

## **SQL y las Bases de datos**

### **Administración de bases de Datos**

**Por**

**Edward Haidyggguert González Rodríguez**  
**Código: 71729158**

**UNIVERSIDAD NACIONAL ABIERTA Y A DISTANCIA - UNAD**  
**ESCUELA DE CIENCIAS BÁSICAS TECNOLOGÍA E INGENIERÍA**  
**Medellín**

## Sistemas de Bases de Datos Relacionales

### Sistemas de Administración de Bases de Datos (DBMS)

Un sistema de administración de bases de datos DBMS (Database Management System, por sus siglas en Inglés) es un sistema basado en computador (software) que maneja una base de datos, o una colección de bases de datos o archivos. La persona que administra un DBMS es conocida como el DBA (Database Administrator, por sus siglas en inglés).

### USOS Y FUNCIONES DE UN DBMS

**Los sistemas de administración de bases de datos son usados para:**

- Permitir a los usuarios acceder y manipular la base de datos proveyendo métodos para construir sistemas de procesamiento de datos para aplicaciones que requieran acceso a los datos.
- Proveer a los administradores las herramientas que les permitan ejecutar tareas de mantenimiento y administración de los datos.

**Algunas de las funciones de un DBMS son:**

- Definición de la base de datos - como la información va a ser almacenada y organizada.
- Creación de la base de datos - almacenamiento de datos en una base de datos definida.
- Recuperación de los datos - consultas y reportes.
- Actualización de los datos - cambiar los contenidos de la base de datos.
- • Programación de aplicaciones de para el desarrollo de software.
- Control de la integridad de la base de datos.
- Monitoreo del comportamiento de la base de datos.

### CARACTERISTICAS DE UN DBMS

#### Control de la redundancia de datos

Este consiste en lograr una mínima cantidad de espacio de almacenamiento para almacenar los datos evitando la duplicación de la información. De esta manera se logran ahorros en el tiempo de procesamiento de la información, se tendrán menos inconsistencias, menores costos operativos y hará el mantenimiento más fácil.



### **Compartimiento de datos**

Una de las principales características de las bases de datos, es que los datos pueden ser compartidos entre muchos usuarios simultáneamente, proveyendo, de esta manera, máxima eficiencia.

### **Mantenimiento de la integridad**

La integridad de los datos es la que garantiza la precisión o exactitud de la información contenida en una base de datos. Los datos interrelacionados deben siempre representar información correcta a los usuarios.

### **Soporte para control de transacciones y recuperación de fallas.**

Se conoce como transacción toda operación que se haga sobre la base de datos. Las transacciones deben por lo tanto ser controladas de manera que no alteren la integridad de la base de datos. La recuperación de fallas tiene que ver con la capacidad de un sistema DBMS de recuperar la información que se haya perdido durante una falla en el software o en el hardware.

### **Independencia de los datos.**

En las aplicaciones basadas en archivos, el programa de aplicación debe conocer tanto la organización de los datos como las técnicas que el permiten acceder a los datos. En los sistemas DBMS los programas de aplicación no necesitan conocer la organización de los datos en el disco duro. Este totalmente independiente de ello.

### **Seguridad**



La disponibilidad de los datos puede ser restringida a ciertos usuarios. Según los privilegios que posea cada usuario de la base de datos, podrá acceder a mayor información que otros.

### **Velocidad**

Los sistemas DBMS modernos poseen altas velocidades de respuesta y proceso

### **Independencia del hardware**

La mayoría de los sistemas DBMS están disponibles para ser instalados en múltiples plataformas de hardware.



## Los Sistemas de Bases de Datos Relacionales (RDBMS)

Los sistemas de bases de datos relacionales RDBMS (Relational Database Management System, por sus siglas en Inglés) tales como Oracle, MySQL, SQL Server, PostgreSQL, Informix, entre otros, le permiten ejecutar las tareas que se mencionan a continuación, de una forma entendible y razonablemente sencilla:

1. Le permiten ingresar datos al sistema.
2. Le permiten almacenar los datos.
3. Le permiten recuperar los datos y trabajar con ellos.
4. Le proveen herramientas para capturar, editar y manipular datos.
5. Le permiten aplicar seguridad.
6. Le permiten crear reportes e informes con los datos.

### DEFINICIÓN Y TERMINOLOGÍA DE UN RDBMS

Los sistemas de base de datos relacionales son aquellos que almacenan y administran de manera lógica los datos en forma de tablas. Una TABLA es, a su vez, un método para presentar los datos en la forma de filas y columnas.

Cada columna representa un campo único de un registro. Varias de estas columnas o campo componen un registro, proveyendo información significativa e interrelacionada. Cada registro es representado en una fila. Una tabla puede consistir en varias columnas. Muchas de las tablas que poseen datos interrelacionados e interdependientes son agrupadas por medio de el establecimiento de relaciones entre ellas. Al administrar las tablas y sus relaciones, encontramos los medios para insertar, borrar, consultar y actualizar la información de un sistema RDBMS.

TABLA EMPLEADOS		
NUM-EMP	NOMBRE-EMP	NUM-DEPT
1001	Andres	AB101
1002	Maria	AB102
1003	Jose	AB103

TABLA DEPARTAMENTOS	
NUM-DEPT	NOMBRE-DEPT
AB101	Finanzas
AB102	Contabilidad
AB103	Ventas

FIGURA 1

En la tabla anterior, la tabla Empleados consiste en tres columnas y tres filas. Las columnas o campo conforman un registro lógico, correspondiente a un empleado. La tabla Empleados está relacionada con la tabla de Departamentos por medio de una columna "Numero de Departamento" que aparece en ambas tablas.

### Llave Primaria

Hemos visto que los datos son almacenados de manera lógica en tablas en la Bases de datos relacionales. Cada tabla tiene un nombre único. Para identificar una fila particular en una tabla, se usa una columna o combinación de columnas. Esta columna debe ser tal que identifique de manera única e inequívoca cada fila. No puede haber mas de dos filas (registros) en una tabla que tengan el mismo valor para la columna que haya sido elegida como llave primaria. Una columna identificada como la llave primaria no puede tener valores duplicados no nulos. Por ejemplo, considerando la tabla de Empleados presentada en la Figura No. 1, podemos ver que cada empleado tiene un único numero de empleado. La columna "NUM-EMP" puede ser escogida como la llave primaria. Similarmente, la columna "NUM-DEPT" en la tabla de Departamentos puede ser igualmente una llave primaria.

### Llave Foránea

La llave primaria y la llave foránea son usadas para establecer relaciones entre tablas. En la Figura No. 1 el dominio de los valores de la columna "NUM-DEPT" de la tabla Empleados se encuentra dentro del rango de valores de la columna "NUM-DEPT" de la tabla Departamentos. Un empleado deber pertenecer a un Departamento que esté listado en la tabla Departamentos. Se considera entonces que la columna "NUM-DEPT" en la tabla Empleados es una llave foránea. De esta manera, la existencia de esta llave foránea en la tabla Empleados controla que no pueda ser ingresado un nuevo registro de un empleado si este no pertenece primero a un Departamento. Si el empleado que desea ingresarse a la tabla trabaja en un Departamento que no esta listado en la tabla Departamentos, primero debe crearse el registro del Departamento en su respectiva tabla, y luego si procedemos a ingresar al empleado. Este tipo de control que impone la asignación de una llave foránea en una tabla es de mucha utilidad para evitar la existencia de registros huérfanos y para evitar la incongruencia de datos, temas que veremos mas adelante. Además, como dijimos al principio, la llave foránea nos permite relacionar dos tablas, lo cual nos permite compartir y repartir la información de manera que no tengamos los mismos datos duplicados en varias tablas. Estos conceptos serán aterrizados en la sección de Normalización de tablas que se estudiará en un capítulo posterior

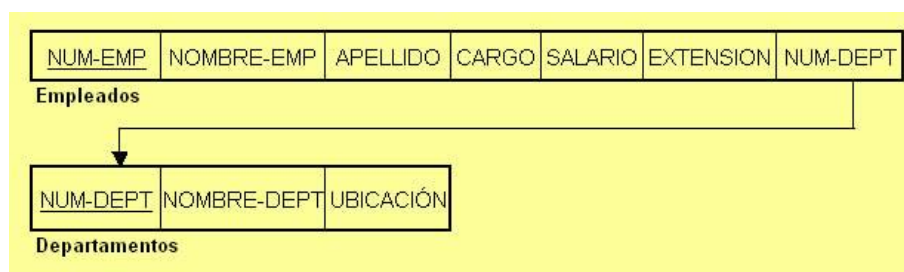


FIGURA 2



En la figura No. 2 hemos establecido la siguiente convención: En los esquemas de tablas, las llaves primarias están subrayadas. Igualmente diagramaremos restricciones de integridad referencial a través de líneas de conexión que van desde cada llave foránea hasta la llave primaria que referencie. Para que haya mejor claridad, la punta de la flecha deberá apuntar hacia la llave primaria de la tabla referenciada.

### Nulos

Un Nulo se puede interpretar como un valor indefinido o como ningún valor. Los nulos son usados en las columnas donde se desconozca su valor. Un nulo no significan espacios en blanco. Un valor "nulo" no puede ser usado para hacer ningún cálculo u operaciones de comparación. Un "nulo" puede ser comparable a un infinito. Un "nulo" no es igual a otro "nulo".

### Vistas

Los RDBMS gestionan la estructura física de los datos y su almacenamiento. Con esta funcionalidad, el RDBMS se convierte en una herramienta de gran utilidad. Sin embargo, desde el punto de vista del usuario, se podría discutir que los RDBMS han hecho las cosas más complicadas, ya que ahora los usuarios ven más datos de los que realmente quieren o necesitan, puesto que ven la base de datos completa. Conscientes de este problema, los RDBMS proporcionan un mecanismo de *vistas* que permite que cada usuario tenga su propia vista o visión de la base de datos. El lenguaje de definición de datos permite definir vistas como subconjuntos de la base de datos. Las vistas, además de reducir la complejidad permitiendo que cada usuario vea sólo la parte de la base de datos que necesita, tienen otras ventajas:

- Las vistas proporcionan un nivel de seguridad, ya que permiten excluir datos para que ciertos usuarios no los vean. Las vistas proporcionan un mecanismo para que los usuarios vean los datos en el formato que deseen.
- Una vista representa una imagen consistente y permanente de la base de datos, incluso si la base de datos cambia su estructura.

## PROPIEDADES DE UN RDMS - REGLAS DE CODD

Un sistema de bases de datos (DBMS) puede ser considerado como relacional si sigue las tres reglas de oro, las cuales se enuncian a continuación:

1. Toda la información debe estar representada en tablas.
2. La recuperación de los datos debe ser posible usando sentencia de SELECT, JOIN y PROJECT.
3. Todas las relaciones entre los datos deben ser explícitamente representadas en los mismos datos.

Para definir los requerimientos de una base de datos relacional RDBMS mas rigurosamente, Codd ha formulado 12 reglas comúnmente conocidas como las *Reglas de Codd*. De un producto se puede decir que es real y completamente relacional si sigue todas la reglas, pero no existe ninguno que efectivamente si las cumpla. Por eso es que se ha generalizado el uso de la regla No. 0 que reza: "Cualquier base de datos relacional verdadera debe ser administrable enteramente a través de sus propias capacidades relacionales".

**OBSERVACIÓN PARA LOS ESTUDIANTES: LOS ENUNCIADOS PRESENTADOS A CONTINUACIÓN CORRESPONDEN A LAS REGLAS TEORICAS QUE ESTABLECIÓ CODD PARA DEFINIR UN SISTEMA DE ADMINISTRACIÓN DE BASES DE DATOS RELACIONALES. SU TERMINOLOGÍA ES MUY TÉCNICA Y PUEDE NO SER ENTENDIDA POR COMPLETO A NO SER QUE SE TENGA UN MANEJO AVANZADO DE LAS APLICACIONES BASADAS EN SQL. SE PRESENTAN A MANERA DE ILUSTRACIÓN TEORICA. MAS ADELANTE, SE OFRECERA UN RESUMEN EN TERMINOS DE FÁCIL COMPRESIÓN.**

### **Regla No. 1 - La Regla de la información**

"Toda la información en un RDBMS esta explícitamente representada de una sola manera, por valores en una tabla". Cualquier cosa que no exista en una tabla no existe del todo. Toda la información, incluyendo nombres de tablas, nombres de vistas, nombres de columnas, y los datos de las columnas deben estar almacenados en tablas dentro de las bases de datos. Las tablas que contienen tal información contituyen el *Diccionario de Datos*.

### **Regla No. 2 - La regla del acceso garantizado**

"Cada ítem de datos debe ser lógicamente accesible al ejecutar una búsqueda que combine el nombre de la tabla, su llave primaria, y el nombre de la columna. Esto significa que dado un nombre de tabla, dado el valor de la llave primaria, y dado el nombre de la columna requerida, deberá encontrarse uno y solamente un valor. Por esta razón la definición de llaves primarias para todas las tablas es prácticamente obligatoria.

### **Regla No. 3 - Tratamiento sistemático de los valores nulos**

"La información inaplicable o faltante puede ser representada a través de valores nulos". Un RDBMS debe ser capaz de soportar el uso de valores nulos en el lugar de columnas cuyos valores sean desconocidos o inaplicables.

### **Regla No. 4 - La regla de la descripción de la base de datos**

"La descripción de la base de datos es almacenada de la misma manera que los datos ordinarios, esto es, en tablas y columnas, y debe ser accesible a los usuarios autorizados". La

información de tablas, vistas, permisos de acceso de usuarios autorizados, etc, debe ser almacenada exactamente de la misma manera: En tablas. Estas tablas deben ser accesibles igual que todas las tablas, a través de sentencias de SQL.

#### **Regla No. 5 - La regla del sub-lenguaje Integral.**

"Debe haber al menos un lenguaje que sea integral para soportar la definición de datos, manipulación de datos, definición de vistas, restricciones de integridad, y control de autorizaciones y transacciones". Esto significa que debe haber por lo menos un lenguaje con una sintaxis bien definida que pueda ser usado para administrar completamente la base de datos.

#### **Regla No. 5 - La regla del sub-lenguaje Integral.**

"Debe haber al menos un lenguaje que sea integral para soportar la definición de datos, manipulación de datos, definición de vistas, restricciones de integridad, y control de autorizaciones y transacciones". Esto significa que debe haber por lo menos un lenguaje con una sintaxis bien definida que pueda ser usado para administrar completamente la base de datos.

#### **Regla No. 7 - La regla de insertar y actualizar**

"La capacidad de manejar una base de datos con operandos simples aplica no solo para la recuperación o consulta de datos, sino también para la inserción, actualización y borrado de datos". Esto significa que las cláusulas SELECT, UPDATE, DELETE e INSERT deben estar disponibles y operables sobre los registros independientemente del tipo de relaciones y restricciones que haya entre las tablas.

#### **Regla No. 8 - La regla de independencia física**

"El acceso de usuarios a la base de datos, a través de terminales o programas de aplicación, debe permanecer consistente lógicamente cuando quiera que haya cambios en los datos almacenados, o sean cambiados los métodos de acceso a los datos". El comportamiento de los programas de aplicación y de la actividad de usuarios vía terminales debería ser predecible basados en la definición lógica de la base de datos, y este comportamiento debería permanecer inalterado, independientemente de los cambios en la definición física de ésta.

#### **Regla No. 9 - La regla de independencia lógica**

"Los programas de aplicación y las actividades de acceso por terminal deben permanecer lógicamente inalteradas cuando quiera que se hagan cambios (según los permisos asignados) en las tablas de la base de datos". La independencia lógica de los datos especifica que los



programas de aplicación y las actividades de terminal deben ser independientes de la estructura lógica, por lo tanto los cambios en la estructura lógica no deben alterar o modificar estos programas de aplicación.

### **Regla No. 10 - La regla de la independencia de la integridad**

"Todas las restricciones de integridad deben ser definibles in los datos, y almacenables en el catalogo, no n el programa de aplicación". Las reglas de integridad son: 1. Ningún componente de una llave primaria puede tener valores en blanco o nulos. (esta es la norma básica d integridad). 2. Para cada valor de llave foránea deberá existir una valor de llave primaria concordante. La combinación de estas reglas aseguran que haya Integridad referencial.

### **Regla No. 11 - La regla de la distribución**

"El sistema debe poseer un lenguaje de datos que pueda soportar que la base de datos esté distribuida físicamente en distintos lugares sin que esto afecte o altere a los programas de aplicación". El soporte para bases de datos distribuidas significa que una colección arbitraria de relaciones, bases de datos corriendo en una mezcla de distintas máquinas y distintos sistemas operativos y que este conectada por una variedad de redes, pueda funcionar como si estuviera disponible como una única base de datos en una sola máquina.

### **Regla No. 12 - Regla de la no-subversión**

"Si el sistema tiene lenguajes de bajo nivel, estos lenguajes de ninguna manera pueden ser usados para violar la integridad de las reglas y restricciones expresadas en un lenguaje de alto nivel(como SQL)". Algunos productos solamente construyen una interfaz relacional para sus bases de datos No relacionales, lo que hace posible la subversión (violación) de las restricciones de integridad. Esto no debe ser permitido.

**TAL COMO LO ANUNCIAMOS, A CONTINUACIÓN PRESENTAREMOS UN RESUMEN EN TERMINOLOGÍA DE FÁCIL COMPRESNSION, QUE ILUSTRA LOS PRINCIPALES CONCEPTOS DE LAS BASES DE DATOS Y LAS REGLAS DE CODD LLEVADAS A LA PRACTICA.**

- Un RDBMS debe proporcionar a los usuarios la capacidad de almacenar datos en la base de datos, acceder a ellos y actualizarlos. Esta es la función fundamental de un RDBMS y por supuesto, el RDBMS debe ocultar al usuario la estructura física interna (la organización de los archivos y las estructuras de almacenamiento).

- Un RDBMS debe proporcionar un catálogo en el que se almacenen las descripciones de los datos y que sea accesible por los usuarios. Este catálogo es lo que se denomina diccionario de datos y contiene información que describe los datos de la base de datos (metadatos). Normalmente, un diccionario de datos almacena: Nombre, tipo y tamaño de los datos, Nombre de las relaciones entre los datos.
- Un RDBMS debe proporcionar un mecanismo que garantice que todas las actualizaciones correspondientes a una determinada transacción se realicen, o que no se realice ninguna. Una transacción en el sistema informático de los empleados de una empresa (por ejemplo) sería dar de alta a un empleado o eliminar un cargo. Una transacción un poco más complicada sería eliminar un Departamento o división y reasignar todos sus empleados a otro Departamento. En este caso hay que realizar varios cambios sobre la base de datos. Si la transacción falla durante su realización, por ejemplo porque falla el hardware (o se va la energía eléctrica) , la base de datos quedará en un estado inconsistente. Algunos de los cambios se habrán hecho y otros no, por lo tanto, los cambios realizados deberán ser deshechos para devolver la base de datos a un estado consistente.
- Un RDBMS debe proporcionar un mecanismo capaz de recuperar la base de datos en caso de que ocurra algún suceso que la dañe. Como se ha comentado antes, cuando el sistema falla en medio de una transacción, la base de datos se debe devolver a un estado consistente. Este fallo puede ser a causa de un fallo en algún dispositivo hardware o un error del software, que hagan que el RDBMS aborte, o puede ser a causa de que el usuario detecte un error durante la transacción y la aborte antes de que finalice. En todos estos casos, el RDBMS debe proporcionar un mecanismo capaz de recuperar la base de datos llevándola a un estado consistente.
- Un RDBMS debe proporcionar un mecanismo que garantice que sólo los usuarios autorizados pueden acceder a la base de datos. La protección debe ser contra accesos no autorizados, tanto intencionados como accidentales.
- Un RDBMS debe ser capaz de integrarse con algún software de comunicación. Muchos usuarios acceden a la base de datos desde terminales. En ocasiones estos terminales se encuentran conectados directamente a la máquina sobre la que funciona el RDBMS. En otras ocasiones los terminales están en lugares remotos, por lo que la comunicación con la máquina que alberga al RDBMS se debe hacer a través de una red. En cualquiera de los dos casos, el RDBMS recibe peticiones en forma de mensajes y responde de modo similar. Todas estas transmisiones de mensajes las maneja el gestor de comunicaciones de datos. Aunque este gestor no forma parte del RDBMS, es necesario que el RDBMS se pueda integrar con él para que el sistema sea comercialmente viable
- Un RDBMS debe proporcionar los medios necesarios para garantizar que tanto los datos de la base de datos, como los cambios que se realizan sobre estos datos, sigan ciertas reglas. La

integridad de la base de datos requiere la validez y consistencia de los datos almacenados. Se puede considerar como otro modo de proteger la base de datos, pero además de tener que ver con la seguridad, tiene otras implicaciones. La integridad se ocupa de la calidad de los datos. Normalmente se expresa mediante restricciones, que son una serie de reglas que la base de datos no puede violar. Por ejemplo, se puede establecer la restricción de que el número de cédula de un empleado no puede tener caracteres alfanuméricos, o por ejemplo que para dar de alta un empleado éste debe pertenecer obligatoriamente a un Departamento. En este caso sería deseable que el RDBMS controlara que no se violen esas reglas cada vez que se ingresen los datos de un empleado a la base de datos de la empresa.

## NORMALIZACIÓN

### INTRODUCCIÓN

Una base de datos tiene que ser diseñada antes de que pueda ser creada y usada. El diseño debe ajustarse a estándares que permitan ahorro de memoria, acceso rápido, fácil mantenimiento, portabilidad, facilidad de futuros mejoramientos, buen desempeño y eficiencia de costos, entre otros. El diseño lógico final de una base de datos debe ser tal que equilibre un desempeño óptimo junto con la integridad de la información. Esto puede ser logrado a través de un proceso conocido como Normalización. La base de datos debe estar en un estado de "Forma completamente normalizada".

### DEFINICIÓN DE NORMALIZACION

Normalización es una serie de reglas que involucra análisis y transformación de las estructuras de los datos en relaciones que exhiban propiedades únicas de consistencia, mínima redundancia y máxima estabilidad.

La necesidad para normalizar puede ser mejor comprendida al mencionar las distintas anomalías o desventajas de los datos NO NORMALIZADOS. Consideremos la tabla en la figura 3. La tabla contiene todos los detalles de los empleados de una compañía, y los detalles del Departamento al que pertenecen.

numEmp	nombre	salario	ingreso	numDept	descDept
1001	Andres	\$ 500.000	1-Ene-01	AB101	Sistemas
1002	Maria	\$ 700.000	16-Mar-02	AB101	Sistemas
1003	Carlos	\$ 350.000	5-Dic-01	AB101	Finanzas
1004	Felipe	\$ 800.000	15-Jun-01	AB103	Finanzas
1005	Oscar	\$ 500.000	1-Ene-03	AB102	Contabilidad
1006	Martha	\$ 700.000	5-Dic-01	AB104	Ventas
1007	Beatriz	\$ 800.000	1-Ene-01	AB110	Gerencia

FIGURA 3

A primera vista, parece conveniente almacenar todos los detalles en una sola tabla. Pero ciertas anomalías se pueden manifestar durante la inserción, actualización y borrado de datos. La normalización provee un método de remover todas estas indeseables anomalías haciendo la base de datos mas confiable y estable.

### **Anomalía de inserción (INSERT)**

Suponga que un nuevo Departamento ha sido creado, el cual no tiene empleados todavía, por lo tanto, en nuestra tabla original, los datos correspondientes al empleado estarían vacíos (nulos), y solo tendríamos la información del Departamento: Columnas "numDept" y "descDept".

### **Anomalía de Actualización (UPDATE)**

Suponga que el número del Departamento de "Sistemas" ha sido cambiado a AB108. Esto involucra tener que cambiar el numero del departamento para todos los empleados que pertenezcan al departamento de "Sistemas", lo cual representa tiempo y recursos de sistema adicionales.

### **Anomalía de borrado (DELETE)**

Si todos los empleados en el Departamento de "Finanzas" abandonan la compañía, todos los registros de estos tendrían que ser borrados. Hecho así, los detalles del Departamento "Finanzas" se perderían. Los datos en la tabla entonces no representan una información correcta sobre el estado de la compañía, y por lo tanto se pierde la integridad de los datos

## **PROPIEDADES DE UNA BASE DE DATOS DESPUÉS DE LA NORMALIZACION**

Una base de datos normalizada debe representar las siguientes propiedades:

- Los requerimientos para almacenamiento de datos se minimizan, dado que el proceso de normalización sistemáticamente elimina la duplicación de los datos.
- Desde que los datos son almacenados en el mínimo número de lugares, las posibilidades de inconsistencias en la información son reducidas al mínimo.
- Las estructuras normalizadas son óptimas para efectuar actualizaciones de los datos. Dado que los datos existen en el mínimo número de lugares, una operación de actualización (UPDATE) necesitará acceder a una mínima cantidad de datos.



## PROCEDIMIENTOS DE NORMALIZACION

El proceso de normalización involucra básicamente tres pasos. Después de cada paso, la base de datos se convierte en formas llamadas "formas normales". Generalmente, la "tercera forma normal" es el estado que debe alcanzar una base de datos para que se diga que está totalmente normalizada. La cuarta y la quinta forma normal también existen, pero no son usadas en el diseño de una base de datos.

A continuación, consideremos un pequeño ejercicio acerca de un Documento de Orden de Compra, el cual trataremos de convertirlo a una forma normalizada. Pero antes explicaremos unas pequeñas reglas:

### Propiedades de una relación

Un tabla debe satisfacer ciertos criterios previos antes de calificar para convertirse en una relación.

#### No duplicados

No debe haber nunca dos columnas o filas totalmente idénticas. Si dos filas son totalmente idénticas, entonces hacen falta algunos atributos que las haga diferentes y distinguibles. Ejemplo: Dos registros de discos compactos en una tienda serían idénticos si son dos copias del último álbum de Shakira, si no fuera porque cada disco compacto tiene un numero código que los hace diferentes.

#### Clave Única

Cada registro tiene que tener una llave única que lo identifique. Cualquier atributo puede ser una llave, pero en lo posible trataremos de elegir como llave única al atributo que tenga una longitud menor y fija, como por ejemplo un numero de ID. Si un atributo es insuficiente para identificar un registro de manera única, entonces más de un atributo puede conformar la llave única. En tal caso, el número de atributos que conformen una llave debe ser el mínimo necesario y suficiente.

#### Insignificancia del orden

La secuencia en la cual los atributos son escritos no debe importar. Podemos escribir el ID del empleado de primero, o el nombre y el apellido de primero, y esto no afectará las relaciones que establezcamos con otras tablas. Por otro lado, los registros deben ser totalmente independiente de su secuencia o posición en la base de datos (dependencia posicional). Esto significa que si intentamos identificar un registro por su posición dentro de la tabla, estaremos creando una llave inválida.



### Forma no-normalizada

Los datos, en su forma elemental, no están normalizados. Por lo tanto, lo primero con lo que debemos comenzar es con los datos elementales o básicos que conformarán el diccionario de datos. El diccionario de datos es creado a partir de los documentos o diagramas de flujo de la compañía. Se deben listar los elementos uno debajo del otro. Así, obtendremos la forma no-normalizada para el ejercicio de ARD (Análisis Relacional de Datos), con el cual deberemos obtener al final distintos grupos de elementos. Mas tarde, dichos grupos se combinarán con los grupos de otros documentos al cual también se les ha hecho el análisis ARD, y se establecerán relaciones entre ellos.

### Introducción a SQL.

Oracle fue la primera compañía que sacó al mercado un producto que usó el lenguaje estructurado de consulta basado en el idioma Inglés, o SQL. SQL le permite a los usuarios finales extraer información por ellos mismos, sin que tengan que usar un lenguaje de programación para cada pequeño reporte o informe. SQL, el cual se pronuncia en el medio informático como "siquel", o sencillamente en sus siglas "S Q L" es una herramienta increíblemente capaz. Usarlo no requiere experiencia previa en programación.

SQL es el lenguaje ANSI (American National Standards Institute) estándar de la industria, usado para manipular información en una base de datos relacional. Este lenguaje es muy simple, aún así poderoso, no procedimental, a diferencia de lenguajes como C o COBOL en los cuales se debe describir exactamente como acceder y manipular los datos. SQL especifica que hay que hacer. Internamente, el lenguaje determina como ejecutar las requisiciones que se le ordenen. Todas las operaciones ejecutadas en la base de datos son desarrolladas usando sentencias de SQL. Para ello, se usan dos tipos de comandos en SQL, los DDL (Data Definition Language - Lenguaje de Definición de Datos) que permiten crear y definir nuevas bases de datos, campos e índices. Y los DML (Data Manipulation Language - Lenguaje de Manipulación de Datos) que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos, al igual que actualizar o borrar los datos de una tabla.

- **Recuperar datos**
- **Actualizar, insertar y borrar datos**
- **Crear, modificar y borrar objetos de la base de datos**
- **Controlar acceso a la base de datos**
- **Garantizar la consistencia de la base de datos**
- **Monitorear la configuración y el desempeño de la base de datos.**

## CARACTERÍSTICAS DEL LENGUAJE ESTRUCTURADO DE CONSULTA - SQL

A manera de resumen, podemos enunciar las siguientes características de SQL:

- Es un lenguaje basado en el idioma Inglés que usa frases en Inglés para manipular la base de datos
- Es no procedimental. Usted especifica la información requerida, mas no las operaciones o la forma de llegar a la información. Cada RDBMS tiene un optimizador de búsquedas, el cual traduce sus cláusulas en SQL y elabora el camino o la ruta óptima para llegar a los datos.
- Cuando se trabaja con los datos, todas las filas afectadas por los comandos se tratan como un solo bloque, no son tratadas una por una
- Adicionalmente, SQL permite que los resultados de una consulta sean los datos de entrada para una nueva.

### COMPONENTES DEL SQL

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

### COMANDOS

Existen dos tipos de comandos SQL:

- Los DDL que permiten crear y definir nuevas bases de datos, campos e índices.
- Los DML que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

Comandos DDL	
Comando	Descripción
CREATE	Utilizado para crear nuevas tablas, campos e índices
DROP	Empleado para eliminar tablas e índices
ALTER	Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

Comandos DML	
Comando	Descripción
<b>SELECT</b>	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado
<b>INSERT</b>	Utilizado para cargar lotes de datos en la base de datos en una única operación.
<b>UPDATE</b>	Utilizado para modificar los valores de los campos y registros especificados
<b>DELETE</b>	Utilizado para eliminar registros de una tabla de una base de datos

## CLÁUSULAS

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

CLAUSULAS	
Cláusula	Descripción
<b>FROM</b>	Utilizada para especificar la tabla de la cual se van a seleccionar los registros
<b>WHERE</b>	Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar
<b>GROUP BY</b>	Utilizada para separar los registros seleccionados en grupos específicos
<b>HAVING</b>	Utilizada para expresar la condición que debe satisfacer cada grupo
<b>ORDER BY</b>	Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico

## TIPOS DE DATOS

Las tablas son los vehículos de almacenamiento dentro de una base de datos relacional. Las columnas de la tabla definen las características de los datos y a su vez le otorgan un nombre a los datos.

Las características de la columna, así como las variables y las constantes, están definidas por un tipo de datos y una longitud. El tipo de dato es un arreglo estático de propiedades con

un valor predefinido. Esas propiedades hacen que el sistema administre a un tipo de datos de manera diferente como lo hace con otro.

TIPOS DE DATO	
Tipo	Descripción
<b>CHAR</b>	Una columna CHAR debe tener un ancho fijo y puede contener cualquier carácter imprimible. Ancho fijo significa que si el dato ingresado es de una longitud menor a la definida, entonces el campo es rellenado con espacios. El máximo tamaño de una columna CHAR es de 2Kb (Kilobytes).
<b>DATE</b>	Una columna DATE puede contener una fecha y hora entre el rango del 1 de enero de 4712 A.C. hasta el 31 de Diciembre de 4712 D.C. El formato estandar de la fecha es DD-MMM-YYYY (01-FEB-2004)
<b>DECIMAL</b>	Los campos decimales pueden contener dígitos de 0 a 9, y opcionalmente un símbolo de negativo. No se admiten valores fraccionarios.
<b>FLOAT</b>	El tipo de datos FLOAT es el mismo tipo que NUMBER.
<b>NUMBER</b>	Una columna NUMBER puede contener un número, con o sin punto decimal y signo negativo, y puede tener desde 1 a 105 dígitos decimales de los cuales solo 38 son significativos.
<b>INTEGER</b>	Este tipo de dato es el mismo NUMBER, con la excepción que solo pueden ser especificados valores enteros.
<b>LONG</b>	El tipo de datos LONG puede contener caracteres hasta 2Gb de tamaño. Solo puede haber una columna LONG por tabla.
<b>RAW</b>	Las columnas RAW pueden almacenar binarios como imágenes gráficas. El máximo tamaño es de 2Kb.
<b>LONG RAW</b>	Es el mismo tipo de datos RAW pero con mayor tamaño.
<b>VARCHAR</b>	Una columna VARCHAR es de longitud variable, pero se debe especificar un ancho fijo. Si el dato ingresado es menor al ancho predefinido, la columna no es rellenada con espacios sino que se ajusta a la longitud del dato.
<b>BLOB</b>	Puede almacenar datos binarios hasta 4 Gb.

## OPERADORES Y CONDICIONES

Los operadores y las condiciones son usados para ejecutar operaciones, tales como la suma y la resta, o comparaciones entre los datos a través de una sentencia de SQL. Los operadores son representados por caracteres simples o por palabras reservadas.

Una condición es una expresión de varios operadores que al ser evaluada retornará VERDADERO, FALSO o DESCONOCIDO.

Los operadores y las condiciones son necesarios en cualquier lenguaje de computación. Ellos te permiten ejecutar cálculos aritméticos, comparaciones de datos, y una amplia variedad de manipulaciones de datos que son necesarios para soportar los requerimientos de las aplicaciones.

## OPERADORES MATEMATICOS

Los operadores aritméticos son usados en SQL para sumar, restar, multiplicar, dividir y negar valores. El resultado de una expresión que use operadores aritméticos es un valor numérico.

Operador	Uso
+, -	Denota una expresión positiva o negativa. Estos son operadores unitarios (Unitario quiere decir que solo operan sobre un operando, por ejemplo, para indicar que un número es negativo se usa el signo "-" delante del número).
*	Multiplica. Este es un operador Binario. (Binario quiere decir que se necesitan dos operandos).
/	Divide. Este es un operador Binario.
+	Suma. Este es un operador Binario.
-	Resta. Este es un operador Binario

## OPERADORES DE CARÁCTER

Las cadenas de caracteres son manipuladas por los operadores de carácter. El más común de los operadores de carácter es el operador de concatenación.

Operador	Uso
	Concatena cadenas de caracteres



El resultado de concatenar dos cadenas de carácter es otra cadena de carácter. Si ambas cadenas son del mismo tipo de datos, la cadena resultante será del mismo tipo de datos original.

## OPERADORES DE COMPARACIÓN

Los operadores de comparación son usados para comparar una expresión contra otra expresión. El resultado de la comparación debe conducir a un valor Falso, Verdadero o Desconocido.

Operador	Uso
<	Menor que
>	Mayor que
<>	Distinto de
<=	Menor ó Igual que
>=	Mayor ó Igual que
!=	No igual a
=	Igual que
BETWEEN X AND Y	Mayor o igual a X y menor igual a Y
LIKE	Retorna verdadero cuando la primera expresión corresponde el patrón de la segunda
IN	Igual a cualquier miembro de
NOT IN	No igual a cualquier miembro de
IS NULL	Es nulo
IS NOT NULL	No es nulo
ALL	Compara un valor dado con todos los valores de una lista dada
ANY, SOME	Compara un valor dado con cada valor de una lista dada
EXISTS	Retorna verdadero si la consulta devuelve por lo menos una fila

## OPERADORES LÓGICOS

Los operadores lógicos son usados para producir un resultado simple a partir de la combinación de dos condiciones por separado.

Operador	Uso
AND	Es el "y" lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR	Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.
NOT	Negación lógica. Devuelve el valor contrario de la expresión.

## OPERADORES DE ARREGLO

Los operadores de arreglo (SET OPERATORS) combinan el resultado de dos consultas en un único resultado simple. Las consultas que contengan operadores de arreglo son conocidas como consultas compuestas.

Operador	Uso
UNION	Retorna todas las filas únicas (distintas) de cada consulta. (sin duplicados)
UNION ALL	Retorna todas las filas de cada consulta, incluyendo las duplicadas.
INTERSECT	Retorna solo las filas que cumplen simultáneamente los criterios de ambas consultas.
MINUS	Retorna todas las filas distintas que están en la primera consulta y que asu vez no estén en la segunda consulta.

## CONSULTAS

### CONSULTA DE SELECCIÓN

Las consultas de selección se utilizan para indicar al motor de datos que devuelva información de las bases de datos, esta información es devuelta en forma de conjunto de registros que se pueden almacenar en un objeto RECORDSET. Este conjunto de registros es modificable.

El comando utilizado para este tipo de consulta es SELECT. Este comando recupera datos almacenados en las filas y columnas de las tablas de una base de datos a través de

procedimientos llamados consultas. Una consulta tiene tres partes principales: La cláusula SELECT, la cláusula FROM, y la cláusula WHERE.

Por lo tanto, la sintaxis básica de una consulta de selección es la siguiente:

**SELECT Campos FROM Tabla WHERE Campo2 = Criterio**

En donde :

**Campos** es la lista de campos que se deseen recuperar

**Tabla** es el origen de los mismos

**Campo2** es el campo que condiciona el resultado

**Criterio** es el valor que debe tener el **Campo2**.

Por ejemplo:

**SELECT Nombre, Apellido, Telefono FROM Clientes WHERE Ciudad = "Barranquilla";**

Esta consulta devuelve un RECORDSET con los campos Nombre, Apellido y Teléfono de la tabla Clientes, que sean de la ciudad de Barranquilla. Nótese que la cláusula WHERE puede omitirse. Si se omite, la consulta retornará todos los Nombres, Apellidos y Teléfonos de la tabla cliente, de todas las ciudades.

En general, podemos generalizar la sintaxis del comando SELECT como

**SELECT** Lista de campos

**FROM** Lista de Tablas

**WHERE** Condiciones de Búsqueda

ID	NOMBRE	APELLIDO	ID_CARGO	FECHA_CONTRATO
PMA42628M	Paolo	Accorti	13	1992-08-27 00:00:00.000
PSA89086M	Pedro	Afonso	14	1990-12-24 00:00:00.000
VPA30890F	Victoria	Ashworth	6	1990-09-13 00:00:00.000
H-B39728F	Helen	Bennett	12	1989-09-21 00:00:00.000
L-B31947F	Lesley	Brown	7	1991-02-13 00:00:00.000
F-C16315M	Francisco	Chang	4	1990-11-03 00:00:00.000
PTC11962M	Philip	Cramer	2	1989-11-11 00:00:00.000
A-C71970F	Aria	Cruz	10	1991-10-26 00:00:00.000
AMD15433F	Ann	Devon	3	1991-07-16 00:00:00.000
ARD36773F	Anabela	Domingues	8	1993-01-27 00:00:00.000
PHF38899M	Peter	Franken	10	1992-05-17 00:00:00.000
PXH22250M	Paul	Henriot	5	1993-08-19 00:00:00.000
CFH28514M	Carlos	Hernandez	5	1989-04-21 00:00:00.000
PDI47470M	Palle	Ibsen	7	1993-05-09 00:00:00.000
KJJ92907F	Karla	Jablonski	9	1994-03-11 00:00:00.000
KFJ64308F	Karin	Josephs	14	1992-10-17 00:00:00.000
MKG44605M	Matti	Karttunen	6	1994-05-01 00:00:00.000

## EJEMPLOS

Supongamos que tenemos una tabla llamada EMPLEADOS con la siguiente estructura, la cual contiene 5000 registros.

TABLA: EMPLEADOS

CE DULA	NO MBRE	APPE LLIDO	CO RREO	DIRE CCION	TELE FONO	CIU DAD	E DAD	SU ELDO
------------	------------	---------------	------------	---------------	--------------	------------	----------	------------

- Queremos hacer una consulta que nos devuelva solamente el nombre y el apellido de las personas de la ciudad de Bogotá. El comando con sus cláusulas quedaría de la siguiente manera:

```
SELECT NOMBRE, APELLIDO FROM EMPLEADOS WHERE CIUDAD = "Bogotá";
```

- Ahora, queremos hacer una consulta que nos de los mismos datos anteriores, pero que además nos muestre las personas de Barranquilla.

```
SELECT NOMBRE, APELLIDO FROM EMPLEADOS WHERE (CIUDAD = "Bogotá"  
OR CIUDAD = "Barranquilla");
```

- Ahora queremos las cédulas, los nombres, los apellidos y los teléfonos solamente de las personas que vivan en Medellín, y que tengan edad entre 20 y 30 años.

```
SELECT CEDULA, NOMBRE, APELLIDO, TELEFONO FROM EMPLEADOS WHERE  
(CIUDAD = "Medellín" AND EDAD > 20 AND EDAD < 30);
```

- Ahora queremos la misma información anterior, pero que aparezca ordenada por el apellido.

```
SELECT CEDULA, NOMBRE, APELLIDO, TELEFONO FROM EMPLEADOS WHERE  
(CIUDAD = "Medellín" AND EDAD > 20 AND EDAD < 30) ORDER BY APELLIDO;
```

Esta misma consulta se puede hacer usando el operador BETWEEN.

```
SELECT CEDULA, NOMBRE, APELLIDO, TELEFONO FROM EMPLEADOS WHERE  
(CIUDAD = "Medellín" AND EDAD BETWEEN 20 AND 30) ORDER BY APELLIDO;
```

- Ahora haremos una consulta más compleja. Queremos nombre, apellido y teléfono, de las personas residentes en Cali, Bucaramanga, Cúcuta y Pereira que sean mayores de 40 años, y que devenguen un sueldo superior a \$350.000, y que ordene la lista por el valor del sueldo.

```
SELECT NOMBRE, APELLIDO, TELEFONO FROM EMPLEADOS WHERE (CIUDAD  
IN ("Cali","Bucaramanga","Cúcuta","Pereira") AND EDAD > 40 AND SUELDO > 350000)  
ORDER BY SUELDO;
```

- Por ultimo, haremos un SELECT más fuerte. Queremos la información del punto anterior, pero esta vez nos debe mostrar los resultados agrupados por ciudades, respetando el ordenamiento por sueldo. Es decir, nos mostrara 4 grupos de resultados, uno por cada ciudad de la lista, y cada grupito estará ordenado por sueldo.

```
SELECT NOMBRE, APELLIDO, TELEFONO FROM EMPLEADOS WHERE (CIUDAD  
IN ("Cali","Bucaramanga","Cúcuta","Pereira") AND EDAD > 40 AND SUELDO > 350000)  
ORDER BY SUELDO GROUP BY CIUDAD;
```

Nótese que simplemente agregamos el operador **GROUP BY** . Y nótese además que no hay necesidad que CIUDAD este dentro de la lista de campos de SELECT para que el agrupamiento se haga.

Como pueden observar, SQL es un lenguaje extremadamente sencillo. En la próxima unidad veremos comandos DDL, que nos permitirán crear tablas, borrar tablas, y el resto de los comandos DML, que nos permitirán insertar, actualizar y borrar registros en una tabla.

## Manipulación de Datos.

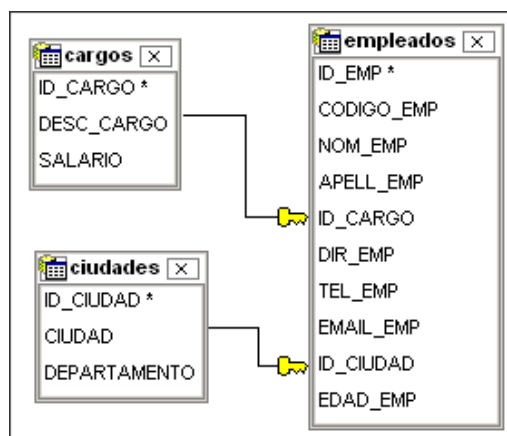
### Creación de Tablas.

Los comandos DDL (Data Definition Language) o de definición de datos en SQL controlan la creación, modificación y eliminación de objetos de la base de datos.

El primer paso para establecer una base de datos (una vez diseñada según los conceptos explicados en la unidad 1), es crear una o más tablas. Después de determinar el diseño de la



base de datos, se pueden crear las tablas que almacenarán los datos. Los datos suelen almacenarse en tablas permanentes.



Por lo tanto, procederemos a crear tablas usando el comando **CREATE TABLE**. La sintaxis general para este comando es:

```
CREATE TABLE 'Nombre_de_tabla' (  
  'Columna1' tipo_de_dato(tamaño) default <expresión> Restricción,  
  'Columna2' tipo_de_dato(tamaño) default <expresión> Restricción,  
  'Columna3' tipo_de_dato(tamaño) default <expresión> Restricción,  
  'Columna4' tipo_de_dato(tamaño) default <expresión> Restricción,  
  ....  
  RESTRICCIÓN DE LLAVE ('Nombre_columna'),  
  RESTRICCIÓN DE LLAVE ('Nombre_columna')  
)
```

**Vamos a explicar todos los componentes de esta sintaxis paso a paso:**

- El comando **CREATE TABLE** es el comando DDL, inmediatamente después de él se coloca el nombre de la Tabla que queremos crear. El nombre de la tabla debe ser un nombre legal, es decir, no debe llevar espacios, ni caracteres como \*/%#!. Se recomienda que el nombre de la tabla sea representativo del contenido (datos) que va a hospedar. Algunos ejemplos de nombre de tabla serían: 'TB\_EMPLEADOS', 'TB\_CATEGORÍAS', 'TB\_AREAS', 'TB\_DIVISIONES', 'TB\_PROVEEDORES', y así por el estilo. Debe tenerse en cuenta que no puede haber dos tablas con el mismo nombre dentro de una misma base de datos.
- Una vez definido el nombre de la tabla, procedemos a crear los campos o columnas de la tabla. A cada campo o columna debe definírsele un tipo de dato, tal como lo vimos en la unidad anterior. El primer elemento en la creación de un campo es el nombre. Aplican las mismas reglas de asignación de nombre que se usaron para la creación de tablas. Como se vió en la unidad pasada, debemos elegir nombres que sean representativos, y que ayuden

a identificar inequívocamente el dato que se almacena. Ejemplos de nombres de tabla pueden ser: 'ID\_EMPLEADO', 'CIUDAD', 'APELLIDO\_EMP', 'NOMBRE\_EMP', 'DIRECCION\_EMP', 'FECHA\_INI', 'FECHA\_NAC', 'TELEFONO', etcétera.

- El tipo de dato que va después de elegido el nombre de la columna, debe corresponder exactamente con el contenido que vayamos a alojar en esta. Así, si tenemos, por ejemplo, el campo NOMBRE\_EMP, el tipo de dato debe ser CHAR que es el que permite que se almacenen caracteres. Si, por ejemplo, el campo es ID\_EMPLEADO, el tipo de dato debe ser INTEGER, el cual permite el ingreso de datos numéricos. Si el campo es FECHA\_NAC, el tipo de dato será DATE, que permite almacenar fechas de calendario.
- Algunos tipos de dato exigen que se les indique un tamaño. Es el caso de CHAR o INTEGER. El tamaño quiere decir que los datos a ingresar no deben sobrepasar ese tamaño. Así para una variable CHAR que se le indique tamaño 20, solo almacenará datos que no sobrepasen 20 caracteres de largo. Para una variable INTEGER que se le indique tamaño 4, solo almacenará números que tengan máximo 4 unidades enteras (ej: 9999).
- Si por alguna razón el usuario intenta ingresar un dato en una tabla, el cual está por encima del tamaño definido en esta, el gestor de la base de datos automáticamente lo trunca, esto es, lo recorta hasta que se ajuste al tamaño definido en la tabla. Esto lo podemos ilustrar con un ejemplo. Digamos que hemos definido un campo tipo CHAR, con tamaño 5, que almacena APELLIDOS, e intentamos ingresar el apellido "RAMIREZ". Al haber definido un tamaño de solo 5 caracteres, el dato que finalmente quedará almacenado será "RAMIR". Por ello es importante que al momento del diseño y creación de la tabla, establezcamos unos tamaños lo suficientemente largos para que puedan almacenar los datos sin lugar a que ocurran truncamientos de la información.
- Para cada campo debemos definir una expresión que corresponda al valor por defecto. Valor por defecto es un concepto en el cual se crean valores predeterminados para los campos de un registro, cuando estos valores no son dados por el usuario. Supongamos que tenemos un campo llamado FECHA DE NACIMIENTO, y supongamos que el usuario no ingresa el dato de dicho campo al momento de alimentar la base de datos. El sistema inmediatamente acude al valor por defecto, que puede ser, por ejemplo 01/01/2000. Esto se hace con el fin de que el campo no quede nulo.
- Para cada campo o columna creados, debemos indicar restricciones en la sintaxis si deseamos que el campo admita valores nulos o no. El concepto de nulo se explicó en la unidad 1. Las restricciones en esta parte de la creación de campos puede ser NULL o NOT NULL. NULL significa que almacena valores nulos. NOT NULL significa que el campo no permite que se dejen valores nulos, es decir, el usuario tiene que ingresar necesariamente un dato.
- Al final de la sintaxis, cuando se han definido cada uno de las columnas (campos) de la tabla, se deben indicar cuales son las restricciones de llave. Esto es, cual es la columna que sirve de llave primaria, y cual es la columna que sirve de llave Foránea. En este paso del proceso también podemos indicar si deseamos que una columna tenga restricción de unicidad. Podemos decir entonces que una columna dada, puede tener alguna de estas tres restricciones de llave: PRIMARY KEY, FOREIGN KEY, o UNIQUE KEY. Cada vez que se define que una columna sea la llave primaria, automáticamente el sistema crea también una restricción UNIQUE para esa misma columna. Como recordaremos en la unidad anterior, las llaves primarias no permiten que haya dos registros con el mismo valor, es decir, no

admite duplicados. Por ello, debe haber una restricción de unicidad (UNIQUE KEY) para cada PRIMARY KEY que garantice que cada elemento de la columna que actué como llave primaria sea único e irrepetible.

- En algunos gestores de Bases de datos como MySQL, la sintaxis exige que para cada llave foránea (FOREIGN KEY) se cree un índice, es decir, se le informe al programa que el campo a ser convertido en llave foránea es de hecho una LLAVE. Esto se logra añadiendo la restricción de llave KEY, seguida del nombre de la columna. En el ejemplo que explicaremos mas adelante veremos esto de manera clara.
- En una tabla solo puede haber una llave primaria, que a la vez es UNIQUE. Sin embargo, puede haber varias columnas que también se les puede aplicar la restricción de unicidad. Es decir, a pesar que solo puede haber una sola llave primaria, si puede haber varias llaves únicas. Lo anterior puede ser entendido mejor en el siguiente ejemplo:

CEDULA_EMPLEADO	APELLIDO	NOMBRE	CODIGO_ACCESO
-----------------	----------	--------	---------------

- En el ejemplo podemos deducir que el campo **CEDULA\_EMPLEADO** es el que actúa como PRIMARY KEY, y por ello, también es un UNIQUE KEY. De esta manera no se admiten duplicados en este campo. No puede haber dos empleados con la misma cédula. Pero también tenemos el campo CODIGO\_ACCESO el cual no es una llave primaria, pero tampoco permite valores duplicados. Esto es, no puede haber dos empleados con el mismo código de acceso. Por ello, a la columna CODIGO\_ACCESO puede aplicársele la restricción de UNIQUE KEY también, para garantizar la unicidad.

### Ejemplos de creación de Tablas.

A continuación, procederemos a hacer un ejemplo práctico de creación de tablas utilizando las siguientes tres tablas como ejemplo

**TABLA EMPLEADOS**

ID_EMP	CODIGO_EMP	HOM_EMP	APELL_EMP	ID_CARGO	DIR_EMP
8550000	2505	ANDRES	PEREZ	10	CR 43 80-80
32454589	2506	JUAN	MARTINEZ	11	CR 50 12-25
72207855	2507	MARTHA	MORALES	50	CLLE 90 58-25
96556489	2508	SANDRA	CARRILLO	25	TRANSV 10 8-79



CONTINUACION  
TABLA

	TEL_EMP	EMAIL_EMP	ID_CIUADAD	EDAD_EMP
	35551010	<a href="mailto:aperez@empresa.com">aperez@empresa.com</a>	BAQ	30
	35554578	<a href="mailto:lmartinez@empresa.com">lmartinez@empresa.com</a>	CTG	41
	35558596	<a href="mailto:mmorales@empresa.com">mmorales@empresa.com</a>	BOG	26
9	35554782	<a href="mailto:scariillo@empresa.com">scariillo@empresa.com</a>	CAL	29

TABLA CARGOS

ID_CARGO	DESC_CARGO	SALARIO
10	GERENTE GENERAL	\$ 10.000
11	GERENTE COMERCIAL	\$ 8.000
15	GERENTE FINANCIERO	\$ 8.000
18	SUBGERENTE RECURSOS HUMANOS	\$ 7.000
20	DIRECTOR PLANEACION	\$ 5.000
25	DIRECTOR SUMNISTROS	\$ 5.000
30	JEFE CONTABILIDAD	\$ 3.000
50	AUXILIAR CONTABLE	\$ 2.000
51	SECRETARIA	\$ 1.000
52	TECNICO DE SISTEMAS	\$ 1.500

TABLA CIUDADES

ID_CIUADAD	CIUADAD	DEPARTAMENTO
BAQ	BARRANQUILLA	ATLANTICO
BOG	BOGOTA	CUNDINAMARCA
BUC	BUCARAMANGA	SANTANDER
CAL	CALI	VALLE DEL CAUCA
CTG	CARTAGENA	BOLIVAR
MED	MEDELLIN	ANTIOQUIA
STA	SANTA MARTA	MAGDALENA
POP	POPAYAN	CAUCA
CUC	CUCUTA	NORTE DE SANTANDER
RIO	RIOHACHA	GUAJIRA





## Análisis del diseño:

Tenemos tres tablas. Una es la tabla **EMPLEADOS**, otra es la tabla **CARGOS**, y la última es la tabla **CIUDADES**.

La tabla **EMPLEADOS** es la tabla principal de nuestra base de datos de ejemplo.

Contiene toda la información relevante del empleado, y tiene además dos campos **ID\_CARGO** y **ID\_CIUDAD** que indican los códigos de cargo y ciudad que se encuentran registrados en otras tablas.

Durante el diseño de la base de datos se consideró separar los datos de Cargo y Ciudad en tablas separadas, ya que contienen información adicional que era necesario que quedara identificada de acuerdo al proceso de **NORMALIZACION**.

Aquí es cuando podemos apreciar la utilidad del proceso de **NORMALIZACION**, el cual conduce a separar la información en distintas tablas para evitar la redundancia de los datos, manteniendo la información disponible a partir de relaciones entre tablas.

Es por ello que en la tabla **EMPLEADOS** tenemos un campo llamado **ID\_CARGO** el cual está relacionado con el campo **ID\_CARGO** de la tabla **CARGOS**. Podemos afirmar entonces que en la tabla **CARGOS** el campo **ID\_CARGO** es la llave primaria, y en la tabla **EMPLEADOS** el mismo campo **ID\_CARGO** es la llave **FORÁNEA**.

De igual manera, el campo **ID\_CIUDAD** de la tabla **CIUDADES** es la llave **PRIMARIA**, y en la tabla **EMPLEADOS** es la llave **FORÁNEA**.



Así las cosas, al tener la tabla **EMPLEADOS** dos llaves foráneas, no podremos ingresar la información de un empleado si antes no se le ha identificado una ciudad de origen y cargo que efectivamente **EXISTAN** en las tablas **CARGOS** y **CIUDADES** respectivamente.

La norma principal de las llaves foráneas es la siguiente: Todo dato que vaya a ser introducido como valor de un campo que actúe de llave foránea, debe existir previamente y corresponder a un registro de la llave primaria en la tabla a la cual se hace la referencia. Por ello es que se acostumbra poblar (llenar con información) primero las tablas que tengan relaciones de llaves foráneas con otra tabla principal, y por último se pobla la tabla principal.

En nuestro ejemplo, debemos entonces poblar primero las tablas **CARGOS** y **CIUDADES**, antes de intentar poblar la tabla **EMPLEADOS**.

Para entender mejor esta norma, pongamos el ejemplo de la llave foránea **ID\_CIUDAD** en la tabla **EMPLEADOS**. Digamos que queremos insertar un registro de un empleado con ciudad de origen "Montería". Al momento de inserción en la tabla **EMPLEADOS** nos encontramos con la restricción de que la ciudad de "Montería" debe tener un ID. Pero resulta que en la tabla **CIUDADES** no aparece ningún ID asociado con la ciudad de Montería. Por lo tanto, al intentar ingresar un ID de ciudad inválido el sistema nos dará un mensaje de error.

Por ello concluimos: Todo empleado debe tener un **ID\_CIUDAD** que esté previamente registrado en la tabla de **CIUDADES**, y debe tener un **ID\_CARGO** que esté previamente





registrado en la tabla de **CARGOS**, todo gracias a que **ID\_CARGO** y **ID\_CIUADAD** son llaves foráneas en la tabla **EMPLEADOS**.

### Sintaxis del ejemplo de creación de Tablas.

Teniendo claro el análisis de la base de datos de ejemplo propuesta, y habiendo estudiado la sintaxis de creación de tablas, procedemos a crear nuestras tablas.

El orden que usaremos es:

**CREAR TABLA CIUDADES**

**CREAR TABLA CARGOS**

**CREAR TABLA EMPLEADOS**

En la sintaxis que veremos a continuación, me he tomado el trabajo de elegir los tipos de dato, tamaños y valores por defecto usando simplemente el sentido común. En la práctica (vida real) el diseñador tiene que escuchar las requisiciones y recomendaciones del cliente que ordena crear la base de datos, antes de aventurarse a elegir tipos, tamaños y valores por defecto de manera arbitraria.

```
CREATE TABLE `ciudades` (  
  `ID_CIUADAD` char(3) NOT NULL default '',  
  `CIUDAD` char(20) NOT NULL default 'SIN DEFINIR',  
  `DEPARTAMENTO` char(20) NOT NULL default 'SIN DEFINIR',  
  PRIMARY KEY (`ID_CIUADAD`),  
  UNIQUE KEY `ID_CIUADAD` (`ID_CIUADAD`)  
)
```

```
CREATE TABLE `cargos` (  
  `ID_CARGO` int(3) NOT NULL default '0',  
  `DESC_CARGO` char(30) NOT NULL default 'SIN DEFINIR',  
  `SALARIO` int(8) NOT NULL default '1000',  
  PRIMARY KEY (`ID_CARGO`),  
  UNIQUE KEY `ID_CARGO` (`ID_CARGO`)  
)
```

```
CREATE TABLE `empleados` (  
  `ID_EMP` int(12) NOT NULL default '0',  
  `CODIGO_EMP` int(4) NOT NULL default '0',  
  `NOM_EMP` char(20) NOT NULL default 'NOMBRE',  
  `APELL_EMP` char(20) NOT NULL default 'NOMBRE',  
  `ID_CARGO` int(3) NOT NULL default '0',  
  `DIR_EMP` char(50) NOT NULL default 'DIRECCION',  
  `TEL_EMP` char(20) NOT NULL default 'TELEFONO',  
  `EMAIL_EMP` char(50) NOT NULL default 'CORREO',  
  `ID_CIUADAD` char(3) NOT NULL default '',  
  `EDAD_EMP` int(2) NOT NULL default '99',  
  PRIMARY KEY (`ID_EMP`),  
  UNIQUE KEY `ID_EMP` (`ID_EMP`),  
  UNIQUE KEY `CODIGO_EMP` (`CODIGO_EMP`),  
  KEY `FORANEA1` (`ID_CARGO`),  
  KEY `FORANEA2` (`ID_CIUADAD`),  
  FOREIGN KEY (`ID_CIUADAD`) REFERENCES `ciudades` (`ID_CIUADAD`),  
  FOREIGN KEY (`ID_CARGO`) REFERENCES `cargos` (`ID_CARGO`)  
)
```

### Observaciones Finales:

- Nótese como en la tabla empleados, antes de definir las llaves foráneas, tenemos la necesidad de especificar en la sintaxis que dichas columnas son llaves. Es por ello que se indican dos líneas con la restricción KEY, la cual le informa al gestor que esas columnas servirán como llaves. Luego, en las dos líneas subsiguientes, a esas llaves se les da la categoría de **FOREIGN KEY** y se hacen las referencias respectivas con las tablas foráneas.
- Nótese que en la tabla empleados hemos definido dos restricciones de unicidad **UNIQUE KEY**. Una es para la llave primaria, la cual como se explico previamente es de carácter obligatorio. Y la segunda llave **UNIQUE** es para el campo **CODIGO\_EMP**, ya que desde el diseño se ha establecido que no puede haber dos empleados con el mismo código. Esta restricción nace por lo tanto del diseño de la tabla, mas no de la sintaxis como tal, como sí ocurre con la llave primaria.
- Para que una llave foránea funcione correctamente, ambos campos que se están referenciando deben **OBLIGATORIAMENTE** tener el mismo tipo de dato y el mismo tamaño. Es decir, cuando definimos **ID\_CARGO** y **ID\_CIUADAD** en las tablas **CARGOS** y **CIUDADES** respectivamente, debemos tener la precaución que durante la creación de la tabla **EMPLEADOS**, las propiedades de **ID\_CARGO** y **ID\_CIUADAD** en esta tabla sean exactamente iguales a las propiedades de sus correspondientes en las tablas foráneas.

### Alterar Tablas.

Una vez que las tablas han sido creadas, puede existir la necesidad de cambiar el formato de una tabla existente. El comando **ALTER TABLE** junto con los comandos auxiliares

**MODIFY** y **ADD**, puede ser usado para modificar el tamaño máximo de una columna de tipo **NUMBER** o **CHAR**, o para agregar una nueva columna, o para habilitar o deshabilitar restricciones de columna.

La sintaxis general es:

```
ALTER TABLE <Nombre de la tabla>  
[MODIFY(<Nueva definición de la columna>)]  
[ADD (<Definición de la columna>)]
```

Los símbolos de corchete [ .. ] no van en la sintaxis. Simplemente indican las opciones entre las varias disponibles. Una orden de **ALTER TABLE** puede llevar un comando de **MODIFY** solamente, o puede ir un comando de **ADD** solamente. La habilitación o deshabilitación de restricciones también es opcional.

Para ver varios ejemplos práctico, volvamos a las tablas creadas en el capítulo anterior.

1. Supongamos que en la tabla **EMPLEADOS** quiero reducir el tamaño del campo de Dirección de empleado de 50 que tenía originalmente, a 40 caracteres.

La orden quedaría, por lo tanto así:

```
ALTER TABLE EMPLEADOS MODIFY ( DIR_EMP CHAR (40));
```

Para usar el comando **MODIFY** adecuadamente, existe una regla que siempre debemos respetar: La columna de vayamos a alterar debe estar vacía para poder reducir su tamaño. Si tratamos de reducir el tamaño de una columna que ya contiene datos en ella, el sistema generará un error y no permitirá el cambio.

Ahora supongamos que quiero alterar la restricción **NOT NULL** del mismo campo **DIR\_EMP** que vimos en el ejemplo anterior, para permitir que haya valores nulos en dicho campo.

La orden quedaría, por lo tanto así:

```
ALTER TABLE EMPLEADOS MODIFY ( DIR_EMP CHAR (40) NULL );
```

Para agregar una nueva columna, usamos el comando **ADD**. Sin embargo, para agregar una columna con la restricción **NOT NULL**, y la tabla ya contiene registros, el sistema nos dará un error. Esto sucede porque al haber registros existentes, la creación de una nueva columna implica que cada uno de los registros existentes deberá contener un valor para la nueva columna, y como estamos indicando que no puede haber valores nulos, el sistema no tiene de donde inventárselos. Entonces, tenemos dos caminos:

1. Al momento de crear la nueva columna indicamos un valor por defecto. (**DEFAULT**) o,
2. Creamos la columna como **NULL**, es decir, que permita valores nulos, luego llenamos los datos de cada uno de los registros para esta nueva columna, y por último, una vez la columna tenga datos, alteramos la columna para convertirla a **NOT NULL**.

Veamos la opción "b" en un ejemplo.

Asumamos que la tabla **CIUDADES** tiene 10 registros. Y supongamos que queremos alterar la tabla para agregar una nueva columna de código postal, que se llame **COD\_POSTAL**, que va a ser de tipo **INTEGER**, con tamaño 5, que no acepte nulos, y que no tenga ningún valor por defecto.

Como habíamos explicado, al intentar crear la columna con la restricción **NOT NULL**, como nos fue solicitado, el sistema dará error, ya que el sistema se ve obligado a que para cada uno de los 10 registros existentes haya necesariamente un valor de Código Postal, y no tiene como inventárselos. Por ello, la restricción **NOT NULL** no puedo usarla durante la creación de la columna. (pero si después de que haya datos en ella).

Procedamos:

```
ALTER TABLE CIUDADES ADD ( COD_ POSTAL INT (5) NULL );
```

Al ser creada esta nueva columna, cada uno de los 10 registros existentes quedo con valor **NULL** en esta columna.

Una vez creada, procedemos a llenar los datos correspondientes a esta columna en cada uno de los 10 registros que tenía la tabla.

Luego, dado que la columna **COD\_POSTAL** ya tiene información, estamos habilitados para convertir dicha columna en **NOT NULL**, lo cual aplicaría para los registros que se inserten de aquí en adelante.

Procedamos:

```
ALTER TABLE CIUDADES MODIFY ( COD_ POSTAL INT (5) NOT NULL );
```

Resumen:

- Podemos agregar una columna en cualquier momento, siempre y cuando NO contenga la restricción **NOT NULL**.
- Si deseamos agregar una columna en la cual queramos que haya una restricción **NOT NULL**, seguiremos los siguientes pasos:
  - ✓ Añadimos la columna permitiendo nulos (**NULL**)
  - ✓ Llenamos con datos la columna creada en cada uno de los registros existentes.
  - ✓ Modificamos la columna creada para que sea **NOT NULL**.
- Podemos cambiar el tipo de datos de una columna.
- Podemos reducir el tamaño de una columna tipo **CHAR**, siempre y cuando la tabla esté vacía.
- Podemos reducir el numero de posiciones decimales de una columna tipo **NUMBER** en cualquier momento, ya que esto no altera el numero como tal. Simplemente formatea la salida del número a la cantidad de posiciones decimales que hayamos especificado. Por ejemplo: En vez de mostrar 3.141695489555, podría mostrar 3.14.

## Eliminar Tablas

La eliminación de tablas es una orden poco común en la administración de una base de datos, ya que ello implica que se pierdan todos los datos contenidos en la tabla.

Además, si una tabla tiene referencias foráneas de otras tablas, inmediatamente se produciría un error en todas las tablas que dependan de la que estoy intentando borrar.

Una explicación clara de esto la podemos deducir usando nuestras tablas de ejemplo de los capítulos anteriores. Como recordaremos, las tablas **CARGOS** y **CIUDADES** son las que contienen los datos de **ID\_CARGO** y **ID\_CIUDAD** que se usan como referencia en la tabla **EMPLEADOS**. Esto es, la tabla empleados es dependiente de las tablas **CARGOS** y **CIUDADES**, ya que posee dos llaves foráneas que apuntan a ellas.

Por lo tanto, si elimino las tablas **CARGOS** y **CIUDADES**, la tabla **EMPLEADOS** quedará huérfana en sus llaves foráneas.

En la práctica lo que se acostumbra en caso que la eliminación de una tabla sea estrictamente necesaria, es eliminar primero las restricciones **FOREIGN KEY** que dependan de las tablas que vayamos a eliminar. De esta manera los datos que originalmente hubieren en la tabla dependiente, permanecerán inalterados, y al ocurrir la eliminación de las otras, no ocurrirá ningún error.

*Nota: La eliminación de una restricción de llave **FOREIGN KEY**, se hace a través de cláusulas avanzadas de **ALTER TABLE**, que pueden ser estudiadas a profundidad en los manuales de referencia de los gestores de **SQL** que tengamos instalados en nuestra máquina.*



En todo caso, si deseo eliminar una tabla, tenemos a disposición el comando DROP TABLE, y su uso es muy sencillo.

La sintaxis general de este comando es:

**DROP TABLE** <nombre de la tabla>

En resumen:

No se puede utilizar **DROP TABLE** para quitar una tabla a la que se haga referencia con una restricción **FOREIGN KEY**. Primero se debe quitar la restricción **FOREIGN KEY** o la tabla de referencia.

ID	NOMBRE	APELLIDO	ID_CARGO	FECHA_CONTRATO
PMA42628M	Paolo	Accorti	13	1992-08-27 00:00:00.000
PSA89086M	Pedro	Afonso	14	1990-12-24 00:00:00.000
VPA30890F	Victoria	Ashworth	6	1990-09-13 00:00:00.000
H-B39728F	Helen	Bennett	12	1989-09-21 00:00:00.000
L-B31947F	Lesley	Brown	7	1991-02-13 00:00:00.000
F-C16315M	Francisco	Chang	4	1990-11-03 00:00:00.000
PTC11962M	Philip	Cramer	2	1989-11-11 00:00:00.000
A-C71970F	Aria	Cruz	10	1991-10-26 00:00:00.000
AMD15433F	Ann	Devon	3	1991-07-16 00:00:00.000
ARD36773F	Anabela	Domingues	8	1993-01-27 00:00:00.000
PHF38899M	Peter	Franken	10	1992-05-17 00:00:00.000
PXH22250M	Paul	Henriot	5	1993-08-19 00:00:00.000
CFH28514M	Carlos	Hernandez	5	1989-04-21 00:00:00.000
PDI47470M	Palle	Ibsen	7	1993-05-09 00:00:00.000
KJJ92907F	Karla	Jablonski	9	1994-03-11 00:00:00.000
KFJ64308F	Karin	Josephs	14	1992-10-17 00:00:00.000
MGK44605M	Matti	Karttunen	6	1994-05-01 00:00:00.000

Como se había explicado previamente, existen tres comandos que caen dentro de la categoría de DML (Data Manipulation Language). Estos son INSERT, UPDATE y DELETE. Cada uno de estos tres comandos es usado para insertar, cambiar o borrar registros en las tablas que hayamos creado dentro de nuestra base de datos.

Nos ocuparemos del comando INSERT, que nos permite insertar nuevos registros en una tabla.

La sintaxis general de INSERT es la siguiente:

```
INSERT INTO <Nombre de la tabla> [<Lista de columnas>]  
VALUES (<Lista de valores>);
```

### Explicación de la Sintaxis:

- Después del comando INSERT INTO debemos indicar el nombre de la tabla. Esta indicación es de carácter obligatoria.
- Luego, de manera opcional (por eso aparece entre corchetes) le indicamos al comando la lista de las columnas sobre las cuales deseamos insertar datos. En este paso hay que hacer la siguiente observación. Si nos abstenemos de usar la lista de las columnas, el gestor SQL entenderá que vamos a ingresar datos para todas y cada una de las columnas, en el estricto orden en el que están definidas en la tabla, los cuales deben estar definidos en la sección "Lista de valores" dentro de la cláusula VALUES. Si por alguna queremos insertar registros definiendo solo valores para algunas columnas y no todas, (o usando un orden distinto) deberemos especificar cuales columnas estamos manipulando, indicándolas en "Lista de columnas".
- Si vamos a introducir valores para todos y cada una de las columnas de una tabla, entonces no es necesario usar la sección "lista de columnas"
- Si usamos una "lista de columnas", el orden que usemos deberá ser respetado tal cual en la "lista de valores". Es decir, si la lista de columnas contiene "ColA, ColB, ColC", la lista de valores deberá seguir el mismo orden "ValorColA, ValorColB, ValorColC".
- Si usamos una lista de columnas para la inserción de un nuevo registro, necesariamente las columnas que no estén dentro de la lista deben tener especificado un valor DEFAULT o un valor NULL. De lo contrario el sistema nos generará un error, ya que para campos que desde el diseño de la tabla tengan la restricción NOT NULL, es obligatorio que ingresemos un valor, es decir, es obligatorio que estos campos NOT NULL estén dentro de la "lista de columnas" durante la inserción de un nuevo registro.
- Cuando insertamos un nuevo registro en una tabla, y no especificamos el valor para una columna determinada, es decir, la columna no está incluida en la "lista de columnas" en el comando INSERT, el sistema automáticamente asignará el valor DEFAULT que hayamos definido durante el diseño de la tabla. Si no existiere un DEFAULT, el valor se asignará NULL (desde que no haya restricción).
- Los valores de la "lista de columna" y "lista de valores" deben estar separados por coma
- En la sección VALUES, si se trata de valores de carácter, estos deben ir encerrados entre comillas sencillas. Si se trata de valores numéricos, no es necesario el uso de comillas.

## Ejemplos de inserción de datos en Tablas.

Veamos algunos ejemplos para entender bien el uso del comando INSERT.

Recordemos la estructura de la tabla CARGOS.

```
CREATE TABLE `cargos` (  
  `ID_CARGO` int(3) NOT NULL default '0',  
  `DESC_CARGO` char(30) NOT NULL default 'SIN DEFINIR',  
  `SALARIO` int(8) NOT NULL default '1000',  
  PRIMARY KEY (`ID_CARGO`),  
  UNIQUE KEY `ID_CARGO` (`ID_CARGO`)  
)
```

Insertar todos los valores de un nuevo registro en la tabla CARGOS:

```
INSERT INTO CARGOS VALUES (56,`MENSAJERO`,1000);
```

Este comando es exactamente lo mismo que haber ordenado:

```
INSERT INTO CARGOS (ID_CARGO,DESC_CARGO,SALARIO) VALUES  
(56,`MENSAJERO`,1000);
```

Pero como habíamos explicado, si estoy insertando valores para todos y cada uno de los campos de la tabla, en el estricto orden en que fueron definidos en la tabla, puedo abstenerme de usar la lista de columnas.

El resultado de cualquiera de los comandos enunciados es:

ID_CARGO	DESC_CARGO	SALARIO
56	MENSAJERO	\$ 1.000

Insertar un nuevo registro indicando solo algunas columnas

```
INSERT INTO CARGOS (ID_CARGO,DESC_CARGO) VALUES (65,`INSPECTOR DE  
SEGURIDAD`);
```

Resultado:

ID_CARGO	DESC_CARGO	SALARIO
65	INSPECTOR DE SEGURIDAD	\$ 1.000

Nótese que el valor de SALARIO no fue especificado en el comando. Por ello, el gestor de la base de datos acude al valor pro defecto que había sido definido durante la creación de la tabla, el cual corresponde a un valor de \$1000.

Otro ejemplo:

**INSERT INTO CARGOS (ID\_CARGO) VALUES (65);**

ID_CARGO	DESC_CARGO	SALARIO
65	SIN DEFINIR	\$ 1.000

Al igual que el caso anterior, para los campos DESC\_CARGO y SALARIO acude a los valores por defecto, dado que en el comando de inserción no fueron definidos.

Inserción de datos que registros cuando una tabla esta referenciada a otra

Consideremos nuevamente los valores de las tablas CARGOS y CIUDADES:

**TABLA CARGOS**

ID_CARGO	DESC_CARGO	SALARIO
10	GERENTE GENERAL	\$ 10.000
11	GERENTE COMERCIAL	\$ 8.000
15	GERENTE FINANCIERO	\$ 8.000
18	SUBGERENTE RECURSOS HUMANOS	\$ 7.000
20	DIRECTOR PLANEACION	\$ 5.000
25	DIRECTOR SUMNISTROS	\$ 5.000
30	JEFE CONTABILIDAD	\$ 3.000
50	AUXILIAR CONTABLE	\$ 2.000
51	SECRETARIA	\$ 1.000
52	TECNICO DE SISTEMAS	\$ 1.500

**TABLA CIUDADES**

ID_CIUADAD	CIUDAD	DEPARTAMENTO
BAQ	BARRANQUILLA	ATLANTICO
BOG	BOGOTA	CUNDINAMARCA
BUC	BUCARAMANGA	SANTANDER
CAL	CALI	VALLE DEL CAUCA
CTG	CARTAGENA	BOLIVAR
MED	MEDELLIN	ANTIOQUIA
STA	SANTA MARTA	MAGDALENA
POP	POPAYAN	CAUCA
CUC	CUCUTA	NORTE DE SANTANDER
RIO	RIOHACHA	GUAJIRA

de la tabla EMPLEADOS:

```
CREATE TABLE `empleados` (
  `ID_EMP` int(12) NOT NULL default '0',
  `CODIGO_EMP` int(4) NOT NULL default '0',
  `NOM_EMP` char(20) NOT NULL default 'NOMBRE',
  `APELL_EMP` char(20) NOT NULL default 'NOMBRE',
  `ID_CARGO` int(3) NOT NULL default '0',
  `DIR_EMP` char(50) NOT NULL default 'DIRECCION',
  `TEL_EMP` char(20) NOT NULL default 'TELEFONO',
  `EMAIL_EMP` char(50) NOT NULL default 'CORREO',
  `ID_CIUADAD` char(3) NOT NULL default '',
  `EDAD_EMP` int(2) NOT NULL default '99',
  PRIMARY KEY (`ID_EMP`),
  UNIQUE KEY `ID_EMP` (`ID_EMP`),
  UNIQUE KEY `CODIGO_EMP` (`CODIGO_EMP`),
  KEY `FORANEA1` (`ID_CARGO`),
  KEY `FORANEA2` (`ID_CIUADAD`),
  FOREIGN KEY (`ID_CIUADAD`) REFERENCES `ciudades` (`ID_CIUADAD`),
  FOREIGN KEY (`ID_CARGO`) REFERENCES `cargos` (`ID_CARGO`)
)
```

Vemos varios ejemplos de inserción en la tabla empleados

**INSERT INTO EMPLEADOS VALUES** (8550000,2505,`ANDRES`,`PEREZ`,10,`CR 43 80-80`,`35551010`,`aperez@empresa.com`,`BAQ`,30)

Resultado:

I D_EMP	CODI GO_EMP	NO M_EMP	APE LL_EMP	ID_ CARGO	DI R_EMP	TE L_EMP	EMA IL_EMP	ID_ CIUADAD	ED AD_EMP
8550000	2505	ANDRES	PEREZ	10	CR 43 80-80	35551010	<a href="mailto:aperez@empresa.com">aperez@empresa.com</a>	BAQ	30

**INSERT INTO EMPLEADOS VALUES** (565656,2800,`CAMILO`,`ROMAN`,20,`CR 80 10-95`,`35558989`,`croman@empresa.com`,`NEI`,30)

Resultado:

**Error: El valor "NEI" no existe como referencia de llave foránea en la tabla CIUDADES.**

En casos como este es cuando vemos la utilidad de las restricciones impuestas en el diseño de la tabla.

El usuario estaba intentando ingresar un valor de código de ciudad que era inválido, dado que no existía previamente en la tabla CIUDADES, y por lo tanto estaba violando la restricción de FOREIGN KEY para dicho campo.





### Un último ejemplo:

```
INSERT INTO EMPLEADOS ( ID_EMP,CODIGO_EMP,ID_CARGO,ID_CIU  
VALUES (68686,5200,48,`BOG`);
```

### Resultado:

ID_E MP	CODIGO_ EMP	NOM_E MP	APELL_ EMP	ID_CAR GO	DIR_E MP	TEL_E MP	EMAIL_ EMP	ID_CIU DAD	EDAD_ EMP
68686 86	5200	NOMB RE	NOMBR E	48	DIRECC ION	TELEFO NO	<a href="#">CORREO</a>	BOG	99

Nótese que en el comando **INSERT** solo ingresamos los valores que eran estrictamente obligatorios, como aquellos de las llaves primarias, llaves unicas y llaves foráneas. El resto de valores los dejamos sin definir, para que el gestor usara los valores por defecto definidos en el diseño de la tabla.

### Modificar los datos de una Tabla.

Hemos visto como insertar datos de nuevos registros en una tabla, pero muchas veces es importante modificarlos. De hecho, el objetivo de una base de datos no es simplemente almacenar datos de forma estática, sino habilitar las herramientas para la actualización de los registros. Esto se logra con los comandos **UPDATE**, que en español significa "Actualizar", y **DELETE**, que en español significa "Borrar" o "Eliminar".

Ejemplos típicos de cuando es necesario usar estos comandos que modifican registros de las tablas de una base de datos, pueden ser los siguientes:

- Cambiar el valor del salario de un cargo, por decisión de la Junta Directiva de una empresa, o por efectos del crecimiento de la inflación, o por acuerdo con los trabajadores en una convención de trabajo.
- Eliminar el registro de un empleado cuando este ha dejado la compañía.
- Cambiar la dirección o teléfono de un empleado. (algo muy común).
- Cambiar el nombre de un departamento de la empresa cuando haya una reestructuración.
- Eliminar un departamento de la empresa, cuando por reestructuraciones dejen de existir.

Ejemplos hay muchos, y cada base de datos nos ira presentando las distintas necesidades, para las cuales debemos estar preparados.

La sintaxis de **UPDATE** es la siguiente:

**UPDATE** <Nombre de Tabla>

**SET** <Nombre de columna a cambiar> = <Nuevo valor>

**WHERE** (<Nombre de Columna que condiciona el cambio> = <condición>)

### Explicación de la sintaxis:

- Toda sentencia UPDATE debe obligatoriamente tener una cláusula WHERE, la cual le indica al gestor cuales son las condiciones de búsqueda de los registros que se van a cambiar. El equivalente en lenguaje Castellano, en un ejemplo real sería: "Cambie el valor del NOMBRE donde la CEDULA sea 50505050".
- Si no existiera una cláusula de WHERE, el comando UPDATE no tendría elementos para hacer ningún cambio. Por lo tanto, WHERE actúa como condicionante de una búsqueda en los registros. Todos los registros resultantes de la búsqueda condicionada con WHERE, son el objetivo del cambio, es decir, son el objetivo de la acción de UPDATE
- SET es la cláusula que indica cual va a ser el nuevo valor para la columna que deseamos cambiar. Si en una misma acción deseamos cambiar el valor de varias columnas, separamos varias cláusulas SET con comas, como se verá en los ejemplos mas adelante.

### Ejemplos de UPDATE :

**TABLA CARGOS**

ID_CARGO	DESC_CARGO	SALARIO
10	GERENTE GENERAL	\$ 10.000
11	GERENTE COMERCIAL	\$ 8.000
15	GERENTE FINANCIERO	\$ 8.000
18	SUBGERENTE RECURSOS HUMANOS	\$ 7.000
20	DIRECTOR PLANEACION	\$ 5.000
25	DIRECTOR SUMNISTROS	\$ 5.000
30	JEFE CONTABILIDAD	\$ 3.000
50	AUXILIAR CONTABLE	\$ 2.000
51	SECRETARIA	\$ 1.000
52	TECNICO DE SISTEMAS	\$ 1.500

Teniendo como ejemplo la tabla CARGOS, supongamos que queremos cambiar el salario de todos los empleados cuyos códigos estén entre 20 y 30, y ponerle un valor único de \$2500.

**UPDATE CARGOS SET (SALARIO =2500) WHERE ( ID\_CARGO >20 AND ID\_CARGO <30)**

Dada la cédula del empleado ANDRES PEREZ, cambiar el teléfono de su residencia por 3598978.

**TABLA EMPLEADOS**

ID_EMP	CODIGO_EMP	NOM_EMP	APELL_EMP	ID_CARGO	DIR_EMP
8550000	2505	ANDRES	PEREZ	10	CR 43 80-80
32454589	2506	JUAN	MARTINEZ	11	CR 50 12-25
72207855	2507	MARTHA	MORALES	50	CLLE 90 58-25
96556489	2508	SANDRA	CARRILLO	25	TRANSV 10 8-79

CONTINUACION  
TABLA

TEL_EMP	EMAIL_EMP	ID_CIUADAD	EDAD_EMP
35551010	<a href="mailto:aperez@empresa.com">aperez@empresa.com</a>	BAQ	30
35554578	<a href="mailto:jmartinez@empresa.com">jmartinez@empresa.com</a>	CTG	41
35558596	<a href="mailto:mmorales@empresa.com">mmorales@empresa.com</a>	BOG	26
9 35554782	<a href="mailto:scarrillo@empresa.com">scarrillo@empresa.com</a>	CAL	29

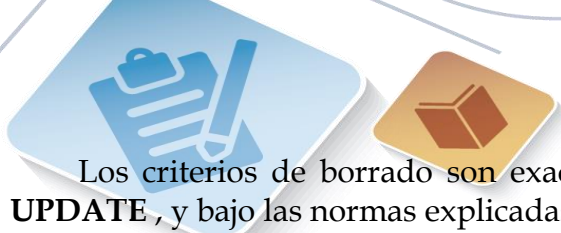
**UPDATE EMPLEADOS SET ( TELEFONO =`3598958`) WHERE ( ID\_EMP =8550000**

### Borrado de Datos.

De la misma manera como se actualizan registros, estos pueden ser igualmente borrados, usando el comando **DELETE** . La sintaxis de **DELETE** es muy parecida a la de **UPDATE** , en el sentido que también usa la cláusula **WHERE** para encontrar a través e una búsqueda los registros que cumplen con los criterios de borrado.

La sintaxis de **DELETE** es como sigue:

**DELETE FROM** <Nombre de la tabla>  
**WHERE** <Criterios de Borrado>




Los criterios de borrado son exactamente de la forma como se usan en un comando **UPDATE**, y bajo las normas explicadas en la unidad 2.

La única restricción a la hora de borrar registros tiene que ver con la existencia de restricciones impuestas a través de las FOREIGN KEY. Esto es, si intentamos borrar un registro de una tabla que este referenciada a otra por medio de una llave foránea, el gestor de la base de datos no permitirá que borremos el registro.

Por lo tanto: Para que un registro pueda ser borrado sin inconvenientes, ninguna tabla debe estar haciendo referencia a él.

Por ejemplo, si en la tabla CIUDADES intentamos borrar el registro de la ciudad de Bogotá, cuya llave primaria es "BOG", el sistema nos dará un error puesto que existen varios empleados de la tabla EMPLEADOS que están usando el valor del campo ID\_CIUDAD como Bogotá, y se violaría la integridad referencial.

En cambio, en la tabla EMPLEADOS, al no ser ella misma referencia de ninguna otra tabla, podemos borrar sin problema cualquiera de sus registros.



## **Bibliografía**

Godoc, E. (2014). SQL. Cornellà de Llobregat, Barcelona: Ediciones ENI.