**PraySoft   
Development Center**

**BASE DE DATOS PARA UNIVERSITARIOS**

**Guia Practica de Aprendizaje SQL**

Publicado por:

Esgrytos

Santo Domingo, República Dominicana.

Copyright © 2016 por PraySoft, Starling A. Germosen, Todos los Derechos Reservados

**No es legal, ni está permitida, la reproducción total o parcial de este material, tampoco su presentación, copia, fotocopia o distribución por cualquier medio, físico, electrónico, mecánico, digital, por registro u otro mecanismo o método, sin una autorización escrita del** Publicado por:

PraySoft.Net

Esgrytos

Santo Domingo, República Dominicana.

Copyright © 2016 por PraySoft, Starling A. Germosen, Todos los Derechos Reservados

**No es legal, ni está permitida, la reproducción total o parcial de este material, tampoco su presentación, copia, fotocopia o distribución por cualquier medio, físico, electrónico, mecánico, digital, por registro u otro mecanismo o método, sin una autorización escrita del titular del Copyright**.

El incumplimiento de este mandato, puede traer consecuencias legales aun cuando fuere por motivos didácticos.

**Importante**

La información de este manual usa un enfoque completamente didáctico, los ejemplos plantean un conocimiento estrictamente básico como introducción a los lenguajes y plataformas descritas, los posibles errores y omisiones no serán jurídicamente responsabilidad del editor.

# Acerca del libro

Este material surge por la necesidad que he visto en las universidades, de utilizar un material tecnificado en el área de las base de datos, pero que este desarrollado con las metodologías de aprendizaje que se aplican a otras áreas, puesto que, los manuales técnicos del área informática, dejan muy de lado los aspectos pedagógicos, ya que desde la óptica de sus autores, tienden a asumir un conocimiento, cuando menos básico, del área que el estudiante está explorando.

Sin embargo, la realidad de nuestras universidades es otra, nuestros estudiantes tienen el mayor deseo de aprender pero poseen muchas lagunas que le dificultan el aprendizaje, por lo cual se necesitan otros respaldos costosos para adquirir la base necesaria del aprendizaje tecnológico o conformarse con calificaciones mediocres aun cuando ese potencial puede ser fácilmente explotado usando los métodos convenientes en cada caso.

Esta guía no pretende ser un manual de Base de Datos para desarrolladores o DBA’S, sino una guía de aprendizaje para el principiante, que ni siquiera ha usado alguna vez un computador.

Este manual es ideal para Ingenieros Industriales, Ingenieros Civiles, Administradores de Empresas y otras carreras que no dependen del desarrollo informático, pero necesitan aprender a resolver problemas informáticos en el día a día y de esa forma administrar correctamente su personal, que puede ser informático.

En las carreras de Sistema, se recomienda este libro, como un manual de consulta para los primeros ciclos de la materia y no para el nivel avanzado del mismo, pues los conceptos y la metodología utilizada es muy básica y sintáctica, lo cual puede resultar aburrido y tedioso si se tiene un conocimiento intermedio o avanzado de administración de Base de Datos.

Si ya tienes experiencia como DBA, te recomendamos que saltes directamente a los ejercicios de cada capítulo y los ejercicios finales.

Los métodos, opiniones, expresiones, links y software usados como demostración, son exclusiva responsabilidad del autor, los mismos pueden variar en el tiempo y pueden no ser aplicables o estar desfasados a la hora de leer este manual.

La información general, definiciones, esquemas y metodología aplicada, está basada en la experiencia del autor y en búsquedas rápidas en la web, por lo que algunas referencias podrían no estar debidamente citadas, si entiendes que se violan tus derechos de autor en alguna parte parcial o total del libro, háznoslo saber al correo electrónico sgermosen@praysoft.net

# Sobre el Autor

****

**Egresado de Ingeniería en Sistemas de la universidad Tecnológica de Santiago en 2010 y posteriormente egresado de la Universidad APEC de Master en Comercio Electrónico, estudia programación desde 2005 con Turbo C y qBasic, luego incursiona en la programación en Visual Basic.Net, C#, Base de Datos Oracle, MS SQL entre otras tecnologías modernas y herramientas de desarrollo.**

**Actualmente se desempeña como Docente y programador independiente de soluciones a la medida para Centros Médicos, Escuelas y Negocios dedicados al sector manufacturero o la venta de artículos.**

**“*Si no puedes explicarlo de forma sencilla, entonces tu no lo entiendes bien*” - Albert Einstein**

A partir de esta premisa, surge la idea de facilitar el aprendizaje de estas tecnologías a los estudiantes universitarios, este material es el resultado de años de dedicación e investigación en el área pedagógica.

**Dedicado a:**

A mis padres por siempre haber interpuesto su salud y hasta su alimentación, para que yo tuviese la preparación académica que hoy me ha llevado donde estoy.

A mi novia, quien me ha acompañado en los momentos más prósperos de mi vida, dándome el ánimo necesario para seguir cuando he pensado desfallecer.

A mis amigos, aquellos que me han apoyado y a los que no, también, pues mi deseo de superación siempre ha estado motivado por el deseo de impresionarlos.

# INTRODUCCIÓN

SQL (Lenguaje de Consulta Estructurado)

El SQL es un lenguaje de consulta, establecido como un lenguaje de alto nivel estándar para sistemas de base de datos relacionales. Este lenguaje lo vas a encontrar en cualquiera de los DBMS relacionales que existen en la actualidad, por ejemplo, ORACLE, SYBASES, SQL SERVER entre otros.

El SQL agrupa tres tipos de sentencias con objetivos particulares, en los siguientes lenguajes:

* Lenguaje de Definición de Datos (DDL, Data Definiton Language)
* Lenguaje de Manipulación de Datos (DML, Data Management Language)
* Lenguaje de Control de Datos (DCL, Data Control Language)

# Lenguaje de Definición de Datos (DDL, Data Definiton Language)

# Conjunto de sentencias del SQL que soportan la definición y declaración de los objetos de la base de datos. Objetos tales como: la base de datos misma “**DATABASE**”, las tablas “**TABLE**”, las Vistas “**VIEW**”, los índices “**INDEX**”, los procedimientos almacenados “**PROCEDURE**”, los disparadores o desencadenadores “**TRIGGER**”,

Reglas “**RULE**”, Dominios “**Domain**” y Valores por defecto “**DEFAULT**”.

# Lenguaje de Manipulación de Datos (DML, Data Management Language)

# Conjunto de sentencias del SQL para manipular los datos que están almacenados en las bases de datos, a nivel de filas (tuplas, rows o registros) y/o columnas (atributos). Ya sea que se requiera que los datos sean modificados, eliminados, consultados o que se agregaren nuevas filas a las tablas de las base de datos.

# Insertar “**INSERT**”, **Actualizar** “UPDATE”, **Eliminar** “DELETE” y Seleccionar “**SELECT**”

# Lenguaje de Control de Datos (DCL, Data Control Language)

# Conjunto de sentencias del SQL para controlar las funciones de administración que realiza el DBMS, tales como la seguridad, o volatilidad de la misma.

# Aceptar transacción “**COMMIT TRANSACTION**”, Rechazar la transacción “**ROLLBACK TRANSACTION**”, otorgar “**GRANT**” revocar “**REVOKE**”

# Pasos para implementar una base de datos:

1. Definir en el disco duro, el área física que contendrá las tablas de la base de datos. Sentencia SQL --> “**CREATE DATABASE MiBaseDeDatos”.**
2. Crear las diferentes tablas de la base de dato. Sentencia SQL
   * **“CREATE TABLE MiTabla”**.
3. Insertar las filas de las diferentes tablas, sin violar la integridad de datos. Sentencia SQL --> “**INSERT INTO MiTabla Values (a,b,c,d)”**
4. Actualizar los datos que cambien con el tiempo en las diferentes tablas. Sentencia SQL --> “**UPDATE MiTabla Set a=NuevoValor,c=NuevoValor”**.
5. Eliminar las filas que ya no se requieran en las diferentes tablas. Sentencia SQL --> “**DELETE MiTabla WHERE a=Condicion**”
6. Realizar las consultas deseadas a las tablas de la base de datos con la finalidad de obtener los datos contenidos dentro de las tablas. Sentencia SQL --> “**SELECT a,b,d FROM MiTabla**”
7. Dar nombre a las consultas. elaboradas en el paso No.6 cuando se requiera ocultar el diseño y columnas de las tablas a través de la creación de vistas lógicas. Sentencia SQL --> “**CREATE VIEW MiView**”.

# Crear una base de datos:

# Sentencia SQL CREATE DATABASE

SINTAXIS:

CREATE DATABASE nombre\_basededatos ON PRIMARY (

NAME = MiBaseDeDatos\_data,

filename = 'c:\MiBaseDeDatos.mdf', */\*Dirección donde se crea\*/*

SIZE = 3mb, */\*Tamaño de la base de datos\*/*

MAXSIZE = 7mb, */\*Tamaño de maximo de la base de datos\*/*

filegrowth = 2 mb */\*crecimiento de la base de datos\*/*

)

LOG ON (

NAME = MiBaseDeDatos\_log, filename = 'c:\MiBaseDeDatos.ldf',

SIZE = 3mb,

MAXSIZE = 7mb,

filegrowth = 2 mb

)

|  |  |
| --- | --- |
| Tipo de Archivo | Extension de nombre de archivos recomendada |
| Archivo de datos principal | .mdf |
| Archivo de datos secundario | .ndf |
| Archivo de registro de transacciones | .ldf |

Antes de proceder a crear la base de datos debemos averiguar si existe otra base de datos VENTASDB para borrarla y crear la nuestra. Recuerde no utilizar estas sentecias en un servidor de producción de SQL SERVER. Corra el siguiente codigo en el Query Analyser de SQLSERVER.

IF DB\_ID('VENTASDB') IS NOT NULL

BEGIN

DROP DATABASE VENTASDB

END

A continuación la sentencia CREATE DATABASE, que usaremos para la creación del espacio que contendrá la base de datos VENTASDB es el siguiente

CREATE DATABASE VENTASDB ON PRIMARY (

NAME=VENTASDB\_data, FILENAME='c:\VENTASDB.mdf', SIZE=5MB, MAXSIZE=10MB, FILEGROWTH=1MB

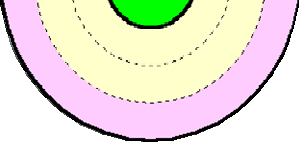
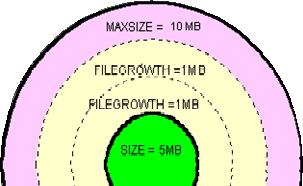
)

LOG ON (

NAME=VENTASDB\_log, FILENAME='c:\VENTASDB.ldf', SIZE=5MB, MAXSIZE=10MB, FILEGROWTH=1MB

)

En el ejemplo que se acaba de presentar se crea un archivo principal de extensión mdf de tamaño inicial de 5 MB, cuando este espacio se agote, este se expandirá en 1 MB más para tener espacio libre y meter más registros o tablas y cuando se agote nuevamente el archivo físico de 1MB, éste se expandirá en 1 MB nuevamente y así en lo sucesivo hasta alcanzar el máximo 10 MB ya de ahí no crecerá más.



**Sentencia CREATE**

Dentro del Lenguaje de Definición (DL) del SQL, la sentencia CREATE permiten la definición o creaciòn de muchos objetos de la base de datos tales como: tablas (esquemas), índices, vistas, dominios, ligaduras de integridad y procedimientos.

En esta oportunidad veremos las sentencias correspondientes a la creación de los esquemas o lo que es lo mismo las tablas que contendrán los datos de la base de datos, La sentencia CREATE TABLE.

La sentencia CREATE TABLE, define el nombre de la tabla, las columnas con su tipo de datos, las ligaduras de intengridad que vigilan el valor que se guarde como dato en las columnas o atributos sean llaves o no.

**Sintaxis:**

CREATE TABLE nombre\_tabla (

campo1 tipo dato [NULL/NOT NULL] | CHECK (expresiónLógica) | [ DEFAULT expresiónConstante],

campo2 tipo dato [NULL/NOT NULL] | CHECK (expresiónLógica) | [ DEFAULT expresiónConstante ],

campo-N,

PRIMARY KEY(campo\_llave),

FOREIGN KEY (campo\_llave) REFERENCES tabla2 (campo\_llave-tabla2)

)

# **Ligaduras**

Esto de las ligaduras es muy importante que le presenten atención. Ayuda a que dejes la vigilancia de la entrada de datos al DBMS y no al que programa la aplicación que a veces puede darse que no haga tal validación en la entrada de los datos. Esto tiene la ventaja de ahorrar líneas de códigos en los programas y no te que se implementan en la sentencia SQL CREATE TABLE .

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tipo: Integridad de Dominio o Columna** | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
|  | | |
| **Especifica un conjunto de valores que son válidos a ingresar sobre una columna específica para una tabla de la base de datos . Esta integridad se verifica a través de una la validación de los valores de datos que se ingresan y el tipo de los datos a introducir (numérico, alfanumérico, alfabético, etc.).** | | |
| **Tipos de Restricción** | **Descripción** | **Cláusula SQL** |
| **Por Defecto** | **Esta restricción asigna un valor específico a una columna cuando el valor para ello no haya sido explícitamente proporcionado para tal columna en una sentencia "INSERT" o de adición de un nuevo registro en la tabla.** | **DEFAULT**  **Por ejemplo, sí las reglas del negocio dicen que no se contratan a menores de edad, en la columna EDAD en la tabla EMPLEADO se restringe a que si una edad para un empleado que ingresa no es señalada explícitamente, el DBMS asigne 18 que es la mayoría de edad.**  **CREATE TABLE EMPLEADO (ID\_EMPL int**  **Primary KEY, EDAD int not null default 18)** |
| **Por Validación** | **Específica los valores de datos que el DBMS acepta le sean ingresados para una columna.** | **CHECK**  **CREATE TABLE EMPLEADO ( ID INT PRIMARY KEY, EDAD INT DEFAULT 18, SEXO VARCHAR(1) CHECK ( SEXO IN ('F','M') ) )**  **Por ejemplo, la columna SEXO, solo permitirá el ingreso del caracter F o M.** |
| **Por Referencia** | **Especifica los valores de datos que son aceptables para actualizar una columna y que están basados en valores de datos localizados en una columna de otra tabla.** | **REFERENCES**  **CREATE TABLE JOBS (job\_id int primary key not null, func\_id int unique)**  **CREATE TABLE EMPLEADO ( ID INT PRIMARY KEY, EDAD INT DEFAULT 18, SEXO VARCHAR(1) CHECK ( SEXO IN ('F','M') ) ,**  **NO\_FUNC int REFERENCES jobs(func\_id))**  **El valor de lo que entre en la columna NO\_FUNC en la tabla EMPLEADO deberá ser uno de los valores contenido en la columna FUNC\_ID en la Tabla JOBS.** |

|  |  |  |
| --- | --- | --- |
| **Tipo: Integridad de Entidad o Tabla** | | |
| **Específica que en una tabla o entidad, todas sus filas tenga un identificador único que diferencie a una fila de otra y también que se establezcan columnas cuyo contenido es un valor único que las hace llaves candidatas para un futuro como por ejemplo: número de cédula, número de seguro social o cuenta de e- mail.** | | |
| **Tipos de Restricción** | **Descripción** | **Cláusula SQL** |
| **Por Llave Primaria** | **Este tipo de restricción se aplica a todas las filas permitiendo que exista un identificador, que se conoce como llave primaria y que se asegura que los usuarios no introduzcan valores duplicados. Además asegura que se cree un índice para mejorar el desempeño. Los valores nulos no están permitidos para este tipo de restricción.** | **PRIMARY KEY**  **CREATE TABLE CLIENTE (NUMCLI INT not**  **null,**  **NOMCLI char(30) not null, DIRCLI char(30), FAX INT, E\_MAIL CHAR(30) UNIQUE not null, SALD\_0\_30 DECIMAL (10,2), SALD\_31\_60 DECIMAL (10,2), SALD\_61\_90**  **DECIMAL (10,2), primary key (NUMCLI) )**  **En este ejemplo, NUMCLI corresponderá a ser la llave primaria de la tabla CLIENTE.** |
| **Por Valor Unico** | **Con esta restricción se previene la duplicación de valores en columnas que tienen valor único y que no son llave primaria pero que pueden ser una llave alternativa o candidata para el futuro. Asegura que se cree (Por parte del DBMS) un índice para mejorar el desempeño. Y al igual que las llaves primarias, no se le está permitido que se introduzcan valores nulos.** | **UNIQUE**  **CREATE TABLE CLIENTE (NUMCLI INT not**  **null,**  **NOMCLI char(30) not null, DIRCLI char(30), FAX INT, E\_MAIL CHAR(30) UNIQUE not null, SALD\_0\_30 DECIMAL (10,2), SALD\_31\_60 DECIMAL (10,2), SALD\_61\_90**  **DECIMAL (10,2), primary key (NUMCLI) )**  **En este ejemplo, E\_MAIL correponderá a ser una columna de valores único en la tabla CLIENTE** |

|  |  |  |
| --- | --- | --- |
| **Tipo: Integridad Referencial** | | |
| **La Integridad Referencial asegura que las relaciones que existe entre llave primaria (en la tabla referenciada) y la llave foránea (en las tablas referenciantes) serán siempre mantenidas. Una fila o registro en la tabla referenciada (tabla donde reside la llave primaria) no puede ser borrada o su llave primaria cambiada si existe una fila o registro con una llave foránea (en la tabla referenciante) que se refiere a esa llave primaria.** | | |
| **Tipos de Restricción** | **Descripción** | **Cláusula SQL** |
| **Por Llave Foránea** | **En esta restricción se define una columna o combinación de columnas en las cuales su valor debe corresponder al valor de la llave primaria en la misma u en otra tabla.** | **FOREIGN KEY**  **CREATE TABLE PEDIDO (NUMPED INT not null PRIMARY KEY,**  **NUMCLI INT not null, FECHA\_PED DATETIME,**  **TOT\_DESC DECIMAL (10,2),**  **FOREIGN KEY (NUMCLI) REFERENCES CLIENTE(NUMCLI) )**  **En este ejemplo de la creación de la tabla PEDIDO, la columna NUMCLI correponderá a ser la llave foránea que hace referecia a la llave primaria NUMCLI (no necesariamente deben llamarse igual las columnas pero deben tener igual tipo de datos) en la tabla CLIENTE** |

A continuación se muestran las sentencias CREATE TABLE que crean las tablas de la Base de Datos VENTASDB, de la cual ya se creo el área al principio de este manual.

Es muy importante el orden en que se crean las tablas. Se debe a que no debe violarse la referencia cruzada entre las ligaduras de integridad del tipo referencial (es decir llave primaria con foráneas). Por ejemplo no puede crear la tabla de PEDIDO primero que la tabla de cliente, pues la tabla de PEDIDO hace referencia a CLIENTE.

Primero crear las tablas que solo tienen llave primaria y luego las tablas que incluyen llaves foráneas. En nuestro caso, primero creamos a las tablas: CLIENTE, ARTICULO y VENDEDOR y después a PEDIDO y a DETALLE\_PED.

|  |
| --- |
| Creación de Tabla Cliente |
| CREATE TABLE CLIENTE  (NUMCLI INT not null,  NOMCLI CHAR(30) not null, DIRCLI char(30), FAX INT, E\_MAIL CHAR(30) DEFAULT ('Desconocido'),  SALD\_0\_30 DECIMAL (10,2), SALD\_31\_60 DECIMAL (10,2), SALD\_61\_90 DECIMAL (10,2), primary key (NUMCLI) ) |
| Creación de Tabla Vendedor |
| CREATE TABLE VENDEDOR (CODVEND INT not null,  NOMVEND char(20) not null,APELLVEND char(20) not null, DIRVEND char(30), TELVEND INT, E\_MAIL CHAR(30)  DEFAULT('Desconocido'),  CUOTA DECIMAL (10,2), VENTASDB DECIMAL (10,2), primary key (CODVEND) ) |
| Creación de Tabla Artículo |
| CREATE TABLE ARTICULO (NUMART char(4) not null PRIMARY KEY, DESCRIPCION CHAR(30), PRECIO DECIMAL (10,2) NOT NULL CHECK (PRECIO >= 0.00), EXISTENCIA INT, CATEGORIA\_ART CHAR (15)) |
| Creación de Tabla Pedido |
| CREATE TABLE PEDIDO (NUMPED INT not null PRIMARY KEY, NUMCLI INT not null, FECHA\_PED DATETIME,  TOT\_DESC DECIMAL (10,2),  FOREIGN KEY (NUMCLI) REFERENCES CLIENTE(NUMCLI) , FOREIGN KEY (CODVEND) REFERENCES VENDEDOR(CODVEND)) |
| Creación de Tabla Detalle\_Ped |
| CREATE TABLE DETALLE\_PED (NUMPED INT not null,  NUMART char (4) not null, CANTIDAD INT CHECK (CANTIDAD >= 0), PRIMARY KEY (NUMPED,NUMART),  FOREIGN KEY (NUMPED) REFERENCES PEDIDO(NUMPED), FOREIGN KEY (NUMART) REFERENCES ARTICULO (NUMART) ) |

## **Eliminación de Tablas:**

La sentencia para eliminar una tabla y por ende todo los objetos asociados con esa tabla como: las vistas, disparadores, etc., DROP TABLE, donde r es el nombre de una tabla existente.

## **DROP TABLE r**

Por ejemplo si deseamos eliminar a la tabla ARTICULO\_PANAMA, revisemos haber si esta ya existe, de la siguiete manera

SELECT \* from ARTICULO\_PANAMA

Posteriormente una vez veriifcada que la tabla existe, se procede a borrar la tabla, escribiendo lo siguiente en l ainterface de consultas del SQL:

## **Sentencia ALTER**

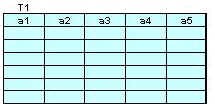
Después que una tabla ha sido utilizada durante algún tiempo, los usuarios suelen descubrir que desean almacenar información adicional con respecto a las tablas. Por ejemplo en la base de datos **VENTASDB**, se podría desear:

Añadir el nombre y numero de de una persona de contacto a cada fila de la tabla CLIENTES para contactar a los clientes.

Añadir una columna de punto de reorden mínimo en la tabla ARTICULO, para que la base de datos pueda alertar automáticamente cuando la cantidad o stock de un producto en particular esta por debajo de lo optimo para la venta.

Por lo general, esta sentencia ALTER TABLE se utiliza sobre tablas que ya poseen desde cientos a miles de filas por ser tablas de un sistemas de Base de Datos que ya esta en producción.

Los cambios que se pueden realizar con la sentencia SQL ALTER TABLE son (ver ejemplo con la figura ) :



**Tabla para ejemplos de sentencia ALTER TABLE.**

1. Añadir una definición de la columna de una tabla. Puede crearse con valores nulos o valores

ALTER TABLE nombre\_de\_la\_tabla ADD nombre\_de\_columna\_nueva TIPO\_DE\_DATO NULL

[ CONSTRAINT nombre\_de\_nueva\_restriccion CHECK / DEFAULT] Ejemplo: ALTER TABLE T1 ADD a6 VARCHAR(30) NULL

1. Eliminar una columna de la tabla. Pero antes de su eliminación deben ser eliminados por ALTER TABLE todas las restricciones que estén definidas sobre esta columna.

Ejemplo: ALTER TABLE T1 DROP COLUMN a4

1. Eliminar la definición de: llave primaria, foránea o restricciones de ligaduras de integridad (check), existentes para una tabla. Esta acción no elimina la a la columna con sus valores, ella permanece tal cual como esta, solo se elimina su definición. Este procedimiento varia de DBMS, con SQL SEVER, Primero se debe averiguar el nombre de la restricción o ligadura de integridad (CONSTRAINT) eliminar se ejecuta el procedimiento almacenado SP\_HELP nombre\_de\_Tabla y en el resultado se busca en la columna constraint\_name a la derecha de constraint\_type para saber el nombre que el DBMS le puso a la restricción de llave primaria y luego se ejecuta la sentencia ALTER TABLE con el nombre de la definición de llave primaria a eliminar.

Ejemplo: ALTER TABLE T1 DROP CONSTRAINT

PK T1 numeroqueasignaelDBMS

Esto elimina cualquiera ligadura de integridad tratada en el tema 4.2.1. Para verificar al ejecutar nuevamente SP\_HELP nombre\_de\_Tabla, en el resultado ya no sale en constraint\_type el nombre de la llave primaria.

Manual Práctico de SQL

1. Definir una llave primaria para una tabla. La columna(s) a la cual se le dará esta responsabilidad debe contener previamente valores únicos por fila.

Ejemplo: ALTER TABLE T1 ADD PRIMARY KEY (a1,a2)

1. Definir una nueva llave foránea para una tabla. La columna a definir como llave foránea debe contener previamente valores que corresponden a la llave primaria de otra tabla. Es similar al descrito en llave primaria, solo que la palabra clave reservada es ADD FOREIGN KEY .
2. Se puede habilitar o inhabilitar los disparadores (trigger) en una tabla.

ALTER TABLE nombre\_de\_la\_tabla ENABLE TRIGGER nombre\_de\_trigger

ALTER TABLE nombre\_de\_la\_tabla DISABLE TRIGGER nombre\_de\_trigger

La persona que puede realizar esta operación de alterar la tabla puede ser: el DBA o el creador de la tabla, u otra persona que el creador o DBA haya autorizado. Para la ultima situación en que a una persona se le otorga el permiso en la operación, el usuario, debe calificar el nombre de la tabla con el nombre de su creador por delante y alterar la definición de la tabla de otro usuario, por ejemplo:

ALTER TABLE nombredueño.nombretabla

**Consultas Simples**

El corazón o poder del Lenguaje SQL es el poder hacer consultas de cualquier tipo a la base de datos en forma no procedural. La sentencia SELECT es muy poderosa y ampliamente rica en sus cláusulas y variantes permitiendo la capacidad de atender en poco tiempo a consultas complejas sobre la base de datos. Está en el especialista desarrollador de aplicaciones conocerlo a profundidad para explotar las bondades y virtudes.

Gracias a esta sentencia es que se pueden realizar todas las operaciones del Algebra Relacional, tratadas en la sección 3.2 y 3.3 del modulo III. En esta sección veremos la sintaxis de como se utiliza.

## La sentencia SELECT, obtiene filas de la base de datos y permite realizar la selección de una o varias filas o columnas de una o varias tablas.

La sintaxis completa de la instrucción SELECT es compleja, pero la voy resumir como sigue (los corchetes cuadrados indican que la cláusula no es obligatoria):

SELECT nombres de las columnas

[INTO nueva Tabla destino para resultados del select\_]

FROM origenTabla

[WHERE condición de Búsqueda]

[GROUP BY nombres de columnas por la cual Agrupar] [HAVING condiciónBúsqueda para Group By ]

[ORDER BY nombre de columnas [ASC | DESC] ]

También se puede unir estas sentencias con otras por el operador UNION entre consultas para combinar sus resultados en un sola tabla de resultados sin nombre.

Manual Práctico de SQL

Veamos cales son las funciones de cada una de las cláusulas de la sentencia SELECT.

La cláusula SELECT: Se usa para listar las columnas de las tablas que se desean ver en el resultado de una consulta. Además de las columnas se pueden listas columnas a calcular por el SQL cuando actué la sentencia. Esta cláusula no puede omitirse.

La cláusula FROM: Lista las tablas que deben ser analizadas en la evaluación de la expresión de la cláusula WHERE y de donde se listarán las columnas enunciadas en el SELECT. Esta cláusula no puede omitirse.

La cláusula WHERE: establece criterios de selección de ciertas filas en el resultado de la consulta gracias a las condiciones de búsqueda. Si no se requiere condiciones de búsqueda puede omitirse y el resultado de la consulta serán todas las filas de las tablas enunciadas en el FROM.

La cláusula GROUP BY: especifica una consulta sumaria. En vez de producir una fila de resultados por cada fila de datos de la base de datos, una consulta sumaria agrupa todas las filas similares y luego produce una fila sumaria de resultados para cada grupo de los nombres de columnas enunciado en esta cláusula. En otras palabras, esta cláusula permitirá agrupar un conjunto de columnas con valores repetidos y utilizar las funciones de agregación sobre las columnas con valores no repetidas. Esta cláusula puede omitirse.

La cláusula HAVING: le dice al SQL que incluya sólo ciertos grupos producidos por la cláusula GROUP BY en los resultados de la consulta. Al igual que la cláusula WHERE, utiliza una condición de búsqueda para especificar los grupos deseados. La cláusula HAVING es la encargada de condicionar la selección de los grupos en base a los valores resultantes en las funciones agregadas utilizadas debidas que la cláusula WHERE condiciona solo para la selección de filas individuales. Esta cláusula puede omitirse.

La cláusula ORDER BY: permitirá establecer la columna o columnas sobre las cuales las filas resultantes de la consulta deberán ser ordenadas. Esta cláusula puede

## **Sentencia SELECT-FROM**

El utilizar la sentencia SELECT, con estas dos cláusulas SELECT - FROM, muestra como resultado a todas las filas existentes en las tablas especificadas en el FROM.

Ejemplo # 1: Seleccionar todas las columnas y filas de la tabla CLIENTE SELECT \* FROM CLIENTE

Da como resultado un total de 8 filas con las 7 columnas que posee la tabla CLIENTE.

Ejemplo # 2: Seleccionar columnas: nomcli y e\_mail de la tabla CLIENTE

## SELECT NUMCLI, NOMCLI, E-MAIL FROM CLIENTE

Da como resultado un total de 8 filas con solo 3 columnas.

Ejemplo # 3: Este ejemplo selecciona las columnas y las muestra con el titulo especificado en AS, es decir con un alias. A NUMCLI lo muestra como NUMERO DE CLIENTE y NOMCLI lo muestra con el nombre especificado en el AS como NOMBRE DEL CLIENTE. Esto permite mostrar una columna con encabezados mas familiares a los usuarios finales.

## SELECT NUMCLI AS 'NUMERO DE CLIENTE', NOMCLI AS 'NOMBRE DE CLIENTE' FROM CLIENTE

Da como resultado un total de 8 filas.

La cláusula AS puede omitirse y el resultado es el mismo.

SELECT NUMCLI 'NUMERO DE CLIENTE', NOMCLI 'NOMBRE DE CLIENTE' FROM CLIENTE

Da como resultado un total de 8 filas.

## SENTENCIA DE FILAS DUPLICADAS (DISTINCT)

Si una consulta incluye la llave primaria (pk) de una tabla en su lista de selección, entonces cada fila de resultados será única (ya que la llave primaria (pk) tiene un valor diferente en cada fila). Si no se incluye la llave primaria en los resultados, pueden producirse filas duplicadas. Veamos el siguiente ejemplo,

Ejemplo # 5: Seleccionar el código de artículos que han sido pedidos. Sin usar la palabra reservada DISTINCT.

El resultado tendría 18 filas y con códigos de productos repetidos.

## SELECT NUMART FROM DETALLE\_PED ORDER BY NUMART

Ejemplo # 6: Seleccionar el código de artículos que han sido pedidos. Utilizando la palabra reservada DISTINCT.

El resultado contiene menos filas, 9 filas y con códigos de productos únicos, es decir no se repiten por las veces que fueron comprados como en el ejemplo #5.

## SELECT DISTINCT NUMART FROM DETALLE\_PED ORDER BY NUMART

## **Columnas Calculadas**

Además de las columnas cuyos valores serán introducidos a la base de datos a través de la sentencia INSERT, una consulta SQL puede incluir en su cláusula SELECT columnas calculadas cuyo valor se calculan a partir de los valores de las otras columnas almacenadas en las tablas. Estas columnas, son especie de una columna virtual pues no existen físicamente en la tabla y sus valores calculados corresponden a los valores por fila.

Esta es una ventaja del SQL que evita que se tengan que diseñar columnas calculadas físicas en la base de datos trayendo perdida en almacenamiento físico y inconsistencia por falta de mantenimiento de la misma.

Ejemplo # 7: Determinar a los clientes con saldo.

Recuerde que el saldo se encuentra almacenado en tres columnas diferentes: SALD\_0\_30 + SALD\_31\_60 + SALD\_61\_90.

## SELECT NUMCLI, NOMCLI,

(SALD\_0\_30 +SALD\_31\_60 + SALD\_61\_90) as "Saldo del Cliente" FROM CLIENTE

WHERE (SALD\_0\_30 +SALD\_31\_60 + SALD\_61\_90) <> 0

Ejemplo # 8: Para determinar el monto de dinero que llevan acumulados en VENTASDB por arriba de sus cuotas los vendedores, la consulta SELECT con columnas calculadas por vendedor es,

## SELECT CODVEND, NOMVEND, APELLVEND, (VENTASDB - CUOTA)

AS 'Monto por Arriba de Coutas' FROM VENDEDOR

WHERE (VENTASDB - CUOTA) > 0

## **Condiciones de Búsqueda (=, <>, >, <, >=, <=, BETWEEN, IN, LIKE, IS NULL, compuestas (AND, OR, NOT))**

SQL usa las conectivas lógicas AND, OR y NOT en la cláusula WHERE. Los operandos de las conectivas lógicas pueden ser expresiones que contengan los operadores de comparación <,<=,>, > =, = y <>. SQL permite usar los operadores de comparación para comparar cadenas y expresiones

aritméticas, así como tipos especiales, tales como el tipo fecha.

## **Condiciones de Búsqueda (=, <>, >, <, >=, <=):**

Ejemplo # 9: Seleccionar a los clientes cuyo número de cliente está contenido entre el rango 1309 y 1950.

## SELECT NUMCLI AS 'NUMERO DE CLIENTE', NOMCLI AS 'NOMBRE DE CLIENTE'

FROM CLIENTE

WHERE NUMCLI >= 1309 AND NUMCLI <= 1950

Se obtiene el mismo resultado utilizando la palabra reservada BETWEEN. Este especifica que un valor sea menor o igual que un valor y mayor o igual que otro valor.

## **Condiciones de Búsqueda con BETWEEN:**

Ejemplo # 10: Para seleccionar a los clientes cuyo número de cliente está contenido entre el rango 1309 y 1950, es:

## SELECT NUMCLI , NOMCLI FROM CLIENTE WHERE NUMCLI BETWEEN 1309 AND 1950

Ejemplo # 11: Para seleccionar los clientes con nombre entre H Y P, la sentencia SELECT es:

## SELECT NUMCLI AS 'NUMERO DE CLIENTE', NOMCLI AS 'NOMBRE DE CLIENTE' FROM CLIENTE

WHERE (NOMCLI BETWEEN 'H' AND 'P')

**Condiciones de Búsqueda Compuestas con AND, OR y NOT:**

Ejemplo # 12: Se puede combinar las conectivas lógicas AND, OR or y NOT y filtrar mayor la selección en la consulta. Por ejemplo al seleccionar a los clientes cuyo número de cliente está contenido entre el rango 1309 y 1950 y que han comprado o incrementado su saldo dentro del mes en curso, es decir su saldo en mes corriente no es cero.

## SELECT NUMCLI AS 'NUMERO DE CLIENTE', NOMCLI AS 'NOMBRE DE CLIENTE', SALD\_0\_30

FROM CLIENTE

WHERE NUMCLI BETWEEN 1309 AND 1950 AND NOT SALD\_0\_30 = 0

Ejemplo # 13: Utilizando el operador OR podemos buscar un cliente cuyo código dudamos si es 824 o 842.

## SELECT NOMCLI, NUMCLI FROM CLIENTE WHERE NUMCLI = 842 OR NUMCLI = 824

Ejemplo # 14: Para seleccionar los vendedores cuyas VENTASDB es mayor que la couta que se espera deben vender y que sus VENTASDB son superiores a B/8000.00.

## SELECT CODVEND, NOMVEND, APELLVEND, VENTASDB, CUOTA FROM VENDEDOR

WHERE VENTASDB > CUOTA AND VENTASDB > 8000

**Condiciones de Búsqueda con LIKE:**

Ejemplo # 15: Seleccionar a todos los clientes que contengan la letra "H" dentro de su nombre.

## SELECT NUMCLI AS 'NUMERO DE CLIENTE', NOMCLI AS 'NOMBRE DE CLIENTE' FROM CLIENTE WHERE NOMCLI LIKE '%H%'

## **Condiciones de Búsqueda con la función SOUNDEX:**

Se emplea en situaciones donde se tiene idea del sonido de un valor de columna pero que no se conoce su correcta escritura (es una función nueva incorporada en el SELECT).

Ejemplo # 16: Seleccionar a todos los clientes cuyo nombre suene parecido al especificado en la función SOUNDEX ( ).

SELECT NUMCLI, NOMCLI FROM CLIENTE WHERE SOUNDEX (NOMCLI) = SOUNDEX ( 'DUIT' )

**Condiciones de Búsqueda con IS NULL:**

Los valores de NULL crean una lógica trivaluada para las condiciones de búsqueda en SQL. Para una fila determinada, el resultado de una condición de busqueda puede ser TRUE o FALSE, o puede ser NULL debido a que una de las columnas utilizadas en la evaluación de la condición de búsqueda contenga un valor NULL.

A veces es útil comprobar explícitamente los valores NULL en una condición de búsqueda y manejarlas directamente.

Ejemplo # 17: Consultar que filas de la tabla ARTICULO tienen valor NULL en algunas de sus columnas: DESCRIPCION, EXISTENCIA Y CATEGORIA\_ART.

## SELECT \*

FROM ARTICULO

WHERE DESCRIPCION IS NULL OR EXISTENCIA IS NULL OR CATEGORIA\_ART IS NULL

## **Ordenación de resultados de Consulta (cláusula ORDER BY)**

Al igual que las filas de una tabla en la base de datos las filas de los resultados de una consulta no están dispuestas en ningún orden particular. Existen situaciones en la que es necesario ver la información en un orden en especial, como en orden alfabetico (ASC, ascendente) u ver a las cifras de dinero listadas de mayor monto a menor (DESC, descendente). Se puede pedir a SQL que ordene los resultados de una consulta incluyendo la cláusula ORDER BY en la sentencia SELECT.

Ejemplo # 18: Para buscar la información de los vendedores por orden de su apellido o nombre, la sentencia select con la cláusula ORDER BY seria la siguiente:

## SELECT \*

FROM VENDEDOR ORDER BY APELLVEND

Ejemplo # 19: Para determinar en una lista resultante del ejemplo #8, quien el nombre del vendedor con mayor el monto de dinero es conveniente ordenar por esta que llevan acumulados en VENTASDB por arriba de sus cuotas los vendedores, la consulta SELECT con columnas calculadas por vendedor es,

## SELECT CODVEND, NOMVEND, APELLVEND, (VENTASDB - CUOTA)

AS 'Monto por Arriba de Coutas' FROM VENDEDOR

WHERE (VENTASDB - CUOTA) > 0 ORDER BY (VENTASDB - CUOTA) DESC

## **Consultas a Múltiples Tablas**

Generalmente el poder de la sentencia SELECT se basa en su capacidad de poder en una sola sentencia consultar múltiples tablas simultáneamente. Esta operación también se le llama JOIN y es lo que en particular llamo un "pegue de tablas en forma horizontal" y ocurre gracias a que existen columnas de conexión ósea atributos de asociación comunes en las tablas que se unen en esta forma, vea figura:



Para que se puedan realizar consultas a múltiples tablas el requisito principal es que las tablas a reunirse en una consulta tengan columnas con valores o dominios comunes, es decir columna de conexión. Si reunimos a las tablas de la figura 4.4.a, de tal forma que la Tabla1 se reúna con la Tabla2 y la Tabla2 se reuniera con la Tabla3, el requisito indispensable sería: la Tabla1 debe tener una columna común en la Tabla2 y Tabla2 debe tener una columna de conexión en la Tabla3. Por lo general estas columnas comunes son: en una tabla la columna es la llave primaria y en la otra tabla la columna asociada es la llave foránea que referencia a la primaria. Para este tipo de consultas a múltiples tablas la cláusula FROM es la responsable de indicar cual será la(s) tabla(s) fuentes.

Existen dos formas de sintaxis permitidas para la escritura de la sentencia SELECT para la reunión de tablas, basada en la figura 4.4.a, estas formas son las siguientes:

FORMA 1:

SELECT a1, a2, a5, b1, b2, b3, c1, c2, c3

FROM Tabla1, Tabla2, Tabla3

WHERE Tabla1.a1 = Tabla2.b2 AND Tabla2. b1 = Tabla3.c1 FORMA 2:

SELECT a1, a2, a5, b1, b2, b3, c1, c2, c3

FROM Tabla1 INNER JOIN Tabla2 ON Tabla1.a1 = Tabla2.b2 INNER JOIN Tabla3 Tabla2. b1 = Tabla3.c1

Note que las columnas comunes en ambas tablas han de calificarse con el nombre de la tabla que pertenecen para evitar errores de ambigüedad.

La Forma 2, tiene la ventaja de liberar a la cláusula WHERE y dejar esta para filtros específicos sobre las filas resultantes de la reunión de tablas.



Manual Práctico de SQL

Ejemplo #21: Si se desea unir la tabla de CLIENTES con PEDIDO (ver figura 4.4.b.), para ambas tablas, la columna de conexión es NUMCLI, la consulta SELECT que realiza esta reunión es la siguiente:

FORMA 1:

SELECT CLIENTE.NUMCLI, NOMCLI, NUMPED, FECHA\_PED, TOT\_DESC FROM CLIENTE, PEDIDO

WHERE CLIENTE.NUMCLI = PEDIDO.NUMCLI FORMA 2:

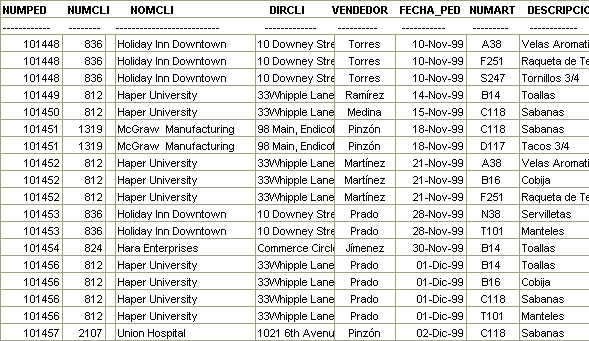
SELECT CLIENTE.NUMCLI, NOMCLI, NUMPED, FECHA\_PED, TOT\_DESC FROM CLIENTE INNER JOIN PEDIDO

ON CLIENTE.NUMCLI = PEDIDO.NUMCLI

Ejemplo # 24: Seleccionar todos los pedidos con su información completa, esto incluye: datos del cliente, datos del pedido, datos del detalle de la compra, datos del articulo de la compra y datos den vendedor que atendió el pedido. Esta consulta requiere que se realice la unión (join) de las cinco tablas: Cliente, Pedido, Detalle\_Ped , Artículo y Vendedor, que son todas las tablas de la base de datos del **VENTASDB.**

SELECT PEDIDO.NUMPED, CLIENTE.NUMCLI,NOMCLI,DIRCLI, APELLVEND AS VENDEDOR, FECHA\_PED,ARTICULO.NUMART, DESCRIPCION,PRECIO,CANTIDAD,(PRECIO\*CANTIDAD) AS TOTAL

FROM CLIENTE INNER JOIN PEDIDO ON CLIENTE.NUMCLI=PEDIDO.NUMCLI INNER JOIN DETALLE\_PED ON PEDIDO.NUMPED = DETALLE\_PED.NUMPED INNER JOIN ARTICULO ON DETALLE\_PED.NUMART=ARTICULO.NUMART INNER JOIN VENDEDOR ON PEDIDO.CODVEND = VENDEDOR.CODVEND ORDER BY PEDIDO.NUMPED



## **Consultas Resumen**

Estas consultas resultan ventajosas para hacer búsquedas a nivel de los valores de las columnas a nivel de todas las filas. Por ejemplo si se desea saber en una tabla de 10,000 filas: el mayor saldo adeudado, la cuota mínima o máxima asignada a los vendedores, el tamaño promedio de pedidos, el total de clientes o vendedores (es decir total de filas).

Ejemplo # 27: Determinar en una misma consulta lo siguiente:

* + Total de Vendedores. COUNT (NOMVEND)
  + Monto Total de COUTAS que deben alcanzarse entre todos los vendedores. SUM(CUOTA).
  + El monto promedio de cuotas asignado a un vendedor. AVG(COUTA)
  + El valor de la cuota máxima asignado al un vendedor. MAX(COUTA)
  + El valor de la cuota mínima asignado a un vendedor. MIN (COUTA) SENTENCIA SQL:

USE VENTASDB

SELECT COUNT (NOMVEND)AS "TOTAL VENDEDORES", SUM(CUOTA)"MONTO DE CUOTAS",AVG(CUOTA) AS "CUOTA PROMEDIO", MAX(CUOTA) "CUOTA MAX.", MIN(CUOTA)"CUOTA MIN." FROM VENDEDOR

El resultado de esta consulta es una fila con los siguientes valores:

TOTAL VENDEDORES MONTO DE CUOTAS CUOTA PROMEDIO CUOTA MAX. CUOTA MIN.

------------------------------ ----------------------------- ---------------------------- --------- ----

------- ---------------

10 25341.00 2534.10

8705.00 200.00

**Otros ejemplos: Sentencia SUM:**

SELECT SUM (cantidad) as 'Suma de las Cantidades' FROM detalle\_ped

**Sentencia AVG:**

SELECT AVG (cantidad) as 'Cantidad Promedio Pedida' FROM detalle\_ped

**Sentencia MIN:**

SELECT MIN (cantidad) as 'Cantidad Mínimo Pedida' FROM detalle\_ped

**Sentencia MAX:**

SELECT MAX (cantidad) as 'Cantidad Máxima Pedida' FROM detalle\_ped

**Sentencia COUNT**

SELECT COUNT (\*) as 'Total de Clientes' FROM CLIENTE

**Sentencia DISTINCT**

SELECT COUNT(DISTINCT numped) as 'Total de Pedidos Solicitados' FROM detalle\_ped

En esta ultima consulta hace que los números de pedidos en DETALLE\_PED , que son repetidos, la cláusula DISTINCT cuente solo números de uno de los que se repiten.

# **CONCLUSIÓN**

Este manual fue hecho con el propósito de ayudar a los estudiantes universitarios que tienen un deseo de aprender, más fuerte cada día.

También para ayudarme a mí mismo en la codificación de SQL.