asignatura SERIAL,

nombre VARCHAR(100),

profesor VARCHAR(200),

CONSTRAINT pk\_asignatura PRIMARY KEY (id\_asignatura)

);

DROP TABLE IF EXISTS matricula CASCADE;

CREATE TABLE matricula (

id\_alumno INTEGER,

id\_asignatura INTEGER,

anio\_escolar VARCHAR(10),

CONSTRAINT pk\_matricula

PRIMARY KEY (id\_alumno, id\_asignatura, anio\_escolar),

CONSTRAINT fk\_matricula\_alumno

FOREIGN KEY (id\_alumno) REFERENCES alumno,

CONSTRAINT fk\_matricula\_asignatura

FOREIGN KEY (id\_asignatura) REFERENCES asignatura

);

DROP TABLE IF EXISTS nota CASCADE;

CREATE TABLE nota (

id\_alumno INTEGER,

id\_asignatura INTEGER,

anio\_escolar VARCHAR(10),

tipo\_evaluacion VARCHAR(1),

nota NUMERIC(4,2),

CONSTRAINT pk\_nota PRIMARY KEY

(id\_alumno, id\_asignatura,

anio\_escolar, tipo\_evaluacion),

CONSTRAINT ck\_nota\_tipo\_evaluacion

CHECK (tipo\_evaluacion IN ('1','2','3','F')),

CONSTRAINT fk\_nota\_matricula

FOREIGN KEY (id\_alumno, id\_asignatura,

anio\_escolar)

REFERENCES matricula (id\_alumno,

id\_asignatura, anio\_escolar)

);