INDICE SENTENCIAS SQL

Una de las sentencias SQL más importantes es SELECT, ya que permite realizar consultas sobre los datos almacenados en la base de datos.

Sintaxis SQL SELECT

```
SELECT * FROM nombretabla

SELECT columna1, columna2 FROM nombretabla
```

Para los ejemplos, tendremos la siguiente tabla de personas denominada “personas”

Estos son los datos almacenados en la tabla “personas”

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
ANTONIO	GARCIA	BENITO
LUIS	LOPEZ	PEREZ

Si queremos consultar todos los datos de la tabla “personas”

```
SELECT * FROM personas
```

Este será el resultado:

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
ANTONIO	GARCIA	BENITO
LUIS	LOPEZ	PEREZ

Si queremos consulta todos los nombres y primer apellido de todas las personas

```
SELECT nombre, apellido1 FROM personas
```

Este será el resultado:

nombre	apellido1
ANTONIO	PEREZ

ANTONIO	GARCIA
LUIS	LOPEZ

DISTINCT

Al realizar una consulta puede ocurrir que existan valores repetidos para algunas columnas. Por ejemplo

```
SELECT nombre FROM personas
```

nombre
ANTONIO
LUIS
ANTONIO

Esto no es un problema, pero a veces queremos que no se repitan, por ejemplo, si queremos saber los nombre diferentes que hay en la tabla personas", entonces utilizaremos **DISTINCT**.

```
SELECT DISTINCT nombre FROM personas
```

nombre
ANTONIO
LUIS

WHERE

La cláusula WHERE se utiliza para hacer filtros en las consultas, es decir, seleccionar solamente algunas filas de la tabla que cumplan una determinada condición.

El valor de la condición debe ir entre comillas simples ''.

Por ejemplo:

Seleccionar las personas cuyo nombre sea ANTONIO

```
SELECT * FROM personas
WHERE nombre = 'ANTONIO'
```

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
ANTONIO		

AND OR

Los **operadores AND y OR** se utilizan para filtrar resultados con 2 condiciones.

El operador **AND** mostrará los resultados cuando se cumplan las 2 condiciones.

```
Condición1 AND condición2
```

El operador **OR** mostrará los resultados cuando se cumpla alguna de las 2 condiciones.

```
Condicion1 OR condicion2
```

En la tabla personas

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
ANTONIO	GARCIA	BENITO
LUIS	LOPEZ	PEREZ

La siguiente sentencia (ejemplo AND) dará el siguiente resultado:

```
SELECT * FROM personas
WHERE nombre = 'ANTONIO'
AND apellido1 = 'GARCIA'
```

nombre	apellido1	apellido2
ANTONIO	GARCIA	BENITO

La siguiente sentencia (ejemplo OR) dará el siguiente resultado:

```
SELECT * FROM personas
WHERE nombre = 'ANTONIO'
OR apellido1 = 'GARCIA'
```

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
ANTONIO	GARCIA	BENITO

También se pueden combinar AND y OR, como el siguiente ejemplo:

```
SELECT * FROM personas
WHERE nombre = 'ANTONIO'
AND (apellido1 = 'GARCIA' OR apellido1 = 'LOPEZ)
```

nombre	apellido1	apellido2
ANTONIO	GARCIA	BENITO

ORDER BY

ORDER BY se utiliza para ordenar los resultados de una consulta, según el valor de la columna especificada.

Por defecto, se ordena de forma ascendente (ASC) según los valores de la columna.

Si se quiere ordenar por orden descendente se utiliza la palabra DES

```
SELECT nombre_columna(s)
FROM nombre_tabla
ORDER BY nombre_columna(s) ASC|DESC
```

Por ejemplo, en la tabla personas :

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ

LUIS	LOPEZ	PEREZ
ANTONIO	GARCIA	BENITO

```
SELECT nombre, apellido1
FROM personas
ORDER BY apellido1 ASC
```

Esta es la consulta resultante:

nombre	**apellido1**
LUIS	LOPEZ
ANTONIO	GARCIA
ANTONIO	PEREZ

Ejemplo de ordenación descendiente (DES)

```
SELECT nombre, apellido1
FROM personas
ORDER BY apellido1 DESC
```

Esta es la consulta resultante:

nombre	**apellido1**
ANTONIO	PEREZ
ANTONIO	GARCIA
LUIS	LOPEZ

INSERT

La sentencia INSERT INTO se utiliza para insertar nuevas filas en una tabla.

Es posible insertar una nueva fila en una tabla de dos formas distintas:

```
INSERT INTO nombre_tabla
```

```
VALUES (valor1, valor2, valor3, .)

INSERT INTO nombre_tabla (columna1, columna2, columna3,.)
VALUES (valor1, valor2, valor3, .)
```

Ejemplo:

Dada la siguiente tabla personas:

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
LUIS	LOPEZ	PEREZ
ANTONIO	GARCIA	BENITO

Si queremos insertar una nueva fila en la tabla personas, lo podemos hacer con cualquiera de las dos sentencias siguientes:

```
INSERT INTO personas
VALUES ('PEDRO', 'RUIZ', 'GONZALEZ')
INSERT INTO personas (nombre, apellido1, apellido2)
VALUES ('PEDRO', 'RUIZ', 'GONZALEZ')
```

Cualquiera de estas sentencias anteriores produce que se inserte una nueva fila en la tabla personas, quedando así dicha tabla:

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
LUIS	LOPEZ	PEREZ
ANTONIO	GARCIA	BENITO
PEDRO	RUIZ	GONZALEZ

UPDATE

La sentencia **UPDATE** se utiliza para modificar valores en una tabla.

La sintaxis de SQL UPDATE es:

```
UPDATE nombre_tabla
SET columna1 = valor1, columna2 = valor2
WHERE columna3 = valor3
```

La cláusula SET establece los nuevos valores para las columnas indicadas.

La cláusula WHERE sirve para seleccionar las filas que queremos modificar.

Ojo: Si omitimos la cláusula WHERE, por defecto, modificará los valores en todas las filas de la tabla.

Ejemplo del uso de SQL UPDATE

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
LUIS	LOPEZ	PEREZ
ANTONIO	GARCIA	BENITO
PEDRO	RUIZ	GONZALEZ

Si queremos cambiar el apellido2 'BENITO' por 'RODRIGUEZ' ejecutaremos:

```
UPDATE personas
SET apellido2 = 'RODRIGUEZ'
WHERE nombre = 'ANTONIO'
AND apellido1 = 'GARCIA'
AND apellido2 = 'BENITO'
```

Ahora la tabla 'personas' quedará así:

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
LUIS	LOPEZ	PEREZ

ANTONIO	GARCIA	**RODRIGUEZ**
PEDRO	RUIZ	GONZALEZ

DELETE

La sentencia **DELETE** sirve para borrar filas de una tabla.

La sintaxis de SQL DELETE es:

```
DELETE FROM nombre_tabla
WHERE nombre_columna = valor
```

Si queremos borrar todos los registros o filas de una tabla, se utiliza la sentencia:

```
DELETE * FROM nombre_tabla;
```

Ejemplo de SQL DELETE para borrar una fila de la tabla personas

nombre	**apellido1**	**apellido2**
ANTONIO	PEREZ	GOMEZ
LUIS	LOPEZ	PEREZ
ANTONIO	GARCIA	**RODRIGUEZ**
PEDRO	RUIZ	GONZALEZ

Si queremos borrar a la persona LUIS LOPEZ PEREZ, podemos ejecutar el comando:

```
DELETE FROM personas
WHERE nombre = 'LUIS'
AND apellido1 = 'LOPEZ'
AND apellido2 = 'PEREZ'
```

La tabla 'personas' resultante será:

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
ANTONIO	GARCIA	**RODRIGUEZ**

LIKE

El operador **LIKE** se utiliza en la cláusula WHERE para buscar por un patrón.

Sintaxis de SQL LIKE

```
SELECT columna(s) FROM tabla WHERE columna LIKE '%patron%'
```

Ejemplos del uso de SQL LIKE

Dada la siguiente tabla 'personas'

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
ANTONIO	GARCIA	RODRIGUEZ
PEDRO	RUIZ	GONZALEZ

Si quiero seleccionar los nombres que empiezan por 'AN' en la tabla 'personas', ejecutaría el comando siguiente:

```
SELECT * FROM personas
WHERE nombre LIKE 'AN%'
```

El character '%' es un comodín, que sirve para uno o más caracteres.

Este es el resultado

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
ANTONIO	GARCIA	RODRIGUEZ

Otro ejemplo de SQL LIKE

Para seleccionar las personas que tienen un 'Z' en su apellido1, ejecutaríamos:

```
SELECT * FROM personas
WHERE apellido1 LIKE '%Z%'
```

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
PEDRO	RUIZ	GONZALEZ

WILDCARDS

Los caracteres Wildcards (comodines) son caracteres especiales que se utilizan para realizar búsquedas especiales, como por ejemplo, buscar palabras que empiecen por una letra determinada (letra%) o que contengan la letra a (%a%), o que contengan alguna vocal ([aeiou]), etc.

Los caracteres Wildcards se utilizan con el operador SQL LIKE en una sentencia SELECT.Los caracteres Wildcards son :

%	sustituye a cero o más caracteres
_	sustituye a 1 carácter cualquiera
[lista]	sustituye a cualquier carácter de la lista
[^lista] o [!lista]	sustituye a cualquier carácter excepto los caracteres de la lista

Ejemplos:

Dada la siguiente tabla 'personas'

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
ANTONIO	GARCIA	RODRIGUEZ
PEDRO	RUIZ	GONZALEZ

Ejemplos Wildcards

Seleccionar las personas cuyo nombre contenga una 'R'

```
SELECT * FROM personas
WHERE nombre LIKE '%R%'
```

Resultado:

nombre	apellido1	apellido2
PEDRO	RUIZ	GONZALEZ

Seleccionar las personas cuyo apellido1 empiece por 'GA'

```
SELECT * FROM personas
WHERE apellido1 LIKE 'PE_EZ'
```

Resultado:

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ

Seleccionar las personas cuyo apellido1 empiece por P o G

```
SELECT * FROM personas
WHERE apellido1 LIKE '[PG]%'
```

Resultado:

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
ANTONIO	GARCIA	RODRIGUEZ

IN

El operador **IN** permite seleccionar múltiples valores en una cláusula WHERE

Sintaxis SQL IN

```
SELECT columna
FROM tabla
WHERE columna
IN (valor1, valor2, valor3, .)
```

Ejemplo SQL IN

Dada la siguiente tabla 'personas'

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
ANTONIO	GARCIA	RODRIGUEZ
PEDRO	RUIZ	GONZALEZ

Queremos seleccionar a las personas cuyo apellido1 sea 'PEREZ' o 'RUIZ'

```
SELECT * FROM personas
WHERE apellido1
IN ('PEREZ','RUIZ')
```

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
PEDRO	RUIZ	GONZALEZ

BETWEEN

El operador BETWEEN se utiliza en la cláusula WHERE para seleccionar valores entre un rango de datos.

Sintaxis de SQL BETWEEN

```
SELECT columna
FROM tabla WHERE columna
BETWEEN valor1 AND valor2
```

Ejemplo de SQL BETWEEN

Dada la siguiente tabla 'personas'

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
ANTONIO	GARCIA	RODRIGUEZ
PEDRO	RUIZ	GONZALEZ

Seleccionar personas cuyo apellido1 esté entre 'FERNANDEZ y 'HUERTAS'

```
SELECT *
FROM personas
WHERE apellido1
BETWEEN 'FERNANDEZ' AND 'HUERTAS'
```

nombre	apellido1	apellido2
ANTONIO	GARCIA	RODRIGUEZ

Seleccionar personas cuyo apellido1 no esté entre 'FERNANDEZ y 'HUERTAS'

```
SELECT *
FROM personas
WHERE apellido1
NOT BETWEEN 'FERNANDEZ' AND 'HUERTAS'
```

nombre	apellido1	apellido2
ANTONIO	PEREZ	GOMEZ
PEDRO	RUIZ	GONZALEZ

SELECT INTO

La sentencia SQL **SELECT INTO** se utiliza para seleccionar datos de una tabla y copiarlos en otra tabla diferente.

Se suele utilizar para hacer una copia de seguridad (backup) de los datos de una tabla.

Sintaxis SQL SELECT INTO

```
SELECT * INTO nuevatabla FROM tablaactual
```

Esta sentencia copiará todos los registros de la tabla 'tablaactual' en la tabla 'nuevatabla'.

La nueva tabla puede incluso estar en una base de datos diferente

```
SELECT *
INTO nuevatabla [IN nuevabasedatos]
FROM tablaactual
```

Si queremos hacer un backup de una tabla en otra

```
SELECT *
INTO personasBackup
FROM personas
```

También se pueden seleccionar sólo algunas columnas

```
SELECT columna1, columna2
INTO personasBackup
FROM personas
```

También se puede incluir una condición (WHERE)

```
SELECT *
INTO personasBackup
FROM personas
WHERE nombre = 'ANTONIO'
```

Se puede utilizar SELECT INTO con JOIN

```
SELECT personas.nombre, personas.apellido1,
departamentos.departamento
INTO personasInformatica
FROM personas INNER JOIN departamentos
ON personas.dep = 'INFORMATICA'
```

ALTER

La sentencia **SQL ALTER** se utiliza para añadir, eliminar o modificar columnas de una tabla.

Sintaxis SQL ALTER

Para añadir una nueva columna a una tabla

```
ALTER TABLE nombretabla
ADD nombrecolumna tipodatocolumna
```

Para borrar una columna de una tabla

```
ALTER TABLE nombretabla
DROP COLUMN (nombrecolumna)
```

Para modificar el tipo de dato de una columna de una tabla

```
ALTER TABLE nombretabla
ALTER COLUMN nombrecolumna tipodatocolumna
```

Ejemplos de SQL ALTER

per	nombre	apellido1	apellido2
1	ANTONIO	PEREZ	GOMEZ
2	ANTONIO	GARCIA	RODRIGUEZ

3	PEDRO	RUIZ	GONZALEZ

Dada la siguiente tabla de 'personas', queremos añadir una nueva columna, denominada 'fechadenacimiento'

```
ALTER TABLE personas
ADD  fechadenacimiento date
```

per	nombre	apellido1	apellido2	fechadenacimiento
1	ANTONIO	PEREZ	GOMEZ	
2	ANTONIO	GARCIA	RODRIGUEZ	
3	PEDRO	RUIZ	GONZALEZ	

Si queremos modificar el tipo de dato de la columna 'fecha', y ponerle tipo 'year' en lugar de tipo 'date'

```
ALTER TABLE personas
ALTER COLUMN fechadenacimiento year
```

Si queremos borrar la columna 'fechadenacimiento', y dejarlo igual que al principio

```
ALTER TABLE personas
DROP COLUMN fechadenacimiento
```

per	nombre	apellido1	apellido2
1	ANTONIO	PEREZ	GOMEZ
2	ANTONIO	GARCIA	RODRIGUEZ
3	PEDRO	RUIZ	GONZALEZ

La sintaxis general es:

ALTER TABLE [esquema.]tabla {ADD|MODIFY|DROP}...

Añadir una columna a una tabla:

ALTER TABLE T_PEDIDOS ADD TEXTOPEDIDO Varchar2(35);

Cambiar el tamaño de una columna en una tabla:

ALTER TABLE T_PEDIDOS MODIFY TEXTOPEDIDO Varchar2(135);

Hacer NOT NULL una columna en una tabla:

ALTER TABLE T_PEDIDOS MODIFY (TEXTOPEDIDO NOT NULL);

Eliminar una columna a una tabla:

ALTER TABLE T_PEDIDOS DROP COLUMN TEXTOPEDIDO;

Valor por defecto de una columna:

ALTER TABLE T_PEDIDOS MODIFY TEXTOPEDIDO Varchar2(135) DEFAULT 'ABC...';

Añade dos columnas:

ALTER TABLE T_PEDIDOS ADD (SO_PEDIDOS_ID INT, TEXTOPEDIDO Varchar2(135));

PARA CAMBIAR LAS RESTRICCIONES DEBEMOS USAR ALTER TABLE

Crear una clave primaria (primary key):

ALTER TABLE T_PEDIDOS ADD CONSTRAINT PK_PEDIDOS PRIMARY KEY (numpedido,lineapedido);

Crear una clave externa, para integridad referencial (foreign key):

ALTER TABLE T_PEDIDOS ADD CONSTRAINT FK_PEDIDOS_CLIENTES FOREIGN KEY (codcliente) REFERENCES T_CLIENTES (codcliente));

Crear un control de valores (check constraint):

ALTER TABLE T_PEDIDOS ADD CONSTRAINT CK_ESTADO CHECK (estado IN (1,2,3));

Crear una restricción UNIQUE:

ALTER TABLE T_PEDIDOS ADD CONSTRAINT UK_ESTADO UNIQUE (correosid);

Borrar una restricción:

ALTER TABLE T_PEDIDOS DROP CONSTRAINT CON1_PEDIDOS;

Deshabilita una restricción:

ALTER TABLE T_PEDIDOS DISABLE CONSTRAINT CON1_PEDIDOS;

habilita una restricción:

ALTER TABLE T_PEDIDOS ENABLE CONSTRAINT CON1_PEDIDOS;

CREATE INDEX

Objetos que forman parte del esquema que hacen que las bases de datos aceleren las operaciones de consultas y ordenación sobre los campos a los que el índice hace referencia.

Sintaxis:

CREATE INDEX nombre ON tabla (columna1, columna2, ..)

Eliminación de índices:

DROP INDEX nombre índice

CREATE SEQUENCE

Sirve para generar automáticamente números distintos. Se utilizan para generar valores para campos que se utilizan como clave forzada (claves cuyo valor no interesa , sólo sirven para identificar los registros de una tabla.

Sintaxis:

CREATE SEQUENCE campo

Otras opciones:

INCREMENT BY

START WITH

MAXVALUE