

EXERCISES QUE TRABAJAREMOS EN EL CUE:

- EXERCISE 1: CREAR BASE DE DATOS.
- EXERCISE 2: EXPRESIONES Y FUNCIONES DEL LENGUAJE SQL.
- EXERCISE 3: ALGEBRA RELACIONAL.

EXERCISE 1: CREAR BASE DE DATOS.

Como ya sabemos, las instrucciones DDL son aquellas encargadas de la creación, manipulación y eliminación de los objetos de una base de datos. En esta oportunidad, conoceremos la sentencia CREATE, que es fundamental para crear Bases de Datos y Tablas.

SENTENCIA CREATE

Esta es una tarea administrativa. Crear la base de datos implica indicar los archivos y ubicaciones que se utilizarán para la misma, además de otras indicaciones técnicas y administrativas, las cuales no se comentarán en este tema.

Lógicamente, solo es posible crear una base de datos si se tienen privilegios de DBA (DataBase Administrator).

El comando SQL de creación de una base de datos con el nombre que se le indique, es el siguiente:

```
1 create database prueba;
```

En gran parte de los sistemas, eso basta para crear la base de datos; en nuestro caso, utilizaremos la herramienta pgAdmin4.

OBJETOS DE LA BASE DE DATOS

Una base de datos es un conjunto de objetos pensados para gestionar datos. Éstos se encuentran contenidos en esquemas, los cuales suelen estar asociados al perfil de un usuario en particular.

En SQL estándar, existe el concepto de catálogo, que sirve para almacenar esquemas, y éstos a su vez, se ocupan para almacenar objetos. Así, el nombre completo de un objeto vendría dado por: catálogo – esquema - objeto. Es decir, los objetos pertenecen a esquemas, y éstos a catálogos.

En casi todos los sistemas de bases de datos hay un catálogo por defecto, de modo que, si no se indica alguno en específico al crear objetos, éstos se almacenan allí. De la misma forma, hay esquemas por defecto.

Los esquemas y catálogos de PostgreSQL están relacionados con las Bases de Datos: cada BD posee un esquema pensado para almacenar sus objetos. Aunque podemos almacenar (si tenemos permisos para ello) objetos en otros esquemas, éstos (tablas, vistas, etc.) estarán normalmente en el nuestro.

CREAR TABLAS

El nombre debe cumplir las siguientes reglas (se comentan de Oracle, en otros SGBD podrían cambiar):

- Deben comenzar con una letra.
- No deben tener más de 63 caracteres.
- Solo se permiten utilizar letras del alfabeto (inglés), números, o el signo de subrayado (también los signos \$ y #, pero de manera especial, por lo que no son recomendados).
- No puede haber dos tablas con el mismo nombre dentro del mismo esquema (pueden coincidir los nombres si están en distintos esquemas).
- No puede coincidir con el nombre de una palabra reservada SQL (por ejemplo, no se puede llamar **SELECT** a una tabla).
- En el caso de que el nombre tenga espacios en blanco, o caracteres nacionales (permitido solo en algunas bases de datos), se suele colocar con comillas dobles. En el estándar SQL se pueden utilizar comillas dobles al poner el nombre de la tabla, a fin de hacerla sensible a las mayúsculas (se diferenciará entre “FACTURAS” y “Facturas”).

Éstas son también las propiedades que debe cumplir cualquier nombre de objeto en una base de datos: nombres de vistas, columnas, restricciones, ...

COMANDO PARA CREAR UNA TABLA:

CREATE TABLE: es la orden SQL que permite crear una tabla. Por defecto, ésta será almacenada en el espacio y esquema del usuario que crea la tabla. Su sintaxis es:

```
1 CREATE TABLE [esquema] nombreDeTabla (  
2     nombreDeLaColumna1 tipoDeDatos [DEFAULT valor]  
3     [restricciones] [...]  
4 );
```

Ejemplo:

Crea una tabla con un solo campo de tipo **VARCHAR**.

```
1 CREATE TABLE proveedores (nombre VARCHAR(25));
```

Esta tabla solo se podrá crear si el usuario posee los permisos necesarios para ello. Si ésta pertenece a otro esquema (suponiendo que el usuario tenga permiso para grabar tablas en ese), se antepone a su nombre, el nombre del esquema:

```
1 CREATE TABLE otroEsquema.proveedores (  
2     nombre VARCHAR(25)  
3 );
```

Se puede indicar un valor por defecto para el atributo, mediante la cláusula **DEFAULT**.

Ejemplo:

```
1 CREATE TABLE Proveedores (  
2     nombre VARCHAR(25),  
3     localidad VARCHAR(30) DEFAULT 'Santiago'  
4 );
```

De este modo, si añadimos un proveedor y no indicamos localidad, se tomará Santiago como referente.

Podemos utilizar **DEFAULT** y usar funciones del sistema. Por ejemplo:

```

1 CREATE TABLE Prestamos(
2   id_prestamo NUMERIC(8),
3   fecha_prestamo DATE DEFAULT current_date
4 );
  
```

TIPOS DE DATOS

A la hora de crear tablas, hay que indicar el tipo de datos de cada campo, y para ello necesitamos conocer los distintos tipos que hay. Éstos pueden ser numéricos, o caracteres alfanuméricos, entre otros:

Nombre	Tamaño	Descripción	Rango
smallint	2 bytes	Número entero de pequeño rango	-32768 a +32767
integer	4 bytes	Opción típica para números enteros	-2147483648 a +2147483647
bigint	8 bytes	Número entero de gran rango	-9223372036854775808 a +9223372036854775807
decimal	variable	Precisión especificada por usuario, exacto	Hasta 131072 dígitos antes del punto decimal; hasta 16383 dígitos después del punto decimal
numeric	variable	Precisión especificada por usuario, exacto.	Hasta 131072 dígitos antes del punto decimal; hasta 16383 dígitos después del punto decimal.
real	4 bytes	Precisión variable, inexacto.	Precisión de 6 dígitos decimales.
double precision	8 bytes	Precisión variable, inexacto.	Precisión de 15 dígitos decimales.
smallserial	2 bytes	Número entero de pequeño rango auto incrementable.	1 a 32767
serial	4 bytes	Número entero auto incrementable.	1 a 2147483647
bigserial	8 bytes	Número entero de gran rango auto incrementable.	1 a 9223372036854775807

Nombre	Descripción
character varying(<i>n</i>), varchar(<i>n</i>)	Cadena de caracteres alfanuméricos de longitud variable limitada.
character(<i>n</i>), char(<i>n</i>)	Cadena de caracteres alfanuméricos de tamaño y longitud fija.
text	Cadena de caracteres alfanuméricos de longitud variable ilimitada.

Ejemplos:

- **Varchar(2000)**, almacena un dato alfanumérico de largo variable, y de máximo 2000 caracteres.
- **Char(10)**, almacena un dato alfanumérico de largo fijo, y de máximo 10 caracteres.
- **Numeric(12,5)**, almacena un número con un máximo de 12 dígitos, incluidos los decimales, donde la cantidad máxima de decimales son 5, y la parte entera con un máximo de 7 dígitos.

SENTENCIA INSERT

Para añadir datos a una tabla, se realiza mediante la instrucción **INSERT**. Su sintaxis fundamental es:

```
1 INSERT INTO tabla [(listaDeColumnas)]  
2 VALUES (valor1 [,valor2 ...])
```

La tabla representa aquella donde queremos añadir el registro, y los valores que siguen a la cláusula **VALUES**, son los que damos a los distintos campos del registro. Si no se especifica la lista de campos, la lista de valores debe seguir el orden de las columnas según fueron creados (para conocer dicho orden, basta invocar al comando **DESCRIBE**).

La lista de campos a rellenar se indica si no queremos rellenar todas las columnas.

Las columnas no rellenadas explícitamente con la orden **INSERT**, se rellenan con su valor por defecto (**DEFAULT**), o con **NULL** si no se indicó valor por defecto alguno. Si alguna columna tiene restricción de obligatoriedad (**NOT NULL**), y no indicamos un valor, ocurrirá un error.

Ejemplos:

```
1 INSERT INTO clientes VALUES( '11111111', 'Pedro', 'Gutiérrez', 'Crespo',  
2 DEFAULT, NULL);  
3 INSERT INTO clientes (run, nombre, apellido1, apellido2)  
4 VALUES( '11111111', 'Pedro', 'Gutiérrez', 'Crespo');
```

Los valores por defecto se indican, durante la creación o modificación de la estructura de una tabla, a través de la palabra clave **DEFAULT**.

ESTABLECIMIENTO DE RESTRICCIONES

Definir restricciones: una restricción es una condición de obligado cumplimiento, para una o más columnas de la tabla.

Éstas se pueden realizar cuando estamos creando (**CREATE**), o modificando (**ALTER**) una tabla. Hay dos maneras de poner restricciones:

- Poner una restricción de columna: en este caso, la restricción se pone seguido a la definición de la columna. Su sintaxis:

```
1 columna tipo [DEFAULT expresión]  
2 [CONSTRAINT nombre] tipo,
```

- Poner una restricción de tabla: en este caso, se ponen al final de la lista de columnas. La única restricción que no se puede definir de esta forma es la de tipo **NOT NULL**. El resto, se harían siguiendo esta sintaxis:

columna1 definición1, columna2 definición2, ..., últimaColumna últimaDefinición.

```
1 [CONSTRAINT nombre] tipo (listaColumnas)
```

```
2 [...otras restricciones...]
```

La diferencia está en que, en el primer caso, no se especifica la lista de columnas al definir la restricción, esto es entendible pues las restricciones de columna se aplicarán a la columna en la que se definen.

PROHIBIR NULOS

La restricción **NOT NULL** permite prohibir los nulos en una determinada tabla. Eso obliga a que la columna deba tener un valor, para que efectivamente, sea almacenado el registro.

Se puede colocar durante la creación (o modificación) del campo, añadiendo la palabra **NOT NULL** tras el tipo:

```
1 CREATE TABLE cliente(  
2   dni VARCHAR2(9) CONSTRAINT nombre_restriccion NOT NULL  
3 );
```

La restricción **NOT NULL** es la única que solo se puede poner seguida al nombre de la columna a la que se aplica. La razón es que solo se puede aplicar a una columna a la vez.

VALORES ÚNICOS

Las restricciones de tipo **UNIQUE** obligan a que el contenido de una, o más columnas, no puedan repetir valores en distintas filas. Ejemplo:

```
1 CREATE TABLE alquiler(  
2   dni VARCHAR(9),  
3   cod_pelicula NUMERIC(5),  
4   CONSTRAINT nombre_restriccion UNIQUE(dni, cod_pelicula)  
5 );
```

CLAVE PRIMARIA

Está formada por las columnas que indican a cada registro de ésta. La clave primaria hace que los campos que la forman no puedan quedar vacíos, ni repetir valores. Además, pasan a formar parte del índice principal de la tabla, el cual se usa para acceder más rápidamente a los datos.

Sean **NOT NULL** (sin posibilidad de quedar vacíos), y que los valores de los campos sean de tipo **UNIQUE** (sin posibilidad de repetición).

Si la clave está formada por un solo campo, basta con:

```
1 CREATE TABLE clientes(  
2   dni VARCHAR(9) CONSTRAINT clientes_pk PRIMARY KEY,  
3   nombre VARCHAR(50)  
4 );
```

Y si la clave está formada por más de un campo:

```
1 CREATE TABLE alquileres(dni VARCHAR(9),  
2   cod_pelicula NUMBER(5),  
3   CONSTRAINT alquileres_pk PRIMARY KEY(dni,cod_pelicula)  
4 );
```

CLAVE SECUNDARIA O FORÁNEA

Se usa para indicar que uno, o más campos de una tabla, están relacionados con la clave principal (o incluso con una clave candidata) de otra tabla y, por lo tanto, no podrán contener valores que no estén relacionados en la otra tabla.

Este es un ejemplo de indicación de clave foránea:

```
1 CREATE TABLE alquileres(  
2   dni VARCHAR2(9) CONSTRAINT alquileres_fk1 REFERENCES   clientes(dni),  
3   cod_pelicula NUMBER(5) CONSTRAINT alquileres_fk2 REFERENCES  
4   peliculas(cod),  
5   CONSTRAINT alquileres_pk PRIMARY KEY(dni,cod_pelicula)  
6 );
```


Esto significa que el campo dni se relaciona con la columna dni de la tabla clientes, y el cod_película con la columna cod de la tabla películas.

Si el campo al que se hace referencia es la clave principal, se puede obviar su nombre.

```
1 CREATE TABLE alquileres(  
2   dni VARCHAR2(9) CONSTRAINT alquileres_fk1 REFERENCES clientes,  
3   cod_pelicula NUMBER(5) CONSTRAINT alquileres_fk2 REFERENCES peliculas,  
4   CONSTRAINT alquileres_pk PRIMARY KEY(dni,cod_pelicula)  
5 );
```

En este caso, se entiende que los campos hacen referencia a las claves principales de las tablas. Si la relación está formada por más de una columna, el orden de los campos debe ser el mismo, aunque, en este caso, es preferible indicar explícitamente el nombre.

De hecho, cuando una relación está formada por más de una columna, se debe indicar el nombre de cada una (como siempre ocurre en las restricciones de más de una columna) tras la lista de columnas de la tabla. Aunque cualquier restricción (sea de una sola columna o no), se puede indicar también al final.

Ejemplo:

```
1 CREATE TABLE existencias(  
2   tipo CHAR2(9),  
3   modelo NUMBER(3),  
4   n_almacen NUMBER(1)  
5   cantidad NUMBER(7),  
6   CONSTRAINT existencias_fk1 FOREIGN KEY(tipo,modelo) REFERENCES  
7   piezas,  
8   CONSTRAINT existencias_fk2 FOREIGN KEY(n_almacen) REFERENCES  
9   almacenes,  
10  CONSTRAINT existencias_pk PRIMARY KEY(tipo,modelo,n_almacen)  
11 );
```

Si la definición de clave secundaria se pone al final, es necesario colocar el texto **FOREIGN KEY** para indicar en qué campos se coloca la restricción de clave foránea. En el ejemplo anterior, es absolutamente necesario (al no indicar explícitamente la lista de columnas en el apartado REFERENCES) que la clave principal de la tabla piezas, a la que hace referencia la clave, la formen las columnas tipo y modelo, y en que estén en ese orden.

Las restricciones de tipo **FOREIGN KEY** provocan una restricción de integridad referencial, en la que no se pueden indicar datos en las claves secundarias, que no existan en las principales relacionadas.

La desventaja es que la integridad referencial provoca algunos problemas, debido a sus efectos secundarios.

Por ejemplo: supongamos que relacionamos el alquiler de habitaciones en una tabla de alquileres, con el dni de la persona que alquila, que será la clave de la tabla clientes. Bien, no podemos borrar una persona de la tabla de clientes que tenga alquileres, y tampoco podremos modificar su dni por la misma razón.

Ante esto, disponemos de la posibilidad de aplicar políticas especiales. Éstas son palabras claves que se colocan tras la cláusula **REFERENCES**, al añadir una restricción de tipo **FOREIGN KEY**.

Así, las políticas que dictan qué hacer cuando se borran datos principales relacionados con claves secundarias, son las siguientes:

- **ON DELETE SET NULL**: coloca nulos en todas las claves secundarias relacionadas.
- **ON DELETE CASCADE**: borra todas las filas relacionadas con aquella que hemos eliminado.
- **ON DELETE SET DEFAULT**: coloca en las filas relacionadas el valor por defecto de esa columna en la columna relacionada.
- **ON DELETE NOTHING**: no hace nada.

Las mismas se pueden aplicar en el caso de modificar claves principales. Así tendremos: **ON UPDATE DO NOTHING**, **ON UPDATE CASCADE**, **ON UPDATE SET NULL** y **ON UPDATE SET DEFAULT**.

Sin embargo, PostgreSQL solo dispone de las políticas **ON DELETE CASCADE** y **ON DELETE SET NULL**. Y por defecto, aplica **DO NOTHING**, tanto para borrar, como para modificar claves primarias. Por lo tanto, no posee ninguna acción para la modificación (**ON UPDATE**) de claves primarias.

Ejemplo de establecimiento de borrado en cascada, y de puesta a null:

```
1 CREATE TABLE alquileres(  
2   dni VARCHAR(9),  
3   cod_pelicula NUMBER(5),  
4   CONSTRAINT alquileres_pk PRIMARY KEY(dni, cod_pelicula),  
5   CONSTRAINT alquileres_fk1 FOREIGN KEY (dni)  
6   REFERENCES clientes(dni) ON DELETE SET NULL,
```

```

7 CONSTRAINT alquileres_fk2 FOREIGN KEY (cod_pelicula) REFERENCES
8 películas(cod) ON DELETE CASCADE
9 );

```

EXERCISE 2: EXPRESIONES Y FUNCIONES DEL LENGUAJE SQL.

EXPRESIONES Y OPERADORES

Una expresión es una combinación de símbolos y operadores, que un motor de base de datos evalúa para obtener un valor de datos único. Las expresiones simples pueden ser: una constante, variable, columna o función escalar. Y los operadores se pueden utilizar para unir dos o más expresiones simples en una compleja.

Veamos cómo se realizan las expresiones:

```

1 A + B, suma dos numeros con el operador es el signo +
2 TO_DATE('3/5/2007','DD/MM/YYYY'), funcion que convierte un String en
3 fecha

```

LOS OPERADORES BÁSICOS DE SQL SON:

Operador	Acción
+	Suma dos números
-	Resta dos números
*	Multiplifica dos números
/	Divide dos números
	Concatena dos cadenas de caracteres
<	Compara si una expresiones es menor que otra
>	Compara si una expresiones es mayor que otra
=	Compara si una expresiones es igual que otra
<=	Compara si una expresiones es menor o igual que otra
>=	Compara si una expresiones es mayor o igual que otra
<>	Compara si una expresiones es distinta a otra
IN(v1, v2, ..., vn)	Comprueba si una expresión existe dentro de un conjunto de expresiones.
LIKE	Comprueba si una expresión contiene patrón indicado
Between	Comprueba si una expresión está dentro de un rango
NOT	Negación
AND	Operador lógico Y
OR	Operador lógico O
IS NULL	Verdadero la expresión que lo antecede en nula

Ejemplo:

```
1 SELECT Nombre, Sueldo * 21/30 APagar, FechaContrato, Edad
2 FROM Empleados
3 WHERE
4 (Sueldo > 1000 /*Sueldo es mayor a 1000*/
5 AND Edad <= 65 /*Y Edad es menor o igual a 65*/
6 AND Vigente <> 'NO' /*Y Vigente es distinto de 'NO'*/
7 AND TipoEmpleado IN (1, 6, 9) /* Y TipoEmpleado es 1, 6 o 9*/
8 AND FechaIngreso Between '2020/01/01' AND '2020/12/31' /* Y Fecha
9 ingreso entre*/
10 AND Nombre LIKE 'A%' /* Y Nombre Comienza por A >=>Comodín*/
11 AND Apellido LIKE '%A%' /* Y Apellido Contiene una B */
12 )
13 OR /*O*/
14 (
15 Clase IS NOT NULL /*La clase no tenga un valor nulo*/
16 AND Tipo IS NULL /*El tipo tenga un valor nulo*/
17 )
```

SENTENCIA SELECT

Este comando permite:

- Obtener datos de ciertas columnas de una tabla (proyección).
- Obtener registros (filas) de una tabla, de acuerdo con ciertos criterios (selección).
- Mezclar datos de tablas diferentes (asociación, join).
- Realizar cálculos sobre los datos.
- Agrupar datos.

La sintaxis es:

```
1 SELECT [ALL | DISTINCT]
2 * | expresión [AS alias] [, expresión [AS alias]] ...
3 FROM tabla [AS alias] [, tabla [AS alias]]...
4 [WHERE condición]
5 [GROUP BY ( expresión | posición)]
6 [HAVING condición]
7 [ORDER BY ( columna | expresión | posición) [ASC | DESC], ...]
```

Dónde:

Operador	Acción
*	El asterisco significa que se seleccionan todas las columnas
ALL	Indica si se deben retornar todas las filas de la consulta
DISTINCT	Hace que no se muestren las filas duplicadas.
tabla	Nombre de una tabla de la base de datos
columna	Nombre de una columna perteneciente a una tabla
expresión	Expresión SQL, hay que recordar que una expresión puede ser una columna
condición	Una condición es una expresión que siempre retorna un valor lógico
alias	Nombre que se le da a la cabecera de la columna en el resultado de esta instrucción o un alias asociado a una tabla dentro de la consulta, la palabra AS es opcional.
WHERE	Cláusula que Indica que filas deben ser devueltas por la consulta, cada registro evalúa la o las condiciones descritas y solo en caso de ser verdadero el registro es incluido en la salida de la consulta.
GROUP BY	Indica las columnas que serán agrupadas para poder obtener por ejemplo consultas totalizadas.
HAVING	Funciona como la cláusula WHERE, pero sobre las columnas contenidas en un GROUP BY
ORDER BY	Permite ordenar el resultado de la consulta en base a las columnas de forma ascendente (ASC) o descendente (DESC)

- Selección de todos los registros de la tabla clientes:

```
1 SELECT * FROM Clientes;
```

- Selección de algunos campos, ordenada por nombre:

```
1 SELECT nombre, apellido1, apellido2
2 FROM Clientes
3 ORDER BY nombre;
```

- Selección de algunas columnas que son renombradas mediante un alias, y lo ordena por nombre (columna 2):

```
1 SELECT id_trabajo AS identificador, nombre AS nombre_trabajador
2 FROM trabajos
3 ORDER BY 2;
```

- Muestra el total de las ventas anuales realizadas en efectivo entre el 2015 y el 2020, y ordenadas por año:

```
1 SELECT Year(FechaVenta) AS Periodo, SUM(MontoVenta) AS VentaAnual
2 FROM Ventas
3 WHERE TipoVenta = 'EFFECTIVO'
4 GROUP BY Year(FechaVenta)
5 HAVING Year(FechaVenta) BETWEEN 2015 AND 2020
6 ORDER BY Year(FechaVenta)
```

FUNCIONES

Todas las funciones pueden ser utilizadas en sentencias SQL para construir expresiones, las cuales serán evaluadas al momento de ejecutarse la sentencia.

Cuando una función no requiere de argumentos, se pueden omitir los paréntesis.

FUNCIONES DE AGREGACIÓN

Éstas permiten devolver un único valor agregado, con la finalidad de poder obtener valores agrupados mediante una operación de cálculo.

Las funciones de agregación básicas que soportan todos los gestores de datos, son las siguientes:

Operador	Acción
COUNT	Devuelve el número total de filas seleccionadas por la consulta.
MIN	Devuelve el valor mínimo del campo que especifiquemos.
MAX	Devuelve el valor máximo del campo que especifiquemos.
SUM	Suma los valores del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.
AVG	Devuelve el valor promedio del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.

- Entrega el total de registros de una tabla:

```
1 SELECT count(*) total_registros FROM tabla
```

- Suma la columna valor de toda la tabla:

```
1 SELECT sum(valor) total FROM tabla
```

- Suma la columna valor del año 2020, agrupada por tipo:

```
1 SELECT tipo, sum(valor) total
```

```
2 FROM tabla
3 WHERE year(fecha) = 2020
4 GROUP BY tipo
```

FUNCIONES NUMÉRICAS

Operador	Acción
ROUND(<i>n</i> , <i>decimales</i>)	Redondea el número al siguiente número con el número de decimales indicado más cercano. <i>ROUND</i> (8.239,2) devuelve 8.24
TRUNC(<i>n</i> , <i>decimales</i>)	Los decimales del número se cortan para que sólo aparezca el número de decimales indicado
MOD(<i>n1</i> , <i>n2</i>)	Devuelve el resto resultado de dividir <i>n1</i> entre <i>n2</i>
POWER(<i>valor</i> , <i>exponente</i>)	Eleva el valor al exponente indicado
SQRT(<i>n</i>)	Calcula la raíz cuadrada de <i>n</i>
SIGN(<i>n</i>)	Devuelve 1 si <i>n</i> es positivo, cero si vale cero y -1 si es negativo
ABS(<i>n</i>)	Calcula el valor absoluto de <i>n</i>
EXP(<i>n</i>)	Calcula <i>eⁿ</i> , es decir el exponente en base <i>e</i> del número <i>n</i>
LN(<i>n</i>)	Logaritmo neperiano de <i>n</i>
LOG(<i>n</i>)	Logaritmo en base 10 de <i>n</i>
SIN(<i>n</i>)	Calcula el seno de <i>n</i> (<i>n</i> tiene que estar en radianes)
COS(<i>n</i>)	Calcula el coseno de <i>n</i> (<i>n</i> tiene que estar en radianes)
TAN(<i>n</i>)	Calcula la tangente de <i>n</i> (<i>n</i> tiene que estar en radianes)
ACOS(<i>n</i>)	Devuelve en radianes el arco coseno de <i>n</i>
ASIN(<i>n</i>)	Devuelve en radianes el arco seno de <i>n</i>
ATAN(<i>n</i>)	Devuelve en radianes el arco tangente de <i>n</i>
SINH(<i>n</i>)	Devuelve el seno hiperbólico de <i>n</i>
COSH(<i>n</i>)	Devuelve el coseno hiperbólico de <i>n</i>
TANH(<i>n</i>)	Devuelve la tangente hiperbólica de <i>n</i>

FUNCIONES DE CARACTERES

Operador	Acción
LOWER(texto)	Convierte el texto a minúsculas (funciona con los caracteres españoles)
UPPER(texto)	Convierte el texto a mayúsculas
INITCAP(texto)	Coloca la primera letra de cada palabra en mayúsculas
RTRIM(texto)	Elimina los espacios a la derecha del texto
LTRIM(texto)	Elimina los espacios a la izquierda que posea el texto
TRIM(texto)	Elimina los espacios en blanco a la izquierda y la derecha del texto y los espacios dobles del interior.
TRIM(caracteres FROM texto)	Elimina del texto los caracteres indicados. Por ejemplo, TRIM('h' FROM nombre) elimina las haches de la columna <i>nombre</i> que estén a la izquierda y a la derecha
SUBSTR(texto,n,m)	Obtiene los <i>m</i> siguientes caracteres del texto a partir de la posición <i>n</i> (si <i>m</i> no se indica se cogen desde <i>n</i> hasta el final)
LENGTH(texto)	Obtiene el largo del texto.
REPLACE(texto, textoABuscar, [textoReemplazo])	<p>Buscar el texto a buscar en un determinado texto y lo cambia por el indicado como texto de reemplazo.</p> <p>Si no se indica texto de reemplazo, entonces esta función elimina el texto que se busca.</p>
LPAD(texto, anchuraMáxima, [carácterDeRelleno])	Rellena el texto a la izquierda (LPAD) o a la derecha (RPAD) con el carácter indicado para ocupar la anchura indicada.
RPAD(texto, anchuraMáxima, [carácterDeRelleno])	<p>Si el texto es más grande que la anchura indicada, el texto se recorta.</p> <p>Si no se indica carácter de relleno se rellena el espacio marcado con espacios en blanco.</p>
REVERSE (texto)	Invierte el texto (le da la vuelta)
INSTR(texto, textoBuscado [,posición inicial [, nAparición]])	Obtiene la posición en la que se encuentra el texto buscado en el texto inicial.
TRANSLATE (texto, caracteresACambiar, caracteresSustitutivos)	<p>Permite transformar caracteres. Se cambia una serie de caracteres por otros que se indiquen en el mismo orden.</p> <p>De tal modo que, el primer carácter a cambiar se cambia por el primer carácter sustitutivo, el segundo por el segundo y así sucesivamente.</p>
ASCII(carácter)	Devuelve el código ASCII del carácter indicado. Si se le pasa más de un carácter, devuelve el código ASCII del primer carácter-
CHR(número)	Devuelve el carácter correspondiente al código ASCII indicado
NVL(valor, sustituto)	Si el valor es NULL, devuelve el valor sustituto; de otro modo, devuelve valor
NVL2(valor,sustituto1, sustituto2)	<p>Variante de la anterior, devuelve el valor sustituto1 si valor no es nulo.</p> <p>Si valor es nulo devuelve el sustituto2</p>
NULLIF(valor1,valor2)	Devuelve nulo si valor1 es igual a valor2. De otro modo devuelve valor1
COALESCE(listaExpresiones)	Devuelve la primera de las expresiones que no es nula.

FUNCIONES DE FECHAS

Operador	Acción
SYSDATE	Obtiene la fecha y hora actuales
SYSTIMESTAMP	Obtiene la fecha y hora actuales en formato TIMESTAMP
ADD_MONTHS(<i>fecha</i> , <i>n</i>)	Añade a la fecha el número de meses indicado por <i>n</i>
MONTHS_BETWEEN(<i>fecha1</i> , <i>fecha2</i>)	Obtiene la diferencia en meses entre las dos fechas (puede ser decimal)
NEXT_DAY(<i>fecha</i> , <i>día</i>)	Indica cual es el día que corresponde a añadir a la fecha el día indicado. El día puede ser el texto ' <i>lunes</i> ', ' <i>martes</i> ', ' <i>Miércoles</i> '... (si la configuración está en español) o el número de día de la semana (1= <i>lunes</i> , 2= <i>martes</i> , ...)
LAST_DAY(<i>fecha</i>)	Obtiene el último día del mes al que pertenece la fecha. Devuelve un valor DATE
EXTRACT(<i>valor</i> FROM <i>fecha</i>)	Extrae un valor de una fecha concreta. El valor puede ser day (día), month (mes), year (año), etc.
GREATEST(<i>fecha1</i> , <i>fecha2</i> ...)	Devuelve la fecha más moderna la lista
LEAST(<i>fecha1</i> , <i>fecha2</i> ...)	Devuelve la fecha más antigua la lista
ROUND(<i>fecha</i> [, ' <i>formato</i> '])	Redondea la fecha al valor de aplicar el formato a la fecha. El formato puede ser: <ul style="list-style-type: none"> ▪ 'YEAR' Hace que la fecha refleje el año completo ▪ 'MONTH' Hace que la fecha refleje el mes completo más cercano a la fecha ▪ 'HH24' Redondea la hora a las 00:00 más cercanas ▪ 'DAY' Redondea al día más cercano
TRUNC(<i>fecha</i> [, ' <i>formato</i> '])	Igual que el anterior pero trunca la fecha en lugar de redondearla.

FUNCIONES DE CONVERSIÓN

TO_DATE(<i>e</i> , <i>f</i>)	Convierte la expresión " <i>e</i> " a fecha dado un formato <i>f</i>
TO_CHAR(<i>e</i> , <i>f</i>)	Convierte la expresión " <i>e</i> " a un String con formato <i>f</i>
TO_NUMBER (<i>t</i>)	Convierte la expresión en número

EXERCISE 3: ALGEBRA RELACIONAL.

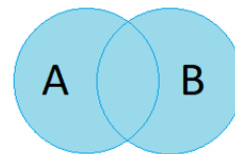
El Álgebra Relacional es un lenguaje, el cual define una serie de operaciones que se realizan utilizando operadores, de los que cada uno puede trabajar sobre uno, o varios conjuntos de datos, produciendo como resultado un nuevo conjunto de datos.

El conjunto de datos resultante de una operación puede a su vez ser utilizado en una nueva, en forma anidada, tal como se hace con las operaciones aritméticas. Esta propiedad es conocida como clausura.

UNIÓN

Opera sobre dos o más tablas, siendo necesario que todas posean la misma estructura, y devolviendo una nueva tabla, cuyo contenido es la combinación de los de todas y cada una de las originales, descartando las filas repetidas.

**SELECT Codigo, Nombre FROM A
UNION
SELECT Codigo, Nombre FROM B**



A	
CODIGO	Nombre
1425	Carlos
2000	Juan
3000	Eduardo

B	
CODIGO	LOCALIDAD
2000	Juan
3000	Eduardo
2128	Maria
2121	Alejandra

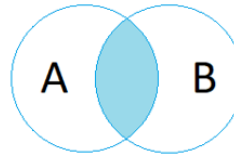


A U B	
CODIGO	LOCALIDAD
1425	Carlos
2000	Juan
3000	Eduardo
2128	Maria
2121	Alejandra

INTERSECCIÓN

Opera sobre dos o más tablas, siendo necesario que todas posean la misma estructura, y devolviendo una nueva tabla, cuyo contenido es las filas comunes a todas las originales, descartando las filas repetidas.

```
SELECT * FROM A
INTERSECT
SELECT * FROM B
```



A	
CODIGO	Nombre
1425	Carlos
2000	Juan
3000	Eduardo

B	
CODIGO	LOCALIDAD
2000	Juan
3000	Eduardo
2128	Maria
2121	Alejandra

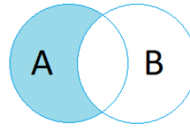


A ∩ B	
CODIGO	LOCALIDAD
2000	Juan
3000	Eduardo

DIFERENCIA

Opera sobre dos tablas, siendo necesario que ambas posean la misma estructura, y devolviendo una nueva tabla, cuyo contenido son las filas que figuran en la primera tabla, y no en la otra.

**SELECT * FROM A
EXCEPT
SELECT * FROM B**



A	
CODIGO	Nombre
1425	Carlos
2000	Juan
3000	Eduardo

B	
CODIGO	LOCALIDAD
2000	Juan
3000	Eduardo
2128	Maria
2121	Alejandra

A - B	
CODIGO	LOCALIDAD
1425	Carlos

PRODUCTO

Opera sobre dos tablas, efectuando un producto cartesiano del contenido de éstas, no siendo necesario que ambas posean la misma estructura, y devolviendo una nueva tabla, cuyo contenido es todas las posibles combinaciones de las filas de una de ellas.

SELECT * FROM A, B

A	
W	X
1	23
78	32
67	5

B	
Y	Z
15	320
7	5


A * B			
W	X	Y	Z
1	23	15	320
1	23	7	5
78	32	15	320
78	32	7	5
67	5	15	320
67	5	7	5

SELECCIÓN

Opera sobre una o más tablas, no siendo necesario que éstas posean la misma estructura, y devolviendo una nueva tabla, cuyo contenido es todas las filas de las tablas indicadas que satisfacen una cierta condición.

```
SELECT * FROM A
WHERE X > 5
```

A	
W	X
1	23
78	32
67	5




A''	
W	X
1	23
78	32

PROYECCIÓN

Opera sobre una o más tablas, no siendo necesario que éstas posean la misma estructura, y devolviendo una nueva tabla, cuyo contenido son todas las filas de las tablas indicadas que satisfacen una cierta condición. La proyección permite indicar cuáles columnas se desea obtener en el resultado.

```
SELECT X, Z FROM A
WHERE X < 50
```

TABLA A		
X	Y	Z
1	23	11
78	32	321
67	5	33
15	320	5
7	5	212
15	320	5

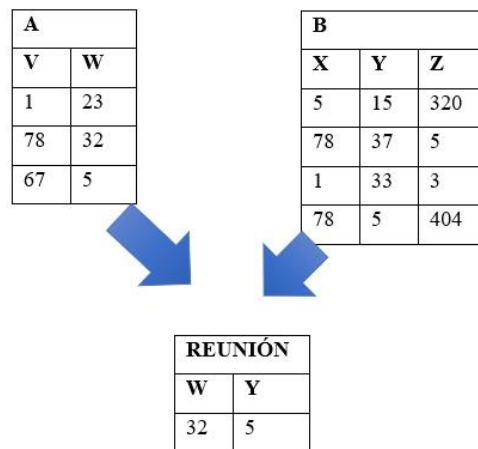


PROYECCIÓN DE X y Z CON X<50	
X	Z
1	11
15	5
7	212

REUNIÓN

Opera sobre dos o más tablas, que poseen estructuras diferentes, y devolviendo una nueva tabla, cuyo contenido es un conjunto de filas con las columnas deseadas provenientes de las diferentes tablas, en el que las filas de éstas son relacionadas mediante alguna condición.

```
SELECT A.W, B.Y  
FROM A  
JOIN B ON (A.V=B.X AND Z>10)
```



DIVISIÓN

Opera sobre dos tablas. Si se divide una tabla B por una tabla A, se obtiene una nueva tabla, cuyas columnas serán aquellas de la tabla B que no existen en la tabla A, y cuyas filas cumplan con estar relacionadas con todas y cada una de las filas de la tabla A.

```

SELECT DISTINCT B1.INDICE
FROM B AS B1
WHERE NOT EXISTS (
    SELECT A1.codigo
    FROM A AS A1
    WHERE NOT EXISTS (
        SELECT B2.codigo
        FROM B AS B2
        WHERE B2.CODIGO = A1. CODIGO AND B2.INDICE = B1.INDICE
    )
)
    
```

B	
CODIGO	INDICE
1425	15%
2000	27%
3000	33%
2128	45%
2121	13%
2000	15%
3000	15%

A
CODIGO
1425
2000
3000

B / A
INDICE
15%

La columna INDICE es la única de la tabla B que no existe en la A, y el valor 15% es el único valor de ésta que aparece en filas que se relacionan con todas las filas de la tabla A; es decir, aquellas en las que CODIGO toma los valores 1425, 2000 y 3000, que son todos los que aparecen en la tabla A.

ASIGNACIÓN

Opera sobre una única tabla, y se utiliza para asignar valores a algunas columnas de filas de la misma.

```
UPDATE A  
SET INDICE = "100%"  
WHERE CODIGO > 2000
```

A	
CODIGO	INDICE
1425	15%
2000	27%
3000	33%
2128	45%
2121	13%
2000	15%
3000	15%



A	
CODIGO	INDICE
1425	15%
2000	27%
3000	100%
2128	100%
2121	100%
2000	15%
3000	100%