



SQL - Conceptos Básicos/Avanzados

Oscar Fernando Aristizábal Cardona
Ingeniero de Sistemas y Telecomunicaciones



SQL - Qué es?

- **¿Qué es SQL?**

- SQL significa Lenguaje de Consulta Estructurado.
- SQL le permite acceder y manipular bases de datos.
- SQL es un estándar ANSI (American National Standards Institute).

- **¿Qué puede hacer con SQL?**

- SQL se pueden ejecutar consultas en una base de datos.
- SQL puede recuperar datos de una base de datos.
- SQL puede insertar registros en una base de datos.
- SQL puede actualizar los registros de una base de datos.
- SQL puede eliminar registros de una base de datos.
- SQL puede crear nuevas bases de datos.
- SQL puede crear nuevas tablas en una base de datos.
- SQL puede crear procedimientos almacenados en una base de datos.
- SQL puede crear vistas de una base de datos.
- SQL puede establecer permisos en las tablas, los procedimientos y las vistas.



SQL - RDBMS

- **SQL en su Sitio Web:**

Para construir un sitio web que muestra algunos datos de una base de datos, se necesita lo siguiente:

- Un programa de base de datos RDBMS (MS Access, SQL Server, MySQL).
 - Un lenguaje de script del lado del servidor, como PHP o ASP.
 - SQL.
 - HTML / CSS.
-
- **RDBMS (Relational Database Management System)** Sistema de Gestión de bases de datos relacionales.
 - RDBMS es la base de SQL, y para todos los sistemas de bases de datos modernas, como MS SQL Server, IBM DB2, Oracle, MySQL, y Microsoft Access.
 - Los datos de RDBMS se almacena en objetos de base de datos llamada tablas.
 - Una tabla es una colección de datos relacionados con las entradas y consta de columnas y filas.



SQL - Sintaxis

- **Tablas en la Base de Datos:**

Una base de datos a menudo contiene una o más tablas. Cada tabla se identifica por un nombre (por ejemplo, "clientes" o "pedidos").

Las tablas contienen registros (filas) con los datos.

Ejemplo: Tabla denominada "personas":

Id	Nombre	Apellido	Telefono	Ciudad
1	Homero	Simpson	188662233	Springfield
2	Marge	Simpson	188662233	Springfield
3	Bart	Simpson	188662233	Springfield

- **Sentencias SQL:**

La mayoría de las acciones que debe realizar en una base de datos se hace con las sentencias SQL.

La siguiente instrucción SQL selecciona todos los registros de la tabla "personas" :

```
SELECT * FROM personas;
```



SQL - DML y DDL

- **DML y DDL**

SQL se puede dividir en dos partes:

El lenguaje de manipulación de datos (**LMD**) y el lenguaje de definición de datos (**DDL**).

Los comandos de consulta y actualización forman parte del **LMD** de SQL:

SELECT - Extrae datos de una base de datos

UPDATE - Actualiza los datos en una base de datos

DELETE - Elimina los datos de una base de datos

INSERT INTO - Inserta nuevos datos en una base de datos

Las instrucciones **DDL** más importantes en SQL son:

CREATE DATABASE - Crea una nueva base de datos.

ALTER DATABASE - Modifica una base de datos.

CREATE TABLE - Crea una nueva tabla.

ALTER TABLE - Modifica una tabla.

DROP TABLE - Elimina una tabla.

CREATE INDEX - Crea un índice (clave de búsqueda).

DROP INDEX - Elimina un índice.



SQL - SELECT

- La instrucción SELECT de SQL:

La instrucción **SELECT** se utiliza para seleccionar datos de una base de datos.

El resultado se almacena en una tabla de resultados, llamado el conjunto de resultados (**result set**).

Sintaxis SELECT de SQL:

```
SELECT nombre_columna(s)  
FROM nombre_tabla;
```

```
SELECT * FROM nombre_tabla;
```



SQL - SELECT DISTINCT

- **La instrucción SELECT DISTINCT de SQL:**

En una tabla, algunas de las columnas puede contener valores duplicados. Esto no es un problema, sin embargo, a veces se tendrá que listar sólo los valores diferentes (Distintos) de una tabla.

La palabra clave **DISTINCT** puede usarse para devolver sólo los valores distintos (diferentes).

Sintaxis SELECT DISTINCT de SQL:

```
SELECT DISTINCT nombre_columna(s)  
FROM nombre_tabla;
```

```
SELECT DISTINCT ciudad FROM personas;
```



SQL - WHERE

- **La cláusula WHERE:**

La cláusula WHERE se utiliza para extraer sólo aquellos registros que cumplen el criterio especificado.

Sintaxis WHERE de SQL:

```
SELECT nombre_columna(s)
FROM nombre_tabla
WHERE nombre_columna operador valor;
```

Con la cláusula **WHERE**, los siguientes operadores pueden ser utilizados:

Operador	Descripción
=	Igual
<>	Diferente, En algunas versiones de SQL se utiliza
>	Mayor que
<	Menor que
>=	Mayor o Igual que
<=	Menor o Igual que
BETWEEN	Entre un rango
LIKE	Buscar un patrón
IN	Especificar múltiples valores posibles para una columna



SQL - AND & OR

- **Los Operadores AND & OR:**

El operador **AND** muestra un registro si tanto la primera condición y la segunda condición es verdadera.

```
SELECT * FROM personas  
WHERE nombre = "Homero" AND ciudad = "Springfield";
```

El operador **OR** muestra un registro, si bien la primera condición o la segunda condición es verdadera.

```
SELECT * FROM personas  
WHERE nombre = "Homero" OR nombre = "Bart";
```

Combinando los 2 operadores:

```
SELECT * FROM personas  
WHERE apellido = "Simpson"  
AND (nombre = "Homero" OR nombre = "Bart");
```



SQL - ORDER BY

- La palabra clave **ORDER BY**:

La palabra clave **ORDER BY** se utiliza para ordenar el conjunto de resultados de una columna especificada.

La palabra clave **ORDER BY** ordena los registros en orden ascendente de forma predeterminada.

Si desea ordenar los registros en un orden descendente, puede utilizar la palabra clave **DESC**.

Sintaxis SQL ORDER BY:

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

```
SELECT * FROM peronas
ORDER BY nombre;
```



SQL - INSERT INTO

• **INSERT INTO**

La instrucción **INSERT INTO** se utiliza para insertar una nueva fila en una tabla.

Sintaxis SQL **INSERT INTO**:

Es posible escribir la instrucción **INSERT INTO** en dos formas.

La primera forma no especifica los nombres de columna, donde los datos se insertan, sólo sus valores.

```
INSERT INTO nombre_tabla  
VALUES(valor1, valor2, valor3, . . .);
```

La segunda forma especifica tanto los nombres de columna y los valores que se incluirá:

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



SQL - UPDATE

La instrucción **UPDATE** se utiliza para actualizar los registros existentes en una tabla.

Sintaxis SQL UPDATE:

```
UPDATE nombre_tabla  
SET columna1 = valor1, columna2 = valor2, . . .  
WHERE alguna_columna = algun_valor;
```

Advertencia UPDATE

Tenga cuidado al actualizar los registros. Nunca omita la cláusula **WHERE**.



SQL - DELETE

- **La instrucción DELETE:**

La sentencia **DELETE** se utiliza para eliminar filas de una tabla.

Sintaxis SQL DELETE:

```
DELETE FROM nombre_tabla  
WHERE alguna_columna = algun_valor;
```

Eliminar todas las filas

Es posible eliminar todas las filas de una tabla sin borrar la tabla. Esto significa que la estructura de la tabla, los atributos, y los índices quedaran intactos:

```
DELETE FROM nombre_tabla;  
  
DELETE * FROM nombre_tabla;
```



SQL - LIMIT

- **La cláusula LIMIT:**

La cláusula **LIMIT** se utiliza para especificar el número de registros a retornar.

La cláusula **LIMIT** puede ser muy útil en tablas grandes con miles de registros. La devolución de un gran número de registros puede tener un impacto en el rendimiento.

Nota: No todos los sistemas de bases de datos soportan la cláusula **LIMIT**.

Sintaxis SQL LIMIT:

```
SELECT nombre_columna(s)
FROM nombre_tabla
LIMIT numero;
```



SQL - LIKE

- **El operador LIKE:**

El operador **LIKE** se utiliza para buscar un patrón específico en una columna.

SQL LIKE Sintaxis:

```
SELECT nombre_columna(s)
FROM nombre_tabla
WHERE nombre_columna LIKE patron;
```

- Si queremos seleccionar los registros que empiezan con "**t**" utilizamos '**t%**'.
- Si queremos seleccionar los registros que terminan con "**t**" utilizamos '**%t**'.
- Si queremos seleccionar los registros que contiene el patrón "**nom**" utilizamos '**%nom%**'.
- También es posible seleccionar los registros que no contienen el patrón "**nom**" utilizamos **NOT LIKE '%nom%'**.



SQL - LIKE - WILDCARDS

- **Comodines en SQL:**

Los comodines SQL pueden sustituir a uno o más caracteres en la búsqueda de datos en una base de datos.

Los comodines en SQL deben ser utilizados con el operador **LIKE** de SQL.

Uso del comodín %

```
>> SELECT * FROM tabla WHERE campo LIKE 'sim%'  
>> SELECT * FROM tabla WHERE campo LIKE '%son'  
>> SELECT * FROM tabla WHERE campo LIKE '%son%
```

Uso del comodín _

```
>> SELECT * FROM tabla WHERE campo LIKE '_ome_o'  
>> SELECT * FROM tabla WHERE campo LIKE 'M_rg_'
```

Uso del comodin [charlist]

```
>> SELECT * FROM tabla WHERE campo LIKE '[abc]%'  
>> SELECT * FROM tabla WHERE campo LIKE '[!abc]%'
```



SQL - IN

- El operador **IN**:

El operador **IN** le permite especificar varios valores en una cláusula **WHERE**.

Sintaxis SQL IN:

```
SELECT nombre_columna(s)
FROM nombre_tabla
WHERE nombre_columna IN (valor1, valor2, . . .);
```



SQL - BETWEEN

- El operador Entre:

El operador **BETWEEN** selecciona un rango de datos entre dos valores. Los valores pueden ser números, texto o fechas.

Sintaxis SQL BETWEEN:

```
SELECT nombre_columna(s)
FROM nombre_tabla
WHERE nombre_columna
BETWEEN valor1 AND valor2;
```

Para mostrar resultados fuera de un rango se usa **NOT BETWEEN**



SQL - ALIAS

- **SQL Alias:**

Usted puede dar a una tabla o a una columna otro nombre utilizando un alias. Esto puede ser muy útil para hacer si tiene los nombres de tablas muy largas o complejas, o nombres de columna.

Un nombre de un alias puede ser cualquier cosa, pero por lo general debería ser corto.

Sintaxis SQL Alias para las tablas:

```
SELECT nombre_columna(s)
FROM nombre_tabla AS nombre_alias;
```

Sintaxis SQL Alias para las columnas:

```
SELECT nombre_columna AS nombre_alias
FROM nombre_tabla;
```



SQL - ALIAS

- Ejemplo SQL con Alias:

```
SELECT c.articulo, u.documento, u.nombres  
FROM usuarios AS u, compras AS c  
WHERE u.documento = c.documento_id;
```

- Ejemplo SQL sin Alias:

```
SELECT compras.articulo, usuarios.documento, usuarios.nombres  
FROM usuarios, compras  
WHERE usuarios.documento = compras.documento_id;
```



SQL - JOIN

- **SQL JOIN:**

La palabra clave **JOIN** se utiliza en una sentencia **SQL** para consultar datos de dos o más tablas, estas tablas están relacionadas entre si.

Una clave primaria es una columna (o una combinación de columnas) con un único valor para cada fila. Cada valor de clave primaria debe ser único dentro de la tabla.

El propósito es enlazar los datos en conjunto, a través de tablas.

Diferentes Tipos de JOIN:

- **JOIN:** Retorna filas cuando hay al menos una coincidencia en ambas tablas.
- **LEFT JOIN:** Retorna todas las filas de la tabla de la izquierda, incluso si no hay coincidencias en la tabla de la derecha.
- **RIGHT JOIN:** Retorna todas las filas de la tabla de la derecha, incluso si no hay coincidencias en la tabla de la izquierda.
- **FULL JOIN:** Retorna filas cuando hay una coincidencia en una de las tablas.



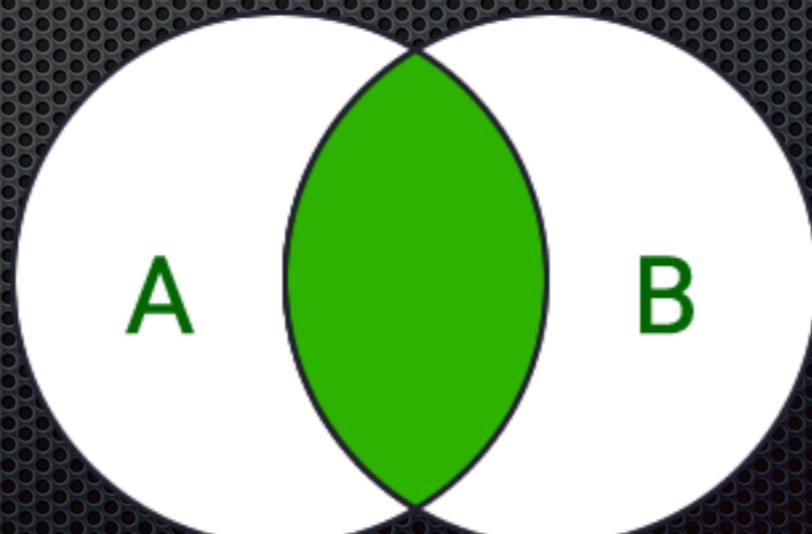
SQL - INNER JOIN

- **SQL INNER JOIN:**

La palabra clave **INNER JOIN** retorna filas cuando hay al menos una coincidencia en ambas tablas.

Sintaxis SQL INNER JOIN:

```
SELECT nombre_columna(s)
FROM nombre_tabla1
INNER JOIN nombre_tabla2
ON nombre_tabla1.nombre_columna = nombre_tabla2.nombre_columna;
```





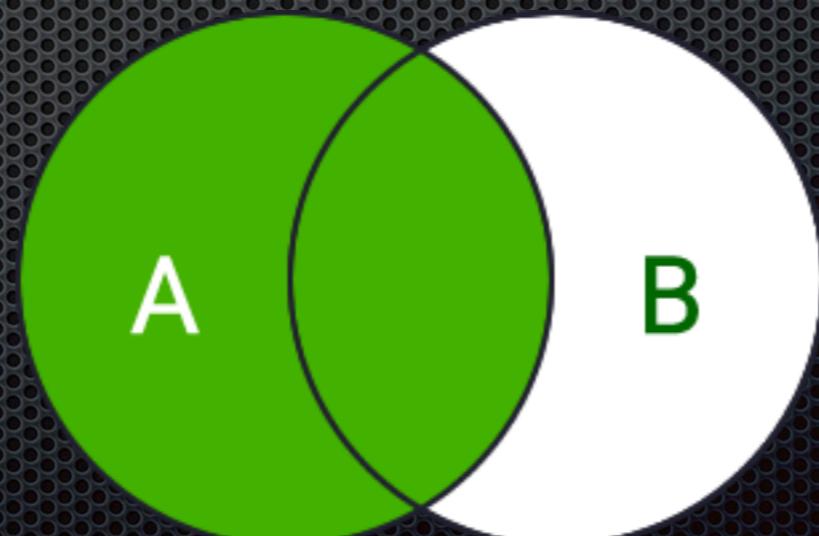
SQL - LEFT JOIN

- **SQL LEFT JOIN:**

La palabra clave **LEFT JOIN** devuelve todas las filas de la tabla de la izquierda (**nombre_tabla1**), incluso si no hay coincidencias en la tabla a la derecha (**nombre_tabla2**).

Sintaxis SQL LEFT JOIN:

```
SELECT nombre_columna(s)
FROM nombre_tabla1
LEFT JOIN nombre_tabla2
ON nombre_tabla1.nombre_columna = nombre_tabla2.nombre_columna;
```





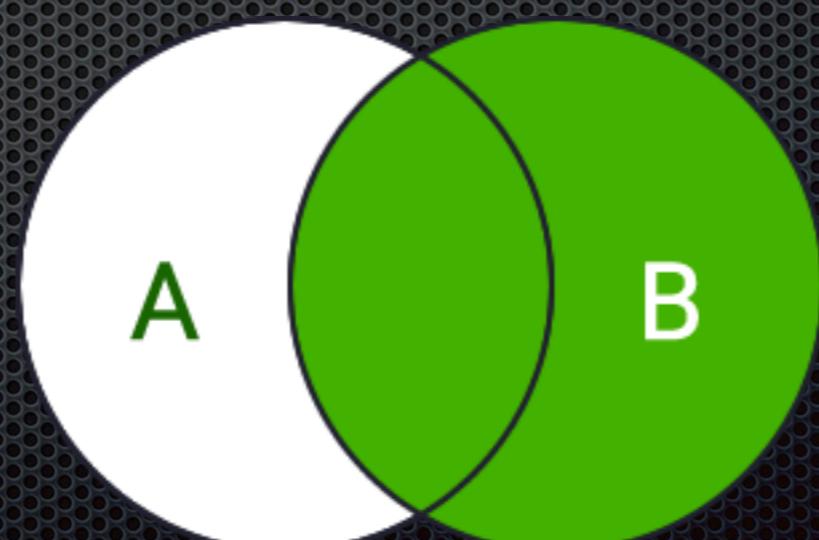
SQL - RIGHT JOIN

- **SQL RIGHT JOIN:**

La palabra clave **RIGHT JOIN** devuelve todas las filas de la tabla de la derecha (**nombre_tabla2**), incluso si no hay coincidencias en la tabla a la izquierda (**nombre_tabla1**).

Sintaxis SQL RIGHT JOIN:

```
SELECT nombre_columna(s)
FROM nombre_tabla1
RIGHT JOIN nombre_tabla2
ON nombre_tabla1.nombre_columna = nombre_tabla2.nombre_columna;
```





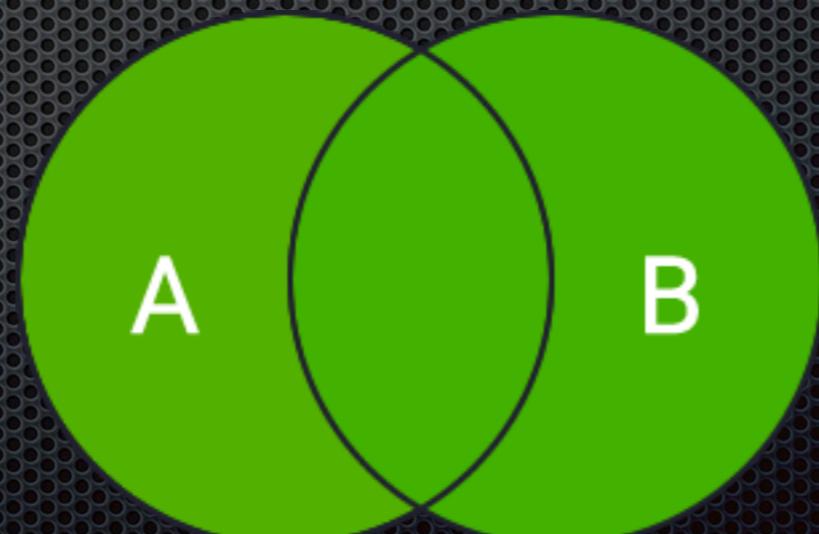
SQL - FULL JOIN

- **SQL FULL JOIN:**

La palabra clave **FULL JOIN** devuelve las filas cuando hay una coincidencia en una de las tablas.

Sintaxis SQL FULL JOIN:

```
SELECT nombre_columna(s)
FROM nombre_tabla1
FULL JOIN nombre_tabla2
ON nombre_tabla1.nombre_columna = nombre_tabla2.nombre_columna;
```





SQL - UNION

- **El operador SQL UNION:**

El operador **UNION** se usa para combinar el conjunto de resultados de dos o más instrucciones **SELECT**.

- Cada instrucción **SELECT** dentro de la Unión debe tener el mismo número de columnas.
- Las columnas también deben tener el mismo tipo de dato.
- Por otra parte, las columnas de cada instrucción **SELECT** deben estar en el mismo orden.

Sintaxis SQL UNION:

```
SELECT nombre_columna(s) FROM nombre_tabla1  
UNION  
SELECT nombre_columna(s) FROM nombre_tabla2;
```

Sintaxis SQL ALL UNION:

```
SELECT nombre_columna(s) FROM nombre_tabla1  
UNION ALL  
SELECT nombre_columna(s) FROM nombre_tabla2;
```



SQL - BACKUP

- **Copias de Seguridad de Tablas:**

La instrucción **INSERT INTO** selecciona datos de una tabla y lo inserta en una tabla diferente.

La instrucción **INSERT INTO** a menudo se utiliza para crear copias de seguridad de las tablas.

Sintaxis SQL INSERT INTO:

```
# Copiar datos a una tabla nueva.  
INSERT INTO tabla_nueva SELECT * FROM tabla_vieja  
  
# Copiar datos a una tabla nueva en una base de datos diferente.  
INSERT INTO nuevabasededatos.nuevatbla SELECT * FROM  
viejabasededatos.tabla_vieja;
```



SQL - CREATE

- **Crear Base de Datos:**

La instrucción **CREATE DATABASE** se utiliza para crear una base de datos.

Sintaxis CREATE DATABASE:

```
CREATE DATABASE nombre_basededatos;
```

- **Crear Tabla:**

La instrucción **CREATE TABLE** se utiliza para crear una tabla en una base de datos.

Sintaxis CREATE TABLE:

```
CREATE TABLE nombre_tabla (
    nombre_columna1 tipo_dato,
    nombre_columna2 tipo_dato,
    nombre_columna3 tipo_dato,
);
```



SQL - CREATE

- Crear Base de Datos: "adsinotas"
- Crear Tablas: "usuarios", "notas", "materias"





SQL - CONSTRAINTS

- **Limitaciones en SQL:**

Las restricciones se utilizan para limitar el tipo de datos que pueden ir en una tabla.

Las restricciones se pueden especificar cuando se crea una tabla (con la instrucción **CREATE TABLE**) o después de crear la tabla (con la instrucción **ALTER TABLE**).

Restricciones mas usadas:

- **NOT NULL**
- **UNIQUE**
- **PRIMARY KEY**
- **FOREIGN KEY**
- **CHECK**
- **DEFAULT**



SQL - CONSTRAINTS

- **Restricción SQL NOT NULL:**

- La restricción **NOT NULL** obliga a cumplir que una columna no puede tener valores nulos o vacíos.
 - La restricción **NOT NULL** se aplica a un campo para contener siempre un valor.
- Esto significa que no se puede insertar un nuevo registro, o actualizar un registro sin añadir un valor a este campo.

```
# Sintaxis NOT NULL:  
CREATE TABLE usuarios ( nombres varchar(64) NOT NULL );
```

- **Restricción SQL UNIQUE:**

- La restricción **UNIQUE** identifica de forma única cada registro de una tabla de base de datos.
 - Una restricción **PRIMARY KEY**, automáticamente tiene una restricción **UNIQUE** definidas en él.
- Tenga en cuenta que puede tener muchas restricciones **UNIQUE** por tabla, pero sólo una restricción **PRIMARY KEY** por tabla.

```
# Sintaxis UNIQUE:  
CREATE TABLE usuarios ( nusuario varchar(64) UNIQUE (nusuario) );
```



SQL - CONSTRAINTS

- **Restricción SQL PRIMARY KEY:**

- La restricción **PRIMARY KEY** identifica de forma única cada registro de una tabla de la base de datos.
- Las llaves primarias deben contener valores únicos.
- Una columna de llave principal no puede contener valores nulos.
- Cada tabla debe tener una llave primaria, y cada tabla sólo puede tener una llave primaria.

```
# Sintaxis PRIMARY KEY:
```

```
# -----
```

```
# Restricción SQL PRIMARY KEY al crear una tabla:
```

```
CREATE TABLE usuarios ( idu int, PRIMARY KEY(idu) );
```

```
# Restricción SQL PRIMARY KEY modificando una tabla:
```

```
ALTER TABLE usuarios ADD PRIMARY KEY(idu);
```

```
# Eliminar restricción SQL PRIMARY KEY modificando una tabla:
```

```
ALTER TABLE usuarios DROP PRIMARY KEY;
```



SQL - CONSTRAINTS

- **Restricción SQL FOREIGN KEY:**

- Una clave foranea en una tabla apunta a una llave primaria de otra tabla.
- La restricción **FOREIGN KEY** se utiliza para prevenir acciones que destruyen los vínculos entre las tablas.

```
# Sintaxis FOREIGN KEY:  
# -----  
  
# Restricción SQL FOREIGN KEY al crear una tabla:  
CREATE TABLE usuarios ( cid int, FOREIGN KEY(cid) REFERENCES categorias(idc) );  
  
# Restricción SQL FOREIGN KEY modificando una tabla:  
ALTER TABLE usuarios ADD FOREIGN KEY(cid) REFERENCES categorias(idc);  
  
ALTER TABLE usuarios ADD CONSTRAINT fk_cid FOREIGN KEY(cid) REFERENCES categorias(idc);  
  
# Eliminar restricción SQL FOREIGN KEY modificando una tabla:  
ALTER TABLE usuarios DROP FOREIGN KEY fk_cid;
```



SQL - CONSTRAINTS

- **Restricción SQL CHECK:**

- La restricción **CHECK** se utiliza para limitar el rango de valores que se pueden ingresar en una columna.

```
# Sintaxis CHECK:  
# -----  
  
# Restricción SQL CHECK al crear una tabla:  
CREATE TABLE usuarios ( idu int, CHECK (idu > 0) );  
  
# Restricción SQL CHECK modificando una tabla:  
ALTER TABLE usuarios ADD CHECK(idu > 0);  
  
ALTER TABLE usuarios ADD CONSTRAINT chk_idu CHECK(idu > 0 AND  
ciudad = 'Manizales');  
  
# Eliminar restricción SQL CHECK modificando una tabla:  
ALTER TABLE usuarios DROP CONSTRAINT chk_idu;
```



SQL - CONSTRAINTS

- **Restricción SQL DEFAULT:**

- La restricción **DEFAULT** se utiliza para insertar un valor predeterminado en una columna.
- El valor por defecto será añadido a todos los nuevos registros, si no se especifica otro valor.

```
# Sintaxis DEFAULT:  
# -----  
  
# Restricción SQL DEFAULT al crear una tabla:  
CREATE TABLE usuarios ( ciudad varchar(32) DEFAULT 'Manizales',  
fecha date DEFAULT GETDATE() );  
  
# Restricción SQL DEFAULT modificando una tabla:  
ALTER TABLE usuarios ALTER ciudad SET DEFAULT 'Manizales';  
  
# Eliminar restricción SQL DEFAULT modificando una tabla:  
ALTER TABLE usuarios ALTER ciudad DROP DEFAULT;
```



SQL - INDEX

- Índices:

- Un índice se pueden crear en una tabla para encontrar datos con mayor rapidez y de manera eficiente.
- Los usuarios no pueden ver los índices, que son utilizados para acelerar las búsquedas y las consultas.
- **Nota:** Actualizando una tabla con índices lleva más tiempo que la actualización de una tabla sin ellos (ya que los índices también necesitan una actualización).

```
# Sintaxis SQL CREATE INDEX:  
# -----  
  
# Crea un índice en una tabla, los valores duplicados están permitidos  
CREATE INDEX idx_nombre ON nombre_tabla(nombre_columna);  
  
# Crea un índice único en una tabla, los valores duplicados no están  
# permitidos  
CREATE UNIQUE INDEX idx_nombre ON nombre_tabla(nombre_columna);
```



SQL - DROP / TRUNCATE

- La declaraciones **DROP** y **TRUNCATE**:

```
# Sintaxis SQL DROP / TRUNCATE:  
# -----  
  
# Eliminar un Indice:  
DROP INDEX nombre_tabla.nombre_indice;  
  
# Eliminar una Tabla  
DROP TABLE nombre_tabla;  
  
# Eliminar una Base de Datos  
DROP DATABASE nombre_basededatos;  
  
# Vaciar toda la información de una Tabla  
# Deja intacta la estructura de la Tabla.  
TRUNCATE TABLE nombre_tabla;
```



SQL - ALTER TABLE

- La Instrucción SQL ALTER TABLE:

- La sentencia **ALTER TABLE** se utiliza para agregar, eliminar o modificar columnas de una tabla existente.

```
# Sintaxis ALTER TABLE:  
# -----  
  
# Añadir una columna a una tabla:  
ALTER TABLE nombre_tabla ADD nombre_columna tipodato;  
  
# Eliminar una columna en una tabla  
ALTER TABLE nombre_tabla DROP COLUMN nombre_columna;  
  
# Cambiar el tipo de datos de una columna en una tabla  
ALTER TABLE nombre_tabla ALTER COLUMN nombre_columna tipdatos;  
  
# Renombrar una Tabla:  
RENAME TABLE nombre_tabla TO nuevonombre_tabla;  
  
# Método con ALTER TABLE para renombrar una Tabla:  
ALTER TABLE nombre_tabla RENAME nuevonombre_tabla;
```



SQL - AUTO_INCREMENT

- **Incremento automático de un campo:**

- Muy a menudo nos gustaría que el valor del campo de una llave primaria se incremente automáticamente cada vez que un nuevo registro se inserta.

```
# Sintaxis AUTO_INCREMENT:  
# -----  
  
CREATE TABLE usuarios (idu int AUTO_INCREMENT);  
  
# Iniciar el auto incremento en otro valor que no sea 1:  
ALTER TABLE usuarios AUTO_INCREMENT = 120;
```



SQL - VIEWS

- **Crear Vistas:**

- En SQL, una vista es una tabla virtual basada en el conjunto de resultados de una consulta SQL.
- Una vista contiene filas y columnas, al igual que una tabla real. Los campos en un vista son campos de una o más tablas reales en la base de datos.

```
# Sintaxis SQL CREATE VIEW:  
# -----  
  
CREATE VIEW nombre_vista AS SELECT nombre_columna(s) FROM nombre_tabla WHERE  
condición;  
  
# Consultar una vista:  
SELECT * FROM nombre_vista;  
  
# Actualizar una vista:  
CREATE OR REPLACE VIEW nombre_vista AS SELECT nombre_columna(s) FROM  
nombre_tabla WHERE condición;  
  
# Eliminar una vista:  
DROP VIEW nombre_vista;
```