SQL

PARA PRINCIPIANTES

PRESTON PRESCOTT

SQL para Principiantes

Preston Prescott

Traducido por Andrés Parraud

“SQL para Principiantes”
Escrito por Preston Prescott
Copyright © 2015 Preston Prescott
Todos los derechos reservados
Distribuido por Babelcube, Inc.
www.babelcube.com
Traducido por Andrés Parraud
“Babelcube Books” y “Babelcube” son marcas registradas de Babelcube Inc.

Tabla de Contenidos

Página de Titulo
Página de Copyright
SQL para Principiantes
Solución | Creación de la Base de Datos Organizacion
Tabla Departamentos
Tabla Estudiantes

SQL para Principiantes

Preston Prescott

Todos los derechos reservados. Ninguna parte de esta publicación o de la información en ella contenida puede ser reproducida o citada en ninguna forma por medios tales como impresión, escaneo, fotocopia o cualquier otro medio sin previo permiso escrito del titular de los derechos de autor.
Aviso legal y términos de uso: Se ha hecho un esfuerzo para asegurar que la información en este libro sea exacta y completa; sin embargo, ni el autor ni el editor garantizan la exactitud de la información, texto y gráficos contenidos en el libro debido a la naturaleza cambiante de la ciencia, la investigación, hechos conocidos y desconocidos e Internet. Ni el autor ni el editor tienen responsabilidad alguna por errores, omisiones o interpretaciones erróneas sobre la materia objeto del presente. Este libro se presenta únicamente para propósitos motivacionales e informativos.

Contenido

Otras Publicaciones del Autor
Capítulo 1: Introducción a SQL y Bases de Datos
Capítulo 2: Tipos de Datos y Creación de Base de Datos/Tablas
Ejercicio 2
Capítulo 3: Drop, Alter, Truncate, & Insert
Ejercicio 3
Capítulo 4: Select, Where, & Order By
Ejercicio 4
Capítulo 5: Operadores SQL I
Ejercicio 5
Capítulo 6: Operadores SQL II
Ejercicio 6
Capítulo 7: Funciones de Agregación, Borrar y Actualizar
Ejercicio 7
Capítulo 8: Relaciones y Consultas Combinadas
Ejercicio 8
Capítulo 9: Sub-consultas de SQL
Ejercicio 9
Capítulo 10: Funciones de Carácter de SQL
Ejercicio 10
Otras Publicaciones del Autor

Otras Publicaciones del Autor

Programación JavaScript: Guía para Principiantes al Lenguaje de Programación JavaScript

http://www.linuxtrainingacademy.com/js-programming

Si en el pasado has intentado aprender programación pero no has tenido demasiado éxito, entonces prueba *Programación JavaScript*. Te enseñará exactamente lo que necesitas saber sobre programación en el lenguaje de guión más difundido del mundo actualmente. Te hará comenzar por el principio y te permitirá construir sobre lo que ya has aprendido en el camino.

Capítulo 1: Introducción a SQL y Bases de Datos

SQL viene de Lenguaje de Consulta Estructurado (por sus siglas en inglés). Oficialmente, se pronuncia “Ese Cu Ele”. SQL, como lo sugiere su nombre, es un lenguaje de consulta usado para interactuar con bases de datos relacionales. SQL se utiliza para ejecutar funciones como insertar datos en una base de datos, recuperarlos, actualizarlos, borrarlos y otras acciones similares. El primer capítulo presenta una visión “a vuelo de pájaro” de SQL, junto con una breve introducción a bases de datos.

Contenido

- **¿Qué son Bases de datos?**
- **Ventajas de las bases de datos relacionales**
- **Tipos de SQL**

I. **Lenguaje de Definición de Datos**
II. **Lenguaje de Manipulación de Datos**

1- ¿Qué son las Bases de Datos?

Un software de aplicación necesita almacenar datos. Hay dos maneras de almacenar los datos usados por cualquier aplicación. La primera opción es almacenarlos en la memoria de la aplicación, donde los datos residirán mientras la aplicación esté corriendo. La otra opción es almacenar los datos en tu disco rígido, y luego, por medio de ciertos procedimientos, llamar los datos a la memoria de la computadora y llevar a cabo una transacción en los datos. La memoria de acceso aleatorio (RAM) de una computadora es limitada; Si el volumen datos que deben procesarse es muy grande, no pueden almacenarse todos simultáneamente en la memoria. En este escenario, se requiere un almacenamiento secundario.

En un disco rígido, los datos pueden ser almacenados en diferentes formatos de archivo. Pueden ser almacenados en forma de archivos de texto, Word, mp4, etc. Sin embargo, se necesita una interfaz uniforme que pueda proveer acceso a diferentes tipos de datos en forma robusta y eficaz bajo un único paraguas. Es aquí donde surge el rol de las bases de datos.

Definición: “Una colección de información almacenada en una computadora de forma que pueda ser accedida, administrada y manipulada con facilidad”.

Las bases de datos almacenan datos en forma de una colección de tablas donde cada tabla almacena datos de una entidad particular. Por ejemplo, si estás desarrollando una aplicación que almacena registros de estudiantes de una universidad, habrá una tabla que contenga registros de estudiantes y una tabla que contenga registros de asignaturas. La información que queremos almacenar sobre los estudiantes será representada en las columnas de la tabla. Cada fila de la tabla contendrá un registro de un estudiante en particular. Cada registro se identificará por cierta columna, la que contendrá un valor único para cada fila.

Supongamos que quieres almacenar el ID, nombre, edad, sexo y departamento de un estudiante. La tabla en la base de datos que contendrá los datos de este estudiante se verá como la que sigue:

SID	SNombre	SEdad	SSexo	SDepartamento
1	Tom	14	Varón	Computación
2	Mike	12	Varón	Electricidad

3	Sandy	13	Mujer	Electricidad
4	Jack	10	Varón	Computación
5	Sara	11	Mujer	Computación

Tabla Estudiantes

Aquí, la letra “S” se ha colocado como prefijo del nombre de cada columna. Esto es sólo una de las convenciones usadas para denotar los nombres de columna. Puedes darle cualquier nombre a las columnas. (Veremos cómo crear tablas y columnas en los próximos capítulos). Es mucho más fácil acceder, manipular y administrar los datos almacenados de este formato. Las consultas SQL pueden ejecutarse sobre los datos almacenados en forma de tablas que tienen relación con otras tablas.

Una base de datos no contiene una tabla única. Por el contrario, contiene múltiples tablas con relaciones entre ellas. Las relaciones mantienen la integridad de la base de datos e impiden la redundancia de los datos. Por ejemplo, si la escuela decide cambiar el nombre del departamento de Computación de “Computación” a “Comp & Soft”, tendrás que actualizar los registros de todos los estudiantes del departamento de Computación. Tendrás que actualizar los registros 1, 4 y 5 de la tabla de estudiantes.

Es fácil actualizar tres registros. Sin embargo, en escenarios de la vida real, hay miles de estudiantes y es una tarea ímproba actualizar los registros de todos esos estudiantes. En escenarios así, las relaciones entre los datos de las tablas se vuelven importantes. Por ejemplo, para resolver el problema de redundancia antes mencionado, podemos crear otra tabla llamada Departamentos y almacenar los registros de todos los departamentos en ella. Esa tabla se verá como esta:

DID	**DNombre**	**DCapacidad**
101	Electricidad	800
102	Computación	500
103	Mecánica	500

Tabla Departamentos

Ahora, en la tabla Estudiantes, en lugar de almacenar el nombre del departamento, se almacenará su id. La tabla Estudiantes será actualizada como sigue:

SID	**SNombre**	**SEdad**	**SSexo**	**DID**
1	Tom	14	Varón	102
2	Mike	12	Varón	101
3	Sandy	13	Mujer	101
4	Jack	10	Varón	102
5	Sara	11	Mujer	102

Tabla Estudiantes

Puede verse que el nombre de la columna del departamento ha sido reemplazada por la columna id de departamento, representada por “DID”. Las filas 1, 4 y 5, que antes tenían asignado el departamento “Computación”, ahora contienen el id del departamento, que es 102. Ahora, si el nombre del departamento es cambiado de “Computación a “Comp & Soft”, este cambio debe hacerse únicamente en un registro de la tabla Departamentos, y todos los estudiantes asociados a él, automáticamente se referirán al nombre actualizado del departamento.

2- Ventajas de las Bases de Datos

Las siguientes son algunas de las principales ventajas de las bases de datos:

- Las bases de datos mantienen la integridad de los datos. Esto significa que los datos se cambian en un único lugar, y que todas las entidades que acceden a ellos obtienen la última versión de los datos.
- Por medio de consultas complejas, las bases de datos pueden ser accedidas, modificadas y manipuladas eficientemente. SQL está diseñado para este propósito.
- Las bases de datos evitan la redundancia de datos. Por medio de tablas y relaciones, las bases de datos evitan la redundancia y los datos pertenecientes a una entidad particular se ubican en un único lugar de la base de datos.
- Las bases de datos ofrecen mejor seguridad y mayor control. Por ejemplo, los nombres de usuario y claves de acceso pueden ser almacenados en tablas con niveles de seguridad extremos.

3- Tipos de Consultas SQL

Basado en su funcionalidad, las consultas SQL pueden ser clasificadas a grandes rasgos en las siguientes categorías principales:

- **Lenguaje de Definición de Datos (DDL)**

Las consultas de Lenguaje de Definición de Datos se utilizan para crear y definir esquemas de base de datos. Las siguientes son algunas de las consultas que pertenecen a esta categoría:

I. *CREATE* – crear tablas y otros objetos en la base de datos
II. *ALTER* – alterar la estructura de bases de datos, principalmente tablas.
III. *DROP* – eliminar objetos, principalmente tablas, de la base de datos
IV. *TRUNCATE* – eliminar todos los registros de una tabla, incluyendo todos los espacios asignados para los registros
V. *COMMENT* – agregar comentarios al diccionario de datos
VI. *RENAME* – cambiar nombres de tablas u otros objetos

- **Lenguaje de Manipulación de Datos**

Las consultas de Lenguaje de Manipulación de Datos (DML) se utilizan para manipular datos dentro de la base de datos. Los siguientes son algunos ejemplos de consultas DML.

I. *SELECT* – Seleccionar datos de tablas de una base de datos
II. *INSERT* – insertar datos en una tabla de la base de datos
III. *UPDATE* – actualizar los datos existentes en toda una tabla
IV. *DELETE* – eliminar todas las filas de una tabla, el espacio para los registros permanece

Capítulo 2: Tipos de Datos y Creación de Base de datos/Tablas

Este capítulo presenta una breve introducción a diferentes tipos de datos. “Tipos de datos” se refiere a formatos en los que los datos son almacenados en una base de datos. Por ejemplo, si quieres almacenar el nombre de una persona, se almacenará en un formato “*varchar*” en la base de datos. En forma similar, el id o la edad de una persona se almacena en formato “*integer*”. Los tipos de datos varían dependiendo del tipo de servidor de base de datos. Sin embargo, los tipos de datos básicos son uniformes en diferentes servidores. Más adelante en este capítulo, te mostraré cómo puedes crear bases de datos y tablas por medio de consultas SQL.

Contenido

- **Tipos de datos SQL**
- **Creación de una Base de datos**
- **Creación de Tablas en una Base de datos**

1- Tipos de datos SQL

- **Tipos de Datos Numéricos Exactos:**

Los siguientes son tipos de datos SQL que almacenan datos numéricos exactos:

TIPO DE DATO	DESDE	HASTA
bigint	-9.223.372.036.854.775.808	9.223.372.036.854.775.807
int	-2.147.483.648	2.147.483.647
smallint	-32.768	32.767
tinyint	0	255
bit	0	1
decimal	-10^38 +1	10^38 -1
numeric	-10^38 +1	10^38 -1
money	-922.337.203.685.477.5808	+922.337.203.685.477.5807
smallmoney	-214.748.3648	+214.748.3647

- **Tipos de Datos Numéricos Aproximados:**

Los siguientes son tipos de datos SQL que almacenan datos numéricos aproximados:

TIPO DE DATO	DESDE	HASTA
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38

- **Tipos de datos SQL de Fecha y Hora**

Estos son algunos de los tipos de datos SQL de fecha y hora:

TIPO DE DATO	DESDE	HASTA

Datetime	Jan 1, 1753	Dec 31, 9999
Smalldatetime	Jan 1, 1900	Jun 6, 2079
date	Almacena el dato como October 10, 1991	
time	Almacena la hora del día como 11:30 P.M.	

- **Tipos de Datos de Cadenas**

Los siguientes son tipos de datos SQL de Cadenas:

TIPO DE DATO	DESDE	HASTA
char	char	Longitud máxima de 8.000 caracteres. (Caracteres de longitud fija no Unicode.)
varchar	varchar	Máximo de 8.000 caracteres. (Datos de longitud variable no Unicode.)
varchar(max)	varchar(max)	Longitud máxima de 231 caracteres, datos de Longitud variable no Unicode (Sólo para Servidor SQL 2005).
text	text	Datos no Unicode de Longitud variable con una longitud máxima de 2.147.483.647 caracteres.

2- Creación de una Base de Datos

A pesar de que la mayoría de los servidores de base de datos vienen con herramientas que permiten a los usuarios crear bases de datos por medio de la interfaz GUI, la consulta del *back-end* que realmente crea la base de datos en el servidor es la misma. Para crear una nueva base de datos, simplemente abre la ventana de consulta de la base de datos que estás usando. Para demostración, yo estoy usando *MS SQL Server Express 2014*.

Nota: Todas las consultas ejecutadas en este libro siguen los estándares *Transact SQL (T-SQL*). Noten también que las declaraciones SQL pueden escribirse en mayúsculas o minúsculas (no son *case sensitive*). Sin embargo, ejecutaré algunas consultas en mayúsculas y algunas en minúsculas sólo para demostrar que ello no afecta a la consulta.

Ingresa la siguiente consulta en tu ventana de consulta:

Consulta 1:

```
CREATE DATABASE Hospital;
```

Esta consulta creará una base de datos llamada “Hospital” en la instancia del servidor de base de datos que estás usando. Ahora, si tu quieres ver si tu base de datos ha sido creada, puedes escribir el siguiente comando en tu ventana de consulta.

Consulta 2:

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

Cuando se ejecuta esta Consulta 2, la ventana de salida mostrará una lista de todas las bases de datos existentes en tu servidor de base de datos. Esta es la salida de la consulta:

name
master
tempdb
model
msdb
MyDB
SchoolDB
Hospital

Puedes ver “Hospital” en la última fila. Tu salida puede ser diferente, dependiendo de la cantidad de bases de datos que ya hayas creado. Echemos un vistazo a lo que está sucediendo en la segunda consulta.

Aquí estoy usando una sentencia “*SELECT*” para seleccionar el nombre de las bases de datos “*FROM*” (desde) la base de datos residente en este sistema. En el servidor SQL, la instancia de la base de datos misma es considerada una base de datos de nivel superior que contiene varias bases de datos definidas por el usuario. Aquí “*sys.databases*” se refiere a la lista de todas las bases de datos residentes en este sistema. En secciones posteriores, veremos los usos de las sentencias “*SELECT*” y “*FROM*” en detalle.

Ahora tratemos de obtener el id y la fecha de creación de todas las bases de datos.

Consulta 3

```
SELECT name, database_id, create_date
FROM sys.databases
```

Aquí puedes ver que, además del nombre, también seleccionamos *database_id* y *create_date*. En la salida podrás ver todas estas propiedades. La salida es la siguiente. Puedes ver la fecha de creación de tu base de datos a continuación de su id, el cual es asignado internamente a todas las bases de datos creadas en un servidor de DB (“DB” es una abreviatura para “Base de Datos”).

name	database_id	create_date
master	1	2003-04-08 09:13:36.390
tempdb	2	2015-02-01 09:08:30.663
model	3	2003-04-08 09:13:36.390
msdb	4	2014-02-20 20:49:38.857
MyDB	5	2015-01-27 01:20:59.790
SchoolDB	6	2015-01-29 00:27:27.153
Hospital	7	2015-02-01 12:39:42.703

3- Creación de Tablas en una Base de Datos

Una vez que has configurado tu base de datos, el siguiente paso es crear tablas en ella porque, a fin de cuentas, debes almacenar los datos en tablas. Para crear tablas en cualquier servidor de base de datos, tienes que ejecutar un par de pasos. Primero, necesitarás posicionarte en la base de datos en la cual quieres crear tu tabla. En la última sección vimos que un servidor puede alojar múltiples bases de datos; sin embargo, es importante especificar la base de datos en la que deseas crear tu tabla. La siguiente consulta cambia el editor de consultas para que ejecute consultas en la base de datos “Hospital”, que creamos en la sección anterior.

Consulta 4

```
USE Hospital
GO
```

Ahora todas las consultas ejecutadas en la ventana de consultas operarán en la base de datos “Hospital”. Supongamos que queremos crear una tabla llamada “Pacientes” en la base de datos “Hospital”. Pacientes tiene los atributos id, nombre, edad y descripción de enfermedad. La siguiente consulta crea una tabla que almacena esos datos.

Consulta 5

```
CREATE TABLE Pacientes
(PatientID int PRIMARY KEY NOT NULL,
PatientName varchar(50) NOT NULL,
PatientAge int NULL,
DiseaseDescription text NULL)
```

La consulta puede parecer intimidante al principio; sin embargo, es bastante simple. “*CREATE TABLE*” es la sentencia usada para crear cualquier tabla en la base de datos, seguida por el nombre de la tabla. Dentro de los paréntesis especificas las columnas de la tabla y sus características. La sentencia “*PatientID int PRIMARY KEY NOT NULL*” establece que una columna llamada “*PatientID*” será creada, y que esa columna debe almacenar datos de tipo entero (“*int*”). La bandera “*PRIMARY KEY*” se refiere al hecho de que esta columna contiene una clave primaria, y la bandera “N*OT NULL*” se refiere a que esta columna no puede estar vacía. De la misma forma, las columnas “*PatientName*”, “*PatientAge*” y “*DiseaseDescription*” han sido agregadas. Aquí, el concepto de clave primaria debe ser explicado.

- *Primary Key* **(Clave Primaria)**

La clave primaria es básicamente una columna que distingue específicamente un registro o fila en una base de datos. No puede haber dos registros en una tabla que tengan la misma clave primaria. Asimismo, la clave primaria jamás puede estar vacía.

En la Consulta 5 especificamos que la clave primaria será *PatientID*, lo que significa que dos pacientes no pueden tener la misma clave primaria.

Ahora, para ver si la tabla “Pacientes” ha sido creada en la base de datos “Hospital”, sólo debes ejecutar la siguiente consulta:

Consulta 6

```
SELECT name
FROM sys.tables
```

Debes establecer “Hospital” como la base de datos actual ejecutando la consulta de arriba. Puedes ver la Consulta 4 para cambiar de base de datos.

Ejercicio 2

Tarea:

Crear una base de datos llamada “Organizacion” y, dentro de ella, crear una tabla llamada Empleados. Empleados debe tener un Id (que debe ser la clave primaria), Nombre, Edad y Salario. Salario y Edad pueden contener valores nulos.

Solución

Creación de la Base de Datos Organizacion

CREATE DATABASE Organizacion;

Cambiar a la DB Organizacion

USE Organizacion

GO

Crear la Tabla Empleados

CREATE TABLE Empleados

(EmployeeID int PRIMARY KEY NOT NULL,

EmployeeName varchar(50) NOT NULL,

EmployeeAge int NULL,

EmployeeSalary money NULL)

Capítulo 3: *Drop, Alter, Truncate,* e *Insert*

En el capítulo 2 estudiamos cómo crear una base de datos y cómo crear tablas dentro de esa base de datos. También vimos algunos de los tipos de datos básicos en SQL. En este capítulo vamos a estudiar las sentencias *Drop, Truncate, Alter* e *Insert*. Una sentencia *drop* se utiliza para borrar una Tabla o Base de datos. *Alter* se usa principalmente para modificar la estructura de una tabla. *Truncate* se usa para borrar todos los datos de una tabla. También debemos estudiar la consulta *insert*, que es utilizada para insertar datos en la tabla.

Contenido

- **Sentencia *Drop***
- **Sentencia *Truncate***
- **Sentencia *Alter***
- **Sentencia *Insert***

1- Sentencia *Drop (Descartar)*

La sentencia *drop* se usa para descartar o borrar una base de datos o tabla y todos los datos, restricciones, índices y gatillos asociados a ella. Antes de ejecutar una consulta *Drop Table*, crea primero otra tabla en la base de datos "Hospital", que creamos en el capítulo anterior. Nombra la nueva tabla como "Departamento". Debe tener columnas para id, nombre y capacidad. La siguiente consulta creará esa tabla:

```
CREATE TABLE Departamento
(DepartmentID int PRIMARY KEY NOT NULL,
DepartmentName varchar(50) NOT NULL,
DepartmentCapacity int NULL)
```

Ahora, si miras las tablas de la DB "Hospital", debe contener dos tablas llamadas "Paciente" y "Departamento". Supón que queremos borrar la tabla "Pacientes" de la base de datos; podemos usar la sentencia *Drop Table*. La siguiente consulta *Drop Table* borra la tabla "Pacientes" de la DB Hospital.

Consulta 1:

```
DROP TABLE Pacientes
```

Recuerda que las consultas SQL son insensibles a las mayúsculas y minúsculas; para borrar la tabla, puedes usar tanto *"drop table"* como *"DROP TABLE"* seguida por el nombre de la tabla. Ahora, si listas todas las tablas de la DB "Hospital", no versa la tabla "Pacientes". Para borrar completamente la base de datos "Hospital", simplemente usas la siguiente sentencia.

Consulta 2:

```
Drop Database Hospital
```

2- Sentencia *Truncate (Cortar)*

La sentencia truncate borra todos los datos dentro de la tabla sin borrar la tabla misma. De hecho, la sentencia *truncate* borra la tabla, incluyendo los datos dentro de ella, y luego vuelve a crear una tabla vacía. Para hacer un *truncate* de la tabla "Pacientes", puedes usar la siguiente sentencia:

Consulta 3:

```
Drop Database Pacientes
```

3- Sentencia *Alter (Modificar)*

La sentencia *alter* se usa para modificar la estructura de una tabla en la base de datos. Usando una sentencia *alter* puedes agregar, quitar o actualizar una columna existente en la tabla.

- **Agregar una Columna en una Tabla**

Supongamos que quieres agregar la fecha de nacimiento del paciente en la tabla "Pacientes" que creamos en el capítulo anterior. Puedes usar una sentencia *alter* como la siguiente:

Consulta 4:

```
ALTER TABLE Pacientes
ADD PatientDOB date
```

La sintaxis de la sentencia *alter* es simple. Tienes que escribir *"ALTER TABLE"* seguido por el nombre de la tabla. A continuación tienes que escribir *ADD* seguido del nombre de la columna que quieres agregar y el tipo de dato que debe almacenar. En la Consulta 4 agregamos la columna *PatientDOB* de tipo *date* (fecha).

Puedes ver todas las columnas dentro de la tabla "Pacientes" usando la siguiente consulta:

Consulta 5

```
SELECT COLUMN_NAME
FROM Hospital.INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = N'Pacientes';
```

Verás la siguiente salida:

COLUMN_NAME
PatientID
PatientName
PatientAge
DiseaseDescription
PatientDOB

Puedes ver la nueva columna agregada *PatientDOB* en la última fila.

- **Borrar una Columna de una Tabla**

La sentencia *alter* puede usarse para borrar de una table una columna en particular. Por ejemplo, si quieres borrar la columna "*DiseaseDescription*" de la tabla "Pacientes", puedes usar la siguiente consulta:

Consulta 6:

```
ALTER TABLE Pacientes
DROP COLUMN DiseaseDescription
```

La primera parte de esta consulta es similar a la usada para agregar una columna. Sin embargo, en la segunda parte usamos *"DROP COLUMN"* seguido del nombre de la columna que queremos borrar.

- **Modificando una columna en una Tabla**

Finalmente, una sentencia *alter* también puede usarse para modificar una columna específica de una tabla. Especificamos que el tipo de dato de *PatientName* es *varchar(50)*. Si queremos cambiarlo a *varchar(100)* y permitir que contenga valores nulos, podemos usar una sentencia *alter*. La siguiente consulta lo demuestra:

Consulta 7:

```
ALTER TABLE Pacientes
ALTER COLUMN PatientName varchar(100) NULL
```

4- Sentencia *Insert (Insertar)*

Hemos aprendido a crear una base de datos, sabemos cómo crear y borrar tablas y sabemos cómo modificar la estructura de una tabla. Ahora es momento de insertar algunos datos ficticios en las tablas que creamos. Antes de ejecutar consultas *insert*, asegúrate de haber creado la base de datos "Hospital" (capítulo 1), la cual debe contener la tabla "Pacientes". Las columnas de la tabla "Pacientes" son las siguientes:

COLUMN_NAME

PatientID
PatientName
PatientAge
DiseaseDescription

Para insertar datos en la tabla “Pacientes”, echa un vistazo a la consulta 8.

Consulta 8:

INSERT INTO Pacientes VALUES (100, 'James', 10, 'Heart Disease')

Para insertar datos en cualquier tabla tenemos que usar *“INSERT INTO”* seguido del nombre de la tabla y luego *“VALUES”* seguido de los valores a insertar separados por comas dentro de paréntesis. Es importante recordar que el orden en que los valores son insertados debe coincidir con el orden de las columnas en la tabla. Por ejemplo, la primer columna en la tabla “Pacientes” es *PatientID*, que almacena datos de tipo *integer*; por lo tanto insertamos un número entero (*integer)* 100, como el primer valor a insertar. La segunda columna es *PatientName*; por lo tanto insertamos una cadena “James”, y así sigue.

- **Cambiar el orden de los valores insertados**

También puedes cambiar el orden en el que quieres insertar los valores en una tabla. Para ello, tienes que mencionar el orden entre paréntesis luego del nombre de la tabla. Por ejemplo, mira la siguiente consulta:

INSERT INTO Pacientes (PatientName, PatientID, DiseaseDescription, PatientAge)
VALUES ('Mike', 101, 'Lung Disease', 25)

Puedes ver que, en el orden especificado, *PatientName* viene primero, seguido de *PatientID, DiseaseDescription* y *PatientAge*. Por lo tanto, los valores se insertan de acuerdo a este orden modificado, donde *“Mike”* viene primero, seguido de “101” y así.

Ejercicio 3

Tarea:

Alterar la tabla “Pacientes” quitando la columna *PatientAge* y agregando una columna para el grupo sanguíneo del paciente. Ingrese algunos datos en la tabla modificada.

Solución

Quitar la Columna *PatientAge*

Alter table Pacientes
Drop Column PatientAge

Agregar la columna para el grupo sanguíneo del paciente

Alter table Pacientes
Add PatientBG varchar(10)

Insertar Datos en la Tabla Modificada

INSERT INTO Pacientes VALUES (100, 'James', 'Heart Disease','B+')

Capítulo 4: *Select, Where,* y *Order By*

Hemos aprendido cómo crear bases de datos, cómo crear una tabla en una base de datos y cómo alterar su estructura. También sabemos cómo insertar datos en tablas. Sin embargo, seguimos sin saber cómo leer la tabla. En la mayoría de los programas de aplicación la lectura de datos y su representación es una tarea fundamental. Afortunadamente SQL incluye consultas y operadores que, cuando se usan combinados, nos ayudan a obtener el conjunto de datos deseados de una base de datos. En SQL la sentencia *"SELECT"* se usa para obtener datos de una base de datos, la cláusula *"Where"* se usa para obtener condicionalmente datos y la cláusula *"Order By"* se usa para cambiar el orden en que los registros de datos son traídos. En este capítulo veremos estos conceptos en acción.

Contenido

- **Sentencia *SELECT***
- **Cláusula *Where***
- **Operadores SQL**

1- Sentencia *SELECT (Seleccionar)*

Antes de ejecutar una sentencia *SELECT* en la base de datos, vamos a insertar algunos datos en la tabla "Pacientes", que creamos en el capítulo anterior. Tu tabla "Pacientes" debe tener cuatro columnas: *PatientID, PatientName, PatientAge, DiseaseDescription*. Ejecuta la siguiente consulta para insertar cinco registros en la tabla "Pacientes".

Consulta 1

```
INSERT INTO Pacientes VALUES
(101, 'James', 10, 'Heart Disease'),
(150, 'Sarah', 15, 'Lung Disease'),
(245, 'Isaac', 21, 'Kidney Disease'),
(250, 'Mike', 17, 'Ear Infection'),
(301, 'Maria', 6, 'Nose Injury')
```

La consulta anterior insertará cinco registros en la tabla "Pacientes". Esta es la forma de insertar múltiples registros por medio de una única consulta.

Si queremos seleccionar todos estos registros, podemos ejecutar la siguiente sentencia *SELECT*:

Consulta 2

```
SELECT * FROM Pacientes
```

La sintaxis de la consulta *SELECT* es la más simple de todas las consultas que hemos ejecutado hasta ahora. Para seleccionar todos los registros de una tabla se usa la consulta *"SELECT * FROM"* seguida por el nombre de la tabla de la que quieres obtener registros. Por ejemplo, si quiero obtener todos los registros de la tabla "Pacientes", ejecuto la Consulta 2. La salida de la Consulta 2 será un conjunto de filas conteniendo todos los registros de la tabla "Pacientes", como se ve a continuación:

PatientID	PatientName	PatientAge	DiseaseDescription
101	James	10	Heart Disease
150	Sarah	15	Lung Disease
245	Isaac	21	Kidney Disease
250	Mike	17	Ear Infection
301	Maria	6	Nose Injury

Puedes ver que todos los registros que hemos insertado en la tabla "Pacientes" usando la Consulta 1 han sido obtenidos usando la sentencia *SELECT* en la Consulta 2.

El asterisco que sigue a la palabra clave *SELECT* trae todas las columnas de la tabla. ¿Y si quieres obtener sólo *PatientName* y *PatientAge* de todos los pacientes de la tabla "Pacientes"? No te preocupes, es bastante simple; debes modificar la sentencia *SELECT* de la siguiente manera:

Consulta 3

SELECT PatientName, PatientAge FROM Pacientes

En la Consulta 3 el asterisco ha sido reemplazado por los nombres de columna de la tabla. La consulta de arriba traerá los valores de las columnas *PatientName* y *PatientAge* de todos los registros en la tabla "Pacientes". La salida de la Consulta 3 es como sigue:

PatientName	PatientAge
James	10
Sarah	15
Isaac	21
Mike	17
Maria	6

- **Uso de *As (Como)* con la sentencia *Select (Seleccionar)***

Si miras la salida de la Consulta 2 y la Consulta 3, verás que los encabezados de la tabla no son apropiados. Es correcto tener una columna de la base de datos llamada *PatientName*, sin espacio entre *Patient* y *Name*. Sin embargo, no parece apropiado en la salida. Puedes cambiar los nombres de las columnas en la salida usando la palabra clave *"AS"* en la sentencia *select*. Por ejemplo, si quieres cambiar el nombre de la columna *PatientName* por *"Patient Name"* y la columna *PatientAge* por *"Patient Age"* en la salida, necesitas modificar la sentencia *SELECT* de la siguiente manera:

Consulta 4

SELECT PatientName AS 'Patient Name', PatientAge AS 'Patient Age' FROM Pacientes

La salida de la Consulta 4 será el siguiente conjunto de registros:

Patient Name	Patient Age
James	10
Sarah	15
Isaac	21
Mike	17
Maria	6

2- Cláusula *Where (Donde)*

En aplicaciones en tiempo real no necesitas a menudo obtener todos los registros de la base de datos. Sólo deseas los registros que cumplan ciertos criterios. Por ejemplo, podrías querer obtener los registros de pacientes de más de 10 años de edad. Es entonces que la cláusula *"Where"* entra en juego. Echa un vistazo a la Consulta 5 para ver esta cláusula en acción.

Consulta 5

SELECT * FROM Pacientes
WHERE PatientAge>10

La sintaxis de una cláusula *"where"* es evidente. Tienes que usar la palabra *"Where"* seguida de la condición que quieres que tus datos cumplan. Queremos obtener los registros en los que *PatientAge* sea mayor que 10, por lo tanto usamos la condición "*PatientAge* > 10". Aquí, el símbolo ">" es el operador SQL. Veremos diferentes operadores SQL en detalle en el próximo capítulo. La salida de la Consulta 5 será un conjunto de registros de la tabla "Pacientes" donde la edad del paciente será mayor que 10. Queda de la siguiente manera:

PatientID	PatientName	PatientAge	DiseaseDescription
150	Sarah	15	Lung Disease
245	Isaac	21	Kidney Disease
250	Mike	17	Ear Infection

3- Order By (Ordenar Por)

La cláusula “*order by*” se usa para cambiar el orden en el cual los registros son traídos de la base de datos. Si quieres obtener los registros de todos los pacientes de la tabla “Pacientes” con sus nombres en orden alfabético, puedes usar la cláusula “*order by*”. La siguiente consulta demuestra el uso de la cláusula “*order by*”:

Consulta 6

SELECT * FROM Pacientes
Order by PatientName

La salida será el conjunto de todos los pacientes de la tabla “Pacientes” ordenados alfabéticamente por su nombre.

PatientID	PatientName	PatientAge	DiseaseDescription
245	Isaac	21	Kidney Disease
101	James	10	Heart Disease
301	Maria	6	Nose Injury
250	Mike	17	Ear Infection
150	Sarah	15	Lung Disease

Puedes traer los registros de pacientes en orden alfabético inverso agregando “*Desc*” luego de la cláusula “*order by*”. Esto traerá los registros en orden alfabético inverso. La siguiente consulta lo demuestra:

Consulta 7

SELECT * FROM Pacientes
Order by PatientName Desc

La salida de la Consulta 7 será la siguiente:

PatientID	PatientName	PatientAge	DiseaseDescription
150	Sarah	15	Lung Disease
250	Mike	17	Ear Infection
301	Maria	6	Nose Injury
101	James	10	Heart Disease
245	Isaac	21	Kidney Disease

Ejercicio 4

Tarea:

En la tabla “Pacientes”, seleccionar *PatientName* y *DiseaseDescription* de todos los pacientes menores de 20 años. Ordenar el resultado por Edad descendente.

Solución

SELECT PatientName, DiseaseDescription from Pacientes
Order by PatientAge Desc

Capítulo 5: Operadores SQL I

En el capítulo 4 estudiamos el uso de cláusula "*where*". Vimos cómo podemos obtener los registros deseados basados en una condición específica usando la cláusula "*where*". Los operadores SQL son básicamente palabras o caracteres clave que se usan para especificar una condición en una cláusula "*where*". Por ejemplo, en el capítulo 4 usamos el operador mayor que ">" para traer todos los registros de pacientes cuya edad fuera mayor que 10. En los capítulos 5 y 6 veremos algunos de los operadores SQL más utilizados.

Los operadores SQL pueden ser categorizados en cuatro grupos:

- Operadores de Comparación
- Operadores Lógicos
- Operadores Conjuntivos
- Operadores de Negación
- Operadores Aritméticos

En este capítulo veremos el funcionamiento de los operadores de Comparación y los Lógicos. En el próximo capítulo estudiaremos los restantes tres tipos de operadores.

1- Operadores de Comparación

Hay seis tipos de operadores de comparación en SQL:

I. Igualdad (=)
II. Desigualdad (<>)
III. Valores menores que (<)
IV. Valores mayores que (>)
V. Valores menores o iguales que (<=)
VI. Valores mayores o iguales que (>=)

Demostraremos el uso de operadores de Desigualdad en la Consulta 1 de este capítulo. Supongamos que quieres traer los registros de todos los pacientes excepto aquellos cuya edad sea 17; Puedes probar el operador de desigualdad como sigue:

Consulta 1

```
SELECT * from Pacientes
where PatientAge <> 17
```

Esta consulta traerá los registros de todos los pacientes excepto los pacientes cuya edad sea igual a 17. Te recomiendo que pruebes el resto de los operadores de comparación tú mismo. Son bastante obvios y se explican a sí mismos.

2- Operadores Lógicos

A continuación, los operadores lógicos de SQL:

I. IS NULL

I. BETWEEN
II. IN
III. LIKE
IV. DISTINCT
V. EXISTS
VI. *ALL* y *ANY*

En este capítulo vamos a ocuparnos de los primeros cinco operadores.

- ***IS NULL (Es Nulo, Vacío)***

El operador *IS NULL* se usa para traer todos los registros de la tabla que tienen un valor nulo en una columna específica. Por ejemplo, si quieres obtener todos los registros de la tabla "Pacientes" que tienen valor nulo en *DiseaseDescription*, puedes usar la siguiente consulta:

Consulta 2

SELECT * from Pacientes
where DiseaseDescription is null

No verás ningún registro en la salida, ya que todos los registros de pacientes que hemos insertado en la tabla "Pacientes" tienen algún valor en la columna *DiseaseDescription.*

- ***BETWEEN (Entre)***

El operador *BETWEEN* se usa para filtrar registros que tienen, en una columna específica, valores dentro de un rango especificado. Por ejemplo, si quieres obtener los registros de pacientes con edades mayores o iguales a 10 y menores o iguales a 20, puedes usar el operador *BETWEEN* de la siguiente manera:

Consulta 3

SELECT * from Pacientes
where PatientAge between 10 and 20

Los registros de salida serán:

PatientID	PatientName	PatientAge	DiseaseDescription
101	James	10	Heart Disease
150	Sarah	15	Lung Disease
245	Isaac	21	Kidney Disease
250	Mike	17	Ear Infection

- ***IN (En)***

El operador *IN* se usa para comparar el valor de una columna específica con una lista de valores. Los registros cuyos valores de esa columna coincidan con uno de los valores de la lista son devueltos. Por ejemplo, si quieres obtener los registros cuyo *PatientID* sea igual a 101, 245 o 301, puedes usar el operador *IN* de la siguiente manera:

Consulta 4

SELECT * from Pacientes
where PatientID IN(101,245,301)

En la salida verás los registros de pacientes cuyo *PatientID* sea igual a 101, 245 o 301.

PatientID	PatientName	PatientAge	DiseaseDescription
101	James	10	Heart Disease
245	Isaac	21	Kidney Disease
301	Maria	6	Nose Injury

- ***LIKE (Como)***

El operador *LIKE* se usa para comparar valores de una columna específica de una tabla con valores similares, usando algunos operadores comodín. Hay dos operadores comodín que se usan con operadores *LIKE*: porcentaje (%) y guión bajo (_).

El operador *LIKE* es uno de los operadores SQL más ampliamente utilizados. Por ejemplo, si estás implementando una funcionalidad de búsqueda en tu aplicación en la que traes registros basado en una cadena similar ingresada en un recuadro de texto y los valores de la base de datos, una de las soluciones es usar operadores *LIKE*. Veamos un ejemplo simple donde *PatientName* contiene la cadena "sa".

Consulta 5

SELECT * from Pacientes
where PatientName LIKE('%sa%')

Esta consulta traerá los registros de todos los pacientes que en *PatientName* contengan la cadena “sa”. Dos pacientes, Sarah e Isaac, cumplen este criterio. La salida será la siguiente:

PatientID	PatientName	PatientAge	DiseaseDescription
150	Sarah	15	Lung Disease
245	Isaac	21	Kidney Disease

En la última consulta especificamos “sa” entre dos signos de porcentaje. Esto significa que la cadena “sa” puede estar presente en cualquier parte del valor de *PatientName*. Si quieres traer registros de todos los pacientes cuyo *PatientName* comience con “sa”, puedes usar el operador *LIKE* de la siguiente manera:

Consulta 6

```
SELECT * from Pacientes
where PatientName LIKE('sa%')
```

Como Sarah es la única paciente cuyo nombre comienza con “sa”, el suyo es el único registro que será traído.

El operador guión bajo (_) representa un único carácter. Por ejemplo, si quieres traer los registros de todos los pacientes con una “a” en la segunda posición de su nombre, puedes usar una combinación de operadores “_” y “%” de la siguiente manera:

Consulta 7

```
SELECT * from Pacientes
where PatientName LIKE('_a%')
```

La consulta de arriba implica que, sin importa qué carácter esté en la primera posición del nombre, los registros de pacientes con el carácter “a” en la segunda posición serán devueltos. La salida de la consulta anterior será la siguiente:

PatientID	PatientName	PatientAge	DiseaseDescription
101	James	10	Heart Disease
150	Sarah	15	Lung Disease
301	Maria	6	Nose Injury

- ***DISTINCT (Distinto, Definido)***

El operador *DISTINCT* se usa para obtener valores DEFINIDOS de una columna específica de una base de datos. Por ejemplo, si quieres obtener todos los valores únicos de la columna *PatientName* de la base de datos, puedes usar el operador *DISTINCT* de la siguiente manera:

Consulta 8

```
SELECT DISTINCT PatientName from Pacientes
```

Ejercicio 5

Tarea:

De la tabla “Pacientes”, seleccione los registros de todos los pacientes cuyo *PatientName* termine con “sa” y tengan una “n” en la tercera posición:

Solución

```
SELECT * FROM Pacientes
where PatientName LIKE ('__n%sa')
```

Capítulo 6: Operadores SQL II

En el capítulo 5 nos presentaron los operadores SQL, que nos ayudan a filtrar registros de la base de datos en base a una condición específica. En el capítulo 5 estudiamos los operadores de comparación y los lógicos. En este capítulo vamos a estudiar los tres tipos de operadores restantes.

Contenido

- **Operadores Conjuntivos**
- **Operadores de Negación**
- **Operadores Aritméticos**

NOTA:

Antes de ejecutar consultas en este capítulo, agrega 5 registros más a la tabla "Pacientes" usando la siguiente sentencia *INSERT*.

```
INSERT INTO Pacientes VALUES
(105, 'Joseph', 26, 'Heart Disease'),
(156, 'Julian', 31, 'Lung Disease'),
(247, 'Margret', 29, 'Heart Disease'),
(259, 'Russ', 47, 'Ear Infection'),
(318, 'Candice', 42, 'Ear Injury')
```

Ahora tu tabla "Pacientes" debe tener los siguientes 10 registros:

PatientID	PatientName	PatientAge	DiseaseDescription
101	James	10	Heart Disease
105	Joseph	26	Heart Disease
150	Sarah	15	Lung Disease
156	Julian	31	Lung Disease
245	Isaac	21	Kidney Disease
247	Margret	29	Heart Disease
250	Mike	17	Ear Infection
259	Russ	47	Ear Infection
301	Maria	6	Nose Injury
318	Candice	42	Ear Injury

1- Operadores Conjuntivos

Los operadores conjuntivos se usan para conectar dos o más condiciones. Los operadores conjuntivos se usan principalmente con una cláusula *"where"*. Hay dos operadores conjuntivos en SQL:

I. *AND*
II. *OR*

- ***AND (Y)***

Una condición *AND* se usa para ejecutar una operación lógica "Y" entre múltiples condiciones especificadas en una cláusula *"where"*. Con el operador *AND* sólo aquellos registros que cumplan todas las condiciones especificadas en múltiples cláusulas *"where"* son traídos de la base de datos. Por ejemplo, si quieres obtener los registros de todos los pacientes mayores de 20 años y cuya *DiseaseDescription* contenga la cadena *"Lung"*, puedes usar la siguiente consulta:

Consulta 1

```
SELECT * FROM Pacientes
where PatientAge > 20 AND DiseaseDescription LIKE ('%lung%')
```

La salida de la consulta anterior es la siguiente:

PatientID	PatientName	PatientAge	DiseaseDescription
156	Julian	31	Lung Disease

- ***OR (O)***

El operador *OR* se usa para ejecutar una operación lógica "O" entre múltiples condiciones especificadas en la cláusula "*where*". Por ejemplo, si quieres obtener los registros de todos los pacientes mayores de 20 años que tengan una enfermedad pulmonar o cardíaca, puedes extender la consulta 1 de la siguiente manera:

Consulta 2

```
SELECT * FROM Pacientes
where PatientAge>20 AND (DiseaseDescription LIKE ('%lung%')
OR DiseaseDescription LIKE ('%Heart%') )
```

Aquí el uso de los paréntesis es particularmente importante. Las últimas dos condiciones con el operador *LIKE* han sido encapsuladas entre paréntesis. De esta manera el operador *AND* funciona entre la primera condición y la salida del *OR* será entre las últimas dos condiciones. La salida de la consulta 2 es la siguiente:

PatientID	PatientName	PatientAge	DiseaseDescription
105	Joseph	26	Heart Disease
156	Julian	31	Lung Disease
247	Margret	29	Heart Disease

Para ver qué pasa si quitas los paréntesis que encierran las últimas dos condiciones unidas por el *OR*, ejecuta la siguiente consulta:

Consulta 3

```
SELECT * FROM Pacientes
where PatientAge>20 AND DiseaseDescription LIKE ('%lung%')
OR DiseaseDescription LIKE ('%heart%')
```

En la consulta 3, por defecto el operador *AND* operará entre la primera y la segunda condición y el operador *OR* operará sobre el resultado de "primera y segunda condición" Y la "tercera" condición. Por ello, en la salida verás los registros de todos los pacientes mayores de 20 años que tengan una enfermedad pulmonar, y también los pacientes que tengan una enfermedad cardíaca, independientemente de su edad.

PatientID	PatientName	PatientAge	DiseaseDescription
101	James	10	Heart Disease
105	Joseph	26	Heart Disease
156	Julian	31	Lung Disease
247	Margret	29	Heart Disease

2- Operadores de Negación

El operador de negación se usa para negar operadores lógicos. Se usan para invertir la salida obtenida por medio de operadores lógicos normales. Simplemente debes agregar el *NOT* antes del nombre del operador. A continuación tienes algunos de los operadores de negación más utilizados:

I. != o <> (*NOT EQUAL*)
II. *NOT BETWEEN*
III. *NOT IN*
IV. *NOT LIKE*
V. *IS NOT NULL*

Para obtener registros de todos los pacientes que no tienen una enfermedad cardíaca, puedes usar la siguiente consulta *NOT LIKE*:

Consulta 4

```
SELECT * FROM Pacientes
```

where DiseaseDescription NOT LIKE ('%heart%')

Los registros de todos los pacientes que tienen otras enfermedades que no sean cardíacas se verán en la salida, que se muestra en la tabla siguiente:

PatientID	PatientName	PatientAge	DiseaseDescription
150	Sarah	15	Lung Disease
156	Julian	31	Lung Disease
245	Isaac	21	Kidney Disease
250	Mike	17	Ear Infection
259	Russ	47	Ear Infection
301	Maria	6	Nose Injury
318	Candice	42	Ear Injury

De forma similar, si quieres obtener los registros de todos los pacientes excepto aquellos menores de 30 años y aquellos mayores de 10 años, puedes usar la siguiente consulta *NOT BETWEEN*:

Consulta 5

SELECT * FROM Pacientes
where PatientAge NOT BETWEEN 10 and 30

Es importante notar aquí que los registros que están en el límite tampoco están incluidos entre los registros obtenidos. Por ejemplo, si tienes un registro con edad 30, no será obtenido. La salida de la Consulta 5 es la siguiente:

PatientID	PatientName	PatientAge	DiseaseDescription
156	Julian	31	Lung Disease
259	Russ	47	Ear Infection
301	Maria	6	Nose Injury
318	Candice	42	Ear Injury

3- Operadores Aritméticos

Hay cuatro tipos de operadores aritméticos en SQL:

I. Adición (+)
II. Sustracción (-)
III. Multiplicación (*)
IV. División (/)

Los operadores aritméticos son usados para ejecutar operaciones matemáticas en columnas que contengan datos numéricos. Por ejemplo, puedes obtener la suma del *PatientID* y *PatientAge* de un paciente usando la siguiente consulta:

Consulta 6

SELECT PatientID + PatientAge as PatientInfo, PatientID,
PatientAge, PatientName, DiseaseDescription FROM Pacientes

En la Consulta 6 obtenemos la suma del *PatientID* y *PatientAge* de un paciente y lo mostramos en la columna *PatientInfo,* seguida por todas las otras columnas de la tabla. La salida de la Consulta 6 es la siguiente:

PatientInfo	PatientID	PatientAge	PatientName	DiseaseDescription
111	101	10	James	Heart Disease
131	105	26	Joseph	Heart Disease
165	150	15	Sarah	Lung Disease
187	156	31	Julian	Lung Disease
266	245	21	Isaac	Kidney Disease
276	247	29	Margret	Heart Disease
267	250	17	Mike	Ear Infection
306	259	47	Russ	Ear Infection

307	301	6	Maria	Nose Injury
360	318	42	Candice	Ear Injury

Ejercicio 6

Tarea:

De la tabla “Pacientes”, obtén los registros de todos los pacientes mayores de 30 años que no tengan enfermedades auditivas ni renales. Multiplica la edad de los pacientes obtenidos por 2 y muestra el resultado en la columna *ExtendedAge*.

Solución

```
SELECT PatientAge * 2 as ExtendedAge, PatientID,
PatientAge, PatientName, DiseaseDescription FROM Pacientes
where PatientAge > 30  AND DiseaseDescription NOT LIKE ('%ear%')
AND DiseaseDescription NOT LIKE('%kidney%')
```

Capítulo 7: Funciones *Aggregate, Delete,* y *Update*

Hasta ahora hemos estado trabajando con registros individuales. Hemos visto cómo insertar registros individuales en una base de datos, cómo recuperar registros de la base de datos según un criterio específico y cómo aplicar diferentes tipos de filtros para generar la salida deseada. ¿Y si queremos traer la edad máxima de los pacientes de la columna *PatientAge*? Las funciones *Aggregate* de SQL nos ayudan a realizar este tipo de tareas. Veremos algunas de las funciones *aggregate* más utilizadas en SQL. También veremos cómo borrar registros de una tabla y cómo podemos actualizar los registros existentes.

Contenido

- **Funciones *Aggregate***

I. ***Count*()**
II. ***Avg*()**
III. ***Sum*()**
IV. ***Max*()**
V. ***Min*()**
VI. ***First*/*Top*()**
VII. ***Ucase*/*Upper*()**
VIII. ***Lcase*/*Lower*()**

- **Sentencia *Delete***
- **Sentencia *Update***

1- Funciones *Aggregate (Agregar)*

Las siguientes son algunas de las funciones más utilizadas:

NOTA:

Todas las consultas en este capítulo se ejecutan en la tabla "Pacientes" con los 10 registros que insertamos en el capítulo anterior (Ver Capítulo 6).

I. *Count*() *(Contar)*

La función *Count*() cuenta la cantidad de filas que satisfacen un criterio específico. Por ejemplo, si quieres contar la cantidad de pacientes que tienen una enfermedad coronaria, puedes usar la función *Count*() de la siguiente manera:

Consulta 1

```
SELECT COUNT(PatientID) as PatientsWithHeartDisease
From Pacientes
WHERE DiseaseDescription LIKE ('%heart%')
```

La consulta de arriba cuenta los pacientes con enfermedad coronaria y muestra el resultado en la columna "*PatientsWithHeartDisease*".

I. *Avg*() (*Average = Promedio)*

La función *Avg*() devuelve el promedio de valores en una cierta columna de la tabla en base a algún criterio. Por ejemplo, si quieres obtener la edad promedio de todos los pacientes con enfermedad coronaria, puedes usar la función *Avg*() de la siguiente manera:

Consulta 2

```
SELECT AVG(PatientAge) as AverageAgeofHeartDisease
From Pacientes
WHERE DiseaseDescription LIKE ('%heart%')
```

I. *Sum*() (Sumar)

La función *Sum*() devuelve la suma de los valores de una columna específica de la tabla según algún criterio. Por ejemplo, si quieres obtener la suma de las edades de todos los pacientes, puedes usar la función *Sum*() de la siguiente manera:

Consulta 3

SELECT Sum(PatientAge) as SumOfAges
From Pacientes

I. *Max*() (*Maximum = Máximo)*

La función *Max*() devuelve el valor máximo entre todos los valores en una columna específica de la tabla. Por ejemplo, si quieres obtener la edad máxima entre todos los pacientes, puedes usar la función *Max*() de la siguiente manera:

Consulta 4

SELECT Max(PatientAge) as MaximumAge
From Pacientes

I. *Min*() (*Minimum = Mínimo)*

De igual forma, para obtener la edad mínima entre todos los pacientes, puede ejecutarse la siguiente consulta:

Consulta 5

SELECT Min(PatientAge) as MinimumAge
From Pacientes

I. *Top/First*() (*Top = Superior – First = Primero)*

Las funciones *Top*() o *First*() devuelven los primeros 'n' entre todos los valores en una columna específica de la tabla, donde "n" puede ser cualquier valor entero. Por ejemplo, si quieres obtener la edad de los primeros tres pacientes de la tabla "Pacientes", puedes usar la función *Top*() de la siguiente manera:

Consulta 6

SELECT Top 3 PatientAge as First3Ages
From Pacientes

La consulta anterior traerá las edades de los primeros tres pacientes. La salida se verá así:

First3Ages

10
26
15

Para obtener las edades de los últimos tres pacientes de la tabla "Pacientes" puedes usar la consulta *Top*() combinada con la cláusula "*Order By*" de la siguiente manera:

Consulta 7

SELECT Top 3 PatientAge as Last3Ages
From Pacientes Order by PatientAge desc

La salida de la Consulta 7 será la siguiente:

Last3Ages

47
42
31

I. *Upper()/Ucase() (Uppercase = Mayúsculas)*

La función *Upper*() o *Ucase*() convierte todos los valores de la columna seleccionada a mayúsculas. Esta función se aplica sólo a columnas que tengan valores *string* o *character*.

Consulta 8

SELECT Upper(PatientName) as PatientNameUpper
From Pacientes
La salida de la Consulta 8 es la siguiente:

PatientNameUpper

JAMES
JOSEPH
SARAH
JULIAN
ISAAC
MARGRET
MIKE
RUSS
MARIA
CANDICE

I. *Lcase*()/*Lower*() (*Lowercase = Minúsculas)*

En forma similar, para convertir valores de columna a minúsculas, se usa la función *Lcase*() o *Lower*(). Como estoy utilizando un servidor MS SQL a efectos de demostración, usaré *Lower*() en mi Consulta.

Consulta 9

SELECT Lower(PatientName) as PatientNameLower
From Pacientes

La salida contendrá todos los valores de *PatientName* en minúsculas, como se ve a continuación:

PatientNameLower

James
Joseph
Sarah
Julian
Isaac
Margret
Mike
Russ
Maria
Candice

2- Sentencia *Delete (Borrar)*

Sabemos cómo insertar registros en una base de datos; también sabemos cómo obtener registros usando la sentencia *SELECT* y cómo filtrar registros usando la cláusula "*where*" con varios operadores SQL. Ahora aprenderemos cómo borrar registros de la tabla.

Para borrar registros se usa la sentencia SQL *DELETE*. Como la sentencia *SELECT*, la sentencia *DELETE* también puede ser usada en combinación con la sentencia *SELECT* para borrar registros filtrados. Veamos cómo podemos borrar todos los registros de una tabla. Para hacer eso se puede ejecutar la siguiente consulta:

Consulta 10

DELETE From Pacientes

Esta consulta borrará todos los registros de la tabla Pacientes. Sin embargo, en ciertos escenarios podríamos querer borrar sólo ciertos registros que satisfagan un criterio específico. Por ejemplo, podrías borrar los registros de todos los pacientes con enfermedad coronaria usando la siguiente consulta:

Nota: Antes de ejecutar esta consulta, deberías tener 10 registros en la tabla "Pacientes", aquellos que insertamos en el capítulo anterior.

Consulta 11

```
Delete From Pacientes
where DiseaseDescription like('%heart%')
```

La consulta de arriba borrará todos los registros de pacientes que contengan "*heart disease*" ("enfermedad coronaria") en la columna *DiseaseDescription*.

3- Sentencia *Update (Actualizar)*

Sabemos cómo insertar, recuperar y borrar registros; en esta sección aprenderemos cómo actualizar un registro existente. Para actualizar un registro, en SQL se usa la sentencia *UPDATE*, seguida por la palabra clave *SET*, que se usa para actualizar un valor existente.

Si en la columna *DiseaseDescription* quieres reemplazar la cadena "*Heart Disease*" ("Enfermedad Coronaria") con la cadena "*Cardiac Disease*" ("Enfermedad Cardíaca"), puedes usar la sentencia UPDATE de la siguiente manera:

Consulta 12

```
UPDATE Pacientes
SET DiseaseDescription = 'Cardiac Disease'
where DiseaseDescription like('%heart%')
```

Ejercicio 7

Tarea:

Borrar los registros de todos los pacientes mayores de 30 que tengan enfermedades auditivas. Cambiar la edad de todos los pacientes con enfermedades pulmonares a 40.

Solución

Borrar Registros

```
Delete From Pacientes
where PatientAge > 30 AND DiseaseDescription LIKE ('%ear%')
```

Updating Records

```
UPDATE Pacientes
set PatientAge = 40
where DiseaseDescription LIKE ('%lung%')
```

Capítulo 8: Relaciones y Consultas de Combinación

Hasta este punto hemos estado ejecutando todas nuestras consultas en una única tabla. Sin embargo, en la vida real las bases de datos tienen cientos, incluso miles, de tablas. Estas tablas están asociadas unas a otras por medio de relaciones. En el Capítulo 1 de este libro dimos un vistazo a cómo las tablas de una base de datos están enlazadas unas a otras. Vimos cómo la tabla "Estudiantes" estaba asociada con la tabla "Departamentos" por medio de la columna DID, que almacenaba el ID del departamento al que pertenecía el estudiante. Este es un tipo de relación en base de datos. En este capítulo estudiaremos diferentes tipos de relaciones de base de datos y consultas de combinación.

Contenido

- **Relaciones de Tablas**

I. **Relaciones Uno-a-Uno**
II. **Relaciones Uno-a-Muchos**
III. **Relaciones Muchos-a-Muchos**

- **Sentencias de Combinación**

I. ***Inner Join***
II. ***Left Join***
III. ***Right Join***
IV. ***Outer Join***

- ***Group By***
- ***Having***

1- Relaciones de Tablas

Las tablas de una Base de datos pueden conectarse unas a otras por medio de tres tipos de relaciones.

- **Relaciones Uno-a-Uno**

En una relación "uno-a-uno" entre dos tablas, por cada registro de la primera tabla hay exactamente un registro en la segunda tabla. La clave primaria de la primera tabla existe como Clave Foránea en la segunda tabla, y viceversa. Por ejemplo, hay una relación "uno-a-uno" entre la tabla "Empleados" y la tabla "Pensión" si un registro de Pensión pertenece a un Empleado y un Empleado puede tener sólo un registro de Pensión. En la mayoría de los escenarios, las relaciones "uno-a-uno" se eliminan combinando los datos en una única columna.

Nota:

Una clave Foránea es básicamente una columna en la tabla que almacena la clave primaria de la tabla con la que está conectada. En el Capítulo 1 vimos que la tabla Estudiantes tenía una columna etiquetada DID, que almacenaba el ID del departamento al que pertenecía el estudiante.

- **Relaciones Uno-a-Muchos**

En una relación "uno-a-muchos", contra un registro de la primera tabla pueden haber muchos registros de la segunda tabla. En una relación "uno-a-muchos" la tabla del lado de los "muchos" de la relación se almacena como clave Foránea y la clave Primaria de la tabla está en el lado "uno" de la relación. La relación entre las tablas "Departamentos" y "Estudiantes" es el ejemplo perfecto de una relación "uno-a-muchos", ya que un

departamento puede tener muchos estudiantes. En la tabla “Estudiantes”, muchos registros pueden tener un departamento.

- **Relaciones Muchos-a-Muchos**

En las relaciones “muchos-a-muchos”, para un registro de la primer tabla puede haber múltiples registros en la segunda; para un registro de la segunda tabla, puede haber múltiples registros en la primera. La relación entre las tablas “Autores” y “Libros” es un buen ejemplo de relaciones “muchos-a-muchos”. Un libro puede ser escrito por múltiples autores, mientras que un autor puede escribir múltiples libros. En la mayoría de los casos, las relaciones “muchos-a-muchos” se separan en relaciones “uno-a-muchos” creando una tabla intermedia que tiene relación “uno-a-muchos” con ambas tablas originales.

2- Sentencias de Combinación

Las sentencias de combinación se usan para seleccionar valores de columnas de dos o más tablas que están conectadas unas con otras. Por ejemplo, puede suceder que tengas que mostrar los nombres de los estudiantes con los nombres de los departamentos a los que pertenecen. Sin embargo, no hay una columna con el nombre del departamento en la tabla “Estudiantes”; sólo contiene el ID del departamento, que funciona como clave Foránea para Departamentos. Por lo tanto, necesitamos un mecanismo para seleccionar los valores de columna de múltiples tablas que están conectadas entre sí. Las consultas *JOIN* nos ayudan a realizar esto.

Antes de ejecutar consultas *JOIN*, corre el siguiente guión en tu ventana de consultas:

Guión 1

```
Create Database Escuela
Use Escuela
Go


CREATE TABLE Estudiantes
(StudID int PRIMARY KEY NOT NULL,
StudName varchar(50) NOT NULL,
StudentAge int NULL,
StudentGender varchar(10) NOT NULL,
DepID int NULL)
CREATE TABLE Departamentos
(DepID int PRIMARY KEY NOT NULL,
DepName varchar(50) NOT NULL,
DepCapacity int NULL)
ALTER TABLE Estudiantes ADD CONSTRAINT StudDepRel FOREIGN KEY (
DepID) references Departamentos(DepID)
INSERT INTO Departamentos Values
(1, 'English', 100),
(2, 'Math', 80),
(3, 'History', 70),
(4, 'French', 90),
(5, 'Geography', 100),
(6, 'Drawing', 150),
(7, 'Architecture', 120)


INSERT INTO Estudiantes Values
(1, 'Alice', 21, 'Male', 2),
(2, 'Alfred', 20, 'Male', 3),
(3, 'Henry', 19, 'Male', 3),
(4, 'Jacobs', 22, 'Male', 5),
```

(5, 'Bob', 20, 'Male', 4),
(6, 'Shane', 22, 'Male', 4),
(7, 'Linda', 24, 'Female', 4),
(8, 'Stacy', 20, 'Female', 1),
(9, 'Wolfred', 21, 'Male', 2),
(10, 'Sandy', 25, 'Female', 1),
(11, 'Colin', 18, 'Male', 1),
(12, 'Maria', 19, 'Female', 3),
(13, 'Ziva', 20, 'Female', 5),
(14, 'Mark', 23, 'Male', 5),
(15, 'Fred', 25, 'Male', 2),
(16, 'Vic', 25, 'Male',null),
(17, 'Nick', 25, 'Male',null)

El guión de arriba creará una nueva base de datos, Escuela, con dos tablas, “Departamentos” y “Estudiantes”. Una relación “uno-a-muchos” se ha definido entre las tablas “Departamentos” y “Estudiantes” usando la siguiente consulta:

ALTER TABLE Estudiantes ADD CONSTRAINT StudDepRel FOREIGN KEY (DepID) references Departamentos(DepID)

La consulta de arriba establece que, en la tabla Estudiantes, se añade una restricción de clave Foránea llamada “*StudDepRel*” (puedes usar cualquier nombre), que establece la columna *DepID* de la tabla Estudiantes como una clave Foránea que referencia a la columna *DepID* de la tabla Departamentos.

Luego de correr el Guión 1, deberías tener en las tablas “Departamentos” y “Estudiantes” la siguiente información:

Tabla Departamentos

DepID	DepName	DepCapacity
1	English	100
2	Math	80
3	History	70
4	French	90
5	Geography	100
6	Drawing	150
7	Architecture	120

Tabla Estudiantes

StudID	StudName	StudentAge	StudentGender	DepID
1	Alice	21	Male	2
2	Alfred	20	Male	3
3	Henry	19	Male	3
4	Jacobs	22	Male	5
5	Bob	20	Male	4
6	Shane	22	Male	4
7	Linda	24	Female	4
8	Stacy	20	Female	1
9	Wolfred	21	Male	2
10	Sandy	25	Female	1
11	Colin	18	Male	1
12	Maria	19	Female	3
13	Ziva	20	Female	5
14	Mark	23	Male	5
15	Fred	25	Male	2
16	Vic	25	Male	NULL
17	Nick	25	Male	NULL

- ***INNER JOIN (Unión Interna)***

INNER JOIN (también llamada *JOIN*) trae datos de la columna seleccionada en ambas tablas sí, y sólo sí, existe un valor en común en ambas tablas en la columna especificada por la condición *JOIN*. Por ejemplo, para obtener los nombres de los estudiantes de la tabla Estudiantes junto con el nombre de su departamento de la tabla Departamentos, se usa la siguiente consulta *INNER JOIN*:

Consulta 1

```
SELECT Estudiantes.StudName, Departamentos.DepName
From Estudiantes
Join Departamentos
On Estudiantes.DepID = Departamentos.DepID
```

La salida de la consulta de arriba es la siguiente:

StudName	DepName
Alice	Math
Alfred	History
Henry	History
Jacobs	Geography

Jacobs	Geography
Bob	French
Shane	French
Linda	French
Stacy	English
Wolfred	Math
Sandy	English
Colin	English
Maria	History
Ziva	Geography
Mark	Geography
Fred	Math

Puedes ver que sólo han sido recuperados aquellos registros de las tablas “Estudiantes” y “Departamentos” que tenían un valor común en la columna *DepID* de la tabla “Estudiantes” y la columna *DepID* de la tabla “Departamentos”. Los últimos dos registros de la tabla Estudiantes” no han sido traídos, ya que no tienen un *DepID* correspondiente. De la misma manera, los últimos dos registros de la tabla “Departamentos” tampoco han sido traídos, ya que no están referenciados a ningún registro de la tabla “Estudiantes”.

- ***LEFT JOIN (Unión Izquierda)***

LEFT JOIN trae todos los registros de la primer tabla y sólo aquellos registros de la segunda tabla que tengan un valor común con la primera, como se especifica en la condición *JOIN*. Por ejemplo, la siguiente consulta trae todos los registros de la tabla “Estudiantes” y sólo aquellos registros de la tabla “Departamentos” que tienen un valor correspondiente en la tabla “Estudiantes”.

Consulta 2

```
SELECT Estudiantes.StudName, Departamentos.DepName
From Estudiantes
Left Join Departamentos
On Estudiantes.DepID = Departamentos.DepID
```

La salida de la Consulta 2 es la siguiente:

StudName	**DepName**
Alice	Math
Alfred	History
Henry	History
Jacobs	Geography
Bob	French
Shane	French
Linda	French
Stacy	English
Wolfred	Math

Sandy	English
Colin	English
Maria	History
Ziva	Geography
Mark	Geography
Fred	Math
Vic	NULL
Nick	NULL

Puedes ver que los últimos dos estudiantes no tienen un *DepID* correspondiente, pero han sido traídos de todas maneras.

- ***RIGHT JOIN (Unión Derecha)***

RIGHT JOIN trae todos los registros de la segunda tabla y sólo aquellos registros de la primera que tengan un valor común con la segunda, como se especifica en la condición *JOIN*. Por ejemplo, la siguiente consulta trae todos los registros de la tabla "Departamentos" y sólo aquellos registros de la tabla "Estudiantes" que tienen un valor de *DepID* correspondiente con la tabla "Departamentos".

Consulta 3

```
SELECT Estudiantes.StudName, Departamentos.DepName
From Estudiantes
Right Join Departamentos
On Estudiantes.DepID = Departamentos.DepID
```

StudName	DepName
Stacy	English
Sandy	English
Colin	English
Alice	Math
Wolfred	Math
Fred	Math
Alfred	History
Henry	History
Maria	History
Bob	French
Shane	French
Linda	French
Jacobs	Geography
Ziva	Geography
Mark	Geography
NULL	Drawing
NULL	Architecture

- ***FULL JOIN (Unión Completa)***

FULL JOIN es la unión de *RIGHT JOIN* y *LEFT JOIN*. *FULL JOIN* trae todos los registros de ambas tablas, ya sea que haya o no una coincidencia entre la clave Foránea y la clave Primaria de la tabla relacionada. Mira la siguiente consulta:

Consulta 4

```
SELECT Estudiantes.StudName, Departamentos.DepName
From Estudiantes
Full Join Departamentos
On Estudiantes.DepID = Departamentos.DepID
```

La salida del código de la consulta 4 es la siguiente:

StudName	DepName
Alice	Math
Alfred	History
Henry	History
Jacobs	Geography
Bob	French
Shane	French
Linda	French
Stacy	English
Wolfred	Math
Sandy	English
Colin	English
Maria	History
Ziva	Geography
Mark	Geography
Fred	Math
Vic	NULL
Nick	NULL
NULL	Drawing
NULL	Architecture

3- Group By (Agrupado por)

La sentencia "*Group By*" nos permite agrupar datos en base al resultado de algunas funciones *aggregate*. Por ejemplo, si quieres mostrar el nombre de cada departamento junto con la edad promedio de los estudiantes de ese departamento, puedes usar una sentencia "*Group By*" de la siguiente manera:

Consulta 5

```
SELECT Departamentos.DepName, AVG(Estudiantes.StudentAge) as AverageStudentAge
From Estudiantes
Right Join Departamentos
```

On Estudiantes.DepID = Departamentos.DepID
Group by DepName

La consulta de arriba calculará la edad promedio de los estudiantes que pertenecen a cada departamento y las mostrará junto a cada nombre de departamento.

DepName	AverageStudentAge
Architecture	NULL
Drawing	NULL
English	21
French	22
Geography	21
History	19
Math	22

4- Having (Que Tiene)

Cómo la cláusula "*Where*" no puede usarse para filtrar datos agrupados por funciones *aggregate*, se introdujo en SQL la sentencia "*Having*". Por ejemplo, si quieres traer sólo los nombres de aquellos departamentos cuyos estudiantes tengan una edad promedio mayor a 20, puedes usar la sentencia "*Having*" de la siguiente manera:

Consulta 7

SELECT Departamentos.DepName, AVG(Estudiantes.StudentAge) as AverageStudentAge
From Estudiantes
Right Join Departamentos
On Estudiantes.DepID = Departamentos.DepID
Group by DepName
Having AVG(Estudiantes.StudentAge) > 20

El resultado de la Consulta 6 es el siguiente:

DepName	AverageStudentAge
English	21
French	22
Geography	21
Math	22

Ejercicio 8

Tarea:

Para cada departamento, mostrar el nombre del departamento y la edad máxima de los estudiantes en ese departamento, si la edad está entre 21 y 24.

Solución

SELECT Departamentos.DepName, Max(Estudiantes.StudentAge) as Age
From Estudiantes
Right Join Departamentos
On Estudiantes.DepID = Departamentos.DepID

Group by DepName
Having Max(Estudiantes.StudentAge) Between 21 AND 24

Capítulo 9: Sub-consultas SQL

Hasta ahora hemos ejecutado consultas SQL únicas para correr funciones *insert, select, update* y *delete*. Sin embargo, hay una forma de ejecutar consultas SQL dentro de otras consultas SQL. Por ejemplo, puedes seleccionar los registros de todos los estudiantes de la base de datos cuya edad sea mayor que la de un estudiante específico. En este capítulo, demostraremos cómo podemos ejecutar sub-consultas o consultas-dentro de-consultas en SQL.

Vamos directamente al primer ejemplo. Recuerda, para este capítulo estaremos usando la base de datos "Escuela", que creamos en el capítulo 8. Deberás tener tablas etiquetadas "Estudiantes" y "Departamentos" con algunos registros.

Supongamos que queremos traer los nombres de todos los estudiantes cuya edad sea mayor que la de "*Stacy*"; podemos hacerlo por dos métodos. Primero, podemos traer la edad de *Stacy*, almacenarla en alguna variable y luego, usando una cláusula "*where*", comparar la edad en nuestra consulta *SELECT*. El segundo enfoque es embeber la consulta que trae la edad de *Stacy* dentro de la consulta que trae las edades de todos los estudiantes. El segundo enfoque utiliza una técnica de sub-consulta. Echa un vistazo a la Consulta 1 para ver las sub-consultas en acción.

Consulta 1

```
Select * From Estudiantes
where StudentAge >
(Select StudentAge from Estudiantes
where StudName = 'Stacy'
)
```

Puedes notar que en la Consulta 1 usamos paréntesis para incluir una sub-consulta en la cláusula "*where*". La consulta de arriba traerá los registros de todos los estudiantes de la tabla "Estudiantes" cuya edad sea mayor que la edad de *Stacy*. La edad de *Stacy* es 20; por lo tanto, en la salida verás los registros de todos los estudiantes mayores de 20. La salida es la siguiente:

StudID	StudName	StudentAge	StudentGender	DepID
1	Alice	21	Male	2
4	Jacobs	22	Male	5
6	Shane	22	Male	4
7	Linda	24	Female	4
9	Wolfred	21	Male	2
10	Sandy	25	Female	1
14	Mark	23	Male	5
15	Fred	25	Male	2
16	Vic	25	Male	NULL
17	Nick	25	Male	NULL

De forma similar, si quieres actualizar el nombre de todos los estudiantes cuyo nombre de departamento sea "*English*", puedes hacerlo usando sub-consultas de la siguiente manera:

Consulta 2

```
Update Estudiantes
Set StudName = StudName + ' Eng'
where Estudiantes.StudID in (
Select StudID
from Estudiantes
Join
Departamentos
On Estudiantes.DepID = Departamentos.DepID
where DepName = 'English'
)
```

En la consulta de arriba, los IDs de estudiante de todos los estudiantes en el departamento "*English*" han sido traídos usando una sentencia *JOIN* en la sub-consulta. Entonces, usando una sentencia *UPDATE*, los nombres de todos aquellos estudiantes han sido actualizados agregando la cadena "*Eng*" al final de sus nombres. Una sentencia *WHERE* se usó para aparear los IDs de los estudiantes traídos usando una sub-consulta.

Ahora, si muestras el nombre del Estudiante junto con su nombre de Departamento, verás "*Eng*" agregado al nombre de los estudiantes que pertenecen al departamento "*English*".

Ejercicio 9

Tarea:

Borrar los registros de todos los estudiantes de la tabla "Estudiantes" cuyo ID de estudiante sea menor que el ID de "*Linda*".

Solución

```
Delete From Estudiantes
where StudID <
(Select StudID from Estudiantes
where StudName = 'Linda'
)
```

Capítulo 10: Funciones de Carácter SQL

Las funciones de carácter SQL se usan para modificar la apariencia de los datos traídos. Las funciones de carácter no modifican los datos reales; en su lugar, sólo realizan ciertas modificaciones a la forma en que los datos son representados. Las funciones de carácter SQL operan sobre los datos de tipo cadena. En este capítulo estudiaremos algunas de las funciones de carácter de SQL más utilizadas.

Nota:

Todas las consultas de este capítulo se ejecutan sobre la base de datos Escuela, creada en el Capítulo 8.

- ***Concatenation* (+) (*Concatenación)***

Las funciones de concatenación se usan para concatenar dos o más cadenas. Para concatenar dos cadenas en SQL se usa el operador '+'. Por ejemplo, podemos unir los nombres de los estudiantes y su edad de la tabla "Estudiantes", y mostrarlos como una columna de la siguiente manera:

```
Select StudName +' '+StudentGender as NameAndGender
from Estudiantes
```

- ***Replace (Reemplazar)***

La función *replace* se usa para reemplazar caracteres en la cadena de salida. Por ejemplo, la siguiente consulta reemplaza "*as*" con "*rs*" en todos los nombres de los estudiantes.

```
Select StudName, REPLACE(StudName, 'ac', 'rs') as ModifiedColumn
From Estudiantes
```

El primer parámetro en la función *replace* es la columna cuyo valor quieres reemplazar; el segundo parámetro es la secuencia de caracteres que quieres reemplazar, seguido por el tercer parámetro, que especifica la secuencia de caracteres que quieres insertar en lugar de la vieja secuencia.

- ***Substring (Subcadena)***

La función *substring* devuelve la cantidad de caracteres comenzando en la posición especificada. La siguiente consulta muestra los primeros tres caracteres de los nombres de los estudiantes.

```
Select StudName, substring(StudName, 1, 3) as SubstringColumn
From Estudiantes
```

El primer parámetro en el método *substring* es el nombre de columna, el segundo parámetro es el índice inicial desde el cual quieres traer la subcadena, y el tercer parámetro es la cantidad de caracteres que quieres traer.

- ***Length (Longitud)***

La función *length* se usa para obtener la longitud (*length*) de los valores de una columna específica. Por ejemplo, para obtener la longitud de los nombres de los estudiantes de la tabla “Estudiantes”, puede ejecutarse la siguiente consulta:

```
Select StudName, Len(StudName) as NameLength
from Estudiantes
```

Puedes notar que, en la consulta de arriba, usamos la función *Len()* para obtener la longitud de los nombres; esto es porque en el servidor SQL, la función *Len()* se usa para calcular la longitud de cualquier cadena.

- ***IFNULL (IF NULL = SI ES NULO)***

La función *IFNULL* verifica si hay un valor Nulo en una columna específica de la tabla. Si existe un valor *NULL*, es reemplazado por el valor especificado como segundo parámetro de la función *IFNULL*. Por ejemplo, la siguiente consulta mostrará 50 como ID de departamento de aquellos estudiantes cuyo ID de departamento sea *NULL*.

```
Select Estudiantes.DepID, IFNULL(Estudiantes.DepID, 50)
from Estudiantes
```

- ***LTRIM (Trim=Recortar)***

La función *LTRIM* recorta todos los espacios vacíos al inicio de los valores en la columna especificada como parámetro de la función *LTRIM*. Por ejemplo, si quieres quitar todos los espacios vacíos antes de los nombres de los estudiantes en la tabla “Estudiantes”, puedes usar la consulta *LTRIM* de la siguiente manera:

```
Select Estudiantes.StudName, LTRIM(Estudiantes.StudName)
from Estudiantes
```

- ***RTRIM***

La función *RTRIM* recorta todos los espacios vacíos al final de los valores en la columna especificada como parámetro de la función *RTRIM*. Por ejemplo, si quieres eliminar todos los espacios vacíos que estén al final de los nombres de los estudiantes en la tabla “Estudiantes”, puedes usar la consulta *RTRIM* de la siguiente manera:

```
Select Estudiantes.StudName, RTRIM(Estudiantes.StudName)
from Estudiantes
```

Ejercicio 10

Tarea:

Reemplaza ‘a’ por ‘z’ en todos los nombres de los estudiantes que terminan con ‘d’. Muestra los nombres reales de los estudiantes y los nombres reemplazados en la pantalla.

Solución

```
Select StudName, REPLACE(StudName, 'a','z')
from Estudiantes
where StudName LIKE ('%d')
```

Otros Libros del Autor

JavaScript Programming: A Beginners Guide to the Javascript Programming Language (*"Programación JavaScript: Una Guía para Principiantes al Lenguaje de Programación Javascript")*

http://www.linuxtrainingacademy.com/js-programming

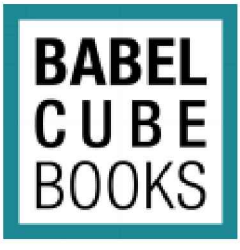
Si has intentado aprender a programar en el pasado pero no has tenido demasiado éxito, prueba *JavaScript Programming*. Te enseñará exactamente lo que necesitas saber sobre programación en el lenguaje de guión más utilizado del mundo actual. Te hará comenzar desde el principio, y te permitirá construir sobre lo que ya hayas aprendido en el camino.

Tus comentarios y recomendaciones son fundamentales

Los comentarios y recomendaciones son cruciales para que cualquier autor pueda alcanzar el éxito. Si has disfrutado de este libro, por favor deja un comentario, aunque solo sea una línea o dos, y házselo saber a tus amigos y conocidos. Ayudará a que el autor pueda traerte nuevos libros y permitirá que otros disfruten del libro.

¡Muchas gracias por tu apoyo!

¿Quieres disfrutar de más buenas lecturas?



Tus Libros, Tu Idioma

Babelcube Books ayuda a los lectores a encontrar grandes lecturas, buscando el mejor enlace posible para ponerte en contacto con tu próximo libro.

Nuestra colección proviene de los libros generados en Babelcube, una plataforma que pone en contacto a autores independientes con traductores y que distribuye sus libros en múltiples idiomas a lo largo del mundo. Los libros que podrás descubrir han sido traducidos para que puedas descubrir lecturas increíbles en tu propio idioma.

Estamos orgullosos de traerte los libros del mundo.

Si quieres saber más de nuestros libros, echarle un vistazo a nuestro catálogo y apuntarte a nuestro boletín para mantenerte informado de nuestros últimos lanzamientos, visita nuestra página web:

www.babelcubebooks.com