Módulo 3. SQL

Introducción

En el módulo 3 conoceremos el lenguaje estructurado de consulta (SQL) y cómo este contiene diferentes lenguajes para la definición (DDL) y manipulación (DML) de los datos. Veremos además los diferentes tipos de datos para luego aprender a crear esquemas, tablas y realizar restricciones. Algunas restricciones comunes son las claves primarias y foráneas, y la creación de índices. Realizaremos asimismo consultas con SQL. Ampliaremos en operaciones cómo proyección, selección, operadores SQL y operaciones de conjunto. Por último veremos consultas con uniones (*joins*) de tablas, tipos de uniones existentes y consultas agrupadas, funciones de grupo y ordenamiento.

Etapa 2 Probemos cómo funciona nuestra base de datos

Video de inmersión

Unidad 1. Creación de estructura (DDL)

Modificación de estructuras de tablas

Tema 1. Introducción. Tipos de datos SQL

Introducción a SQL

Del inglés *Structured Query Language* (SQL) es el lenguaje estándar para trabajar con bases de datos relacionales y es soportado prácticamente por todos los productos en el mercado. Originalmente, SQL fue desarrollado en IBM *Research* a principios de los años setenta. Se usó por primera vez a gran escala en un prototipo de IBM denominado System R, luego fue aplicado en numerosos productos comerciales de IBM y de otros fabricantes. En este capítulo presentamos una introducción al lenguaje SQL, los tipos de datos y cómo generar la estructura de una base de datos. Esto último es lo que se conoce por sus siglas en inglés como DDL (*Data Definition Language*). En cuanto a la actualización, se especifica también otro lenguaje, el DML (*Data Manipulation Language*) que nos permite modificar el estado de la base de datos.

Para comprender mejor un concepto, además de su definición, siempre es útil conocer su contexto e historia. Para poder entender mejor qué es SQL veamos un artículo define SQL y muestra algo de historia del estándar ANSI.

¿Qué es el lenguaje SQL?

Fuente: **CampusMVP.** (2014) *¿Qué es el lenguaje SQL?* Recuperado de: https://www.campusmvp.es/recursos/post/Que-es-el-lenguaje-SQL.aspx

Situación actual de los estándares SQL3 y SQL/MM

Tipos de datos en SQL

Las bases de datos almacenan datos mediante registros, pueden ser distintos en su característica, soporte, etc. Así, como un aspecto previo es necesario conocer la naturaleza de los distintos tipos de datos.

Los DBMS proveen distintos tipos de datos con los cuales podemos trabajar, sin embargo, es necesario especificar cuál nos conviene más y para eso debemos de saber sus características en búsqueda, capacidad, uso de los recursos, etc.

Cada DMBS introduce tipos de valores de campos que no se encuentran precisamente presentes en otras. A pesar de esto, existe

un conjunto de tipos que están presentes en la totalidad de las implementaciones de bases de datos. En la tabla 1 podemos ver estos tipos comunes:

Tabla 1: Tipos de datos genéricos en SQL

Alfanuméricos	Contienen cifras y letras. Presentan una longitud limitada (*varchar*, *varchar2*, *text*). En general son similares, pero pueden variar en cada DBMS
Numéricos	Existen de varios tipos, principalmente, enteros (sin decimales) y reales (con decimales). Algunos ejemplos son *short, int, bigint, decimal*.
Booleanos	Poseen dos formas: Verdadero y falso (Sí o No). *Boolean*.
Fechas	Almacenan fechas facilitando posteriormente su explotación. Por lo general, los DBMS también incorporan funciones para manipular fechas, comparar y obtener un valor particular (día, mes, año).
Memos	Son campos alfanuméricos de longitud ilimitada. Presentan el inconveniente de no poder ser indexados (veremos más adelante lo que esto quiere decir).
Autoincrementales	Son campos numéricos enteros que incrementan en una unidad (o más) su valor para cada registro incorporado. Su principal utilidad es la de identificador, ya que resultan exclusivos de un registro.

Fuente: elaboración propia

En este curso en particular, recomendamos el uso del motor Postgre SQL. Como en todos los administradores de bases de datos, implementa los tipos de datos definidos para el estándar SQL3 y aumenta algunos otros. Detallando algunos de los tipos mencionados antes veamos los tipos definidos por el estándar SQL3 que se muestran en la tabla 2.

Tabla 2: Tipos de datos en Postgre SQL

Tipos de datos del estándar SQL3 en *Postgre* SQL		
Tipo en Postgres	**Correspondiente en SQL3**	**Descripción**
Bool	**Boolean**	valor lógico o booleano (*true*/*false*)
char(n)	**character(n)**	cadena de caracteres de tamaño fijo
Date	**Date**	fecha (sin hora)
float4/8	**float(86#86)**	número de punto flotante con precisión 86#86
float8	**real, double precisión**	número de punto flotante de doble precisión
int2	**Smallint**	entero de dos *bytes* con signo
int4	**int, integer**	entero de cuatro *bytes* con signo

int4	**decimal(87#87)**	número exacto con 88#88
int4	**numeric(87#87)**	número exacto con 89#89
money	**decimal(9,2)**	cantidad monetaria
time	**time**	hora en horas, minutos, segundos y centésimas
Time stamp	**Interval**	intervalo de tiempo
timestamp	**timestamp with time zone**	fecha y hora con zonificación
varchar(n)	**character varying(n)**	cadena de caracteres de tamaño variable

Fuente: elaboración propia.

Tema 2. Creación y relaciones entre tablas

Tablas

La estructura más elemental de almacenamiento en un Sistema de Administración de Base de Datos (DBMS) es la **tabla**.

Los datos son almacenados en las tablas por medio de filas y columnas. La definición de las tablas se realiza con un nombre que la identifica unívocamente y un conjunto de columnas. Una vez creada se le pueden insertar datos en las filas. Sobre las filas de las tablas se pueden hacer operaciones de consulta, eliminación o actualización de datos.

☰ Columna

☰ Fila

☰ Campo

Según vimos antes, es necesario un lenguaje que nos permita crear tanto la estructura, es decir, las tablas, como el acceso, consulta y actualización de los datos, además de la inserción, selección y eliminación de tuplas o registros. SQL provee ambos lenguajes (DDL y DML). Comencemos viendo unos ejemplos de cómo crear tablas a través de SQL.

Esquema o base de datos

Para crear nuestras tablas primero tendremos que hacer un esquema de base de datos y teniendo en cuenta el motor utilizado cada esquema puede tener varias bases de datos. Previo a las tablas veamos las sentencias necesarias para crear el esquema.

```
CREATE SCHEMA {<nombre_esquema>|AUTHORIZATION<usuario>}
[<lista_elementos_esquema>];
```

La nomenclatura utilizada en esta sentencia y, de aquí en adelante, es la siguiente:

- Las palabras en negrita son palabras reservadas del lenguaje.
- La notación [...] quiere decir que lo que hay entre los corchetes se podría poner o no.
- La notación {A|...|B} quiere decir que tenemos que escoger entre todas las opciones que hay entre las llaves. Pero tenemos que poner una obligatoriamente.
- La notación <A> quiere decir que A es un elemento pendiente de definición.

Como ya hemos visto, la estructura de almacenamiento de los datos del modelo relacional son las tablas. Para crear una tabla hay que utilizar la sentencia *create table*. Veamos el formato:

```
CREATE TABLE <nombre_tabla> (
<definición_columna>
    [, <definición_columna>...] [, |
<restricciones_tabla>]
    );
```

Donde la definición de columna es:

```
<nombre_columna>{<tipo_datos>|<dominio>} [<def_defecto>] [<restricciones_columna>]
```

El proceso que hay que seguir para crear una tabla es el siguiente:

1. Decidir el nombre de la tabla (nombre tabla).
2. Dar nombre a cada uno de los atributos que formarán las columnas de la tabla (nombre columna).
3. Asignar a cada una de las columnas un tipo de datos predefinido o bien un dominio definido por el usuario. También se puede dar definiciones por defecto y restricciones de columna.
4. Una vez definidas las columnas, solo habrá que dar las restricciones de tabla.

Si queremos crear una tabla para almacenar los empleados de una empresa, donde cada empleado tiene obligatoriamente que tener un DNI y fecha de ingreso y opcionalmente se almacenará su nombre, su apellido y su edad, podríamos tener la siguiente sentencia SQL (Dataprix, 2009, https://bit.ly/2UYOFnY).

```
CREATE TABLE EMPLEADO (
   DNI CHAR(12) NOT NULL,
   NOMBRE CHAR(50),
   APELLIDO CHAR(50),
   EDAD INT,
   FECHA_INGRESO TIMESTAMP NOT NULL
);
```

Las columnas que **no** pueden tener valores nulos, se indica con las palabras reservadas ***not null***.

Esquemas y/o base de datos

Restricciones de columna

En cada una de las columnas de la tabla, una vez que les hemos dado un nombre y hemos definido el dominio, podemos imponer ciertas restricciones que siempre se tendrán que cumplir. Las restricciones que se pueden dar son las que aparecen en la tabla que aparecen a continuación:

Tabla 3: Restricciones de columna

Restricciones de columna	
Restricción	Descripción
Not null	La columna no puede tener valores nulos.

Unique	La columna no puede tener valores repetidos. Es una clave alternativa.
Primary key	La columna no puede tener valores repetidos ni nulos. Es la clave primaria.
References <nombre_tabla> [(<nombre_columna>)]	La columna es la clave foránea de la columna de la tabla especificada.
Constraint [<nombre_restricción>] *Check* (<condiciones>)	La columna tiene que cumplir las condiciones especificadas.

Fuente: elaboración propia con base en Dataprix, 2009.

Restricciones de columna

Tema 3. Claves primarias, secundarias y foráneas

Clave primaria: la clave primaria está compuesta por una o más columnas que permiten identificar de manera unívoca a cada fila de una tabla, por ejemplo, el código de barras de un artículo. La clave primaria es única por tabla y siempre debe contener un valor.

Clave foránea: la clave foránea está compuesta por una o más columnas que hacen referencia a una **clave primaria** de otra tabla o de la misma, por eso la relación entre las tablas que las contienen se conoce como padre/hijo. Las claves foráneas tienen el fin de ayudar al cumplimiento de las reglas de diseño de la base de datos relacional. Se puede contar con más de una clave foránea por tabla. Estas claves surgen de la necesidad de relacionar diferentes tablas. Como vimos en el ejemplo de curso-alumno (en la lectura 2) la forma de relacionar una tabla con otra es a través de este tipo de claves.

Restricciones de tabla

Una vez hemos dado un nombre, definido un dominio y hemos impuesto ciertas restricciones para cada una de las columnas, podemos aplicar restricciones sobre toda la tabla que siempre se tendrán que cumplir. Las restricciones que se pueden dar son las siguientes:

Tabla 4: Restricciones de tabla

Restricciones de tabla	
Restricción	Descripción
Unique (<nombre_columna> [, <nombre_columna>...])	El conjunto de las columnas especificadas no puede tener valores repetidos. Es una clave alternativa.
Primary key (<nombre_columna> [, <nombre_columna>...])	El conjunto de las columnas especificadas no puede tener valores nulos ni repetidos. Es una clave primaria.

Foreing key (<nombre_columna>[, <nombre_columna>...]) *References*<nombre_tabla> [(<nombre_columna2> [, <nombre_columna2>...])]	El conjunto de las columnas especificadas es una clave foránea que referencia la clave primaria formada por el conjunto de las columnas 2 de la tabla dada. Si las columnas y las columnas 2 se llaman exactamente igual, entonces no habría que poner columnas2.
Constraint[<nombre_restricción>] *Check*(<condiciones>)	La tabla tiene que cumplir las condiciones especificadas.

Fuente: elaboración propia con base en Dataprix, 2009.

Ejercicio propuesto: agregar a la tabla de empleado vista antes restricciones de primary key al atributo DNI y de unique al par nombre-apellido.

Tema 4. Índices y vistas

Vistas

Los índices y las vistas son mecanismos que facilitan el acceso a los datos. Para el caso de los índices, como vimos en el módulo 1, sirven para mejorar la performance de la base de datos, ya que son estructuras auxiliares de rápido acceso. El caso de las vistas nos permite resumir o conjugar diferentes consultas agrupándolas con un mismo nombre lógico. Veamos la estructura de la consulta SQL y algunos ejemplos.

```
CREATEVIEW<nombre_vista>[(lista_colum
nas)]AS(consulta) [WITH CHECKOPTION];
```

Vistas

Lo primero que tenemos que hacer para crear una vista es decidir qué nombre le queremos poner (nombre_vista). Si buscamos cambiar el nombre de las columnas, o bien poner nombre a alguna que en principio no tenía. Lo podemos hacer en lista de columnas. A partir de ahí solo nos quedará definir la consulta que formará nuestra vista.

Las vistas no existen realmente como un conjunto de valores almacenados en la BD, son tablas ficticias denominadas derivadas (no materializadas). Se construyen a partir de tablas reales (materializadas) almacenadas en la BD y conocidas con el nombre de tablas básicas (o tablas de base) (Dataprix, 2009). La no-existencia real de las vistas hace que puedan ser actualizables o no.

Creamos una vista que de, por ejemplo, para cada cliente el número de proyectos que tiene encargados el cliente.

```
CREATEVIEWproyectos_por_cliente(codigo_cli, numero_proyectos) AS (SELECT
c.codigo_cli, COUNT(*)
FROM proyectos p, clientes c
WHERE p.codigo_cliente = c.codigo_cli GROUP BY
c.codigo_cli);
```

Más adelante detallaremos, pero vemos en esta vista varios componentes nuevos. La consulta propiamente dicha (*select*) retorna lo indicado en el enunciado y nos permite consultar los datos directamente utilizando la vista "proyectos por cliente" si tuviésemos las siguientes extensiones:

Tabla 5: Clientes

Clientes					
codigo_cli	**nombre_cli**	**Nif**	**dirección**	**ciudad**	**teléfono**
10	ECIGSA	38.567.893-C	Aragón 11	Barcelona	NULL
20	CME	38.123.898-E	Valencia 22	Gerona	972.23.57.21
30	ACME	36.432.127-A	Mallorca 33	Lérida	973.23.45.67

Fuente: Programación, 2015, https://bit.ly/2OXHQ23

Tabla 6: Proyectos

Proyectos						
codigo_proy	**nombre_proy**	**precio**	**fecha_inicio**	**fecha_prev_fin**	**fecha_fin**	**codigo_cliente**
1	GESCOM	1,0E+6	1-1-98	1-1-99	NULL	10
2	PESCI	2,0E+6	1-10-96	31-3-98	1-5-98	10
3	SALSA	1,0E+6	10-2-98	1-2-99	NULL	20
4	TINELL	4,0E+6	1-1-97	1-12-99	NULL	30

Fuente: Programación, 2015, https://bit.ly/2OXHQ23

Y si mirásemos la extensión de la vista proyectos por clientes veríamos lo que encontramos en la siguiente tabla.

Tabla 7: Proyectos por clientes

proyectos_por_clientes	
codigo_cli	**numero_proyectos**
10	2
20	1
30	1

Fuente: elaboración propia.

Índices

Los índices, si bien siempre están asociados a una tabla, son objetos independientes, ya sea desde el punto de vista lógico como del punto de vista físico. Ellos se pueden crear o eliminar sin que afecten a la información almacenada en una tabla.

El DBMS utiliza el índice como se puede utilizar el índice de un libro. El índice contiene una entrada por cada valor que aparece en las columnas indexadas de la tabla y punteros (*orowid*) a las filas que contienen esos valores. Cada puntero conduce directamente a la fila apropiada; en consecuencia, se evita el barrido total de la tabla (Oracle, 2017, https://bit.ly/2SZuT9a).

En el índice los valores de los datos están dispuestos en orden ascendente o descendente, de modo que el DBMS es capaz de

buscar rápidamente el índice para encontrar un valor particular. A continuación, puede seguir el puntero para localizar la fila que contiene el valor.

Índices

Desventajas de los índices: consume espacio adicional en disco. Deben actualizarse cada vez que se agrega una fila a la tabla. Impone una sobrecarga adicional en *inserts* y *updates*.

Tipos de índices

Algunos productos DBMS soportan dos o más tipos diferentes de índices, que están optimizados para diferentes tipos de acceso a la base de datos.

Árbol B (*B*Tree*): es el tipo predeterminado en casi todos los productos de DBMS. Utiliza una estructura de árbol de entradas de índice y bloques de índice (grupos de entradas de índice) para organizar los valores de datos que contiene en orden ascendente o descendente. Proporciona una búsqueda eficiente de un valor único o de un intervalo de valores, como lo es la búsqueda requerida para un operador de comparación de desigualdad o una operación de prueba de intervalo (*between*).e

La sentencia asigna un nombre al índice y especifica la tabla para la que se crea el índice. La sentencia también especifica las columnas a indexar y si deben ser indexadas en orden ascendente o descendente. Veamos unos ejemplos de creación de índices:

- Una columna indexada: cree un índice único para la tabla *offices*.

CREATE UNIQUE INDEX OFC_MGR_IDX ON OFFICES (MGR);

- Dos columnas indexadas: cree un índice para la tabla orders.

CREATE INDEX ORD_PROD_IDX ON ORDERS (MFR, PRODUCT);

Si crea un índice para una tabla y posteriormente decide que no es necesario, la instrucción *drop index* elimina el índice de la base de datos. La instrucción elimina el índice creado en el ejemplo anterior: ***DROP INDEX ORD_PROD_IDX*.**

Unidad 2. Consultas y manipulación de datos (DML)

Para manipular los datos de cualquier base de datos se cuenta con dos operaciones básicas: las de recuperación y las de inserción o actualización de datos. Para realizar estas operaciones utilizamos el lenguaje SQL, específicamente el sublenguaje que permite manipular los datos (*Data Manipulation Language*).

La sentencia del lenguaje utilizada para la recuperación de datos a partir de la base de datos es la sentencia *select*. SQL posee diversos operadores que pueden ser utilizados al escribir una consulta SQL o *select*. La sentencia *select*, además, posee cláusulas que permiten restringir y ordenar el conjunto resultado, lo que permite presentar el resultado de una consulta según el formato requerido por el usuario final de la base de datos. SQL también provee la facilidad de invocar funciones, programas que reciben argumentos y calculan o retornan un valor en la sentencia *select*. La invocación de funciones brinda mayor flexibilidad y potencia al lenguaje a la vez que provee un mecanismo de extensión.

> Los datos en una base de datos relacional se almacenan en diferentes tablas que se encuentran relacionadas por valores en común. Utilizando la sentencia *select* es posible escribir consultas que unen dos o más tablas relacionadas y presentan la información de manera unificada.
>
> Aparte del acceso a datos almacenados en diferentes tablas es posible agregar o agrupar datos según valores

comunes, de esta manera se pueden calcular valores sumarizados en el grupo de filas. La sentencia *select* posee una cláusula para especificar el agrupamiento de datos y existen diferentes funciones de grupo para llevar a cabo cálculos en el grupo. Este tipo de consultas son las que se denominan **agrupadas**. (Oracle, 2017, https://bit.ly/39VnGDf)

Estructura de una consulta:

Select {lista de columnas}

From Tablas

Where Condiciones - Filas ***Group By*** {lista de columnas} ***Having*** Condiciones - Grupo ***Order By*** {lista de columnas}

Todas las palabras marcadas en negrita son reservadas del lenguaje SQL y el orden que se ve es el que debemos seguir al momento de crear nuestras consultas.

Tema 1. Selección y proyección

Para hacer consultas sobre una tabla con el SQL es preciso utilizar la sentencia *select from* que tiene el siguiente formato:

SELECTnombre_columna_a_seleccionar [nombre_columna_a_seleccionar...]
FROM tabla_a_consultar
WHERE restricciones;

La selección es el proceso por el cual elegimos qué registros tomamos de una tabla en particular. Esto se hace a través del ***where***, es opcional, ya que si no se escribe indica que queremos todos las tuplas o registros de una tabla. La proyección es la que nos permite elegir qué campos o atributos mostrar de una determinada tabla, se hace a través del ***select***. Por último, pero no menos importante, aparece el ***from***. A través de esta palabra indicamos sobre qué tabla estamos realizando la consulta. Veamos unos ejemplos.

Si tenemos una tabla clientes (código, nombre, dirección, ciudad) y queremos listar todos los clientes mostrando todas las columnas, simplemente basta con escribir:

*Select**
From clientes;

Vemos que con "*" indicamos todas las columnas. Ahora, si quisiéramos ver solo el código del cliente y el nombre, sería así:

Select código, nombre
From clientes;

Por último, si quisiéramos solo ver aquellos códigos de clientes de la ciudad de Buenos Aires, se vería:

Select código
From clientes Where ciudad = "Buenos Aires".

Condiciones y operadores

Para definir las condiciones en la cláusula *where* podemos utilizar los operadores que dispone SQL.

Tabla 8: Condiciones y operadores

Operadores lógicos	
Not	Para la negación de condiciones

And	Para la conjunción de condiciones
Or	Para la disyunción de condiciones

Fuente: elaboración propia

Tabla 9: Condiciones y operadores

Operadores de comparación	
=	Igual
<	Menor
>	Mayor
<=	Menor o igual
>=	Mayor o igual
<>	Distinto

Fuente: elaboración propia.

Tema 2. Operaciones de conjuntos

Las operaciones de conjuntos son operaciones derivadas de la matemática de conjuntos. Existen múltiples operaciones de conjunto. En este tema veremos que SQL reserva una palabra en particular, la unión, intersección y diferencia. En el próximo tema veremos las reuniones (del inglés *join*) que se llevan a cabo entre diferentes tablas. Las desarrolladas a continuación tienen como condición necesaria que ambos conjuntos posean la misma cantidad y el mismo tipo de columnas.

Unión

La cláusula *union* permite unir consultas de dos o más sentencias *select from*. Su formato es el siguiente:

```
SELECT <nombre columnas>
FROM <tabla>
[WHERE <condiciones>]
UNION [ALL]
SELECT <nombre_columnas>
FROM <tabla>
[WHERE <condiciones>];
```

Si se utiliza la opción *all* aparecen todas las filas obtenidas al hacer la unión. No se escribirá esta opción si se quieren eliminar las filas repetidas. Lo más importante de la unión es que somos nosotros los que tenemos que vigilar que se haga entre columnas definidas sobre dominios compatibles; es decir, que tengan la misma interpretación semántica. Como ya se ha dicho, SQL no ofrece herramientas para asegurar la compatibilidad semántica entre columnas.

La intersección

Para hacer la intersección entre dos o más sentencias *select from*, se puede utilizar la cláusula *intersect* cuyo formato es el siguiente:

```
SELECT <nombre_columnas>
FROM <tabla>
[WHERE <condiciones>]
INTERSECT [ALL]
SELECT <nombre_columnas>
FROM <tabla>
[WHERE <condiciones>];
```

Si se utiliza la opción *all* aparecen todas las filas obtenidas al hacer la intersección. No se escribirá esta opción si se quieren eliminar las filas repetidas. En la intersección es de suma importancia tener en cuenta que somos nosotros los que tenemos que vigilar que se aplique entre columnas definidas sobre dominios compatibles, es decir, que tengan la misma interpretación semántica.

La diferencia

Para encontrar la diferencia entre dos o más sentencias *select from* podemos utilizar la cláusula *except* que tiene este formato:

```
SELECT <nombre_columnas>
FROM <tabla>
[WHERE <condiciones>]
EXCEPT [ALL]
SELECT <nombre_columnas>
FROM <tabla>
[WHERE <condiciones>];
```

Si se utiliza la opción *all*, aparecen todas las filas obtenidas al hacer la diferencia. No se escribirá esta opción si se quieren eliminar las filas repetidas. Lo más importante de la diferencia es que somos nosotros los que tenemos que vigilar que se haga entre columnas definidas sobre dominios compatibles.

Modificación de registros

Veamos cómo podemos insertar y modificar los registros de una base de datos:

Inserción de Registros

Para hacer agregar registros en una tabla con el SQL es preciso utilizar la sentencia *insert into* que tiene el siguiente formato:

```
INSERT INTO nombre_tabla
[(nombre_columna [,….])
VALUES ({expresión|DEFAULT} [,….]);
```

Veamos unos ejemplos.

Si tenemos una tabla clientes (código, nombre, dirección, ciudad) y queremos insertar un registro simplemente basta con escribir:

```
INSERT INTO clientes VALUES (1, “PICAPIEDRAS PEDRO”, “AV.RIVADAVIA 12334”, “MENDOZA”);
```

O si no quiero insertar todas las columnas escribiría:

```
INSERT INTO clientes (código, nombre)
VALUES (2, “PICAPIEDRAS VILMA”);
```

Eliminación o borrado de Registros

Para eliminar o borrar registros en una tabla con el SQL es preciso utilizar la sentencia *delete from* que tiene el siguiente formato:

```
DELETE FROM nombre_tabla
WHERE restricciones o condiciones;
```

Veamos unos ejemplos.

Si queremos borrar el registro de la tabla Clientes cuyo código sea igual a 1, simplemente basta con escribir:

```
DELETE FROM clientes WHERE CODIGO=1;
```

Si quiero borrar todos los registros de una tabla escribiría:

```
DELETE FROM CLIENTES;
```

Para borrar todos los registros de una tabla se puede utilizar la sentencia Truncate, que es mas rápida que un delete:

```
TRUNCATE TABLE CLIENTES;
```

Si le agrego CASCADE borra los registros que hacen referencia a la tabla via foreign key

```
TRUNCATE TABLE CLIENTES CASCADE;
```

A diferencia del comando DELETE, donde se guardan los ID's eliminados en el log de transacciones, un TRUNCATE TABLE solo marca las páginas que contenían los datos, para que se puedan reocupar y no queda logueado en el log de transacciones por lo tanto no tiene vuelta atrás.

Actualización de Registros

Para actualizar registros de una tabla con el SQL es preciso utilizar la sentencia *update* que tiene el siguiente formato:

```
UPDATE nombre_tabla
SET { column_name = { expression | DEFAULT } |
      ( column_name [, ...] ) = [ ROW ] ( { expression | DEFAULT } [, ...] ) |
      ( column_name [, ...] ) = ( sub-SELECT )
    }
WHERE restricciones o condiciones;
```

Veamos unos ejemplos.

Si queremos actualizar el nombre del registro de la tabla Clientes cuyo código sea igual a 1, simplemente basta con escribir:

```
UPDATE clientes
SET NOMBRE=”PICAPIEDRAS CARLOS”
WHERE CODIGO=1;
```

Si actualizar más de un dato de una tabla escribiría:

```
UPDATE clientes
SET NOMBRE=”PICAPIEDRAS CARLOS”, LOCALIDAD=”CORDOBA”
WHERE CODIGO=1;
```

Otros ejemplos: incrementa la cantidad de ventas del vendedor que administra la cuenta:

```
UPDATE empleado SET cant_ventas = cant_ventas + 1 FROM cuentas
 WHERE cuentas.nombre = 'Acme Corporation'
 AND empleado.id = cuentas.vendedor;
```

Realizo la misma operación pero con un sub-select:

```
UPDATE empleado SET cant_ventas = cant_ventas + 1 WHERE id =
 (SELECT vendedor FROM cuentas WHERE name = 'Acme Corporation');
```

En las operaciones de conjunto las columnas de los conjuntos a operar deben ser las mismas.

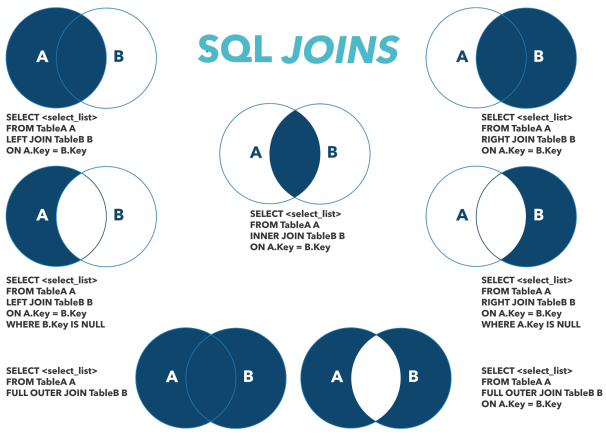
Verdadero

Falso

Justificación

Tema 3. Uniones entre tablas (joins)

Figura 1. SQL Joins



Fuente: [imagen sin título sobre uniones entre tablas], 2019, https://bit.ly/3wwHW5J

Muchas veces queremos consultar datos de más de una tabla haciendo combinaciones de columnas de diferentes tablas. En SQL es posible listar más de una tabla especificándolo en la cláusula *from*.

Combinación

La combinación consigue crear una sola tabla a partir de las tablas especificadas en la cláusula *from*. Hace coincidir los valores de las columnas relacionadas de estas tablas.

Ejemplo de una combinación (*join*):

Se quiere saber el NIF, el código y el precio del proyecto que se desarrolla para el cliente número 20:

```
SELECT proyectos.codigo_proy, proyectos.precio, clientes.nif FROM clientes,
proyectos
WHERE clientes.codigo_cli = proyectos.codigo_cliente
AND clientes.codigo_cli = 20;
```

El resultado sería el siguiente:

Tabla 10: Resultado

proyectos.codigo_proy	proyectos.precio	clientes.nif
3	1000000.0	38.123.898-E

Fuente: elaboración propia.

Si trabajamos con más de una tabla puede pasar que la tabla resultante tenga dos columnas con el mismo nombre. Por eso, es obligatorio especificar a qué tabla corresponden las columnas a las que nos referimos, se nombra la tabla a la que pertenecen antes de ponerlas (por ejemplo, clientes.codigo_cli). Para simplificarlo se utilizan los alias que, en este caso, se definen en la cláusula *from*.

Nota: un alias se define automáticamente después de nombrar la tabla dentro del *from* (ejemplo *from* persona p, p es un alias para persona).

La opción *on*, además de expresar condiciones con la igualdad en caso de que las columnas que queremos relacionar tengan nombres diferentes, nos ofrece la posibilidad de expresar condiciones con los otros operadores de comparación que no sean el de igualdad. También podemos utilizar una misma tabla dos veces con alias diferentes, para poder distinguirlas.

Combinación natural

La combinación natural (*natural join*) de dos tablas consiste básicamente en hacer una equicombinación entre columnas del mismo nombre y eliminar las columnas repetidas. La combinación natural, utilizando SQL: 1992, se haría de la siguiente manera:

```
SELECT <nombre_columnas_seleccionar>FROM <tabla1>NATURAL JOIN <tabla2> [WHERE
<condiciones>];
```

Ejemplo de una combinación natural: se quiere saber el código y el nombre de los empleados que están asignados al departamento que tiene por teléfono 977.333.852.

```
SELECT codigo_empl,nombre_empl
FROM empleados NATURAL JOIN departamentos WHERE telefono ="977.333.852";
```

Combinación interna y externa

Cualquier combinación puede ser interna o externa. La combinación interna (*inner join*) solo se queda con las filas que tienen valores idénticos en las columnas de las tablas que compara. Esto puede hacer que se pierda alguna fila interesante de alguna de

las dos tablas, por ejemplo, porque se encuentra *null* en el momento de hacer la combinación. Su formato es el siguiente:

```
SELECT <nombre_columnas_seleccionar>
FROM <tabla1> [NATURAL] [INNER] JOIN <tabla2>
                                {ON <condiciones>|
        USING (<nombre_columna> [,<nombre_columna>...])} [WHERE
<condiciones>];
```

Si no se quiere perder ninguna fila en una tabla determinada se puede hacer una combinación externa (*outer join*). Permite obtener todos los valores de la tabla que hemos puesto a la derecha, los valores de la que hemos puesto a la izquierda o todos los valores de ambas tablas. Su formato es el siguiente:

```
SELECT <nombre_columnas_seleccionar>
FROM <tabla1> [NATURAL] {LEFT|RIGHT|FULL} [OUTER] JOIN <tabla2>
                        {ON <condiciones>|
     USING (<nombre_columna> [,<nombre_columna>...])} [WHERE
<condiciones>];
```

Combinaciones con más de dos tablas

Si se quieren combinar tres tablas o más solo hay que añadir todas las tablas en el *from* y los vínculos necesarios en el *where*. Hay que ir haciendo combinaciones por parejas de tablas y la tabla resultante se convertirá en la primera pareja de la siguiente.

Veamos ejemplos de los dos casos. Supongamos que se quieren combinar tablas de empleados, proyectos y clientes:

```
SELECT *
FROM empleados, proyectos, clientes
WHERE num_proy = codigo_proy AND codigo_cliente = codigo_cli;
```

o bien,

```
SELECT *
FROM empleados, proyectos, clientes
WHERE num_proy = codigo_proy AND codigo_cliente = codigo_cli;
```

Tema 4. Consultas agrupadas y ordenadas

Funciones de agregación

SQL ofrece las siguientes funciones de agregación para efectuar diferentes operaciones con los datos de una BD:

Tabla 11: Funciones de agregación

Funciones de agregación	
Función	Descripción
Count	Nos da el número total de filas seleccionadas
Sum	Suma los valores de una columna
Min	Nos da el valor mínimo de una columna
Max	Nos da el valor máximo de una columna

AVG	Calcula la media de una columna

Fuente: elaboración propia.

En general las funciones de agregación se aplican a una columna, excepto la función de agregación *count* que normalmente se aplica a todas las columnas de la tabla o tablas seleccionadas: *count* (*) contaría las filas de la tabla o tablas que cumplan las condiciones. Veamos un ejemplo de uso de la función *count* que aparece en la cláusula *select* para hacer la consulta: ¿cuántos departamentos están ubicados en la ciudad de Lérida?

```
SELECT COUNT(*) AS numero_dpt FROM departamentos
WHERE ciudad_dpt = "Lerida";
```

La respuesta a esta consulta es la siguiente:

numero_dept

Consultas con agrupación de filas de una tabla

Las cláusulas que añadimos a la sentencia *select from* permiten organizar las filas por grupos:

- La cláusula *group by* nos sirve para agrupar filas según las columnas que indique esta cláusula.
- La cláusula *having* especifica condiciones de búsqueda para grupos de filas; lleva a cabo la misma función que antes hacía la cláusula *where* para las filas de toda la tabla, pero ahora las condiciones se aplican a los grupos obtenidos.

Presenta el siguiente formato:

```
SELECT <nombre_columnas_seleccionar>
FROM <tabla_consultar> [WHERE <condiciones>]


GROUP BY <columnas_según_las_que_agrupar> [HAVING
<condiciones_por_grupos>]
[ORDER BY <nombre_columna> [DESC][, <nombre_columna> [DESC]...]];
```

Prestemos especial atención a que en las sentencias SQL se van añadiendo cláusulas a medida que la dificultad o la exigencia de la consulta lo requiere.

Consulta con agrupación de filas: se quiere saber el sueldo medio que ganan los empleados de cada departamento.

```
SELECT nombre_dpt, ciudad_dpt, AVG(sueldo) AS sueldo_medio FROM empleados
GROUP BY nombre_dpt, ciudad_dpt;
```

Factores de agrupación

Los factores de agrupación de la cláusula `group by` tienen que ser, como mínimo, las columnas que figuran en `select`, exceptuando las columnas afectadas por las funciones de agregación.

La cláusula *group by* podríamos decir que empaqueta a los empleados de la tabla empleados según el

departamento en que están asignados. En la siguiente figura podemos ver los grupos que se formarían después de agrupar.

Tabla 12: Empleados

empleados						
codigo_empl	nombre_empl	apellido_empl	sueldo	nombre_dpt	ciudad_dpt	num_proy
1	Maria	Puig	100000.0	DIR	Gerona	1
2	Pedro	Mas	90000.0	DIR	Barcelona	4
3	Ana	Ros	70000.0	DIS	Lerida	3
4	Jorge	Roca	70000.0	DIS	Barcelona	4
5	Clara	Blanc	40000.0	PROG	Tarragona	1
6	Laura	Tort	30000.0	PROG	Tarragona	3
8	Sergio	Grau	30000.0	PROG	Tarragona	NULL
7	Roger	Salt	40000.0	NULL	NULL	4

Fuente: elaboración propia.

Y así con la agrupación anterior es sencillo obtener el resultado de esta consulta:

Tabla 13: Resultados

nombre_dpt	ciudad_dpt	sueldo_medio
DIR	Gerona	100 000.0
DIR	Barcelona	90 000.0
DIS	Lérida	70 000.0
DIS	Barcelona	70 000.0
PROG	Tarragona	33 000.0
NULL	NULL	40 000.0

Fuente: elaboración propia.

Ejemplo de uso de la función de agregación **SUM**

Veamos un ejemplo de uso de una función de agregación SUM de SQL que aparece en la cláusula *having* de *group by*: queremos saber los códigos de los proyectos en los cuales la suma de los sueldos de los empleados es mayor de EUR 180 000.

```
SELECT
num_proy FROM
empleados GROUP BY
num_proy
    HAVING SUM(sueldo)>180000.0;
```

Nuevamente, antes de mostrar el resultado de esta consulta veamos gráficamente qué grupos se formarían en este caso:

Tabla 14: Empleados

empleados						
codigo_empl	nombre_empl	apellido_empl	sueldo	nombre_dpt	ciudad_dpt	num_proy
1	Maria	Puig	100000.0	DIR	Gerona	1
5	Clara	Blanc	40000.0	PROG	Tarragona	1
3	Ana	Ros	70000.0	DIS	Lerida	3
6	Laura	Tort	30000.0	PROG	Tarragona	3
2	Pedro	Mas	90000.0	DIR	Barcelona	4
4	Jorge	Roca	70000.0	DIS	Barcelona	4
7	Roger	Salt	40000.0	NULL	NULL	4
8	Sergio	Grau	30000.0	PROG	Tarragona	NULL

Fuente: elaboración propia.

Consultas ordenadas

Si se quiere que, al hacer una consulta, los datos aparezcan en un orden determinado, hay que utilizar la cláusula *order by* en la sentencia *select* que tiene el siguiente formato:

```
SELECT <nombre_columnas_seleccionar>
FROM <tabla_consultar> [WHERE
<condiciones>]
ORDER BY <nombre_columna_según_la_que_ordenar> [DESC]
                                    [, <nombre_columna_según_la_que_ordenar>
[DESC]...];
```

Ejemplo: se quiere consultar los nombres de los empleados ordenados según el sueldo que ganan, y si ganan el mismo sueldo, ordenados alfabéticamente por el nombre.

```
SELECT codigo_empl, nombre_empl,
apellido_empl, sueldo FROM empleados
ORDER BY sueldo, nombre_empl;
```

Esta consulta daría la siguiente respuesta:

Tabla 15: Resultados

codigo_empl	nombre_empl	apellido_empl	sueldo
6	Laura	Tort	30 000.0
8	Sergio	Grau	30 000.0
5	Clara	Blanc	40 000.0
7	Roger	Salt	40 000.0
3	Ana	Ros	70 000.0
4	Jorge	Roca	70 000.0
2	Pedro	Más	90 000.0
1	María	Puig	10 000.0

Fuente: elaboración propia.

Si no se especifica nada más el orden que se seguirá será ascendente, pero si se quiere seguir un orden descendente hay que añadir *desc* detrás de cada factor de ordenación expresado en la cláusula *order by*:

También se puede explicitar un orden ascendente poniendo la palabra clave ASC (opción por defecto).

```
ORDER BY <nombre_columna> [DESC][, <nombre_columna> [DESC]...];
```

Microactividades

La plataforma que utilizaremos será: https://extendsclass.com/postgresql/e176003

Si utilizamos pgAdmin todas las tablas, datos, funciones, procedimientos almacenados y *triggers* serán guardados en nuestra computadora de forma tal que podremos acceder a ellos indiferentemente si cerramos el programa o reiniciamos nuestros sistemas.

En el caso de utilizar la herramienta *online*, deberán guardar las consultas que vayan realizando, ya que, al cerrar el navegador se perderán todos los datos y se deberá copiar, pegar y ejecutar las consultas que ya habíamos ejecutado (y que guardado en un documento de texto cualquiera de los disponibles). De esta forma podrán avanzar en las próximas microactividades sin necesidad de volver a escribirlas una a una nuevamente.

A lo largo de todas las microactividades se debe crear y trabajar sobre un sistema para una universidad.

Desde la universidad Puls-ar nos solicitan llevar el registro de los alumnos y las materias que ellos cursan. Cuando los alumnos se inscriben se les solicita su DNI, nombre, apellido y dirección de correo electrónico.

Dado de alta en la universidad ellos ya están en condiciones de anotarse a las materias disponibles (teniendo en cuenta que las materias tienen un nombre y un año de cursado).

1. Identifique las entidades y sus atributos, nombrarlos siempre en minúsculas.
2. Crear las tablas de las entidades encontradas sabiendo que el año de cursado se representa como un número entero, los nombres de las materias nunca exceden los 30 caracteres, el nombre, apellido y email no tendrán más de 50 caracteres y el DNI lo consideraremos como cadena de caracteres. Ningún atributo de las entidades admite valor nulo y se utilizará como clave primaria en cada tabla un entero autoincremental.
3. Encontrar la/s relaciones entre tablas y, de ser necesario, crear la estructura.
4. Agregar los datos de alumnos y materias para tener la siguiente estructura:

Tabla 1. Alumnos

id. Alumno	dni	nombre	apellido	email
1	12345678	Pedro	Picapiedras	pedro@picapiedras.com
2	21334441	Vilma	Picapiedras	vilma@picapiedras.com
3	54354332	Pablo	Marmol	pablo@marmol.com
4	54353444	Betty	Marmol	betty@marmol.com
5	54354355	Pebbles	Picapiedras	pebbles@picapiedras.com
6	54354353	Ban Ban	Marmol	banban@marmol.com
7	54354335	Stephen	King	stephen@redrose.com
8	89786786	Clara	González	clara.gonzalez@gmail.com
9	12344325	Alicia	Pascualli	aliciapasc@hotmail.com
10	24665412	Andrés	Calamaro	calamaro@gmail.com
11	24665413	Fabiana	Cantilo	fabiana_cantilo@gmail.com
12	24665414	Gustavo	Cerati	cerati@sodastereo.com

Fuente: elaboración propia.

Tabla 2. Materias

Id. Materia	nombre	anio_cursado
1	Física 1	1
2	Física 2	2
3	Matemáticas 1	1
4	Matemáticas 2	2
5	Algebra 1	1
6	Algebra 2	2
7	Química General	1
8	Química Inorgánica	1
9	Química Orgánica	2
10	Termo Física	2
11	Integración 1	1
12	Integración 2	2
13	Programación Lógica	1
14	Base de Datos	1

Fuente: elaboración propia.

5. Inscribir a los alumnos en las materias para obtener la siguiente estructura.

Tabla 3. Ejercicio

ALUMNO	MATERIA
Pedro Picapiedras	Física 1
Pedro Picapiedras	Matemática 2
Pedro Picapiedras	Algebra 1
Pedro Picapiedras	Integración 1
Pedro Picapiedras	Base de Datos
Vilma Picapiedras	Física 1
Vilma Picapiedras	Física 2
Vilma Picapiedras	Algebra 1
Vilma Picapiedras	Algebra 2
Vilma Picapiedras	Termo Física
Vilma Picapiedras	Base de Datos
Stephen King	Matemática 1
Stephen King	Matemática 2
Stephen King	Química Inorgánica 1
Stephen King	Química Orgánica 2
Stephen King	Integración 1

Fuente: elaboración propia.

6. Obtener la cantidad de alumnos que cursan Física 1

Tabla 4. Cantidad

cantidad
2

Fuente: elaboración propia.

7. Obtener la cantidad de materias en la que está inscripto el alumno Stephen King.

8. Obtener un listado de la cantidad de alumnos inscriptos a cada materia ordenando según nombre, materia y año de cursado ascendente.

Tabla 5. Ejercicio

Año	Anio_cursado	Cantidad
Algebra 1	1	2
Algebra 2	2	1
Base de Datos	1	2
Física 1	1	2
Física 2	2	1
Integración 1	1	2
Matemática 1	2	1
Matemática 2	1	2
Química Inorgánica 1	2	1
Química Orgánica 2	1	1
Termo Física	2	1

Fuente: elaboración propia.

9. Modificar la tabla alumnos de forma tal que ahora el campo email admita hasta 80 caracteres.

10. Modificar la tabla alumnos agregando la fecha de ingreso del alumno.

11. Modificar la tabla alumnos: setearle a la fecha de ingreso como defecto la fecha del día si el usuario no ingresa ninguna. Luego inserte un nuevo alumno y verifique el correcto funcionamiento: DNI 1122334411, nombre=Minnie, apellido=Mouse, email=minnie@mousel.com.

12. De la consulta obtenida anteriormente crear una vista llamada alumnos_cantidad_materias y posteriormente utilizar la misma para obtener solo las materias donde haya más de 1 inscripto.

Tabla 6. Ejercicio

nombre	anio_cursado	cantidad
Algebra	1	2
Base de Datos	1	2
Física	1	2
Integración	1	2
Matemáticas	2	2
Termo Física	2	2

Fuente: elaboración propia.

13. Agregue los registros que correspondan para que el alumno Ban Ban esté en las materias:

Tabla 7. Ejercicio

6	54354353	Ban Ban	Marmol	banban@marmol.com

Fuente: elaboración propia.

Tabla 8. Ejercicio

id	nombre	anio_cursado
1	Física	1
2	Física	2
3	Matemáticas	1
4	Matemáticas	2
5	Algebra	1
6	Algebra	2
7	Química General	1
8	Química Inorgánica	1
9	Química Orgánica	2
10	Termo Física	2

Fuente: elaboración propia.

14. Muestre la cantidad de alumnos que tienen más de 4 materias usando la vista ya creada.

Resolución

Video de habilidades

Video 1: Algebra relacional

Fuente: Gabriela Flores. (2013). Algebra Relacional (proyección). [Video de YouTube]. Recuperado de https://www.youtube.com/watch?v=CDhl_IvH2Uo

Debemos lograr llevar las operaciones de conjunto o del algebra relacional al plano de la información que se almacena en la base de datos con el fin de poder cruzar y organizar la información. Las bases de datos basan todas sus operaciones en al alegra relacional.

Preguntas de habilidades video 1

La proyección no permite eliminar atributos.

- Verdadero.
- Falso.
- **Justificación**

La proyección brinda un subconjunto horizontal de los atributos de la relación

- Verdadero.
- Falso

Justificación

¿Qué operación restringe de manera horizontal o a nivel de filas o registros?

Unión

Producto Cartesiano

Selección

Diferencia

Proyección

Justificación

¿No es posible combinar dos operaciones del álgebra relacional?

Verdadero

Falso

Justificación

La operación que combina todas las tuplas de la relación T con cada una de las tuplas de la relación R es Producto Cartesiano

Verdadero

Falso

Justificación

Videos 2: Ejemplos de consultas SQL

Fuente: Merche Marqués Andrés. (2014). Consultas en SQL: SELECT, FROM, WHERE. [Video de YouTube]. Recuperado de https://www.youtube.com/watch?v=U8lgFaLW_Qg

Es muy importante adquirir práctica en el armado de las consultas y sentencia SQL. La sentencia que posee más complejidad es el Select por eso haremos más hincapié en ella. Tenemos que ser capaces de poder combinar más de una tabla y saber hacer las agrupaciones o restricciones necesarias.

Preguntas de habilidades video 2

Si deseo buscar la materia administración de redes ¿qué sentencia debo escribir y mostrar todos los campos de la tabla?

Select * from clases where materia = ‘administracion de redes’

Select Count* from clases where materia = ‘administracion de redes’

¿Qué sentencia me permite eliminar las filas duplicadas?

Distinct

Union

Sum

Avg

Count

Justificación

Se deseamos ordenar los datos por una columna en particular debemos escribir Order by (campo por el cual ordenar)

Verdadero

Falso

Justificación

Escribe una sentencia que muestre todas las materias de programacion , mostrando como resultado solo la columna materia

Select materia from clases where materia like ‘programacion%’

Select materia from where materia like ‘programacion%’

Escribe una sentencia que muestre la materia elaboración de texto avanzado y la materia manejo de db , mostrando como resultado todas las columnas de la tabla clases.

Select * from clases where materia = ‘elaboración de texto avanzado’ or materia = ‘manejo de db’

From* clases where materia = ‘elaboración de texto avanzado’ or materia = ‘manejo de db’

Cierre

A lo largo de la lectura estudiamos las operaciones cómo proyección, selección, operadores SQL y operaciones de conjunto y aprendimos sobre uniones (*joins*) de tablas, tipos de uniones existentes y consultas agrupadas, funciones de grupo y ordenamiento.

Glosario

Referencias

[Imagen sin título sobre uniones entre tablas], (2019). Recuperado de https://deployadmin.com/2019/07/07/sql-joins-memo/

CampusMVP. (2014) *¿Qué es el lenguaje SQL?* Recuperado de https://www.campusmvp.es/recursos/post/Que-es-el-lenguaje-SQL.aspx

Dataprix. (2009) *Creación de tablas*. Recuperado de https://www.dataprix.com/es/book/export/html/654

Oracle. (2017) *Objetos de bases de datos*. Recuperado de http://docplayer.es/2135056-5-objetos-de-base-de-datos.html

Programación*. (2015) SQL Creación y borrado de vistas Creación de una vista en BDUOC. Recuperado de https://ayudaparaprogramacion.blogspot.com/2015/07/sql-creacion-y-borrado-de-vistas.html*