Lenguaje SQL para Novatos

Edición Septiembre 2024

Eugenio H. Martínez

Instituto CyP Soft

Introducción

Este libro, `Lenguaje SQL para Novatos`, está dirigido a estudiantes noveles en programación SQL y gestión de base de datos.

Orientado para todos los niveles: principiantes y expertos.
Se encuentra disponible en GitHub como documento PDF bajo el nombre *LenguajeSQL-SentenciasBasicas.pdf*
Su enlace es: https://github.com/torrentelinux/torrentarium/blob/master/base/biblioteca/LenguajeSQL-SentenciasBasicas.pdf

Conceptos básicos

SQL (Structured Query Language) es un lenguaje estructurado para la consulta de los datos organizados en una base de datos.

Una consulta es una instrucción que solicita la recuperación de la información.
Una base de datos relacional se basa en el modelo de datos relacional.
El modelo relacional usa una colección de tablas para representar tanto los datos como sus relaciones.
Una base de datos relacional consta de tablas, diagramas, vistas, funciones y procedimientos.
Una tabla consta de columnas y filas.
Una fila es un conjunto de columnas.
Una columna posee un nombre y un dato único distinguible.
El dato puede ser: un valor nulo, un dígito binario, un carácter, una secuencia de caracteres, un número entero, un número real, una fecha, una imagen gráfica, un sonido, un video, etc.

Tabla Modelo				
	columna1	columna2	columna3	columna4
fila1	dato1	dato2	dato3	dato4
fila2	dato5	dato6	dato7	dato8
fila3	dato9	dato10	dato11	dato12

Ilustración 1: Tabla modelo

Transacciones

Consiste en una secuencia de instrucciones de consulta o de actualización. Una transacción comienza implícitamente cuando se ejecuta una instrucción SQL.

Una de las siguientes instrucciones debe finalizar la transacción:

COMMIT WORK ➔ las actualizaciones realizadas por la transacción pasan a ser permanentes en la base de datos.
ROLLBACK WORK ➔ deshace las actualizaciones realizadas por la transacción, pasando el estado de la base de datos a su estado anterior a la ejecución de la primera instancia de la transacción.

Sentencias básicas

CREATE TABLE <nombre_tabla>(<nombre_columna> <tipo>, …, PRIMARY KEY(<nombre_columna>)[, FOREIGN KEY(<nombre_columna>) REFERENCES <nombre_tabla2>(nombre_columna_tabla2)])

Forma 1: CREATE TABLE Localidades(codigo INT, localidad CHAR(25), PRIMARY KEY(codigo))
Forma 2: CREATE TABLE Localidades(codigo INTEGER, localidad CHAR(32) NOT NULL, provincia CHAR(1) NOT NULL, PRIMARY KEY(codigo), FOREIGN KEY(provincia) REFERENCES Provincias(codigo))

Para la Forma 1: crea la tabla Localidades con dos columnas y una clave primaria(PK=Primary Key)) en la base de datos vigente.
Para la Forma 2: crea la tabla Localidades con tres columnas, la columna codigo es la clave primaria(PK) y la columna provincia es la clave foránea(FK=Foreign Key) que referencia a la columna codigo de la tabla Provincias.
Tanto INT como INTEGER se refieren al mismo tipo de dato entero. La frase NOT NULL significa no nulo, se utiliza para indicar que la columna no acepta valores nulos.

Tabla `Localidades - Forma 1`

Fila	Nombre_columna	Tipo	Nombre_columna	Tipo
1	[PK]codigo	INT	localidad	CHAR(25)

Tabla `Localidades - Forma 2`

Fila	Nombre_columna	Tipo	Nombre_columna	Tipo	Nombre_columna	Tipo
1	[PK]codigo	INTEGER	localidad	CHAR(32)	[FK]Provincia	CHAR(1)

INSERT INTO <nombre_tabla> VALUES (dato1, dato2, …, datoN)

INSERT INTO Localidades VALUES (4000, ‘San Miguel de Tucumán’)

Inserta una fila con datos en la tabla Localidades.

Tabla `Localidades`

Fila	codigo	localidad
1	4000	San Miguel de Tucumán

DELETE FROM <nombre_tabla>

DELETE FROM Localidades

Borra todas las filas de la tabla Localidades.

Tabla `Localidades`

Fila	codigo	localidad

DELETE FROM <nombre_tabla> WHERE <expresión>

Forma 1: DELETE FROM Localidades WHERE localidad = ‘San’
Forma 2: DELETE FROM Localidades WHERE codigo BETWEEN 4100 AND 4200

Borra selectivamente las filas de la tabla Localidades.

DROP TABLE <nombre_tabla>

DROP TABLE Localidades

Elimina la tabla `Localidades` de la base de datos vigente.

ALTER TABLE <nombre_tabla> ADD <nombre_columna> <tipo>

ALTER TABLE Localidades ADD provincia CHAR(30)

Agrega la columna `provincia` a la tabla `Localidades`.

Tabla `Localidades`

Fila	Nombre_columna	Tipo	Nombre_columna	Tipo	Nombre_columna	Tipo
1	[PK]codigo	INT	localidad	CHAR(25)	provincia	CHAR(30)

UPDATE <nombre_tabla> [SET <nombre_columna> = [datoN [WHERE <expresión>]] | [<expresión_consulta>]

Forma 1: UPDATE Localidades
Forma 2: UPDATE Localidades SET codigo = 4101
Forma 3: UPDATE Localidades SET localidad = ‘San Isidro de Lules’ WHERE codigo = 4101
Forma 4: UPDATE Ordenes SET precio = (SELECT precio FROM Partes WHERE Ordenes.stock_nro = Partes.stock_nro)

Actualiza la tabla completamente, por nombre de la columna y en caso de que una expresión sea verdadera.
También se acepta una expresión de consulta para concretar una actualización en una tabla.

SELECT <nombre_columna1>, …, <nombre_columnaN> FROM <nombre_tabla1>, …, <nombre_tablaN> WHERE <expresión>

Forma 1: SELECT codigo FROM Localidades WHERE localidad = ‘Famaillá’
Forma 2: SELECT * FROM Localidades

La cláusula SELECT se utiliza para la extracción de información a partir de una o varias tablas. El símbolo ‘*’ significa “todas las columnas”.
En la “forma 1” se realiza una consulta a Localidades por la columna `codigo` de la localidad `'Famaillá'`.
En la “forma 2” se consulta toda la tabla Localidades.

SELECT <nombre_columna1>, …, <nombre_columnaN> FROM <nombre_tabla> ORDER BY <nombre_columna>, …, <nombre_columnaN> { ASC | DESC }

Forma 1: SELECT * FROM Localidades ORDER BY provincia DESC
Forma 2: SELECT * FROM Localidades ORDER BY provincia ASC

Se realiza una consulta ordenada a la tabla por el/los campo/s seleccionado/s en forma descendente o ascendente.

SELECT <nombre_columna1>, …, <nombre_columnaN>
FROM <nombre_tabla1> INNER JOIN <nombre_tabla2> ON <nombre_tabla1.clave> = <nombre_tabla2.clave>
SELECT provincias.codigo, provincias.provincia, localidades.codigo, localidades.localidad
FROM provincias INNER JOIN localidades ON localidades.provincia = provincias.codigo

Se reúnen dos tablas para obtener una única consulta. La reunión es de tipo interna.

Tabla `Provincias`				
Fila	**Nombre_columna**	**Tipo**	**Nombre_columna**	**Tipo**
1	[PK]`codigo`	CHAR(1)	`provincia`	CHAR(32)

Tabla `Localidades`						
Fila	**Nombre_columna**	**Tipo**	**Nombre_columna**	**Tipo**	**Nombre_columna**	**Tipo**
1	[PK]`codigo`	INT	`localidad`	CHAR(32)	[FK]`provincia`	CHAR(1)

SELECT <nombre_columna1>, …, <nombre_columnaN>
FROM <nombre_tabla1> LEFT OUTER JOIN <nombre_tabla2> ON <nombre_tabla1.clave> = <nombre_tabla2.clave>
SELECT provincias.codigo, provincias.provincia, localidades.codigo, localidades.localidad
FROM provincias LEFT OUTER JOIN localidades ON localidades.provincia = provincias.codigo

Se reúnen dos tablas para obtener una única consulta. La reunión es de tipo externa.

SELECT CAST(<nombre_columna> AS <tipo>) FROM <nombre_tabla>
SELECT CAST(precio AS double) FROM public.frutas

Fuerza el tipo de dato numérico original a otro tipo de dato numérico especificado por el usuario. Por ejemplo de tipo money a tipo double.

CREATE VIEW <nombre_vista> AS <consulta>
CREATE VIEW CodigoPostalxLocalidadxProvincia AS
SELECT Localidades.codigo, Localidades.localidad, Provincias.provincia FROM Provincias
INNER JOIN Localidades ON Provincias.codigo = Localidades.provincia

Crea una vista a partir de una expresión de consulta y se accede a las tablas Provincias y Localidades.

CREATE INDEX <nombre_índice> ON <nombre_tabla | nombre_relación>(<nombre_columna>)
CREATE INDEX indice_vendedor ON Vendedor(sector_de_vtas)

En este caso, sobre la tabla Vendedor se crea el 'indice_vendedor' y la clave de búsqueda es 'sector_de_vtas'.
Se puede apreciar que mediante CREATE INDEX se crea un índice a partir de una columna específica de la tabla. La clave de búsqueda es indicada en 'nombre_columna'.

CREATE UNIQUE INDEX <nombre_índice> ON <nombre_tabla | nombre_relación>(<nombre_columna>)
CREATE UNIQUE INDEX indice_vendedor ON Vendedor(sector_de_vtas)

La cláusula UNIQUE permite declarar a 'nombre_columna' como clave candidata (AK=Alternative Key) de 'nombre_tabla'. Se evita la duplicidad de los datos en las filas.

DROP INDEX <nombre_tabla.nombre_índice>
DROP INDEX Vendedor.indice_vendedor

Se elimina el índice indicado en 'nombre_indice' de la tabla 'nombre_tabla'. En este caso, se elimina el índice 'indice_vendedor' de la tabla 'Vendedor'.

CREATE DATABASE <nombre_basedatos>
CREATE DATABASE CodigosPostales

Crea la base de datos CodigosPostales. En Microsoft SQL Server, se utilizan las extensiones para los archivos:
.MDF ➔ Master Data File (principal).
.NDF ➔ Secundario.
.LDF ➔ Log Data File (registro de transacciones).

ALTER DATABASE <nombre_basedatos> MODIFY NAME = <nuevo_nombre_basedatos>
ALTER DATABASE CodigosPostales MODIFY NAME = CodigosPostales_ARG

Renombra la base de datos CodigosPostales a CodigosPostales_ARG.

DROP DATABASE <nombre_basedatos>
DROP DATABASE CodigosPostales

Quita la base de datos del servidor SQL y elimina a CodigosPostales. Antes de ser eliminada, la misma debe estar sin uso.

Sentencias avanzadas

ALTER USER <nombre_usuario> PASSWORD <contraseña_usuario>
ALTER USER postgres PASSWORD '*951p753'

Cambia la contraseña para el usuario 'postgres'. ALTER USER es un alias de ALTER ROLE en PostgreSQL.
Las contraseñas de cada usuario están almacenadas en la tabla “postgres/pg_catalog/pg_authid”.

SELECT <nombre_columna1>, …, <nombre_columnaN> FROM <nombre_tabla1>

WHERE <nombre_columna> [NOT] LIKE '[%]texto_patrón[%]'

Ejemplo 1: SELECT * FROM libros WHERE autor LIKE '%chatz%'
Ejemplo 2: SELECT * FROM libros WHERE autor LIKE '%ch_tz%'
Ejemplo 3: SELECT titulo, precio FROM libros WHERE CAST(precio AS VARCHAR) LIKE '1_.%'
Ejemplo 4: SELECT titulo, precio FROM libros WHERE CAST(precio AS VARCHAR) LIKE '%.00'
Ejemplo 1: Hace una consulta a la tabla 'libros' por la columna 'autor' que contenga la porción de textos 'chatz' en dicha columna.
Ejemplo 2: Hace una consulta a la tabla 'libros' por la columna 'autor' que comience por los caracteres 'ch', una letra cualquiera y termina en 'tz' en dicha columna.
Ejemplo 3: Hace una consulta a la tabla 'libros' por las columnas 'titulo' y 'precio' cuyos precios están comprendidos entre 10.00 y 19.99, el campo 'precio' que es un campo de tipo *decimal* se convierte a *varchar*.
Ejemplo 4: Hace una consulta a la tabla 'libros' por las columnas 'titulo' y 'precio' cuyos precios no incluyan centavos.
El caracter '%' es un símbolo comodín utilizado para la búsqueda de porciones o patrones de textos en la columna. Puede estar al principio o al final del texto también en ambos extremos del mismo. Otro símbolo comodín es '_' que reemplaza a un caracter.
El operador LIKE se emplea con tipos de datos *char*, *varchar*, *date*, *time* y *timestamp*.

Terminal de PostgreSQL

La terminal de comandos de PostgreSQL se llama `psql: terminal interactiva`.

En Windows, funciona como una ventana de textos. Es aconsejable abrirla siendo Administrador para poder contar con todos los derechos administrativos que otorga el S.O. Windows.
Para conocer sus diferentes opciones, ejecute desde `cmd`: `psql --help`
Para iniciar sesión en el servidor local con el rol 'postgres' escriba lo siguiente: `psql -U postgres`
Y va a responder así:

```
[C:\laragon\www]psql -U postgres
Contraseña para usuario postgres:
psql (14.5)
Digite «help» para obtener ayuda.

postgres=#
```

Una vez que ha ingresado al sistema, puede tipear los siguientes comandos para conocerlo:

```
help → solicita una ayuda global
\?   → solicita una ayuda de los comandos reconocidos por psql
\h   → solicita una ayuda del lenguaje SQL
\q   → sale del sistema
\du  → muestra los roles de usuarios registrados en postgresql
\dg  → idem a '\du'
\! cls    → borra la pantalla (Windows)
\! clear  → borra la pantalla (Linux)
\conninfo → muestra información vigente de la conexión al servidor/base de datos/usuario
\connect "postgres" → se conecta a la base de datos 'postgres'
select * from pg_database;    → muestra las bases de datos contenidas en el servidor
select * from pg_authid;      → muestra el perfil de los roles admitidos
select * from pg_config;      → muestra la configuración del servidor de base de datos
select * from pg_file_settings;    → muestra la configuración actual de postgresql.conf
select version();   → muestra el n° de versión del servidor de base de datos
```

Cuando el servidor PosgreSQL no está en ejecución, sea cual sea el motivo, `psql` va a avisar de la siguiente manera:

```
[C:\laragon\www]psql -U postgres
psql: error: falló la conexión al servidor en «localhost» (::1), puerto 5432: Connection
refused (0x0000274D/10061)
        ¿Está el servidor en ejecución en ese host y aceptando conexiones TCP/IP?
falló la conexión al servidor en «localhost» (127.0.0.1), puerto 5432: Connection refused
(0x0000274D/10061)
        ¿Está el servidor en ejecución en ese host y aceptando conexiones TCP/IP?

[C:\laragon\www]
```

Otra situación llamativa que se puede dar al querer ejecutar a `psql` es cuando los códigos de página de consola difieren entre lo que presenta `cmd` y lo que exige `psql`:

```
[C:\laragon\www]psql -U postgres
Contraseña para usuario postgres:
psql (14.5)
ADVERTENCIA: El código de página de la consola (850) difiere del código
             de página de Windows (1252).
             Los caracteres de 8 bits pueden funcionar incorrectamente.
             Vea la página de referencia de psql «Notes for Windows users»
             para obtener más detalles.
Digite «help» para obtener ayuda.

postgres=#
```

Para resolver esta situación, haga lo siguiente en `cmd`:

```
[C:\laragon\www]chcp 1252
Página de códigos activa: 1252

[C:\laragon\www]psql -U postgres
Contraseña para usuario postgres:
psql (14.5)
Digite «help» para obtener ayuda.
```

```
postgres=#
```

Listo, `psql` quedó conforme con la visualización de los caracteres de 8 bits.

Ejercicio modelo

Se creará una base de datos a modo de ilustración práctica para explicar el uso del lenguaje SQL.

La base de datos se llamará `CodigosPostales` y contendrá dos tablas: `Provincias` y `Localidades`.
La tabla `Provincias` tendrá las columnas definidas así:

Columna 1: `codigo`, de tipo CHAR, su longitud es de un carácter, no acepta valores nulos y es la clave primaria(PK).
Columna 2: `provincia`, de tipo CHAR, su longitud es de 32 caracteres, no acepta valores nulos.

La tabla `Localidades` tendrá las columnas definidas así:

Columna 1: `codigo`, de tipo INT, su longitud es de 4 dígitos, no acepta valores nulos y es la clave primaria(PK).
Columna 2: `localidad`, de tipo CHAR, su longitud es de 32 caracteres, no acepta valores nulos.
Columna 3: `provincia`, de tipo CHAR, su longitud es de un carácter, no acepta valores nulos y es la clave foránea(FK).

Las dos tablas estarán relacionadas por las columnas `Provincias.codigo` y `Localidades.provincia`.
Ahora abra el intérprete de comandos SQL de su sistema instalado que puede ser:

- En PostgreSQL se llama `terminal interactiva(psql.exe)`.
- En Microsoft SQL Server se llama `analizador de consultas SQL(isqlw.exe)`.

Ejecute los siguientes pasos:

1. Crear la base de datos CodigosPostales
   ```
   CREATE DATABASE CodigosPostales
   ```
2. Trabajar con la base de datos recién creada
   ```
   USE CodigosPostales
   ```
   En PostgreSQL, escriba: `\connect "CodigosPostales"`
3. Crear la tabla Provincias
   ```
   CREATE TABLE Provincias(codigo CHAR(1), provincia CHAR(32) NOT NULL, PRIMARY
   KEY(codigo))
   ```
4. Crear la tabla Localidades
   ```
   CREATE TABLE Localidades(codigo INTEGER, localidad CHAR(32) NOT NULL, provincia
   CHAR(1) NOT NULL, PRIMARY KEY(codigo), FOREIGN KEY(provincia) REFERENCES
   Provincias(codigo))
   ```
5. Agregar valores a la tabla Provincias
   ```
   INSERT INTO Provincias VALUES('A', 'Salta')
   INSERT INTO Provincias VALUES('B', 'Buenos Aires')
   INSERT INTO Provincias VALUES('C', 'Capital Federal')
   INSERT INTO Provincias VALUES('D', 'San Luis')
   INSERT INTO Provincias VALUES('E', 'Entre Ríos')
   INSERT INTO Provincias VALUES('F', 'La Rioja')
   INSERT INTO Provincias VALUES('G', 'Santiago del Estero')
   INSERT INTO Provincias VALUES('H', 'Chaco')
   INSERT INTO Provincias VALUES('J', 'San Juan')
   INSERT INTO Provincias VALUES('K', 'Catamarca')
   INSERT INTO Provincias VALUES('L', 'La Pampa')
   INSERT INTO Provincias VALUES('M', 'Mendoza')
   INSERT INTO Provincias VALUES('N', 'Misiones')
   INSERT INTO Provincias VALUES('P', 'Formosa')
   INSERT INTO Provincias VALUES('Q', 'Neuquén')
   INSERT INTO Provincias VALUES('R', 'Río Negro')
   INSERT INTO Provincias VALUES('S', 'Santa Fé')
   INSERT INTO Provincias VALUES('T','Tucumán')
   INSERT INTO Provincias VALUES('U', 'Tierra del Fuego')
   INSERT INTO Provincias VALUES('V', 'Chubut')
   INSERT INTO Provincias VALUES('W', 'Corrientes')
   INSERT INTO Provincias VALUES('X', 'Córdoba')
   INSERT INTO Provincias VALUES('Y', 'Jujuy')
   INSERT INTO Provincias VALUES('Z', 'Santa Cruz')
   ```
6. Agregar valores a la tabla Localidades
   ```
   INSERT INTO Localidades VALUES(1000, 'Capital Federal', 'C')
   INSERT INTO Localidades VALUES(1900, 'La Plata', 'B')
   INSERT INTO Localidades VALUES(3000, 'Santa Fé', 'S')
   INSERT INTO Localidades VALUES(3080, 'Esperanza', 'S')
   INSERT INTO Localidades VALUES(2300, 'Rafaela', 'S')
   INSERT INTO Localidades VALUES(3100, 'Paraná', 'E')
   INSERT INTO Localidades VALUES(3300, 'Posadas', 'N')
   INSERT INTO Localidades VALUES(3400, 'Corrientes', 'W')
   INSERT INTO Localidades VALUES(3500, 'Resistencia', 'H')
   INSERT INTO Localidades VALUES(3600, 'Formosa', 'P')
   INSERT INTO Localidades VALUES(4000, 'San Miguel de Tucumán', 'T')
   INSERT INTO Localidades VALUES(4100, 'Famaillá', 'T')
   INSERT INTO Localidades VALUES(4132, 'Ingenio La Fronterita', 'T')
   INSERT INTO Localidades VALUES(4200, 'Santiago del Estero', 'G')
   INSERT INTO Localidades VALUES(4220, 'Termas de Río Hondo','G')
   INSERT INTO Localidades VALUES(4400, 'Salta', 'A')
   INSERT INTO Localidades VALUES(4440, 'San José de Metán', 'A')
   INSERT INTO Localidades VALUES(4600, 'San Salvador de Jujuy', 'Y')
   INSERT INTO Localidades VALUES(4700, 'San Fernando del Valle de Catamarca', 'K')
   ```

```
INSERT INTO Localidades VALUES(5000, 'Córdoba', 'X')
INSERT INTO Localidades VALUES(5300, 'La Rioja', 'F')
INSERT INTO Localidades VALUES(5400, 'San Juan', 'J')
INSERT INTO Localidades VALUES(5500, 'Mendoza', 'M')
INSERT INTO Localidades VALUES(5700, 'San Luis', 'D')
INSERT INTO Localidades VALUES(6300, 'Santa Rosa', 'L')
INSERT INTO Localidades VALUES(8300, 'Neuquén', 'Q')
INSERT INTO Localidades VALUES(8500, 'Viedma', 'R')
INSERT INTO Localidades VALUES(9300, 'Rawson', 'V')
INSERT INTO Localidades VALUES(9400, 'Río Gallegos', 'Z')
INSERT INTO Localidades VALUES(9410, 'Tierra del Fuego', 'U')
```

7. Realizar una consulta a la tabla Provincias. Se visualizará todo el contenido de Provincias
   `SELECT * FROM Provincias`
8. Realizar una consulta a la tabla Localidades. Se visualizará todo el contenido de Localidades
   `SELECT * FROM Localidades`

Hasta aquí todo debería estar funcionando muy bien. Ya se tiene la tabla `Provincias` cargada completamente y la tabla `Localidades` tiene cargada algunas localidades de la Argentina, si prefiere usted podría seguir agregando más localidades a esta tabla.

Configuración de PostgreSQL

Los archivos de configuración de PostgreSQL se encuentran almacenados en el directorio de los datos.

Los mismos se denominan: `'pg_hba.conf'`, `'pg_ident.conf'` y `'postgresql.conf'`.
Por ejemplo, en Windows, están en "`C:\Program Files\PostgreSQL<version>\data`". Dicho lugar puede cambiar según la procedencia de la aplicación.
En el caso de Laragon ver. 6.0 su directorio de datos está en "`C:\laragon\data\postgresql-14\`".
Laragon es un entorno de desarrollo integral para la gestión y administración de bases de datos en SQL que incluye un servidor de páginas webs y ofrece soporte a PHP, Python, MySQL y otros módulos más. También sirve para crear sitios webs completos.

Referencias

La siguiente lista de material bibliográfico, impreso y documentos en línea, que se utilizó para la confección de este libro digital:

1. Fundamentos de bases de datos. Silberschatz, Korth y Sudarshan. 5ta. Edición, 2006, en español. Editorial McGraw-Hill/Interamericana de España, S.A.U. https://www.mheducation.es/fundamentos-de-bases-de-datos-9788448190330-spain-group (material impreso).
2. PHP 6, sitios dinámicos con el lenguaje más robusto. Francisco Minera. 1ra. Edición, 2010, en español. Coedición Fox Andina y Gradi S.A. https://www.redusers.com/noticias/publicaciones/libro-programacion-php6/ (material impreso).
3. PostgreSQL 8.1.0 Documentation. The PostgreSQL Global Development Group, U.S.A. Copyright (c) 1996 – 2005. https://www.postgresql.org/files/documentation/pdf/8.1/postgresql-8.1-A4.pdf (documento PDF).
4. Comando `Alter User` de PostgreSQL. https://www.postgresql.org/docs/current/sql-alteruser.html
5. Conversión incorrecta con el tipo de dato money. https://stackoverflow.com/questions/54924969/bug-in-pg-jdbc-driver-money-db-type-bad-value-for-type-double
6. La sentencia `SELECT` en Oracle Database 21c. https://docs.oracle.com/en/database/oracle/oracle-database/21/sqlrf/SELECT.html
7. Referencia del lenguaje SQL de Oracle Database 21c. https://docs.oracle.com/en/database/oracle/oracle-database/21/sqlrf/toc.htm
8. El operador LIKE con ejemplos. https://www.srcodigofuente.es/aprender-sql/operador-like
9. El lenguaje SQL. https://es.wikipedia.org/wiki/SQL
10. Laragon, gestión de servidores de base de datos. https://laragon.org
11. Códigos postales de Argentina. https://codigo-postal.co/argentina/