UT3 Diseño Físico Lenguaje SQL: DDL

Sitio:	Aula Virtual I.E.S. Ramón Arcas Meca
Curso:	SDAW1 - Bases de Datos 2023-24 (Diego Heredia)
Libro:	UT3 Diseño Físico Lenguaje SQL: DDL
Imprimido por:	ALBERTO CÁNOVAS LÓPEZ
Día:	miércoles, 13 de marzo de 2024, 11:29

Tabla de contenidos

1. SQL

.

2. CREATE DATABASE

Instrucción CREATE DATABASE

La instrucción CREATE DATABASE se usa para crear una nueva base de datos SQL.

Sintaxis

MySQL / MariaDB / SQL Server:

```
CREATE DATABASE databasename;
```

También se permite una sintaxis especial para evitar errores si la Base de datos ya existía.

```
CREATE DATABASE IF NOT EXISTS databasename;
```

MySQL / MariaDB

Durante la creación de la base de datos podemos especificar el juego de caracteres y la intercalación (`COLLATE`) a utilizar. Estos conceptos los estudiaremos más adelante.

```
CREATE DATABASE db_name
    [[DEFAULT] CHARACTER SET nombre_charset]
    [[DEFAULT] COLLATE nombre_intercalacion];
```

SQL Server:

En SQL Server también se puede especificar la intercalación (`COLLATE`) que deseamos utilizar.

```
CREATE DATABASE db_name
    [COLLATE nombre_intercalacion;
```

Este SGBD durante la creación de la base de datos admite más parámetros, podéis consultar la sintaxis completa en la web de microsoft. Veamos un ejemplo:

```
USE master ;
GO
CREATE DATABASE Sales
ON
( NAME = Sales_dat,
    FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\MSSQL\DATA\saledat.mdf',
    SIZE = 10,
    MAXSIZE = 50,
    FILEGROWTH = 5 )
LOG ON
( NAME = Sales_log,
    FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\MSSQL\DATA\salelog.ldf',
    SIZE = 5MB,
    MAXSIZE = 25MB,
    FILEGROWTH = 5MB ) ;
GO
```

Este ejemplo crea la base de datos `Sales`. Debido a que no se usa la palabra clave `PRIMARY`, el primer archivo (`Sales_dat`) se convierte en el principal. Como no se especifica MB ni KB en el parámetro SIZE del archivo Sales_dat , se utiliza **MB** y el tamaño se asigna en megabytes.

`Sales_log` será el archivo donde se almacenará el registro de sucesos de la base de datos, su tamaño se asigna en megabytes porque el sufijo MB se ha indicado explícitamente en el parámetro SIZE .

Cada vez que se crea, modifica o quita una base de datos de usuario, se debe hacer una copia de seguridad de la misma.

Oracle:

La creación de una Base de datos en Oracle es algo aún más complicado. La sintaxis completa será:

```
CREATE DATABASE nombreDB opciones
```

Dónde las opciones son:

```
    DATAFILE filespec AUTOEXTEND OFF
    DATAFILE filespec AUTOEXTEND ON [NEXT int K | M] [MAXSIZE int K | M]
    MAXDATAFILES int
    EXTENT MANAGEMENT LOCAL
    DEFAULT TEMPORARY TABLESPACE tablespace [TEMPFILE filespec]
                                        [EXTENT MANAGEMENT LOCAL] [UNIFORM [SIZE int K | M]]
    UNDO TABLESPACE tablespace [DATAFILE filespec]
    LOGFILE [GROUP int] filespec
    MAXLOGFILES int
    MAXLOGMEMBERS int
    MAXLOGHISTORY int
    MAXINSTANCES int
    ARCHIVELOG | NOARCHIVELOG
    CONTROLFILE REUSE
    CHARACTER SET charset
    NATIONAL CHARACTER SET charset
    SET TIMEZONE = 'time_zone_region'
    SET TIMEZONE = '{+|-} hh:mm'
    FORCE LOGGING
    USER SYS IDENTIFIED BY password
    USER SYSTEM IDENTIFIED BY password
```

Se puede poner mas de un `DATAFILE` o `LOGFILE` separando los nombres de fichero con comas `DATAFILE filespec1, filespec2, filespec3`

Si no se especifican claves, Oracle establece `"change_on_install"` para `SYS` y `"manager"` para `SYSTEM`.

Después de crear la base de datos podemos cambiar entre los modos `ARCHIVELOG`, `NOARCHIVELOG` con la sentencia `ALTER DATABASE`

CREATE DATABASE Ejemplo

La siguiente instrucción SQL crea una base de datos llamada `testDB`:

Ejemplo

```
CREATE DATABASE testDB;
```

> ❗Debemos asegurarnos de tener privilegios de administrador antes de crear cualquier base de datos.

3. Ver las Bases de Datos

Si deseamos obtener un listado de las bases de datos creadas, podemos ejecutar la siguiente instrucción o comando SQL:

MySQL / MariaDB:

```
SHOW DATABASES;
```

Dónde podríamos obtener como resultado:

Database
information_schema
mysql
performance_schema
prueba

SQL Server:

SQL es algo más complejo que *MySQL* y para realizar un listado de las bases de datos disponibles, tendríamos que ejecutar la sentencia:

```
SELECT name FROM sys.databases
GO
```

Oracle:

Oracle no tiene un modelo de base de datos simple como *MySQL* o *SQL Server*. Lo más cercano es consultar los espacios de tabla y los usuarios correspondientes dentro de ellos.

Por ejemplo, si tenemos un espacio de tablas `DEV_DB` con todas nuestras *'bases de datos'* reales dentro, podremos ejecutar la siguiente consulta

```
SELECT TABLESPACE_NAME FROM USER_TABLESPACES;
```

Que nos podría dar como resultado todos los *tablespaces* disponibles:

TABLESPACE_NAME
SYSTEM
SYSAUX
UNDOTBS1
TEMP
USERS
EXAMPLE
DEV_DB

Y dentro de un espacio de tabla especifico (por ejemplo `DEV_DB`) podríamos consultar los usuarios mediante una consulta del tipo:

```
select USERNAME from DBA_USERS where DEFAULT_TABLESPACE = 'DEV_DB';
```

Que podría dar un resultado como:

USERNAME
ROLES
DATAWARE
DATAMART
STAGING

Si queremos mostrar todos los usuarios junto a los tablespaces a los que pertenecen:

```
select USERNAME, DEFAULT_TABLESPACE from DBA_USERS;
```

4. USAR una Base de Datos

Todo Sistema Gestor de Bases de Datos (SGBD) que se precie de serlo, debe permitir la gestión de diferentes bases de datos, identificadas cada una de ellas por un nombre diferente.

Por ese motivo aparece el concepto de Base de Datos de **trabajo**, **en uso** o **seleccionada**. Todas las sentencias SQL que ejecutemos en un SGBD se realizarán sobre la base de datos que tengamos en USO.

Dependiendo del sistema, si no hemos creado ninguna Base de Datos, la Base de Datos en uso será la que contiene los metadatos sobre los objetos almacenados en ese SGBD y se llamará de diferente forma.

MySQL / MariaDB:

El usuario administrador por defecto es `root`. Cuando nos conectemos podemos seleccionar la base de datos a utilizar mediante la instrucción:

```
USE nombreBD;
```

SQL Server:

El usuario administrador por defecto es `sa` (Super Admin). Cuando nos conectemos podemos seleccionar la base de datos a utilizar mediante la instrucción:

```
USE nombreBD
GO
```

Cuando nos conectamos, si no hemos establecido conexión con ninguna base de datos estaremos en la base de datos `MASTER`

Oracle:

Durante la instalación de Oracle se instalan dos cuentas administrativas y otras dos con permisos especiales para tareas de optimización y monitorización de la base de datos:

- `SYS` : Inicialmente posee la contraseña **CHANGE_ON_INSTALL** que, lógicamente, hay que cambiar inmediatamente en la instalación. SYS toma rol de DBA (es decir, de superadministrador) y es en su esquema donde se crea el diccionario de datos; por lo que no conviene de ninguna manera crear otro tipo de elementos en su esquema; es decir, el usuario SYS no debe crear tablas, ni vistas no ningún otro objeto de la base de datos.
- `SYSTEM` : Posee también el rol DBA y se crea durante la instalación. La contraseña **MANAGER** que tiene por defecto se debería cambiar en la instalación. En su esquema se suelen crear tablas y vistas administrativas (pero no se deberían crear otro tipo de tablas).
- `SYSMAN` : Usado para realizar tareas administrativas con la aplicación Database Control del Enterprise Manager.
- `DBSMNP` : Usuario que tiene permisos para monitorizar Enterprise Manager.

En Oracle, la conexión a la base de datos con un usuario me determinará que "base de datos" estoy utilizando.

5. Restricciones SQL

Las restricciones en SQL *(SQL constraints)* se usan para especificar reglas que deben cumplir los datos en una tabla.

Creación de restricciones SQL (Constraints)

Las restricciones se pueden especificar cuando la tabla se crea con la instrucción `CREATE TABLE`, o después de que la tabla se haya creado con la instrucción `ALTER TABLE`.

Sintaxis

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```

SQL Constraints

Las restricciones de SQL se utilizan para especificar reglas para los datos en una tabla.

Las restricciones se utilizan para limitar el tipo de datos que pueden ir a una tabla. Esto garantiza la precisión y fiabilidad de los datos en la tabla. Si hay alguna violación entre la restricción y la acción sobre los datos, la acción se cancelará.

Las restricciones pueden ser a nivel de **columna** o de **tabla**. Las restricciones de nivel de columna se aplican a una columna, y las restricciones de nivel de tabla se aplican a toda la tabla.

Las siguientes restricciones se usan comúnmente en SQL:

- `NOT NULL` : garantiza que una columna no pueda tener un valor *NULL*.
- `UNIQUE` : garantiza que todos los valores de una columna sean diferentes.
- `PRIMARY KEY` : identifica a la clave primaria de la tabla, que en realidad es como una combinación de *NOT NULL* y *UNIQUE*. Identificará de forma única a cada fila en una tabla.
- `FOREIGN KEY` : será una restricción que identificará de forma única a una fila o registro en otra tabla, a la cual se referencia.
- `CHECK` : asegura que todos los valores en una columna satisfagan una condición específica.
- `DEFAULT` : establece un valor predeterminado para una columna cuando no se especifica ningún valor.
- `INDEX` : los indices pueden convertirse en restricciones si especifican valores únicos para la columna o columnas que lo forman. Se utilizan para crear y recuperar datos de la base de datos más rápidamente, es decir, para mejorar la eficiencia en el acceso a los datos.

Vamos a ir estudiándolas una por una en esta unidad.

6. SQL NOT NULL Constraint

Restricciones SQL NOT NULL

Por defecto, una columna puede contener valores `NULL` *(ausencia de valor)*.

La restricción `NOT NULL` impone a una columna la condición de **NO** aceptar valores NULL.

Esto exige que un campo contenga siempre un valor, lo que significa que no se puede insertar un nuevo registro o actualizarlo sin agregar un valor a este campo.

SQL NOT NULL en CREATE TABLE

Durante la creación de la tabla `Persons` , la siguiente sentencia SQL se asegura de que las columnas `ID`, `LastName`, y `FirstName` no puedan aceptar valores nulos *(NOT NULL)*:

Ejemplo

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```

SQL NOT NULL en ALTER TABLE

Para crear una restricción **NOT NULL** sobre la columna `Age` cuando la tabla `Persons` ya ha sido creada, podríamos usar la siguiente sentencia SQL:

```
ALTER TABLE Persons
MODIFY Age int NOT NULL;
```

7. SQL UNIQUE Constraint

Restricciones SQL UNIQUE

La restricción `UNIQUE` asegura que todos los valores en una columna son diferentes.

Las restricciones **UNIQUE** y **PRIMARY KEY** proporcionan una garantía de unicidad para una columna o conjunto de columnas.

Una restricción **PRIMARY KEY** tiene automáticamente una restricción **UNIQUE**.

Sin embargo, se pueden tener muchas restricciones **UNIQUE** por tabla, pero solo una restricción **PRIMARY KEY** por tabla.

SQL UNIQUE Constraint en CREATE TABLE

La siguiente sentencia SQL crea una restricción **UNIQUE** sobre la columna `ID` cuando la tabla `Persons` se está creando:

> Como sabemos, SQL es un lenguaje “estandar”, pero a la hora de implementarlo, algunos fabricantes han optado por pequeñas diferencias en su sintaxis. A partir de ahora, voy a indicar esas diferencias entre los *Sistemas Gestores de Bases de Datos* más utilizados en la actualidad, cuando existan diferencias.

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL UNIQUE,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

MySQL:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    UNIQUE (ID)
);
```

Para nombrar una restricción UNIQUE y para definir una restricción UNIQUE en varias columnas, podremos usar la siguiente sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT UC_Person UNIQUE (ID,LastName)
);
```

SQL UNIQUE Constraint en ALTER TABLE

Para crear una restricción UNIQUE en la columna `ID` cuando la tabla ya está creada, usamos la siguiente sentencia SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD UNIQUE (ID);
```

Para establecer un nombre a una restricción UNIQUE y para definir una restricción UNIQUE en varias columnas, usaremos la siguiente sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
```

Eliminar una restricción UNIQUE

Para eliminar una restricción UNIQUE, usamos la siguiente sentencia SQL:

MySQL:

```
ALTER TABLE Persons
DROP INDEX UC_Person;
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT UC_Person;
```

8. SQL PRIMARY KEY Constraint

Restricciones SQL PRIMARY KEY

La restricción `PRIMARY KEY` (*clave primaria*) identifica de forma exclusiva cada registro en una tabla.

Las claves primarias deben contener valores ÚNICOS (*UNIQUE*) y no pueden contener valores NULL.

Una tabla solo puede tener UNA clave primaria y en la tabla, esta clave primaria puede consistir en una o varias columnas (campos).

SQL PRIMARY KEY en CREATE TABLE

La siguiente sentencia SQL crea una CLAVE PRIMARIA en la columna `ID` cuando se crea la tabla `Persons`:

MySQL:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

Para permitir dar un nombre a una restricción PRIMARY KEY o para definir una restricción PRIMARY KEY en varias columnas, usaremos la siguiente sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

> **Nota:** En el ejemplo anterior solo hay UNA CLAVE PRIMARIA (`PK_Person`). Sin embargo, el VALOR de la clave primaria se compone de DOS COLUMNAS (`ID` + `LastName`).

SQL PRIMARY KEY en ALTER TABLE

Para crear una restricción PRIMARY KEY en la columna `ID` cuando la tabla ya está creada, usaremos la siguiente sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD PRIMARY KEY (ID);
```

Para permitir nombrar una restricción PRIMARY KEY o para definir una restricción PRIMARY KEY en varias columnas, usaremos la siguiente sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);
```

> **Nota:** Si utilizamos la instrucción ALTER TABLE para agregar una clave primaria, las columnas de la clave primaria ya deben haberse declarado que no contienen valores NULL (cuando se creó la tabla por primera vez).

Eliminar una restricción PRIMARY KEY

Para eliminar una restricción PRIMARY KEY, usaremos la siguiente sintaxis SQL:

MySQL:

```
ALTER TABLE Persons
DROP PRIMARY KEY;
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT PK_Person;
```

> **Nota:** debemos observar que en la mayoria de los Sistemas Gestores de Bases de Datos, para poder eliminar una clave primaria necesitamos saber su nombre.

9. SQL FOREIGN KEY Constraint

Restricciones SQL FOREIGN KEY

Una `FOREIGN KEY` es una clave utilizada para vincular dos tablas.

Una FOREIGN KEY es un campo (o colección de campos) en una tabla que se refiere a la CLAVE PRIMARIA (`PRIMARY KEY`) en otra tabla.

La tabla que contiene la clave externa se llama tabla secundaria o hija, y la tabla que contiene la clave candidata se denomina tabla principal, padre o referenciada.

Supongamos las siguientes dos tablas:

`Persons` table:

PersonID	LastName	FirstName	Age
1	Heredia	Diego	30
2	Lopez	Juan	23
3	Gonzalez	Enrique	40

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Observamos que la columna `PersonID` en la tabla pedidos `Orders` apunta a la columna `PersonID` en la tabla personas `Persons`.

La columna `PersonID` en la tabla `Persons` es la CLAVE PRIMARIA en la tabla `Persons`.

La columna `PersonID` en la tabla `Orders` es una CLAVE AJENA o **FOREIGN KEY** en la tabla `Orders`.

La restricción FOREIGN KEY se usa para evitar acciones que destruirían los enlaces entre tablas.

La restricción FOREIGN KEY también evita que se inserten datos no válidos en la columna de clave ajena, porque debe ser uno de los valores contenidos en la tabla a la que apunta o referencia.

SQL FOREIGN KEY en CREATE TABLE

La siguiente sentencia SQL crea una FOREIGN KEY en la columna `PersonID` cuando se crea la tabla `Orders`:

MySQL:

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Orders (
    OrderID int NOT NULL PRIMARY KEY,
    OrderNumber int NOT NULL,
    PersonID int FOREIGN KEY REFERENCES Persons(PersonID)
);
```

Para permitir nombrar una restricción FOREIGN KEY o para definir una restricción FOREIGN KEY en varias columnas, usaremos la siguiente sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
    REFERENCES Persons(PersonID)
);
```

SQL FOREIGN KEY en ALTER TABLE

Para crear una restricción FOREIGN KEY en la columna `PersonID` cuando la tabla `Orders` ya está creada, usaremos la siguiente sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

Para permitir nombrar una restricción FOREIGN KEY o para definir una restricción FOREIGN KEY en varias columnas, usaremos la siguiente sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
ADD CONSTRAINT FK_PersonOrder
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

Eliminar una restricción FOREIGN KEY

Para eliminar una restricción FOREIGN KEY, usaremos la siguiente sintaxis SQL:

MySQL:

```
ALTER TABLE Orders
DROP FOREIGN KEY FK_PersonOrder;
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
DROP CONSTRAINT FK_PersonOrder;
```

10. SQL CHECK Constraint

Restricciones SQL CHECK

La restricción `CHECK` se usa para limitar el rango de valores que se puede colocar en una columna.

Si definimos una restricción CHECK en una sola columna, solo permite ciertos valores para esta columna.

Si definimos una restricción CHECK en una tabla, puede limitar los valores en ciertas columnas según los valores en otras columnas de la misma fila o registro.

SQL CHECK en CREATE TABLE

La siguiente sentencia SQL crea una restricción CHECK en la columna `Age` cuando se crea la tabla `Persons`. La restricción CHECK asegura que la edad de una persona debe ser mayor de 18 años:

MySQL:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int CHECK (Age>=18)
);
```

Para permitir nombrar una restricción CHECK o para definir una restricción CHECK en varias columnas, usaremos la siguiente sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Lorca')
);
```

SQL CHECK en ALTER TABLE

Para crear una restricción CHECK en la columna `Age` cuando la tabla ya está creada, usaremos la siguiente sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CHECK (Age>=18);
```

Para permitir nombrar una restricción CHECK o para definir una restricción CHECK en varias columnas, usaremos la siguiente sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Lorca');
```

Eliminar una restricción CHECK

Para eliminar una restricción CHECK, usaremos la siguiente sintaxis SQL:

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT CHK_PersonAge;
```

MySQL:

```
ALTER TABLE Persons
DROP CHECK CHK_PersonAge;
```

11. SQL DEFAULT Constraint

Restricciones SQL DEFAULT

La restricción `DEFAULT` se usa para proporcionar un valor predeterminado para una columna.

El valor predeterminado se agregará a todos los registros nuevos SI no se especifica ningún otro valor.

SQL DEFAULT en CREATE TABLE

La siguiente sentencia SQL establece un valor por defecto (*DEFAULT*) para la columna `City` cuando se crea la tabla `Persons`:

My SQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Lorca'
);
```

La restricción DEFAULT también se puede usar para insertar valores del sistema, mediante el uso de funciones como `GETDATE()` :

```
CREATE TABLE Orders (
    ID int NOT NULL,
    OrderNumber int NOT NULL,
    OrderDate date DEFAULT GETDATE()
);
```

SQL DEFAULT en ALTER TABLE

Para crear una restricción DEFAULT en la columna `City` cuando la tabla ya está creada, usaremos la siguiente sintaxis SQL:

MySQL:

```
ALTER TABLE Persons
ALTER City SET DEFAULT 'Lorca';
```

SQL Server:

```
ALTER TABLE Persons
ADD CONSTRAINT df_City
DEFAULT 'Lorca' FOR City;
```

MS Access:

```
ALTER TABLE Persons
ALTER COLUMN City SET DEFAULT 'Lorca';
```

Oracle:

```
ALTER TABLE Persons
MODIFY City DEFAULT 'Lorca';
```

Eliminar una restricción DEFAULT

Para eliminar una restricción DEFAULT, usaremos la siguiente sintaxis SQL:

MySQL:

```
ALTER TABLE Persons
ALTER City DROP DEFAULT;
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ALTER COLUMN City DROP DEFAULT;
```

12. Índices con CREATE INDEX

Sentencia SQL CREATE INDEX

La instrucción `CREATE INDEX` se usa para crear índices en tablas.

Los índices se utilizan para recuperar datos de la base de datos más rápidamente que de otra manera. Los usuarios no pueden ver los índices, solo se usan para acelerar las búsquedas o consultas.

> **Nota:** Actualizar una tabla con índices lleva más tiempo que actualizar una tabla sin ellos (porque los índices también necesitan una actualización). Por lo tanto, solo debemos crear índices en columnas que se buscarán con frecuencia.

Sintaxis de CREATE INDEX

Crea un índice en una tabla permitiendo valores duplicados:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

Sintaxis de CREATE UNIQUE INDEX

Crea un índice único en una tabla. No se permiten valores duplicados:

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

> **Nota:** La sintaxis para crear índices varía entre las diferentes bases de datos. Por lo tanto debemos verificar la sintaxis correcta para crear índices según la base de datos que estemos utilizando.

Ejemplo de CREATE INDEX

La siguiente instrucción SQL crea un índice llamado `idx_lastname` en la columna `LastName` en la tabla `Persons`:

```
CREATE INDEX idx_lastname
ON Persons (LastName);
```

Si deseamos crear un índice en una combinación de columnas, podemos insertar los nombres de las columnas entre paréntesis, separados por comas:

```
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
```

Sentencia DROP INDEX

La instrucción DROP INDEX se usa para eliminar un índice en una tabla.

MS Access:

```
DROP INDEX index_name ON table_name;
```

SQL Server:

```
DROP INDEX table_name.index_name;
```

DB2/Oracle:

```
DROP INDEX index_name;
```

MySQL:

```
ALTER TABLE table_name
DROP INDEX index_name;
```

13. CONCEPTOS SOBRE LOS DATOS

JUEGO DE CARACTERES o CHARSET

En una base de datos se almacenan caracteres (cada letra, número, símbolo, es un carácter), y estos deben ser reconocibles por quien los lee.

Como existen miles de caracteres en el mundo y su uso varía dependiendo de la región geográfica, lengua, etc. **debemos colocarle a nuestra base de datos un charset para que admita los caracteres que nos interesa**.

Por ejemplo, si hablas chino mandarín, querrás tener un charset que admita los caracteres propios de esta lengua. En el caso del español, es necesario que la base de datos reconozca la ñ o las tildes, por ejemplo.

Teniendo esto claro, verás la **importancia de escoger el charset adecuado al momento de crear una base de datos**, ya que después de hacer la conversión puede ser algo complejo.

MySQL / MariaDB

El Sistema Gestor de Bases de Datos (SGBD) tendrá un juego de caracteres por defecto que será el que se aplique siempre que en la creación de una base de datos no se especifique.

Si queremos especificar el juego de caracteres durante la creación de una base de datos lo podemos hacer utilizando la siguiente sintaxis.

```
CREATE DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]
```

Como las bases de datos están en constante mejora, hasta hace poco el **Utf8** era el recomendado, pero ahora puede que haya otros más recomendables, se recomienda usar **Utf8mb4** que es lo actual y además, **es totalmente compatible**.

El por qué de la aparición de este juego de caracteres es porque el Utf8 inicial de *MySQL* estaba formado por 3 bytes en lugar de los 4 que son necesarios. Puedes consultar sus caracteristicas en (https://dev.mysql.com/doc/refman/8.0/en/charset-unicode-utf8mb4.html)

Los Sistemas de Gestión de Contenidos (CMS o Content management system) actuales, para agregar nuevas funcionalidades requieren usar el **Utf8mb4**. Un ejemplo típico de esto, son los emojis, sin él no tendrás **soporte completo** para ellos. **Utf8mb4 soporta hasta 4 bytes en cada carácter**

Estos emojis no podrás colocarlos en tu web: 🤗😎ES si el juego de caracteres del campo donde se almacenan es Utf8, puesto que no se almacenarán correctamente en la Base de Datos.

SQL Server:

Los tipos `Unicode` son: `nchar`, `nvarchar` y `ntext`.

Tipos de datos de texto **NO UNICODE** son: `char`, `varchar` y `text`.

Para el caso de este Sistema Gestor de Bases de Datos (SGBD) el Juego de caracteres lo define la intercalación que estemos utilizando, así que lo veremos en el apartado siguiente.

Oracle:

Los tipos `Unicode` son: `nchar`, `nvarchar2` y `nclob`.

Tipos de datos de texto **NO UNICODE** son: `char`, `varchar2` y `clob`.

Puedes consultar toda la información sobre como elegir un juego de caracteres en Oracle en la web de Oracle
https://docs.oracle.com/en/database/oracle/oracle-database/18/nlspg/choosing-character-set.html#GUID-BF26E01D-AB92-48FC-855A-69A5B3AF9A92

COLLATION o INTERCALACIÓN

También llamado *Cotejamiento*, son las reglas que **permiten comparar caracteres en la base de datos y decidir como se ordenan**.

Por ejemplo: el manejo de las doble L: "`LL`", se manejará y ordenará diferente dependiendo del cotejamiento o intercalación que tengamos.

Evidentemente, se trata de un concepto que **afecta sólo a los campos de texto**.

El cotejamiento está intimamente relacionado con el charset que tengamos.

Cada Sistema Gestor de Bases de Datos (SGBD) define sus propias intercalaciones, así por ejemplo

MySQL / MariaDB:

Tiene definida, entre otras, la siguiente intercalación:

Charset: Utf8mb4 ➞ **Collation**: Utf8mb4_unicode_520_ci

Los nombres de las intercalaciones en MySQL han empezado a tener un sentido, por ejemplo:

- `_ci` indicará que en esa intercalación se considerán **iguales las mayúsculas que las minúsculas**, o lo que es lo mismo, *case insensitive*.
- `_cs` indicará que en esa intercalación son **diferentes las mayúsculas que lass minúsculas**, o lo que es lo mismo, *case sensitive*.
- `_ai` indicará que en esa intercalación se considerán **iguales las letras acentuadas o no**, o lo que es lo mismo, *accent insensitive*.

SQL Server

Las intercalaciones en SQL Server, especifican tres propiedades:

1. Cómo se ordenan los tipos de datos `Unicode` (`nchar`,`nvarchar` y `ntext`). El orden define la secuencia en la cual son ordenados los caracteres y la forma en que son evaluados en operaciones de comparación.
2. Cómo se ordenan los tipos de datos **NO UNICODE** (`char`,`varchar` y `text`).
3. La página de códigos del juego de caracteres usado para almacenar los tipos de datos **NO UNICODE**.

En SQL Server también existen normas para poner nombre a las intercalaciones. Existiendo dos tipos.

Por compatibilidad, se mantienen intercalaciones de versiones antiguas de SQL Server, que no utilizan caracteres Unicode. La sintaxis de estos nombres será:

```
SQL_<alfabeto>[_Pref_]_CP<codepage>_<estilo>
```

Donde:

- `<alfabeto>` : identifica el alfabeto o el idioma cuyas reglas de ordenación se aplican cuando se especifica el orden del diccionario. Por ejemplo: `Latin1_General` o `Polish`.
- `_Pref_` : es opcional y **especifica la preferencia de mayúsculas**. Aunque la comparación distinga entre mayúsculas y minúsculas, la versión en mayúsculas de una letra se ordenará antes que la versión en minúsculas, cuando no existe ningún otro tipo de diferenciación.
- `<codepage>` : especifica un número de uno a cuatro dígitos que identifica la página de códigos que la intercalación utiliza. `CP1` especifica la página de códigos 1252; en los demás casos se especifica el número de página de códigos completo. Por ejemplo, `CP1251` especifica la página de códigos 1251 y `CP850` especifica la página de códigos 850.
- `<estilo>` : en cuanto a estilo podemos definir.
  - *Case Sensitivity* `CI` especifica que no se diferencia mayúsculas de minúsculas, `CS` especifica que se diferencia mayúsculas de minúsculas.
  - *Accent Sensitivity* `AI` especifica que no se distinguen acentos, `AS` especifica que se distinguen acentos.
  - `BIN` especifica el criterio de ordenación binario que se va a utilizar.

Para enumerar las intercalaciones de SQL Server admitidas por el servidor, ejecutamos la consulta siguiente.

```
SELECT * FROM sys.fn_helpcollations()
WHERE name LIKE 'SQL%';
```

`SQL_EBCDIC284_CP1_CS_AS` : se describe como *Modern-Spanish, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 215 on Code Page 1252 for non-Unicode Data*

> ⚠ *Atención*❗*:* estas intercalaciones que se mantienen por compatibilidad, no son para juegos de caracteres *Unicode*, por tanto, no deberíamos utilizarlas para la creación de nuevas Bases de Datos.

El otro tipo de intercalaciones, son las que Microsoft asocia de forma predeterminada **según la configuración regional de Windows** una **intercalación predeterminada** distinta.

Por ejemplo:

Configuración regional de Windows	LCID* de Windows	LCID* de SQL	Intercalación predeterminada
Afrikáans (Sudáfrica)	0x0436	0x0409	Latin1_General_CI_AS
Inglés (Reino Unido)	0x0809	0x0409	Latin1_General_CI_AS
Spanish (Traditional Sort) - Spain	0x0409	0x0409	SQL_Latin1_General_CP1_CI_AS
Español (España)	0x0c0a	0x0c0a	**Modern_Spanish_CI_AS**
Español (España, tradicional)	0x040a	0x040a	Traditional_Spanish_CI_AS
Español (Estados Unidos)	0x540a	0x0409	Latin1_General_CI_AS

**LCID: Identificador de Codigo de Idioma*

Si deseamos conocer cual es lal intercalación que está activa en nuestro MS SQL Server, podemos ejecutar la siguiente consulta:

```
SELECT SERVERPROPERTY('collation');
```

Que en nuestro caso, nos va a devolver `SQL_Latin1_General_CP1_CI_AS` , ello se debe a que durante la configuración de SQL Server, la opción de intercalación de instalación predeterminada viene determinada por la configuración regional del sistema operativo (SO). Para la compatibilidad con versiones anteriores, la intercalación predeterminada se establece en la **versión más antigua** disponible que esté asociada a cada configuración regional concreta. **Por tanto, esta no es siempre la intercalación recomendada.** Para aprovechar al máximo las características de SQL Server, podríamos cambiar la configuración de instalación predeterminada para que use las intercalaciones de Windows.

En nuestro caso, la imagen docker que utilizamos, la hicieron para la configuración regional del sistema operativo *"Inglés (Estados Unidos)"* (página de códigos 1252), cuya intercalación predeterminada durante la instalación es `SQL_Latin1_General_CP1_CI_AS` y se puede cambiar a la intercalación de Windows homóloga más cercana, `Latin1_General_100_CI_AS`, **o podemos tenerlo en cuenta a la hora de crear nuestras Bases de Datos en este servidor**.

Para saber más sobre como se definen las intercalaciones en **Microsoft SQL Server**, podemos consultar la siguiente documentación (https://docs.microsoft.com/es-es/sql/relational-databases/collations/collation-and-unicode-support?view=sql-server-ver15), o ver ejemplos de uso de intercalaciones en (https://www.red-gate.com/simple-talk/sql/sql-development/questions-sql-server-collations-shy-ask/)

Oracle:

¿Dónde se definen estos conceptos?

Cada instancia o motor de Sistema Gestor de Bases de Datos (SGBD), tendrá definido un Juego de Caracteres y una intercalación por defecto para la base de datos del sistema y las bases de datos que creen los usuarios.

En el momento de crear una Base de Datos si no se especifica lo contrario, se utilizarán los valores por defecto, pero en ese momento o posteriormente se puede definir que **la intercalación (Collation) de cada BASE DE DATOS pueda ser diferente**.

Siendo más fino, dentro de una base de datos se pueden aplicar también **intercalaciones diferentes en distintas TABLAS**, e incluso se podrán aplicar a nivel de **COLUMNAS** de una tabla.

Además los Sistema Gestor de Bases de Datos (SGBD) como **SQL Server** u **Oracle** también permiten especificar un tipo de intercalación en el criterio de ordenación o comparación de una consulta, por ejemplo: (más informarción)

```
SELECT apellidos FROM TPRUEBA
ORDER BY apellidos
COLLATE Traditional_Spanish_ci_ai ASC;
```

13.1. Ejemplo

```
CREATE TABLE APELLIDOS(
    id INT IDENTITY PRIMARY KEY,
    apellido VARCHAR(50) NOT NULL);

INSERT INTO APELLIDOS(apellido)
  VALUES('Carrión'), ('Carrion'), ('carrión'), ('carrion'),
    ('Chaves'), ('Chavira'), ('Cúneo'), ('Cuneo'), ('cúneo');
```

14. DROP DATABASE

Instrucción DROP DATABASE

La instrucción `DROP DATABASE` se usa para **eliminar** una base de datos SQL existente.

Sintaxis:

```
DROP DATABASE databasename;
```

> ⚠ *Atención*❗*:* Debemos tener cuidado antes de eliminar una base de datos. 🗑 Eliminar una base de datos supone la pérdida de la información completa almacenada en la base de datos.

DROP DATABASE Ejemplo

La siguiente instrucción SQL elimina la base de datos existente “testDB”:

```
DROP DATABASE testDB;
```

15. CREATE TABLE

Instrucción CREATE TABLE

La instrucción `CREATE TABLE` se usa para crear una nueva tabla en una base de datos.

Sintaxis:

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ….
);
```

Los parámetros **column** especifican los nombres de las columnas, campos o atributos de la tabla.

El parámetro **datatype** especifica el tipo de datos que la columna puede contener (por ejemplo, *varchar*, *entero*, *fecha*, etc.).

Más adelante veremos una descripción general de los tipos de datos disponibles, en el apartado **TIPOS DE DATOS**.

SQL CREATE TABLE Ejemplo

El siguiente ejemplo crea una tabla llamada *Personas* (`Persons`) que contiene cinco columnas *PersonID*, *Apellido*, *Nombre*, *Dirección* y *Ciudad*, `PersonID`, `LastName`, `FirstName`, `Address` y `City` respectivamente.

Ejemplo:

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

La columna `PersonID` es de tipo `int` y contendrá un número entero.

Las columnas `LastName`, `FirstName`, `Address` y `City` son de tipo `varchar` , almacenarán caracteres y la longitud máxima para estos campos es de 255 caracteres.

La tabla vacía *"Personas"* `Persons` ahora se verá así:

PersonID	LastName	FirstName	Address	City

Ahora en la tabla `Persons` se pueden insertar datos con la instrucción de SQL `INSERT INTO`.

CREATE TABLE utilizando otra tabla

También se puede crear una copia de una tabla existente usando `CREATE TABLE`.

La nueva tabla obtiene las mismas definiciones de columna. Se pueden utilizar todas las columnas o especificar algunas de ellas.

Si creamos una nueva tabla usando una tabla existente, la nueva tabla será rellenada con los valores existentes de la tabla anterior.

Sintaxis:

```
CREATE TABLE new_table_name AS
    SELECT column1, column2,...
    FROM existing_table_name
    WHERE ....;
```

La siguiente sentencia SQL crea una nueva tabla llamada `TestTable` (que es un copia de la tabla *Clientes* `Customers`):

Ejemplo:

```
CREATE TABLE TestTable AS
SELECT customername, contactname
FROM Customers;
```

---

16. DROP TABLE

Instrucción DROP TABLE

La instrucción `DROP TABLE` se usa para **eliminar** una tabla existente en una base de datos.

Sintaxis:

```
DROP TABLE table_name;
```

⚠ *Atención*❗*:* Eliminar una tabla provocará la pérdida de todos los datos almacenados en esa tabla

SQL DROP TABLE Ejemplo

La siguiente instrucción SQL elimina la tabla existente *Remitentes* `Shippers`:

Ejemplo:

```
DROP TABLE Shippers;
```

---

17. ALTER TABLE

Instrucción ALTER TABLE

La instrucción `ALTER TABLE` se usa para agregar, eliminar o modificar columnas en una tabla existente.

La instrucción `ALTER TABLE` también se usa para añadir o eliminar restricciones (`CONSTRAINTS`) en una tabla existente.

ALTER TABLE - ADD Column

Para agregar una columna en una tabla, usamos la siguiente sintaxis:

```
ALTER TABLE table_name
ADD column_name datatype;
```

La siguiente sentencia SQL agrega una columna *Correo electrónico* `Email` a la tabla "Clientes" `Customers`:

```
ALTER TABLE Customers
ADD Email varchar(255);
```

ALTER TABLE - DROP COLUMN

Para eliminar una columna en una tabla, usamos la siguiente sintaxis

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

⚠ *Atención!:* Algunos sistemas gestores de bases de datos no permiten eliminar una columna

Ejemplo:

La siguiente sentencia SQL elimina la columna *Correo electrónico* `Email` de la tabla *Clientes* `Customers`:

```
ALTER TABLE Customers
DROP COLUMN Email;
```

ALTER TABLE - ALTER/MODIFY COLUMN

Para cambiar el tipo de datos de una columna en una tabla, usaremos la siguiente sintaxis:

SQL Server / MS Access:

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype;
```

MySQL / MariaDB / Oracle (versiones <10G):

```
ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```

Oracle 10G y superiores:

```
ALTER TABLE table_name
MODIFY column_name datatype;
```

Ejemplo ALTER TABLE

Supongamos que tenemos la tabla `Persons` con el siguiente contenido:

ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Ahora queremos agregar una columna para la fecha de nacimiento, llamada `DateOfBirth` en la tabla *Personas* `Persons`.

Usaremos la siguiente instrucción SQL:

```
ALTER TABLE Persons
ADD DateOfBirth date;
```

Observaremos que la nueva columna, `DateOfBirth`, es de tipo `date` y va a almacenar una fecha. El *datatype* especifica qué tipo de datos puede almacenar la columna.

La tabla *Personas* ahora se verá así:

ID	LastName	FirstName	Address	City	DateOfBirth
1	Hansen	Ola	Timoteivn 10	Sandnes	
2	Svendson	Tove	Borgvn 23	Sandnes	
3	Pettersen	Kari	Storgt 20	Stavanger	

Ejemplo de Cambio del Tipo de Datos

Ahora queremos cambiar el tipo de datos de la columna denominada `DateOfBirth` en la tabla *Personas* `Persons`.

Usamos la siguiente instrucción SQL:

```
ALTER TABLE Persons
ALTER COLUMN DateOfBirth year;
```

Observaremos que la columna `DateOfBirth` ahora es de tipo `year` (año) y va a almacenar un año en un formato de dos o cuatro dígitos.

Ejemplo DROP COLUMN

A continuación, queremos eliminar la columna denominada `DateOfBirth` en la tabla `Persons`.

Usamos la siguiente instrucción SQL:

```
ALTER TABLE Persons
DROP COLUMN DateOfBirth;
```

La tabla "*Personas*" ahora se verá así:

ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

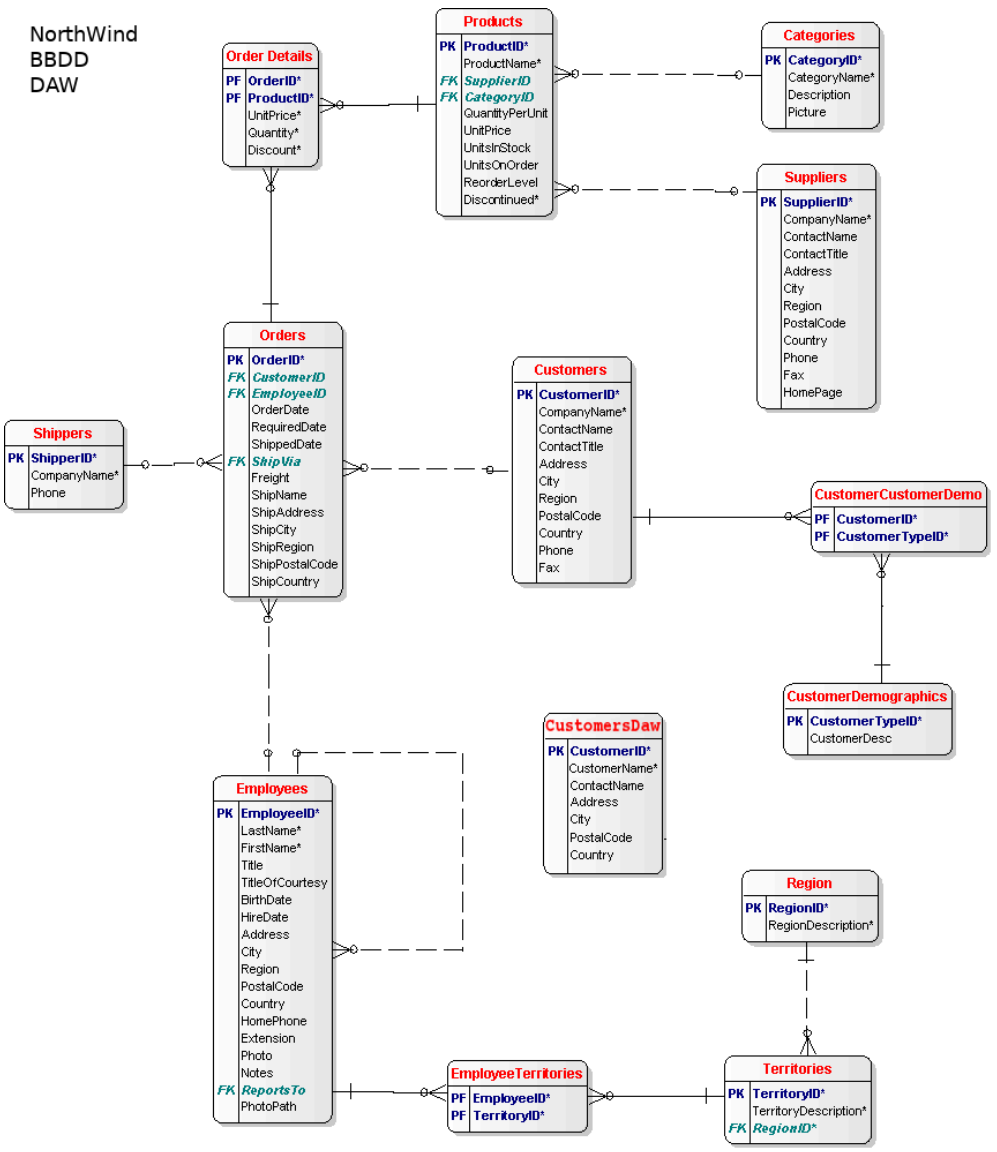
18. Base de Datos de Ejemplo

Dentro de los apuntes de nuestro módulo `Base de Datos` de **Desarrollo de Aplicaciones Web**, para realizar las explicaciones de las instrucciones del lenguaje SQL, vamos a trabajar con una base de datos de Ejemplo a la que hemos añadido una serie de tablas. Aquí se muestra como construir dichas tablas y un ejemplo de su contenido.

Un Saludo (***Diego Heredia***)

Base de Datos de Ejemplo

Sobre la base de datos `Northwind` vamos a crear una serie de tablas con datos reducidos que nos van a servir para ilustrar los contenidos que quiero explicaros.



Si deseas ver el contenido de dichas tablas, se incluye al final de esta página, pero puedes pulsar sobre el enlace del nombre de la tabla y te llevará directamente.

- `CustomersDaw` : una pequeña muestra de la tabla `Customers`.
- `Customers` : tabla `Customers` completa.

Script para generar las nuevas tablas

Para construir las tablas ejecuta el siguiente código SQL, teniendo previamente instalada la base de datos `Northwind`.

```
USE Northwind
GO

DROP CustomersDaw IF EXISTS

SELECT TOP 5 ROW_NUMBER() OVER (ORDER BY CustomerID) AS CustomerID,
  CompanyName as CustomerName,
  ContactName, Address, City, PostalCode, Country
INTO CustomersDaw
FROM Customers
GO

ALTER TABLE CustomersDaw
ALTER COLUMN CustomerID INT NOT NULL;
GO
```

```

ALTER TABLE CustomersDaw
ADD PRIMARY KEY (CustomerID);
GO
```

CustomersDaw

A continuación se ***muestra*** el contenido de la tabla `CustomersDaw`, que es una pequeña selección del contenido de la tabla *Clientes* `Customers` :

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Customers

A continuación se ***muestra*** el contenido de la tabla *Clientes* `Customers` de la base de datos `Northwind` :

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
7	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France
8	Bólido Comidas preparadas	Martín Sommer	C/ Araquil, 67	Madrid	28023	Spain
9	Bon app’	Laurence Lebihans	12, rue des Bouchers	Marseille	13008	France
10	Bottom-Dollar Marketse	Elizabeth Lincoln	23 Tsawassen Blvd.	Tsawassen	T2F 8M4	Canada
11	B’s Beverages	Victoria Ashworth	Fauntleroy Circus	London	EC2 5NT	UK
12	Cactus Comidas para llevar	Patricio Simpson	Cerrito 333	Buenos Aires	1010	Argentina
13	Centro comercial Moctezuma	Francisco Chang	Sierras de Granada 9993	México D.F.	05022	Mexico
14	Chop-suey Chinese	Yang Wang	Hauptstr. 29	Bern	3012	Switzerland
15	Comércio Mineiro	Pedro Afonso	Av. dos Lusíadas, 23	São Paulo	05432-043	Brazil
16	Consolidated Holdings	Elizabeth Brown	Berkeley Gardens 12 Brewery	London	WX1 6LT	UK
17	Drachenblut Delikatessend	Sven Ottlieb	Walserweg 21	Aachen	52066	Germany
18	Du monde entier	Janine Labrune	67, rue des Cinquante Otages	Nantes	44000	France
19	Eastern Connection	Ann Devon	35 King George	London	WX3 6FW	UK
20	Ernst Handel	Roland Mendel	Kirchgasse 6	Graz	8010	Austria
21	Familia Arquibaldo	Aria Cruz	Rua Orós, 92	São Paulo	05442-030	Brazil

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
22	FISSA Fabrica Inter. Salchichas S.A.	Diego Roel	C/ Moralzarzal, 86	Madrid	28034	Spain
23	Folies gourmandes	Martine Rancé	184, chaussée de Tournai	Lille	59000	France
24	Folk och fä HB	Maria Larsson	Åkergatan 24	Bräcke	S-844 67	Sweden
25	Frankenversand	Peter Franken	Berliner Platz 43	München	80805	Germany
26	France restauration	Carine Schmitt	54, rue Royale	Nantes	44000	France
27	Franchi S.p.A.	Paolo Accorti	Via Monte Bianco 34	Torino	10100	Italy
28	Furia Bacalhau e Frutos do Mar	Lino Rodriguez	Jardim das rosas n. 32	Lisboa	1675	Portugal
29	Galería del gastrónomo	Eduardo Saavedra	Rambla de Cataluña, 23	Barcelona	08022	Spain
30	Godos Cocina Típica	José Pedro Freyre	C/ Romero, 33	Sevilla	41101	Spain
31	Gourmet Lanchonetes	André Fonseca	Av. Brasil, 442	Campinas	04876-786	Brazil
32	Great Lakes Food Market	Howard Snyder	2732 Baker Blvd.	Eugene	97403	USA
33	GROSELLA-Restaurante	Manuel Pereira	5ª Ave. Los Palos Grandes	Caracas	1081	Venezuela
34	Hanari Carnes	Mario Pontes	Rua do Paço, 67	Rio de Janeiro	05454-876	Brazil
35	HILARIÓN-Abastos	Carlos Hernández	Carrera 22 con Ave. Carlos Soublette #8-35	San Cristóbal	5022	Venezuela
36	Hungry Coyote Import Store	Yoshi Latimer	City Center Plaza 516 Main St.	Elgin	97827	USA
37	Hungry Owl All-Night Grocers	Patricia McKenna	8 Johnstown Road	Cork		Ireland
38	Island Trading	Helen Bennett	Garden House Crowther Way	Cowes	PO31 7PJ	UK
39	Königlich Essen	Philip Cramer	Maubelstr. 90	Brandenburg	14776	Germany
40	La corne d'abondance	Daniel Tonini	67, avenue de l'Europe	Versailles	78000	France
41	La maison d'Asie	Annette Roulet	1 rue Alsace-Lorraine	Toulouse	31000	France
42	Laughing Bacchus Wine Cellars	Yoshi Tannamuri	1900 Oak St.	Vancouver	V3F 2K1	Canada
43	Lazy K Kountry Store	John Steel	12 Orchestra Terrace	Walla Walla	99362	USA
44	Lehmanns Marktstand	Renate Messner	Magazinweg 7	Frankfurt a.M.	60528	Germany
45	Let's Stop N Shop	Jaime Yorres	87 Polk St. Suite 5	San Francisco	94117	USA
46	LILA-Supermercado	Carlos González	Carrera 52 con Ave. Bolívar #65-98 Llano Largo	Barquisimeto	3508	Venezuela
47	LINO-Delicateses	Felipe Izquierdo	Ave. 5 de Mayo Porlamar	I. de Margarita	4980	Venezuela
48	Lonesome Pine Restaurant	Fran Wilson	89 Chiaroscuro Rd.	Portland	97219	USA
49	Magazzini Alimentari Riuniti	Giovanni Rovelli	Via Ludovico il Moro 22	Bergamo	24100	Italy
50	Maison Dewey	Catherine Dewey	Rue Joseph-Bens 532	Bruxelles	B-1180	Belgium
51	Mère Paillarde	Jean Fresnière	43 rue St. Laurent	Montréal	H1J 1C3	Canada

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
52	Morgenstern Gesundkost	Alexander Feuer	Heerstr. 22	Leipzig	04179	Germany
53	North/South	Simon Crowther	South House 300 Queensbridge	London	SW7 1RZ	UK
54	Océano Atlántico Ltda.	Yvonne Moncada	Ing. Gustavo Moncada 8585 Piso 20-A	Buenos Aires	1010	Argentina
55	Old World Delicatessen	Rene Phillips	2743 Bering St.	Anchorage	99508	USA
56	Ottilies Käseladen	Henriette Pfalzheim	Mehrheimerstr. 369	Köln	50739	Germany
57	Paris spécialités	Marie Bertrand	265, boulevard Charonne	Paris	75012	France
58	Pericles Comidas clásicas	Guillermo Fernández	Calle Dr. Jorge Cash 321	México D.F.	05033	Mexico
59	Piccolo und mehr	Georg Pipps	Geislweg 14	Salzburg	5020	Austria
60	Princesa Isabel Vinhoss	Isabel de Castro	Estrada da saúde n. 58	Lisboa	1756	Portugal
61	Que Delícia	Bernardo Batista	Rua da Panificadora, 12	Rio de Janeiro	02389-673	Brazil
62	Queen Cozinha	Lúcia Carvalho	Alameda dos Canàrios, 891	São Paulo	05487-020	Brazil
63	QUICK-Stop	Horst Kloss	Taucherstraße 10	Cunewalde	01307	Germany
64	Rancho grande	Sergio Gutiérrez	Av. del Libertador 900	Buenos Aires	1010	Argentina
65	Rattlesnake Canyon Grocery	Paula Wilson	2817 Milton Dr.	Albuquerque	87110	USA
66	Reggiani Caseifici	Maurizio Moroni	Strada Provinciale 124	Reggio Emilia	42100	Italy
67	Ricardo Adocicados	Janete Limeira	Av. Copacabana, 267	Rio de Janeiro	02389-890	Brazil
68	Richter Supermarkt	Michael Holz	Grenzacherweg 237	Genève	1203	Switzerland
69	Romero y tomillo	Alejandra Camino	Gran Vía, 1	Madrid	28001	Spain
70	Santé Gourmet	Jonas Bergulfsen	Erling Skakkes gate 78	Stavern	4110	Norway
71	Save-a-lot Markets	Jose Pavarotti	187 Suffolk Ln.	Boise	83720	USA
72	Seven Seas Imports	Hari Kumar	90 Wadhurst Rd.	London	OX15 4NB	UK
73	Simons bistro	Jytte Petersen	Vinbæltet 34	København	1734	Denmark
74	Spécialités du monde	Dominique Perrier	25, rue Lauriston	Paris	75016	France
75	Split Rail Beer & Ale	Art Braunschweiger	P.O. Box 555	Lander	82520	USA
76	Suprêmes délices	Pascale Cartrain	Boulevard Tirou, 255	Charleroi	B-6000	Belgium
77	The Big Cheese	Liz Nixon	89 Jefferson Way Suite 2	Portland	97201	USA
78	The Cracker Box	Liu Wong	55 Grizzly Peak Rd.	Butte	59801	USA
79	Toms Spezialitäten	Karin Josephs	Luisenstr. 48	Münster	44087	Germany
80	Tortuga Restaurante	Miguel Angel Paolino	Avda. Azteca 123	México D.F.	05033	Mexico
81	Tradição Hipermercados	Anabela Domingues	Av. Inês de Castro, 414	São Paulo	05634-030	Brazil
82	Trail's Head Gourmet Provisioners	Helvetius Nagy	722 DaVinci Blvd.	Kirkland	98034	USA
83	Vaffeljernet	Palle Ibsen	Smagsløget 45	Århus	8200	Denmark
84	Victuailles en stock	Mary Saveley	2, rue du Commerce	Lyon	69004	France

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
85	Vins et alcools Chevalier	Paul Henriot	59 rue de l'Abbaye	Reims	51100	France
86	Die Wandernde Kuh	Rita Müller	Adenauerallee 900	Stuttgart	70563	Germany
87	Wartian Herkku	Pirkko Koskitalo	Torikatu 38	Oulu	90110	Finland
88	Wellington Importadora	Paula Parente	Rua do Mercado, 12	Resende	08737-363	Brazil
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland

19. SELECT

SELECT Instrucción

La instrucción `SELECT` se usa para seleccionar datos de una base de datos.

Los datos devueltos se almacenan en una tabla de resultados, llamada conjunto de resultados o `result-set`.

Sintaxis:

```
SELECT column1, column2, …
FROM table_name;
```

Aquí, `column1`, `column2`, … son los nombres de los campos de la tabla que deseamos mostrar u obtener. Si deseamos seleccionar **todos los campos** disponibles en la tabla, useremos la siguiente sintaxis:

```
SELECT * FROM table_name;
```

SELECT de una columna

La siguiente instrucción SQL selecciona `CustomerName` y `City` columnas de la tabla `CustomersDaw`:

Ejemplo

SELECCIONE el nombre del cliente, ciudad de los clientes;

Ejemplo:

```
SELECT CustomerName, City FROM Customers;
```

SELECT * Ejemplo

Si lo que deseamos es seleccionar todas las columnas de una tabla, por ejemplo, la tabla CustomersDaw podemos utilizar el caracter * para hacerlo:

```
SELECT * FROM CustomersDaw;
```

📝❓*Ejercicio SEL_COL_1 ❓:*

> Escribe la sentencia o instrucción SQL para mostrar los campos `City` y `PostalCode` de la tabla `CustomersDaw`

20. SELECT DISTINCT

Instrucción SELECT DISTINCT

La instrucción `SELECT DISTINCT` se usa para devolver solo los valores distintos o diferentes.

Dentro de una tabla, una columna (o un grupo de ellas) a menudo contiene/n muchos valores duplicados; a veces solo deseamos enumerar los diferentes valores (`distinct`).

SELECT DISTINCT Sintaxis

```
SELECT DISTINCT column1, column2, …
FROM table_name;
```

Sobre nuestra tabla de ejemplo CustomersDaw vamos a ir realizando distintas consultas para ilustrar su funcionamiento.

SELECT Ejemplo sin DISTINCT

La siguiente instrucción SQL selecciona TODOS (incluidos los duplicados) valores de la columna *País* `Country` en la tabla `CustomersDaw`:

```
SELECT Country FROM CustomersDaw;
```

|Country| :-: |Germany| |Mexico| |Mexico| |UK| |Sweden|

Ahora, usemos la palabra clave `DISTINCT` con la instrucción SELECT anterior y veremos el resultado.

La siguiente instrucción SQL selecciona solo los valores `DISTINCT` de Columna *País* `Country` en la tabla `CustomersDaw`:

```
SELECT DISTINCT Country FROM CustomersDaw;
```

|Country| :-: |Germany| |Mexico| |UK| |Sweden|

La siguiente instrucción SQL cuenta o enumera el número de diferentes (`DISTINCT`) países de `CustomersDaw`:

```
SELECT COUNT(DISTINCT Country) FROM CustomersDaw;
```

⚠ *Atención❗:* `COUNT (DISTINCT column_name)` no es compatible con Microsoft Access. En este caso la solución sería ejecutar:

```
SELECT Count(*) AS DistinctCountries
FROM (SELECT DISTINCT Country FROM CustomersDaw);
```

📝Ejercicio SEL_DIST_1❓:

> Selecciona y después cuenta las distintas ciudades (`City`) que hay en la tabla `Customers` , ojo **NO** en `CustomersDaw`. Todas las columnas deben tener un nombre de campo.

21. WHERE

Cláusula WHERE

La cláusula `WHERE` se usa para filtrar registros.

La cláusula WHERE se usa para extraer solo aquellos registros que cumplen una condición especificada.

WHERE Sintaxis

```
SELECT column1, column2, …
FROM table_name
WHERE condition;
```

La cláusula WHERE no solo se usa en la instrucción SELECT , sino que también se utiliza en actualizaciones `UPDATE`, eliminaciones `DELETE` y muchos otros sitios.

Trabajaremos con nuestra tabla de ejemplo CustomersDaw

WHERE Ejemplo

La siguiente instrucción SQL selecciona a todos los clientes del país “México”, en la tabla *ClientesDaw* `CustomersDaw` :

Ejemplo:

```
SELECT * FROM CustomersDaw
WHERE Country='Mexico';
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

Campos de texto versus campos numéricos

SQL requiere comillas simples alrededor de **valores de texto** (la mayoría de los sistemas de bases de datos también permitirá comillas dobles, pero **SQL Server no**).

Sin embargo, los **valores de campos numéricos** no deben estar entre comillas:

```
SELECT * FROM CustomersDaw
WHERE CustomerID=1;
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany

Operadores en la cláusula WHERE

Los siguientes operadores se pueden utilizar en la cláusula WHERE:

Operador	Descripción
`=`	Igual
`>`	Mayor que
`<`	Menor que
`>=`	Mayor o igual que
`<=`	Menor o igual que
`<>`	No es igual (o distinto). Ojo: En algunas versiones de SQL este operador puede escribirse como `!=`
`BETWEEN`	Entre un cierto rango
`LIKE`	Busca un patrón
`IN`	Para especificar múltiples valores posibles para una columna

Ejercicio SEL_WHE_1 ❓:

Sobre nuestra tabla CustomersDaw selecciona todos los registros donde la columna Ciudad tenga el valor “Berlin”. Prueba después a que el valor sea “berlin” y posteriormente “Berlín” y da una explicación a lo que te esté pasando.

22. Operadores AND, OR y NOT

Operadores AND, OR y NOT

La cláusula WHERE se puede combinar con operadores `AND`, `OR` y `NOT`.

Los operadores `AND` y `OR` se utilizan para filtrar registros basandose en más de una condición:

- El operador **Y-Lógico** `AND` muestra un registro si **todas** las condiciones que están separadas por `AND` son VERDADERAS.
- El operador **O-Lógico** `OR` muestra un registro si **alguna** de las condiciones que está separada por `OR` es VERDADERA.

El operador `NOT` muestra un registro si la(s) condición(es) **NO ES(SON) VERDADERA(S)**.

⚠ *Atención❗:* Existe una precedencia de operadores que establece como se van a aplicar los mismos, pero siempre **podremos agrupar las condiciones utilizando tantos (paréntesis) como sea necesario**.

AND Sintaxis

```
SELECT column1, column2, …
FROM table_name
WHERE condition1 AND condition2 AND condition3 …;
```

OR Sintaxis

```
SELECT column1, column2, …
FROM table_name
WHERE condition1 OR condition2 OR condition3 …;
```

NOT Sintaxis

```
SELECT column1, column2, …
FROM table_name
WHERE NOT condition;
```

Para estos ejemplos, vamos a trabajar sobre la tabla Customers completa.

AND Ejemplo

La siguiente instrucción SQL selecciona todos los campos de “Clientes” donde el país es *Alemania* `Germany` y no tienen numero de FAX:

```
SELECT * FROM Customers
WHERE Country='Germany' AND FAX IS NULL;
```

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
KOENE	Königlich Essen	Philip Cramer	Sales Associate	Maubelstr. 90	Brandenburg	NULL	14776	Germany	0555-09876	NULL
MORGK	Morgenstern Gesundkost	Alexander Feuer	Marketing Assistant	Heerstr. 22	Leipzig	NULL	04179	Germany	0342-023176	NULL
QUICK	QUICK-Stop	Horst Kloss	Accounting Manager	Taucherstraße 10	Cunewalde	NULL	01307	Germany	0372-035188	NULL

OR Ejemplo

La siguiente instrucción SQL selecciona todos los campos de “Clientes” donde la ciudad es “Berlin” o (`OR`) “Leipzig”:

```
SELECT * FROM Customers
WHERE City='Berlin' OR City='Leipzig';
```

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin	NULL	12209	Germany	030-0074321	030-0076545
MORGK	Morgenstern Gesundkost	Alexander Feuer	Marketing Assistant	Heerstr. 22	Leipzig	NULL	04179	Germany	0342-023176	NULL

📝Ejercicio SEL_LOG_1❓:

Escribe la instrucción SQL para seleccionar todos los campos de “*Clientes*” Customers donde país sea “*Italia*” o “*España*” con la ciudad “*Sevilla*”:

NOT Ejemplo

La siguiente instrucción SQL selecciona todos los campos de “Clientes” donde país NO es “Alemania” de la tabla CustomersDaw

```
SELECT * FROM CustomersDaw
WHERE NOT Country='Germany';
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Combinando AND, OR y NOT

También puede combinar los operadores `AND`, `OR` y `NOT`.

La siguiente instrucción SQL selecciona todos los campos de “Clientes” `Customers` donde el país es “Alemania” Y la ciudad debe ser “Berlín” O “Leipzig” (usamos paréntesis para formar expresiones complejas):

```
SELECT * FROM Customers
WHERE Country='Germany' AND (City='Berlin' OR City='Leipzig');
```

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin	NULL	12209	Germany	030-0074321	030-0076545
MORGK	Morgenstern Gesundkost	Alexander Feuer	Marketing Assistant	Heerstr. 22	Leipzig	NULL	04179	Germany	0342-023176	NULL

Ejercicio SEL_LOG_2 ?:

> Combina los operadores lógicos de otra forma para obtener los mismos resultados que en la consulta:

```
SELECT * FROM Customers
WHERE Country='Germany' AND (City='Berlin' OR City='Leipzig');
```

23. ORDER BY

La palabra clave ORDER BY

La palabra clave `ORDER BY` se usa para ordenar el conjunto de resultados en forma ascendente o orden descendiente.

La palabra clave ORDER BY ordena los registros en orden ascendente de forma predeterminada. A ordene los registros en orden descendente, use la palabra clave `DESC`.

ORDER BY Sintaxis

```
SELECT column1, column2, …
FROM table_name
ORDER BY column1, column2, … ASC|DESC;
```

ORDER BY Ejemplo

La siguiente instrucción SQL selecciona a todos los clientes de la tabla `CustomersDaw`, ordenados por la columna “País” `Country`:

```
SELECT * FROM CustomersDaw
ORDER BY Country;
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK

ORDER BY DESC Ejemplo

La siguiente instrucción SQL selecciona a todos los clientes de la tabla `CustomersDaw`, ordenados de forma **DESCENDENTE** por la columna “País” `Country`:

```
SELECT * FROM CustomersDaw
ORDER BY Country DESC;
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany

ORDER BY Ejemplo con varias columnas

La siguiente instrucción SQL selecciona a todos los clientes de la tabla `CustomersDaw` , ordenada por la columna `Country` y `CustomerName`. Esto significa que ordena por *País*, pero si algunas filas tienen el mismo *País*, los ordena por *CustomerName*:

```
SELECT * FROM CustomersDaw
ORDER BY Country, CustomerName;
```

Por defecto, las columnas siempre se ordenan ascendentemente `ASC`. Esta instrucción es análoga a la siguiente, dónde **indicamos explícitamente el tipo de ordenación de cada una de las columnas**.

```
SELECT * FROM CustomersDaw
ORDER BY Country ASC, CustomerName ASC;
```

El resultado de ambas consultas será:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK

Ejercicio SEL_ORD1 ❓:

> ¿Cómo realizarías un listado de todos los clientes de `CustomersDaw` ordenados de manera descendente por *País* y para los coincidentes, ordenados de manera también descendente por *Código Postal*?.

Ejercicio SEL_ORD2 ❓:

> Seleccionar las columnas `ContactName`, `Address`y `City` de todos los registros de la tabla `Customers`, ordenando el resultado alfabéticamente por la columna Ciudad `City`, donde la dirección `Address` empiece por `B` o `E`.

24. INSERT INTO

⚠ *Atención*❗: Si estás usando **SQL Server** para seguir estos apuntes, cuando ejecutamos la sentencia `SELECT INTO` para generar la tabla `CustomersDaw`, se genera una tabla sin **clave primaria**. He añadido la creación de esa clave primaria como unas instrucciones que se ejecutarán a continuación del `SELECT INTO`, pero si tu tabla actual no tiene clave primaria, antes de continuar vamos a **establecer una restricción de Clave Primaria** usando el siguiente código SQL:

```
ALTER TABLE CustomersDaw
ALTER COLUMN CustomerID INT NOT NULL;

ALTER TABLE CustomersDaw
ADD PRIMARY KEY (CustomerID);
```

*Nota[1]: es necesario indicar que la columna `CustomerID` no admite valores nulos `NOT NULL` para que **SQL Server** nos deje añadir la clave primaria posteriormente*

*Nota[2]: si estás usando **MySQL** o **MariaDB** en el script de creación de la base de datos `Northwind` si he añadido la creación de esta clave primaria, sobre la tabla `CustomersDaw`*

Instrucción INSERT INTO

La instrucción `INSERT INTO` se usa para insertar nuevos registros en una tabla.

INSERT INTO Sintaxis

Es posible escribir la instrucción INSERT INTO de dos maneras.

La primera forma especifica los nombres de columna y los valores que van a ser insertados:

```
INSERT INTO table_name (column1, column2, column3, …)
VALUES (value1, value2, value3, …);
```

Si agregamos valores para todas las columnas de la tabla, no necesitamos especificar los nombres de columna en la consulta SQL. Sin embargo, hay que asegurarse de que el orden de los valores está en el mismo orden que las columnas en la tabla.

La sintaxis de este tipo de INSERT INTO sería la siguiente:

```
INSERT INTO table_name
VALUES (value1, value2, value3, …);
```

INSERT INTO Ejemplo

La siguiente instrucción SQL inserta un nuevo registro de *Cliente* en la tabla `CustomersDaw`

```
INSERT INTO CustomersDaw (CustomerName, ContactName, Address, City,PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

⚠ *Atención*❗: **Observa que hemos omitido el campo `CustomerID` en la inserción.**, este campo podría haber sido AUTOINCREMENTAL o una SECUENCIA y asignarse de forma automática, como veremos más adelante, pero al no ser así:

Cuando ejecutemos esta sentencia en **SQL Server** nos va a producir el siguiente mensaje de ERROR: `Cannot insert the value NULL into column 'CustomerID', table 'Northwind.dbo.CustomersDaw'; column does not allow nulls. INSERT fails.` esto se debe a que si indicamos las columnas en la orden `INSERT INTO` no podemos olvidarnos de ninguna columna que deba ser **NO NULA** `NOT NULL`, y todas las claves primarias lo son.

MariaDB por su parte actua de forma diferente, asignando a la columna que va a ser clave primaria `CustomerID` un valor `0` por omisión y permitiendo la inserción de dicho cliente, siempre que no existiera ya otro cliente con `CustomerID = 0`, en cuyo caso nos aparecería el mensaje de error `ERROR 1062 (23000): Duplicate entry '0' for key 'PRIMARY'`

Por tanto, para realizar una inserción correcta, debemos indicar también el valor para el campo `CustomerID` :

```
INSERT INTO CustomersDaw (CustomerID, CustomerName, ContactName, Address, City,PostalCode, Country)
VALUES (6, 'Fruteria Paco', 'Francisco Martinez', 'Av.Juan Carlos I, 23', 'Lorca', '30800', 'Spain');
```

Si realizamos una consulta a nuestra tabla, veremos como se ha insertado el nuevo registro.

```sql
SELECT * FROM CustomersDaw;
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Fruteria Paco	Francisco Martinez	Av.Juan Carlos I, 23	Lorca	30800	Spain

INSERT solo de algunas columnas

También es posible insertar solo datos en columnas específicas.

La siguiente instrucción SQL insertará un nuevo registro, pero solo insertará datos en las columnas `CustomerID`,`CustomerName`, `City` y `Country`:

```sql
INSERT INTO CustomersDaw (CustomerID, CustomerName, City, Country)
VALUES (7,'Diego', 'Lorca', 'Spain');
```

Si mostramos ahora los clientes de la tabla `CustomersDaw` veremos que están los siguientes:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Fruteria Paco	Francisco Martinez	Av.Juan Carlos I, 23	Lorca	30800	Spain
7	Diego	NULL	NULL	Lorca	NULL	Spain

Ejercicio INS_1 ?:

Inserta un cliente en la tabla `CustomersDaw` cuyo `CustomerID` sea `8` y relleno con los datos que te inventes, no debes dejar ningún campo sin valor.

25. Valor NULL

¿Qué es un valor NULO?

Un campo con un valor `NULL` es un campo sin valor.

Si un *campo en una tabla es opcional*, es posible **insertar** un nuevo registro o **actualizar** ese registro *sin indicar ningún valor para ese campo*. En ese caso, el valor para ese campo se almacenará como `NULL`.

> ⚠ *Atención*: Un valor `NULL` es diferente de un valor `0` o de la cadena vacia `''`. Si un campo tiene un valor `NULL` será porque ha sido dejado en blanco durante la creación de ese registro.

Comprobaciones de valor NULL

No es posible comprobar valores NULL con los operadores de comparación como `=`, `<`, `>` o `<>`

Debemos de utilizar en su lugar los operadores `IS NULL` e `IS NOT NULL`.

Operador IS NULL

El operador `IS NULL` se usa para comprobar si un campo no contiene ningún valor.

```
SELECT  column_names
FROM table_name
WHERE column_name IS NULL;
```

IS NULL Ejemplo

La siguiente sentencia SQL muestra el `CompanyName` y el `ContactName` de todos los *clientes* `Customers` sin *código postal* `PostalCode`:

```
SELECT CompanyName, ContactName
FROM Customers
WHERE PostalCode IS NULL;
```

CompanyName	ContactName
Hungry Owl All-Night Grocers	Patricia McKenna

Operador IS NOT NULL

El operador `IS NOT NULL` se usa para buscar valores no vacios o no nulos.

```
SELECT _column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

IS NOT NULL Ejemplo

Vamos a contar cuantos *clientes* de la tabla `Customers` tienen *código postal* `PostalCode`.

```
SELECT COUNT(*)
FROM Customers
WHERE PostalCode IS NOT NULL;
```

Nos dará como resultado `90` clientes, de los **91** que hay en total.

Funciones IFNULL(), ISNULL(), COALESCE() y NVL()

Funciones para mostrar campos nulos

En ocasiones deseamos que a la hora de mostrar los resultados de una consulta, no nos aparezcan los valores `NULL`, sino un valor diferente que lo sustituya, por ejemplo la cadena vacia, el `0` o cualquier otro valor posible.

Para ayudarnos en esa tarea, cada sistema gestor de bases de datos ha implementado un función en la que indicamos el nombre del campo que deseamos mostrar y a continuación, separado por coma, cual es el valor que deseamos que se muestre si dicho campo es `NULL`.

Vamos a estudiar el siguiente ejemplo:

> Mostrar el *Coste Total de los Productos* cuyo precio por unidad `UnitPrice` sea mayor de **60 euros**. Para calcular ese Coste Total, multiplicaremos el precio unitario `UnitPrice` por las unidades en existencias `UnitsInStock`. Además supongamos que pudiera haber productos cuyas unidades en existencia fueran `NULL`. Más concretamente supongamos que el **ProductID=29** tiene `UnitsInStock=0`

La consulta:

```
SELECT ProductID, ProductName, UnitPrice, UnitsInStock, UnitPrice * UnitsInStock AS CosteTotalProducto
FROM Products
WHERE UnitPrice > 60;
```

Generaría el siguiente resultado:

ProductID	ProductName	UnitPrice	UnitsInStock	CosteTotalProducto
9	Mishi Kobe Niku	97,0000	29	2813,0000
18	Carnarvon Tigers	62,5000	42	2625,0000
20	Sir Rodney's Marmalade	81,0000	40	3240,0000
29	Thüringer Rostbratwurst	123,7900	NULL	NULL
38	Côte de Blaye	263,5000	17	4479,5000

SQL Server

La función **ISNULL()** de SQL Server nos devolverá el valor alternativo indicado como segundo parametro, cuando el primero sea nulo `NULL`.

```
SELECT ProductID, ProductName, UnitPrice, UnitsInStock, UnitPrice * ISNULL(UnitsInStock, 0) AS CosteTotalProducto
FROM Products
WHERE UnitPrice > 60;
```

ProductName	UnitPrice	UnitsInStock	CosteTotalProducto
Mishi Kobe Niku	97,0000	29	2813,0000
Carnarvon Tigers	62,5000	42	2625,0000
Sir Rodney's Marmalade	81,0000	40	3240,0000
Thüringer Rostbratwurst	123,7900	NULL	0,0000
Côte de Blaye	263,5000	17	4479,5000

También podemos utilizar la función **COALESCE()** que producirá el mismo resultado:

```
SELECT ProductName, UnitPrice, UnitsInStock, UnitPrice * COALESCE(UnitsInStock, 0) AS CosteTotalProducto
FROM Products
WHERE UnitPrice > 60;
```

MySQL / MariaDB

En estos SGBD, la función que realiza esta tarea se denomina `IFNULL()`

```
SELECT ProductName, UnitPrice, UnitsInStock, UnitPrice * IFNULL(UnitsInStock, 0) AS CosteTotalProducto
FROM Products
WHERE UnitPrice > 60;
```

O también podremos usar la función `COALESCE()`

```
SELECT ProductName, UnitPrice, UnitsInStock, UnitPrice * COALESCE(UnitsInStock, 0) AS CosteTotalProducto
FROM Products
WHERE UnitPrice > 60;
```

Oracle

La función `NVL()` en Oracle obtiene el mismo resultado:

```
SELECT ProductName, UnitPrice, UnitsInStock, UnitPrice * NVL(UnitsInStock, 0) AS CosteTotalProducto
FROM Products
WHERE UnitPrice > 60;
```

MS Access

La función de MS Access `IsNull()` sólo tiene un parámetro y devolverá TRUE (`-1`) si la expresión es un valor `NULL` o en otro caso, devolverá FALSE (`0`). Para conseguir el mismo resultado que en la consulta anterior, tendríamos que aplicar otra función adicional anidada, la función `IIF`.

> *Nota: Las **funciones** las estudiaremos con mayor profundidad el trimestre siguiente.*

```
SELECT ProductName, UnitPrice, UnitsInStock, UnitPrice * IIF(IsNull(UnitsInStock), 0, UnitsInStock)) AS CosteTotalProducto
FROM Products
WHERE UnitPrice > 60;
```
