UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES "CAMPUS ARAGÓN"

SQL LENGUAJE Y APLICACIONES

T E S I S

QUE PARA OBTENER EL TITULO DE :

INGENIERO EN COMPUTACIÓN

P R E S E N T A :

ALEJANDRO CHÁVEZ RODRÍGUEZ

ASESOR:
ING. SILVIA VEGA MUYTOY

MÉXICO **2002**

PAGINACION DISCONTINUA

Agradecimientos

A Dios

Que ha estado conmigo en cada etapa de mi vida, en cada problema, en cada alegría, en todo momento como mi amigo, mi padre y mi guía.
Hoy, quiero decirle....."Gracias amigo"

A mis padres

Fausto y Elena que han trabajado sin descanso para darme una carrera, para brindarme una educación. Son la piedra angular de mi vida, les debo todo; lo que soy, lo que fui y lo que seré. Gracias a ustedes es que llegó a cumplir esta meta porque me enseñaron como vivir, como estudiar, como alcanzar y lo más importante; como valorar. Los amo.

A mis hermanos

Porque en cada uno de ellos he encontrado valores importantes; de Caro la constancia y el carácter, de Mari la fuerza para afrontar la vida ante los retos y de Javi la iniciativa y el hacer con calidad las cosas. Los quiero mucho hermanos.

A mi amor

Alejandra, que me ha dado el amor, la fuerza para seguir adelante, la inyección de ganas y superación con la cual llego hoy a cumplir un sueño que alguna vez sentí lejos y ahora se vuelve una realidad. Gracias por ser mi novia, mi amiga, mi apoyo, mi confidente y por compartir un momento tan trascendental en mi vida. Nunca olvides que eres parte de este sueño y de otros más. Te amo Ale.

A mis mejores amigos

Isidro, Jorge y Omar que han pasado tantas vivencias conmigo, que han sabido apoyarme y sobre todo y lo más importante han sabido ser amigos; amigos para toda la vida.
Nunca olvido una frase que compartimos en cada termino de semestre y que siempre nos hizo mantener los pies en la tierra y que sé; seguirá en nuestras mentes mientras tengamos vida ..... "Y que sabemos ?".

A la otra parte de mi familia

Que ayudo en mucho a mi desarrollo como persona y que sé, siempre contaré con ellos; algunos de ellos ya no están aquí, pero siempre estarán en mi corazón

Rosario Reyes, Angela Tomaidis, Agustin Chávez, Audelia Paniagua , Trinidad Ramírez(†), Catalina Jimenez S.(†), Abel Rodríguez, José Manuel Arturo, Cristobal Chacon, Federico Chávez., Gudalupe Chávez, primos, tíos, etc.

A mi tía

Rosario que desde que era un niño me ayudó, me fomentó el estudio y fue la primera profesional que conocí en mi vida, la cual me hizo querer a la UNAM desde mi niñez. Gracias tía por tanto apoyo.

A mis profesores

Que nos regalaron su experiencia, su conocimiento y ante todo su amistad. En especial a mi asesora Silvia Vega que me brindó todo su apoyo en cualquier momento que lo necesite, a los profesores que fueron maestros míos y jurados Roberto Blanco y Gabriela Gonzáles y al profesor Luis Flores que hizo unas clases inolvidables y que fue y es un gran amigo. Gracias.

Más amigos

A todas aquellas personas que han estado conmigo en alguna etapa de mi vida y que me han demostrado su apoyo y su amistad.
Amigos de TELMEX, de SICE, de la Universidad, del CCH, de Ingles, etc.
Gracias amigos.

ÍNDICE DE CONTENIDO

Capitulo 5. SQL en Oracle con SQL*PLUS.

Introducción general.

Todo el entorno de un Ingeniero en Computación es muy basto y requiere que cada uno de nosotros busque un área de especialización; un área de desarrollo donde pueda realizar grandes proyectos, donde estimule la imaginación, la creatividad y donde pueda manifestar sus inquietudes. Cada área requiere la interrelación con distintas profesiones y personas, con medios que pueden ser ajenos a nosotros pero que requieren de las herramientas, conocimientos y capacidad que nosotros hemos desarrollado a través de nuestra carrera universitaria. En mi caso busque la opción para desarrollarme profesionalmente respecto a las bases de datos; primera porque es un campo de trabajo muy basto y que avanza a pasos grandes y el otro motivo es que tenía muchas dudas con respecto a la gran cantidad de software que existe para bases de datos, a la gran cantidad de personas y recursos que se involucran en este medio y porque algún día llegó a mis manos un disco compacto donde decía explícitamente "Demo de SQL ". La vez que vi este CD intente probar el programa y no entendí nada, todo era confuso para mí y la duda permaneció.
Otra experiencia se añade; la búsqueda de trabajo; donde en entornos de bases de datos se requería el conocimiento del lenguaje llamado SQL y su relación con plataformas como Oracle. Yo entendí en esos momentos que era de vital importancia el conocer SQL si quería conseguir un trabajo en el área de bases de datos o con relación al desarrollo de software. Las interrogantes fueron entonces ¿Cómo empezar?, ¿En que libros busco?, ¿Tengo que saber teoría de bases de datos?, ¿Necesito algún programa especial?; etc.
Debido a estas situaciones y ahora que me desenvuelvo en un medio de trabajo que involucran SQL; decidí realizar mi tesis titulada "SQL Lenguaje y Aplicaciones" para apoyar a mis compañeros que están estudiando la carrera y que al leer esta tesis puedan tener una perspectiva más clara, puedan y sepan utilizar las principales instrucciones de este lenguaje y tengan bases para conseguir un trabajo. Esta es la aportación que ofrece mi tesis a la universidad.
Debido a las interrogantes que se presentan cuando se aprende un lenguaje nuevo como SQL, y en base al trabajo que desarrollo todos los días que incluye la capacitación de personal en el lenguaje; decidí plantear los siguientes objetivos para mi tesis.
Mi objetivo principal es realizar una tesis que enseñe los aspectos elementales del lenguaje SQL para que cualquier persona que tenga estudios introductorios de bases de datos pueda leer la tesis y aplicar las principales instrucciones del lenguaje SQL para crear, modificar, manipular y consultar las bases de datos, conocer qué es y cuál es la función principal del lenguaje SQL, conocer las principales instrucciones del lenguaje SQL en sus dos ámbitos teórico y práctico y saber dónde y cómo utilizar estas instrucciones en dos plataformas importantes como lo son Oracle y Access
Como parte del proceso de investigación de mi desarrollo de tesis decidí utilizar dos metodologías importantes que me parecen adecuadas para desarrollar esta tesis. La primera de estas metodologías es la inductiva; conociendo cada una de las principales instrucciones del lenguaje SQL hasta formar un conjunto de conocimientos más grandes y estructurados que me permitan demostrar que puedo realizar una gran cantidad de operaciones de manipulación de datos; es decir parto de la particularidad de cada instrucción para llegar a la generalidad de un lenguaje o conjunto de instrucciones que me permitan manipular datos. Esta no fue la única metodología en la que me apoye para desarrollar mi trabajo, esta también la que considero a mi manera personal la más importante y la que permite

comprender de mejor manera este lenguaje y en general cualquier lenguaje; la experimentación ya que podemos nosotros ver en un libro la sintaxis; pero no sabemos como se comporta, no sabemos que puede pasar en un medio real, con gran cantidad de información tal vez 1,000 o 15,000 registros.
Gran parte de la tesis hay ejercicios probados con datos, además que sirva de estímulo en la persona que lea mi trabajo a que se ejercite y visualice la funcionalidad de cada instrucción. Solo practicando es la manera de comprender el comportamiento, el uso y la aplicación de cada instrucción del lenguaje SQL. En muchas ejercicios hay notas pequeñas de algunas experiencias reales con respecto a esa instrucción.
Con estas dos metodologías he apoyado mi trabajo para darle el equilibrio que requiere con respecto a la teoría y la práctica.

La tesis comprende 3 partes básicas que son:

Parte 1.Teoría de bases de datos relacionales.
- Capitulo 1. Breve Teoría de Bases de Datos

Parte 2.Lenguaje SQL(Instrucciones DDL y DML)
- Capitulo 2. Selección de datos mediante SQL. Instrucciones DML
- Capitulo 3. Creación de Bases de datos. Instrucciones DDL

Parte 3.SQL en motores de bases de datos relacionales (ACCESS, ORACLE)
- Capitulo 4. SQL en Access por medio de Visual Basic(Visdata y SQLVB6)
- Capitulo 5. SQL en ORACLE con SQL *Plus

En la primera parte se busca dar un repaso a la teoría de bases de datos, al modelo relacional y al "Sistema de Gestión de Bases de Datos" (SGBD), ya que el lenguaje se maneja dentro de un entorno de bases de datos relacionales y por esta razón es importante el mostrar y recalcar los aspectos fundamentales en donde se va a desenvolver el lenguaje SQL; cabe aclarar que no busco extenderme demasiado en el tema y la profundidad ya que mi objetivo fundamental de la tesis es el de dar a conocer las principales instrucciones de SQL y su aplicación y no el realizar una tesis de teoría de bases de datos.
En esta parte hay apertura al lenguaje SQL; se muestra donde comienza a ser importante, se muestra el objetivo y la función del lenguaje, se muestra que es SQL, se muestra porque es una herramienta básica ya que el SGBD define funciones, lenguajes facilidades e interfaces para el manejo de las bases de datos.
La segunda parte es la más importante de la tesis puesto que es donde se muestra el uso de las instrucciones del lenguaje SQL, ejemplos con ejercicios, notas sobre algunas características de la instrucción y se muestra también al resultado en algunos casos a los que se tiene que llegar. Se analizan dos tipos de ordenes; DDL y DML y la diferencia que existe en su funcionalidad.
Finalmente la tercera parte tiene como objetivo el mostrar como se puede hacer uso del lenguaje SQL en dos plataformas importantes y muy actuales como lo son Oracle(por medio de SQL *Plus) y Access por medio de Visual Basic Ver. 6.0). Uso estas dos plataformas debido a que Oracle es para el manejo de una gran cantidad de información y Access es más limitado; y la idea es el mostrar como SQL puede ser una herramienta tanto para aplicaciones grandes(a nivel corporativo) así como aplicaciones de un solo usuario. Es importante aclarar que aquí no se busca enseñar el lenguaje de Visual Basic, ni enseñar el uso de la Base de datos Oracle o de Access. Cada uno de ellos requeriría al menos una tesis

para su estudio. Mi objetivo es sólo mostrar como interactúa con estas plataformas, si se ven algunas instrucciones propias de Visual Basic, pero son sólo las más esenciales para el trabajo con SQL.
Estos tres partes tienen el objetivo de resolver el planteamiento del problema básico que involucra la manipulación de datos, los entornos de trabajo y plataformas del lenguaje SQL y por supuesto el conocimiento de las principales instrucciones del lenguaje SQL. De esto se desprende la hipótesis principal de mi tesis:
"Teniendo el fundamento teórico y sabiendo aplicar las principales instrucciones del lenguaje SQL, se tendrá la capacidad para realizar una gran cantidad de operaciones para manipular datos de las más importantes bases de datos relacionales."
Finalmente; las técnicas de recopilación que utilice para obtener información fueron de diversa índole. En primera instancia use la bibliografía básica que venía en mis apuntes de bases de datos; subsecuentemente en buscadores de Internet como Altavista y Yahoo encontrando una gran variedad de artículos relacionados al tema, pláticas con gerentes de mi empresa y diversos desarrolladores, la experiencia adquirida en mi trabajo como desarrollador de sistemas en plataforma Oracle, y diversos libros de bases de datos, SQL, Visual Basic, Oracle. Esta información fue filtrada en base a los objetivos que había planteado para el capitulado; cada libro sigue una línea y lleva consigo aspectos que no atañen a lo que nos interesa; se tuvo que analizar cada uno omitiendo temas que estuvieran fuera del objetivo de la tesis ,confronte gran cantidad de información para obtener las características coincidentes de cada autor y omitir las diferencias más radicales.
Los aspectos antes mencionados forman la tesis que ahora presento.

Capitulo 1 Breve Teoría de Bases de Datos

Introducción.

El almacenamiento, manipulación, recuperación y uso de la información en forma adecuada es de gran importancia para cualquier organización. La investigación, la planificación y la toma de decisiones exigen una información precisa, oportuna, completa, coherente y adaptada a las necesidades de cada usuario. Las bases de datos juegan un papel de gran relevancia en casi todas las áreas, incluyendo medicina, ingeniería, leyes, educación, política, etc.

El lenguaje SQL opera en las bases de datos; se desenvuelve en él y es donde se puede ver y aplicar la potencialidad de este lenguaje; por ello es oportuno introducir el estudio de las bases de datos analizando sus inicios, sus conceptos, su definición, sus modelos para dar una base donde se sustente el estudio y la importancia del lenguaje SQL. Cabe destacar que este estudio no es extenso,debido a que el objetivo de la tesis es la de mostrar el uso y la aplicación del lenguaje SQL y no de un estudio extenso de la teoría de bases de datos. El objetivo de este capítulo es mostrar los aspectos más importantes de las bases de datos en forma clara y concreta para ir llevando al punto exacto donde se hace necesario el uso de este lenguaje.

1.1 Inicios de los sistemas.

1.1.1 Inicios(enfoque del modelo anterior)

Si se analizan los sistemas y aplicaciones en el pasado, se puede ver que existía una gran cantidad de archivos para cada aplicación. Los datos se obtenían varias veces y se encontraban repetidos en los distintos archivos. Esto generaba una gran redundancia, que por supuesto se traducía en recursos malgastados e información de mala calidad. En el siguiente cuadro(Fig. 1) se muestra un ejemplo gráfico del enfoque anterior que se le ha llamado "Sistemas Orientados al Proceso".

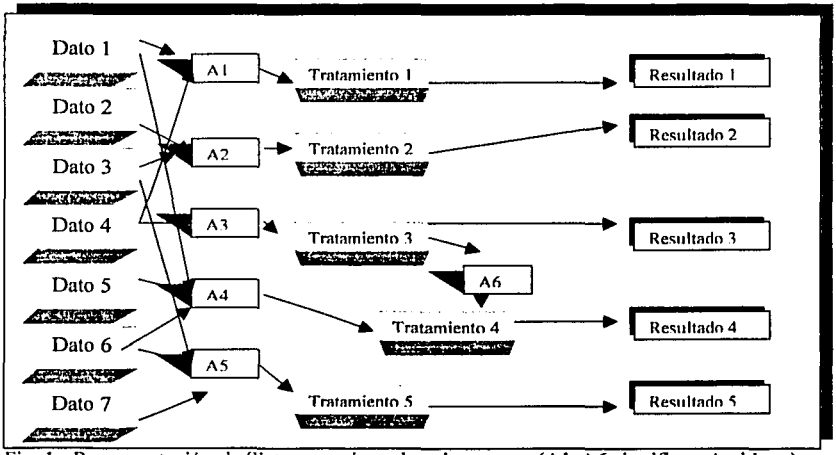

Fig. 1 Representación de Sistemas orientados al proceso. (A1-A6 significan Archivos)

1.1.2 Desventajas del modelo Anterior.

En este siguiente ejemplo se analizarán las desventajas que se obtenían al hacer uso del enfoque anterior(Sistemas Orientados al Proceso).

Ejemplo 1. Se desea almacenar la información de los pedidos de queso que ha hecho un cliente por mes y que tipo de queso ha solicitado en cada uno de sus pedidos. La tabla 1 muestra el enfoque tradicional.

Código del Cliente	Nombre del Cliente	Mes de Pedido	Tipo de Queso	Costo
1	Alejandra Mercado	2	Cottage	350.00
1	Omar García	2	Panela	250.00
2	Isidro López	2	Queso Crema	300.00
23	Jorge Ubaldo	3	Sierra	200.00
1	Omar García	3	Cottage	350.00
2	Isidro López	3	Manchego	400.00
1	Omar García	4	Oaxaca	200.00
24	Jorge Rojas	5	Queso Crema	300.00

Tabla 1. Forma de almacenar datos basandose en el enfoque tradicional.

En esta tabla se puede observar los siguientes aspectos:

- Se repiten varios datos, como el Nombre del Cliente, el Mes de Pedido, el Tipo de queso y el Código del Cliente.
- Si se desea cambiar el Nombre de uno de los Clientes, se tiene que hacer en cada lugar donde aparezca, esto significa que se debe buscar en toda la tabla.
- Si se desea sacar el Costo Total por Año de cada cliente se tiene que leer toda la tabla secuencialmente.

En este ejemplo no parece muy difícil hacer esto puesto que sólo son ocho renglones, pero se puede imaginar lo que pasaría con cien mil renglones o más sería demasiado el tiempo que se requeriría para completar la tarea, el espacio de almacenamiento tendría que ser inmenso, el costo sería muy grande y mucha de la información sería incorrecta y se puede agregar el tiempo para ejecutar cada uno de los procesos.

Como se puede ver en este enfoque tradicional; las aplicaciones se analizan e implementan con entera independencia unas de otros, y los datos no se suelen transferir entre ellas sino que se duplican siempre que otros procesos o tareas los requieran.
Aunque más graves son las inconsistencias que a menudo se presentan en estos sistemas, debido a que la actualización de los datos en las distintas aplicaciones no se llevan al mismo tiempo.
Este tipo de desventajas se remarcan más cuando por ejemplo; se presentan demandas inesperadas de información o se requiere información importante para una toma de decisiones.
Este tipo de problemas que afrontaban día a día las organizaciones, la necesidad de mejorar todos estos aspectos negativos y por supuesto el liderazgo en las distintas ramas llevó al desarrollo de nuevas tecnologías que mejoraran por mucho a las ya existentes. Se necesitaba una gestión más racional del conjunto de datos, surgiendo así un nuevo enfoque que se apoya sobre una base de datos en la cual los datos son recogidos y almacenados una sola vez, con independencia de los tratamientos(Fig. 2).
Estos nuevos sistemas que van surgiendo "Sistemas Orientados hacia los Datos" van substituyendo poco a poco a los viejos sistemas orientados al proceso.

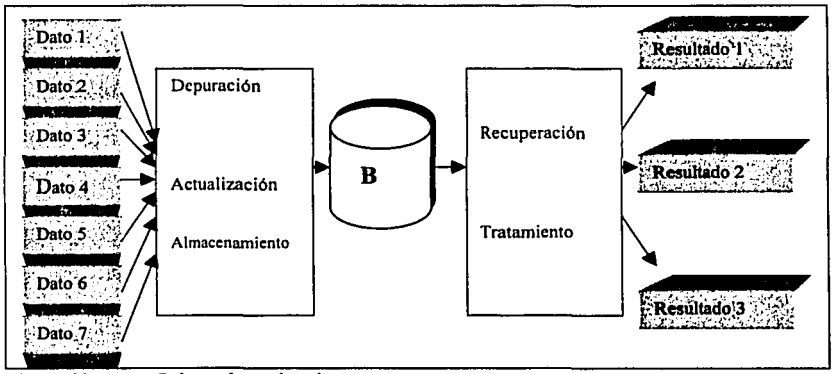

Fig. 2 Sistemas Orientados a los datos.

1.2 Ventajas y Desventajas de las Bases de Datos.

1.2.1 Ventajas de las Base de Datos.

Estos sistemas presentan una gran variedad de ventajas; por ejemplo:
La independencia que presentan los datos con respecto a los tratamientos y a la inversa que lleva a que un cambio en los datos no imponga un nuevo diseño en las bases de datos. Esta característica da la ventaja de la flexibilidad que es tan importante ya que da la oportunidad de ir adaptando el sistema a la evolución que la misma organización vaya marcando, puesto que en estos tiempos conforme las organizaciones se vayan adaptando a los cambios tan fuertes y rápidos más será la posibilidad de escalar puestos y mantenerse en el liderazgo.
Otro aspecto muy importante es la coherencia en los datos; esto se consigue debido a que al usar los mismos datos para los tratamientos, al recoger y almacenar una sola vez los datos; los resultados serán claros y acorde a lo que se deseaba obtener puesto que no hacen uso de diferentes datos. Como consecuencia no hay redundancia y la actualización de los datos se lleva al mismo tiempo disminuyendo así los tiempos de procesamiento de la información y mejorando la calidad de ésta.
La disponibilidad de los datos es más organizada y con más control ya que aquí nadie es dueño de los datos, todos pueden hacer uso de ellos y compartirlos siempre y cuando estén autorizados.

1.2.2 Desventajas de las Base de Datos.

Es importante tomar en cuenta las desventajas que puede traer la implantación de una base de datos. Una de ellas es la instalación que puede llevar implícito un alto costo en equipo de cómputo y en los programas que se necesitan. También hay que tomar en cuenta el personal que se requiere para instalar la base de datos, el diseño y la administración. Ahora cuando se implanta un sistema de bases de datos lleva consigo un proceso largo con constantes cambios y adaptaciones para conseguir satisfacer las necesidades de la empresa, ese proceso es algo que deben tomar en cuenta las personas que deciden implantar un sistema de estos en su empresa. Éstos son algunos de los problemas que se presentan al instalar una base de datos y que siempre debe formar parte del análisis de los que deciden implantarlo.

1.3 Definición de una Base de Datos.

En base a algunos aspectos que se han mencionado y a otros que se verán con más detalle adelante podemos dar una definición de la base de datos.
Una base de datos es una colección de datos interrelacionados almacenados en un soporte secundario no volátil y donde estos datos tienen una redundancia controlada. A su vez los datos van a ser compartidos por una gran diversidad de usuarios y por muchas aplicaciones. La estructura de la base de datos (definición) se va a almacenar con los datos y se va apoyar en un modelo de datos el cual va a permitir captar las interrelaciones y restricciones existentes en el mundo real.
Se van a tener procedimientos de actualización y recuperación que no atentan contra la seguridad de los datos.

El por qué de esta definición?
En una base de datos lo que se busca como objetivo principal es almacenar datos y proveerlos al usuario cada que los necesite; pero datos con calidad (información), datos que den el soporte para una adecuada toma de decisiones; por tal razón requieren no estar aislados, requieren estar relacionados para funcionar como información.

En la definición se puede notar el uso de la palabra "Modelo"; y es que para almacenar estos datos relacionados requerimos de una representación de la realidad, (del mundo real), para estar más apegados a lo que se define en nuestro mundo.
Estas bases de datos pretender atender a múltiples usuarios y a diferentes aplicaciones en contraposición a los sistemas orientados al proceso donde cada archivo esta diseñado para responder a las necesidades de una determinada aplicación.
La definición o descripción del conjunto de datos contenidos en la base (lo que se denomina estructura o esquema de la base de datos) deben ser únicas y estar integrada con los mismos datos.
El Sistema de gestión de bases de datos (SGBD) es el conjunto de programas que permiten la implantación, acceso y mantenimiento de las bases de datos. El SGBD junto con la base de datos y con los usuarios, constituyen el "Sistema de Base de Datos".[1]

1.4 Niveles de abstracción.

Una característica bien importante del enfoque de bases de datos es que proporciona cierto nivel de abstracción de los datos, al ocultar detalles de almacenamiento que la mayoría de los usuarios no necesitan conocer. Los modelos de datos son el principal instrumento para ofrecer dicha abstracción. Primero se describirá las características de estos tres niveles de abstracción y luego se hablará sobre los modelos de datos.

En la mayoría de los sistemas existen dos estructuras fundamentales: la lógica (vista del usuario) y la física(forma en que se encuentran los datos en el almacenamiento). En las bases de datos aparece un nuevo nivel de abstracción que se ha denominado de diversas formas: nivel conceptual, lógico global, etc. Esta estructura intermedia pretende una representación global de los datos que se interponga entre las estructuras lógica y física.

Estructura Lógica del Usuario: Esquema Externo
Más cercana a los usuarios. Debido a que en un esquema externo es el punto de vista y como lo observa un usuario; en este esquema deberán encontrarse reflejados sólo aquellos datos e interrelaciones que necesite el correspondiente usuario. También habrá que especificarse las restricciones de su uso, como puede ser el derecho a insertar, borrar determinados datos o el acceso a los mismos.

Estructura Lógico Global: Esquema Conceptual
En este esquema, por ser la visión global de los datos, deberá incluirse la descripción de todos los datos e interrelaciones entre éstos, así como las restricciones de integridad y confidencialidad. Este esquema conceptual describe la estructura de toda la base de datos para una comunidad de usuarios. Describe atributos, vínculos, restricciones, operaciones.

Estructura Física: Esquema Interno

[1] En secciones posteriores se extenderá el tema de "Sistema de Bases de Datos".

El contenido del esquema interno depende en gran medida de cada SGBD(Sistema de Gestión de Base de Datos); se pueden distinguir tres aspectos que deben especificarse en él.

- Estrategia de almacenamiento. Aquí se incluye la asignación de espacios de almacenamiento para el conjunto de datos y la estrategia para optimizar tiempos de respuesta y espacios de memoria.
- Caminos de acceso. Aquí se incluye la especificación de claves e índices.
- Miscelánea. Aquí se incluye técnicas de compresión de datos, de criptografiado, la correspondencia entre esquema interno y esquema conceptual, optimización etc.

1.5 Modelo de Datos

Como definición para partir en el modelo de datos(MD); se puede decir que es un conjunto de conceptos que permiten describir, a distintos niveles de abstracción, la estructura de una base de datos, a la cual se le llama esquema. Dependiendo del nivel de abstracción que se encuentre la estructura descrita; el modelo puede ser externo, global o interno.

Los modelos externos nos permiten representar los datos que necesita cada usuario en particular con las estructuras propias del lenguaje de programación que va a emplear. En el caso de los modelos globales, ayudan a describir los datos para el conjunto de usuarios; a nivel de conjunto (optimizar los recursos a nivel empresa). En el caso de los modelos internos o físicos están orientados a la máquina (se busca la eficiencia de los recursos informáticos).

En este caso para el estudio de la tesis se pondrá más énfasis en el modelo global; además de que este modelo utiliza los mismos conceptos que el externo y en los internos son muy particulares.

Estos modelos globales se clasifican a su vez en conceptuales y convencionales.

- Los modelos conceptuales o de alto nivel facilita la descripción global del conjunto de información; son modelos de análisis no de implementacion. A estos modelos también se les llama semánticos. La finalidad de estos modelos es la ayuda al diseño de bases de datos en la fase de representación. Ejemplos de estos modelos son: Entidad/Interrelación, Infológico, etc.
- Los modelos convencionales se encuentran soportados por los SGBD(Sistemas de Gestión de Bases de Datos) y están orientados a describir los datos a nivel lógico. Por lo que sus conceptos son propios de cada SGBD (tablas o relaciones en el caso del modelo relacional ó árboles en el jerárquico).

El modelo de datos (sea lógico o físico) es el instrumento que se aplica a los datos para obtener el esquema(Fig. 3)

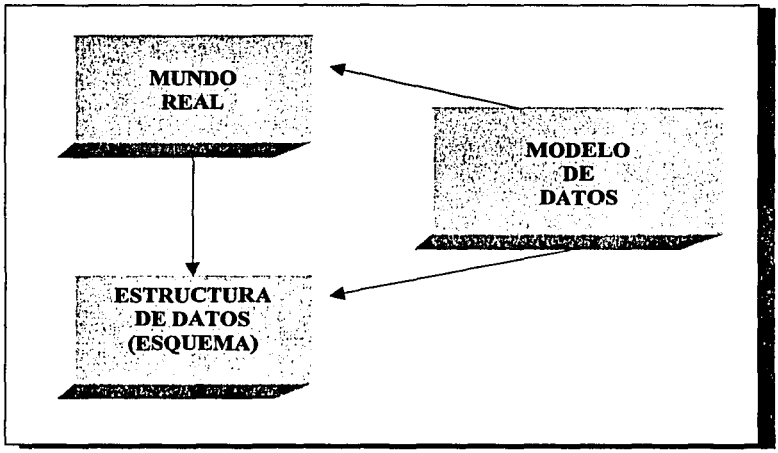

Fig.3 Aplicación de un "Modelo de Datos" a un mundo real para obtener un esquema.

Es sumamente importante para este estudio el poder diferenciar entre esquema y ocurrencia; ya que permite entender de mejor manera el modelo de datos.

En el caso de esquema se debe entender como descripción de la estructura de la base de datos, y ocurrencia del esquema, que son los datos que se encuentran almacenados en un determinado momento. El esquema no varía mientras no varíe el mundo real que éste describe; mientras que la ocurrencia del esquema; es decir, los datos contenidos en él, son distintos en el transcurso del tiempo.

Con todas estas bases se puede decir entonces en forma más estricta que un modelo de datos es un "Conjunto de conceptos, reglas y convenciones que permiten describir y manipular (consultar y actualizar) los datos de un cierto mundo real que se desea almacenar en la Base de Datos".

Los modelos son la base para los lenguajes, aunque el nivel de abstracción es menor, ya que el lenguaje es el modelo más una sintaxis; por ejemplo el lenguaje SQL es el resultado de aplicar una determinada sintaxis al modelo relacional.

1.5.1 Propiedades del Modelo de Datos.

Hay dos tipos de propiedades; estáticas(suele entenderse como estructura) y dinámicas (que son las operaciones que se aplican a los datos o valores almacenados en las estructuras).

- Estáticas: Se compone de
  - Elementos Permitidos
    - Objetos (entidades, relaciones, registros)
    - Interrelaciones
    - Propiedades o características de los objetos o de las interrelaciones (atributos, campos, etc.)
    - Dominios; conjunto de valores sobre los que se definen las propiedades.
  - Restricciones
    - No todos los valores están permitidos en el mundo real; por ejemplo un niño no puede ser viudo o tener cuarenta años. Este tipo de limitaciones que unas veces vienen impuestas por el mismo modelo de datos y otras nos las impone el mundo que estamos modelando, se denominan restricciones. Las restricciones que impone el mismo modelo se llaman restricciones inherentes, y las responden a la necesidad de dar al sistema un apreciación lo más cercana y real al mundo son llamadas restricciones de integridad o semánticas. Con semántica se quiere referir al significado de los datos y con integridad a la corrección de los mismos y a su consistencia respecto al mundo real del cual proceden

- Dinámicas

  Los valores que toman los distintos objetos de un esquema en cualquier momento deben de cumplir las restricciones de integridad así como las posibles restricciones asociadas al cambio de estado. La componente dinámica del modelo consta de un conjunto de operadores que se definen sobre la estructura del correspondiente modelo de datos.

  Una operación tiene dos componentes:

  1. Localización también llamada selección, consiste en localizar una ocurrencia de un objeto indicando un camino(sistema navegacional), o bien un conjunto de ocurrencias especificando una condición(de especificación).
  2. Acción: que se realiza sobre la(s) ocurrencia(s) previamente localizada(s) mediante una operación de localización, y puede consistir en una recuperación o en una actualización (inserción, borrado o modificación).

SQL reúne ambas operaciones en un único operador.
Así en la siguiente sentencia SQL:

```
SELECT Titulo, Autor
FROM Libros_Mayas
WHERE Fecha_Edicion='2000'
```

La localización y la acción de recuperar se expresa mediante un único mandato con el verbo inglés "SELECT", el objetivo son las propiedades (atributos en el modelo relacional) "Titulo" y "Autor" del objeto(relación) "Libro_Mayas" y la condición es que la "Fecha_Edición" sea igual a 2000. Con esta sentencia se recuperan (no sólo se localizan) el titulo y el autor de todos los libros Mayas editados en el 2000.

Los modelos de datos en el diseño de bases de datos.
Los modelos de datos proporcionan paso a paso la estructuración; del mundo real a la base de datos física. Cuando se esta diseñando una base de datos es de gran importancia distinguir la fase de modelado conceptual donde se describe el mundo real de acuerdo con un modelo altamente semántico e independiente del SGBD y la fase de diseño lógico en donde se va a obtener un esquema que responda a la estructura lógica específica del SGBD.

En la Fig. 4 se ve el proceso que un modelo de datos permite realizar.
En el mundo real existen objetos y asociaciones entre ellos; los dos tienen propiedades y hay reglas que imponen ciertas limitaciones. Es necesario una abstracción del mundo real a fin de obtener el esquema conceptual.

En la primera parte con la ayuda del esquema, se obtiene el esquema conceptual. Después aplicando las reglas del modelo de datos propios del SGBD que se va a utilizar, se obtiene el esquema lógico o esquema de bases de datos y ya de éste se pasa al esquema interno, donde el objetivo es conseguir la máxima eficiencia de los recursos informáticos y al problema específico. Por último se implementa la base de datos física en los medios secundarios de almacenamiento. La estructura física se va a llenar con los valores (ocurrencias) que se obtienen por observaciones de lo que pasa mundo real.

Es importante saber que hay herramientas CASE que fueron realizadas para el diseño de bases de datos, al disponer de datos semánticos (en general basados en el Modelo Entidad/Relación) que faciliten el diseño conceptual y realizan la transformación al modelo relacional.

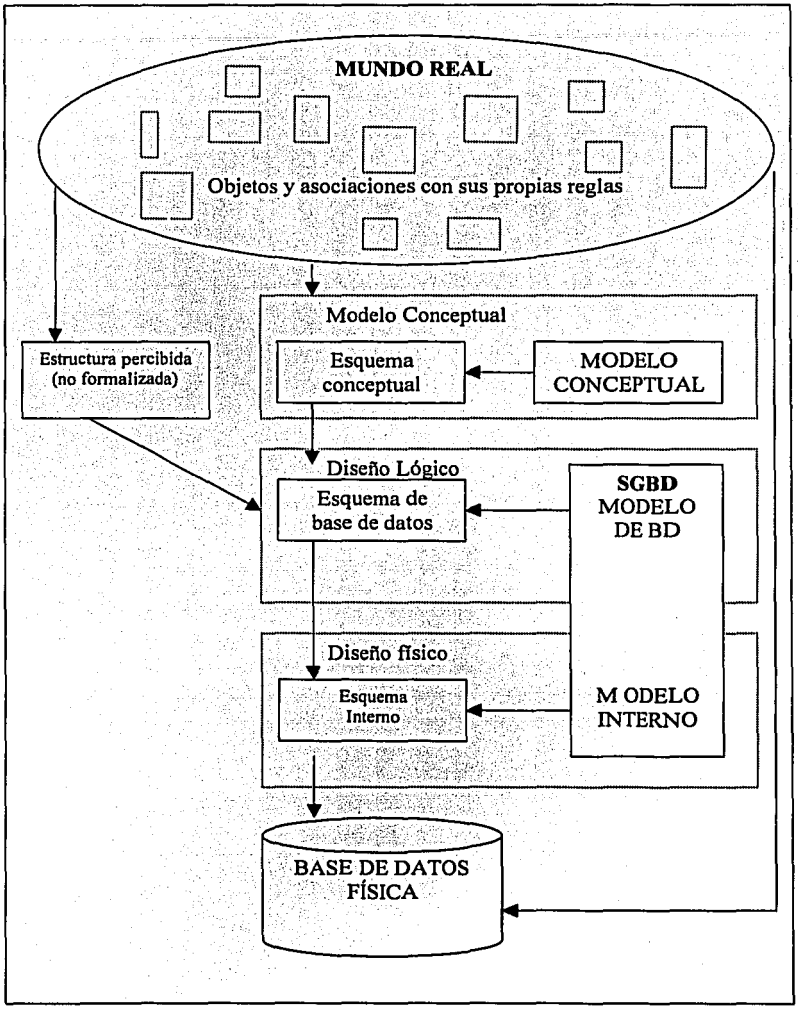

Fig. 4 Transformación del mundo real a la base de datos física.

1.5.2 Modelo Relacional

Por el objetivo de la tesis de mostrar el uso del lenguaje SQL es de gran importancia como una base fundamental de este lenguaje el hablar sobre el "Modelo Relacional" que es de donde sale el SQL; donde se define.
Este modelo relacional tiene dos partes muy importantes que son la parte estática y la parte dinámica; las dos van de la mano y ofrecen una gran potencialidad a los lenguajes que se definan dentro de este modelo de datos. Naturalmente sólo se hablará de los conceptos más importantes de este modelo que sustentan a SQL; puesto que el objetivo es explicar cómo se puede aplicar el lenguaje SQL; cómo se usa; como se interactúa con el.
Y no es el objetivo el explicar teoría de bases de datos.

1.5.2.1 Introducción.

Una aportación importante fue la de Codd en el año de 1960 con la teoría matemática de las relaciones en el campo de las bases de datos. En ésta aportación se presenta un nuevo modelo de datos con una serie de objetivos fundamentales que eran: independencia física, independencia lógica, flexibilidad, uniformidad y sencillez.
Para lograr estos importantes objetivos Codd introduce el término de relación(tabla) como estructura básica del modelo y con respecto a la parte dinámica del mismo se propone un conjunto de operadores que se aplican a las relaciones.
De aquí se lograron realizar algunos productos basados en este modelo de datos como fueron:

- Query By Example –QBE- un lenguaje de acceso relacional a ficheros VSAM de IBM.
- Oracle, primer SGBD relacional comercial, el cual soporta como lenguaje de definición y manipulación de datos el SQL.
- Ingres nacido en la Universidad de Berkeley

1.5.2.2 Estructura del Modelo Relacional.

El modelo relacional ofrece una manera única de representar los datos: como una tabla bidimensional denominada relación.
En el siguiente cuadro Fig. 5 se puede apreciar algunos de los elementos más importantes de este modelo.
En el cuadro se representa la relación EMPLEADO, en donde aparece la estructura del modelo relacional. Aparecen las columnas Nombre, Nacionalidad y Estudios que representan los atributos y son las propiedades de la tabla; un conjunto de Filas o registros que se les denomina tuplas y los dominios que es de donde los atributos toman sus valores; el grado representa el número de atributos que posee la tabla y la cardinalidad el número de tuplas que hay en la tabla.
Aunque una relación se puede representar en forma de tabla; la relación tiene elementos muy importantes que la distinguen de la tabla. Ejemplo de esto es que no se admiten filas duplicadas, las filas y las columnas no están ordenadas y es plana, es decir que en el cruce de una fila y una columna sólo puede haber un valor.

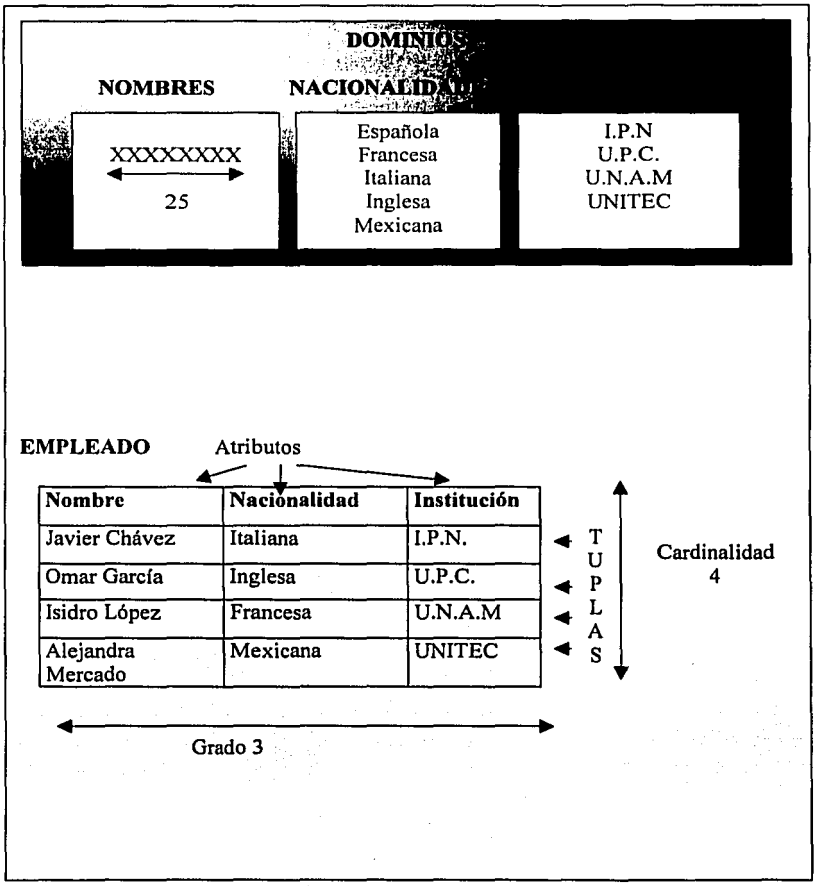

Fig. 5 Representación de la relación Empleado.

1.5.2.3 Atributos y el Dominio.

En la parte superior de una relación se ven los atributos. Los atributos de una relación sirven de nombre a las columnas de la relación. Por lo regular describen el significado de las entradas de la columna situada por debajo de ellos. Un dominio D es un conjunto finito de valores del mismo tipo e indivisibles V1,V2,.....Vn caracterizados por un nombre. El dominio entonces tiene un nombre y un tipo de dato; así el tipo de datos del dominio de nacionalidades puede ser de tipo de dato carácter con una longitud de diez ó quince caracteres.

Estos dominios se pueden definir por extensión o por intensión. Por ejemplo el dominio de las edades de los empleados se puede definir por extensión como entero de longitud dos comprendido entre dieciocho y sesenta y cinco; sin embargo si el dominio de las nacionalidades por intensión sería muy limitado semánticamente ya que permitiría toda combinación de diez o quince letras aun cuando no formaran un nombre valido de nacionalidad; por ello sería preferible definir este dominio por intensión con los distintos nombres de las nacionalidades que permitiera la base de datos.
El dominio contiene todos los posibles valores que puede tomar un atributo y es estático; puesto que estos valores no varían en el tiempo.

1.5.2.4 Definición de una relación

Una relación o instancia de la relación r del esquema de relación R(A1,A2,...,An), denotado también como r(R) es un conjunto de n-tuplas r=(t1,t2,....,tm) Cada n-tupla t es una lista ordenada de n valores t = <v1,....,Vn>, donde cada valor vi, i<=1<=n, es un elemento de dom(Ai) o es un valor nulo(carece de valor).

Ejemplo. Ver Tabla 2.
EMPLEADOS

Nombre	Nacionalidad	Estudios	Teléfono
Alejandra Mercado	Mexicana	Universidad	53164534
Isidro López López	Francesa	Universidad	56963815
Omar Garcia Diaz	Inglesa	Primaria	56963824

Tabla 2. Tres tuplas en la tabla de EMPLEADOS .

Cada tupla representa una entidad de estudiante en particular. La definición de relación puede replantearse así: Una relación r(R) es un subconjunto del producto cartesiano de los dominios que definen r:

$r(R) \subseteq (dom(A1)xdom(A2)x...xdom(An))$

El número total de tuplas en el producto cartesiano es:

|dom(A1)| * |dom(A2)|*...*|dom(An)|

Una instancia de relación refleja sólo las tuplas validas que representan un estado en particular del mundo real. A medida que el mundo real cambia, también lo hace la relación, transformándose en otro estado de relación (el esquema R es relativamente estático y no cambia excepto muy pocas veces).

Notación

- Un esquema de relación R de grado n se denota R(A1,A2,...,An)
- Una n-tupla t en una relación r(R) se denota t=<v1,...v2>, donde vi es el valor correspondiente del atributo Ai

- t[Ai] se refiere al valor vi en t para el atributo Ai
- t[Au,Aw,...,Az] donde Au, Aw,...,Az es una lista de atributos de R, se refiere a las subtuplas de valores <Vu,Vw,...Vz) de t correspondientes a los atributos espedificados de la lista
- Las letras Q,R,S denotan nombres de relación
- Las letras q,r,s denotan estados de relación
- Las letras t,u,v denotan tuplas
- Los nombres de los atributos se califican con el nombre de la relación a la cual pertenecen. Por ejemplo EMPLEADO.Nombre o EMPLEADO.Estudios
- Para la tupla t = <Alejandra Mercado, Mexicana, Universidad, 53164534>, tenemos t[Nombre]=<Alejandra Mercado>, t[Nacionalidad,Estudios,Teléfono]=<Mexicana, Universidad, 53164534>

1.5.2.5 Clases de relación

En el siguiente cuadro Fig. 6 se observa una de las clasificaciones

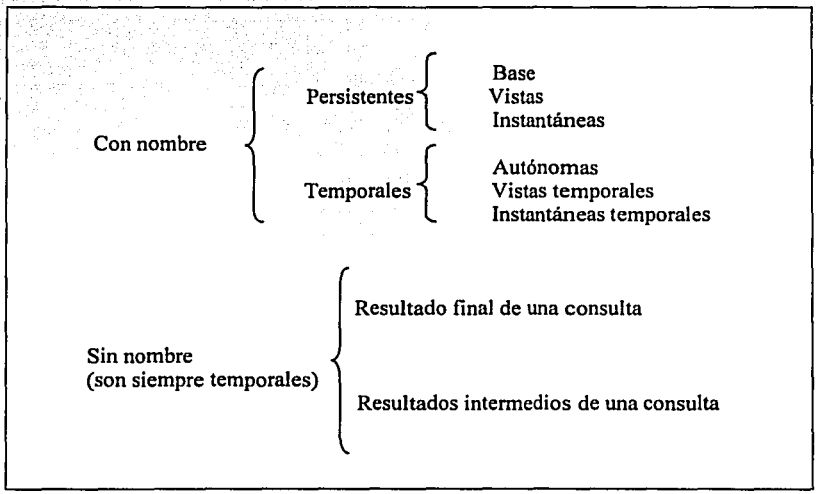

Fig. 6 Clasificación de las relaciones

- Persistentes: Son aquellas relaciones cuya definición permanece en la base de datos. A su vez éstas se dividen en:

- Relaciones Base(Nivel Conceptual en el modelo ANSI): Existen por si mismas, no en función de otras relaciones, y se crean especificando explícitamente su esquema de relación (nombre y conjunto de pares: atributo/dominio)
- Vistas (Nivel Externo en el modelo ANSI): Son relaciones derivadas que se definen dando un nombre a una expresión de consulta. No tiene datos almacenados, sino lo único que almacenan es su definición en términos de otras relaciones con nombre.
- Instantáneas (Nivel Interno en el modelo ANSI): Son relaciones derivadas; pero tienen datos propios almacenados, los cuales son el resultado de ejecutar la consulta especificada o de guardar una relación base.
- Temporales: A diferencia de las relaciones persistentes, una relación temporal desaparece de la base de datos en un cierto momento sin necesidad de una acción de borrado del usuario.

Las relaciones sin nombre son los resultados de las consultas que se suelen utilizar como resultados intermedios; pero que nunca se almacenan; como en una calculadora se suma dos cantidades; se obtiene un resultado y en base a él se obtiene otro más; nunca se guarda el resultado sólo sirve para continuar el proceso de calculo.

1.5.2.6 Claves

Ahora se hablara de un termino de gran importancia en este modelo relacional; que son las claves.

Un conjunto de atributos que identifican unívocamente a cada tupla de una relación se le llama clave candidata. Siempre hay al menos una clave candidata, ya que al ser una relación un conjunto no existen dos tuplas iguales.

En la relación puede tener más de una clave candidata, entre las cuales se debe distinguir:

- Clave primaria (Primary key). Es la clave candidata que el usuario escogerá, por consideraciones ajenas al modelo relacional, para identificar las tuplas de la relación.
- Claves alternativas. Son aquellas claves candidatas que no han sido escogidas como claves primarias.
- Clave ajena (Foreign key):.Se denomina clave ajena de una relación R2 a un conjunto no vacío de atributos cuyos valores han de coincidir con los valores de la clave candidata de una relación. Estas deben de estar definidas sobre el mismo dominio.

1.5.2.7 Restricciones

Las restricciones como su nombre lo indica son acciones no permitidas dentro del entorno que en este caso es la base de datos. Hay de dos tipos; las restricciones inherentes y las semánticas.

Restricciones Inherentes.

Estas no son definidas por los usuarios sino obligadas por el propio modelo; sus características son las siguientes:

- El orden de las tuplas no es significativo
- El orden de los atributos no es significativo
- No hay dos tuplas iguales
- Cada atributo sólo puede tomar un único valor del dominio sobre el que esta definido.
- Regla de integridad de entidad. Ningún atributo que forme parte de la clave primaria de una relación puede tomar un valor nulo.

Restricciones Semánticas o de Usuario.

Son ventajas que ofrece el modelo al usuario a fin de que éste pueda reflejar el esquema, lo más fielmente posible, la semántica del mundo real.

Se basara en el estándar SQL92 para el estudio de las restricciones semánticas.

Características:

- Clave primaria(Primary Key). Aquí se permite declarar un atributo o un conjunto de atributos como clave primaria de una relación, por lo que sus valores no se podrán repetir ni se admitirán los nulos. Aunque se sabe que el hecho de que la llave no sea un valor nulo; o sea que es obligatorio que tenga un valor correspondiente a una restricción inherente al modelo; la declaración de un atributo como clave primaria de una relación es una restricción semántica
- Unicidad(UNIQUE) Los valores de un conjunto de atributos no pueden repetirse en una relación
- Obligatoriedad(NOT NULL) de una o más atributos
- Integridad Referencial (FOREIGN KEY). Si una relación R2 (relación que referencia) tiene un descriptor que es una clave candidata de la relación R1 (relación referenciada), todo valor de dicho descriptor debe concordar con un valor de la clave candidata de R1. Esta es una importante restricción semántica que viene impuesta por el mundo real, siendo el usuario quien la define al describir el esquema relacional y el modelo la reconoce sin necesidad de que se programe ni de que se tenga que escribir algún procedimiento. Con respecto a las consecuencias del borrado y modificación realizadas sobre tuplas de la relación referenciada; donde se puede encontrar según SQL92; (ver Fig. 7)

    - Operación restringida(NO ACTION):Esta operación es muy importante y que suele suceder a menudo; donde se tratan de borrar tuplas en donde uno de los campos contiene un valor que referencia a otro. El borrado de tuplas de la relación que contiene la clave referenciada o la modificación de esta, solo se permite si no existen tuplas con este valor en la relación que contiene la clave ajena. En muchos sistemas al querer realizar esto se nos envía un mensaje

donde se indica que hay registros asociados a este valor y por tanto no se va a permitir el borrado o la modificación a menos que modifiquemos estos registros o los borremos y ya no tenga a donde referenciar.

- Operación con transmisión en cascada(CASCADE): Esta operación va muy ligada con la anterior; puesto que el borrado o modificación de tuplas que contiene la clave candidata referenciada lleva consigo el borrado o modificación en cascada de las tuplas de la relación que contiene la clave ajena
- Operación con puesta a nulos(SET NULL): El borrado de tuplas de la relación que contiene la clave candidata referenciada lleva consigo poner a nulo los valores de las claves ajenas de la relación que referencia.

- Retricciones de Rechazo: Aquí el usuario formula una condición mediante un predicado definido sobre un conjunto de atributos, de tuplas o de dominios el cual debe ser verificado.
  - Verificación(CHECK) Comprueba en toda operación de actualización si el predicado es cierto o falso. Por ejemplo en Oracle ya sea en Forms, Reports y en procesos se permite elaborar enunciados para evitar que información que no es adecuada para nuestra base de datos sea introducida.
  - Aserción(ASSERTION) Es igual a la anterior pero puede afectar a varios elementos.

AUTOR

autor_id	autor_nombre	autor_fechanac
5	Javier Chávez Rodríguez	25/06/1965
11	Jose Manual Molina	22/10/1963

LIBRO

libro_ISBN	libro_titulo	libro_autorid	libro_aniopublic	libro_descripción
1-2156486-5-5	Backs	5	1996	Sample-Pleas
1-4569853-5-0	El Verano	6	1996	Novela
3-2356984-5-5	Cakes, Pie	7	1992	

Fig. 7 Clave ajena de la tabla LIBRO que referencia con AUTOR

1.6 Modelo Entidad - Relación(E - R)

El modelo conceptual de datos es el primer paso para el diseño de bases de datos, se lleva a cabo durante las etapas de análisis y diseño del ciclo de desarrollo de sistemas.
El objetivo del modelo conceptual de datos es desarrollar el modelo entidad - relacion que representa los requerimientos de información.En el diseño de bases de datos los requerimientos de información son refelejados en un modelo entidad - relacion.
Pero que es el modelo entidad - relacion?. Un modelo entidad entidad - relación (E-R) es una forma efectiva para integrar y documentar los requerimientos de información de cualquier empresa, en un formato preciso y claro, da una imagen clara del alcance del alcance de los requerimientos de información, ademas de que puede ser desarrollado y mejorado. Al ser una forma gráfica es fácil de entender al proveer una estructura adecuada para la integración de múltiples aplicaciones.

1.6.1 Componentes del modelo entidad relación.

- Entidades: Son los aspectos importantes acerca de los cuales se necesita conocer información, una clase o categoría de cosas. Aspectos como EMPLEADO, DEPARTAMENTO, PROYECTO pueden ser representativos de entidades.
- Relaciones: Representan la asociación entre dos entidades. Una relación es bidireccional. Como ejemplo la relación entre un INSTRUCTOR y un CURSO es: "Cada curso puede ser enseñado por uno y solamente un instructor"
  Los requerimientos de una relación son: el nombre, una opción y una cardinalidad.
- Atributos: Información específica que necesita ser almacenada. Los atributos describen entidades y son las piezas específicas de información que necesitan ser conocidas desde el punto de vista de la empresa. Ejemplos posibles de atributos de una entidad como empleado pueden ser: nombre, fecha de nacimiento, salario, etc.

1.6.2 Estandares de diagramación de entidades

Los siguientes son los estandares más comunes en este diagrama:

1. Cajas de cualquier dimensión con esquinas redondeadas
2. Un nombre único para cada entidad.
3. Nombre de la entidad en mayusculas y en singular
4. Nombre de sinónimo, entre parentesis(opcional)
5. Nombre de los atributos en minusculas.
6. La relación se dibuja como una línea puntada uniendo 2 entidades cuando es opcional
7. La relación se dibuja como una línea continua si la relación es obligatoria.
8. Respecto al grado de la relación, linea continua sin "pata de gallo" es relación una a una.
9. Respecto al grado de la relación, linea continua con "pata de gallo" en un lado y del otro simple; es relación muchas a una.
10. Respecto al grado de la relación, linea continua con "pata de gallo" en un lado y del otro "pata de gallo"; es relación muchas a muchas.

1.6.3 Ocurrencias

Cada entidad debe tener múltiples ocurrencias o instancias. Por Ejemplo la entidad EMPLEADO tiene una ocurrencia para cada empleado de la empresa: Isidro Lopez, Jorge Ubaldo y Alejandra Mercado.

Cada instancia de la entidad tiene valores específicos para cada atríbuto de la entidad.

1.6.4 Ejemplo de modelo (E - R)

Este esquema surge de la necesidad del cliente; esta necesidad es transformada a esquema por un analista en sistemas que por medio de diversas juntas trata de obtener el mayor cúmulo de información sobre las reglas del negocio del cliente y plantear el mundo completo que implica el negocio para darle un adecuado manejo a las necesidades de éste. En la tesis se maneja el siguiente ejemplo general de una base de datos de una librería. Y se maneja como ejemplo para ver la aplicación del modelo en las necesidades de esta librería. Y se da seguimiento en el uso de las instrucciones DDL y DML para su mejor entendimiento.
El diagrama esta a nivel general y en las instrucciones DML se maneja una parte del esquema y en las instrucciones DDL se maneja la otra parte.

Ejemplo. Se han contratado una consultoría de sistemas para crear un sistema capaz de manejar las necesidades del negocio de una librería. En base a las diversas juntas que se han tenido con los dueños de la librería se han obtenido datos relevantes y básicos para la creación del sistema.

Esta librería es grande y por consecuencia tiene varias sucursales en el Distrito Federal y continua en expansión. Tiene así mismo contrato con varias editoriales que son los que les van a proveer los libros.
Por otro lado cuenta con clientes importantes que adquieren una gran cantidad de ejemplares y por supuesto los clientes cotidianos que van a diario a adquirir un libro. Ellos por necesidad desean saber todos los datos de las editoriales con las que tiene contrato, desean saber los datos sobre sus clientes, los datos sobre sus empleados, los datos sobre los autores de los libros, los datos sobre las distintas sucursales que tienen en el D.F. y que puedan tener en otros lados, desean tener un control sobre los pedidos que hacen los clientes y cuántas unidades venden por semestre como un histórico de ventas.

Estas observaciones han sido determinadas como primordiales para fijar las directrices del sistema. En las distintas fases de la construcción del sistema se ha determinado como urgente construir la parte correspondiente al control de autores, editoriales, ventas y libros y como segunda fase la construcción de lo que corresponde a empleados, clientes y pedidos. Aún así se conceptualizó de la siguiente manera el esquema general del sistema y queda aparte las fases de construcción de éste(Fig. 8).

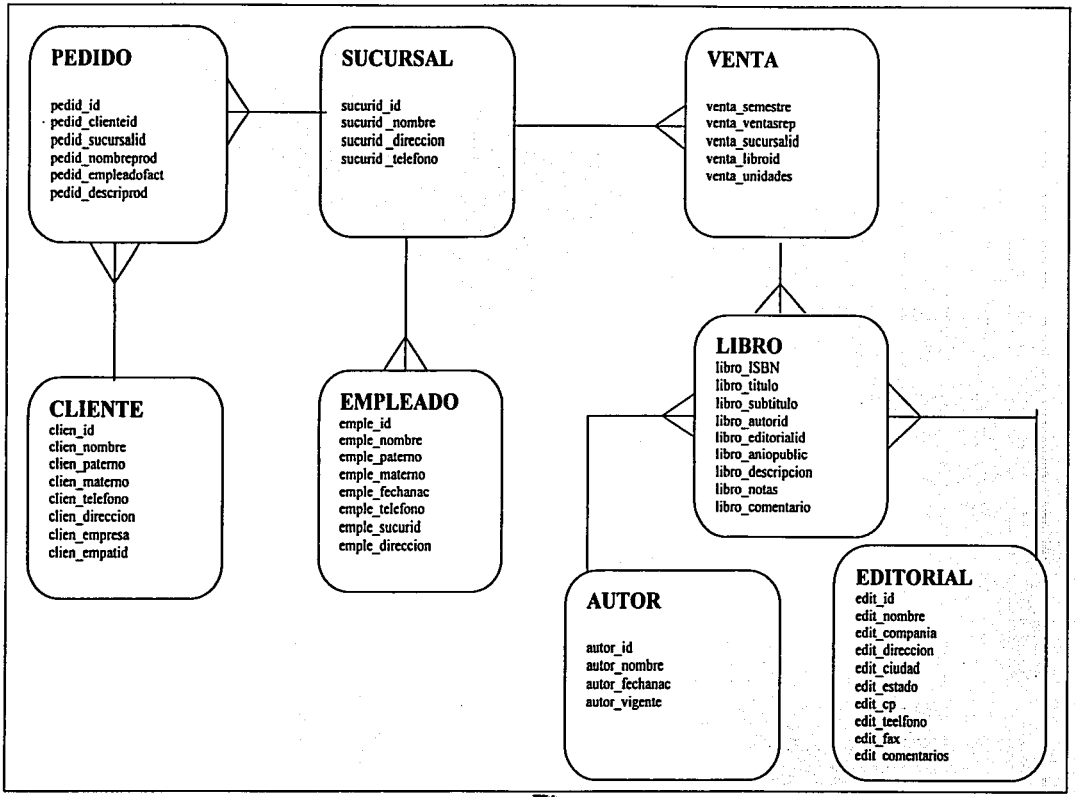

Fig. 8 Diagrama (E - R) de la base de datos Libros.

1.7 Sistema de Gestión de Bases de Datos(SGBD) o Sistema de Administración de Bases de Datos(DBMS)

Se analizará lo que es un SGBD o DBMS, junto con sus funciones, lenguajes, interfaces, herramientas y la gran relación que va a tener con el lenguaje SQL.

1.7.1 Definición .

El SGBD o DBMS son dos términos que se refieren a lo mismo, al sistema de gestión o administración de la base de datos, sólo que distintos autores manejan indistintamente estos términos. Este sistema agrupa varios programas, procedimientos y lenguajes que permiten manipular y describir los datos almacenados en una base de datos. La finalidad del SGBD es establecer las adecuadas interfaces entre los distintos tipos de usuarios y la base de datos.

1.7.2 Funciones

Tres Funciones son las básicas del SGBD, y son:

- Función de descripción o definición.
- Función de manipulación
- Función de control

La primera de estas funciones; la de definición es la que permite trabajar sobre la estructura de la base, la que permite especificar los elementos que la integran, como ejemplo los metadatos que son elementos que no sólo contienen datos sino también información relacionada con la base, permite especificar las relaciones que existen entre ellos, las reglas de integridad semántica. Esta función es realizada por el lenguaje de definición (DDL o Data Definition Language por sus siglas en Ingles o Lenguaje de Definicion de Datos) propia de cada SGBD, debe de dar las facilidades para realizar adecuadamente esta función.

La segunda función; la de manipulación es la que trabaja con lo que tuvo que haber hecho la primera función que fue la de crear toda la estructura, la integridad y las relaciones en la base de datos, ahora es necesario cargar los datos en la estructura previamente cargada para que la base de datos sea operativa. Pero los usuarios tendrán que hacer uso de los datos, recuperar los datos o de hacer cambios sobre éstos; posiblemente cambios de nombres, direcciones etc., en algún registro; entonces tienen que hacer actualizaciones en la base; todo esto es lo que se va a encargar de hacer la función de manipulación. Entonces las dos operaciones básicas son:

- La Consulta
- La Actualización

La primera de ellas; la consulta puede tener dos variantes; una consulta selectiva, donde en base a ciertas condiciones se va a tener información especifica, la segunda es una consulta a la totalidad de los datos. Normalmente se ocupa la consulta selectiva en base a las necesidades de información que se requiere; es muy común utilizar una serie de condiciones para poder recuperar información especifica.

Con respecto a la actualización; que se puede ver como la puesta al día de los datos se encuentran tres importantes formas de actualización y son:

1. Inserción
2. Borrado
3. Modificación de los datos

En la primera operación; con un ejemplo se puede entender esta operación. Cuando se contrata nuevo personal en una empresa, es necesario registrar a los nuevos empleados, si esta empresa posee una base de datos entonces es necesario dar de alta a los empleados o sea insertar nuevos registros en la base.
En la segunda operación, que es la de borrado continuando con el mismo ejemplo; si en esta empresa se ven en la necesidad de despedir algunos de sus trabajadores, entonces tienen que dar de baja en la base de datos a los empleados, entonces se borran datos de la base.
Finalmente en la operación de modificación de los datos, dando seguimiento al ejemplo que se planteo; si por ejemplo algunos de los trabajadores cambiaron de domicilio, entonces es necesario poner al día los datos de estos trabajadores; se hace una actualización o modificación a los datos. Esta función se llevara a cabo por medio de un lenguaje de manipulación de datos(DML Data Manipulation Languaje ó Lenguaje de Manipulación de Datos por sus siglas en Ingles).
La función de control reúne todas las interfaces que necesitan los diferentes usuarios para comunicarse con la base y da una serie de procedimientos para el administrador. Existen funciones de servicio, como cambiar la capacidad de los archivos, obtener las estadísticas de utilización, cargar archivos, etc., y sobre todo las relacionadas con la seguridad física copias de seguridad, rearranque en caso de caída y protección frente a accesos no autorizados.
Todos los lenguajes de los SGBD tienen que tener un enfoque distinto según el tipo de usuario, ya que las necesidades de un programador o del administrador van a ser distintas a las de un usuario final. Medios potentes requieren la gente encargada de la base y medios más flexibles y autocontenidos serán para personas que no necesiten el saber un lenguaje para poder extraer información de la base.
En el caso de los programadores y administradores, el conjunto de sentencias de manipulación del SGBD que pueden ser llamadas desde un lenguaje de programación, permitiendo así el acceso a la base de datos, se suele denominar sublenguaje de datos y también lenguaje huésped o lenguaje embebido. Los SGBD admiten varios lenguajes de tipo anfitrión para manipulación de los datos como Cobol, Ensamblador, Fortran, PL/I, Basic, C. La mayoría de los SGBD disponen de lenguajes de cuarta generación, poco procedimentales, que permiten el acceso a la base de datos, en general, mediante sentencias embebidas en el lenguaje de cuarta generación y escritas en un lenguaje de datos como SQL. En el caso del SGBD relacionales el SQL es un estándar muy extendido que proporciona todas estas facilidades.
En contrapartida el usuario final requerirá medios simples para comunicarse con la base, lo que se puede conseguir mediante un lenguaje de manipulación autocontenido que tenga una sintaxis sencilla.

1.7.3 Lenguajes

Basándose en las funciones; se tienen dos importantes. Los lenguajes de definición de datos y los lenguajes de manipulación de datos. En el caso de los lenguajes de definición de datos son lenguajes autocontenidos y no requieren de la aplicación o intervención de ningún lenguaje de programación.
Con respecto a la estructura lógica de la base el administrador deberá contar con instrumentos de descripción que le permitan asignar nombres a los campos, a los registros, establecer sus longitudes y características, así como las relaciones entre estos elementos y su integridad semántica. Normalmente el SGBD podría encargarse de (a partir de la

estructura global) definir la estructura interna adecuada sin intervención del usuario (administrador); pero donde si permite hacer o cambiar algunas partes de la configuración. Con respecto a la estructura externa de la base de datos el SGBD deberá poner a disposición de los usuarios medios que le permitan recuperar o actualizar los datos contenidos en la base.
Por lo tanto los lenguajes de definición de datos sirven para describir la información elemental y la manera como esta estructurada la base de datos.
Hablando de los lenguajes de manipulación de datos tienen el objetivo principal de facilitar el manejo de la información. Es la función de manipulación la que permite orientarse a determinados conjuntos de datos(información específica) en base a un criterio basado en condiciones. Con estas condiciones el SGBD debe ocuparse de acceder al soporte físico de donde se van a tomar los datos e indicarle una salida a un dispositivo.
Es importante retomar el hecho de que la base de datos tiene diferentes tipos de usuarios, y es importante disponer de distintos instrumentos para satisfacer las necesidades de cada uno. Así como el programador necesita de un lenguaje de manipulación que se embeba en un lenguaje de programación, el usuario que no es informático deberá tener un programa con el mismo objetivo pero mucho mas sencillo. Por esto el SGBD tiene lenguajes autocontenidos que ofrecen muchas facilidades a los usuarios que no poseen muchos conocimientos de computación.
En el siguiente ejemplo de una sentencia de un lenguaje autocontenido –SQL- en la que la descripción de los datos a recuperar se realiza de una manera muy simple, solamente se indica el nombre de los campos y el criterio de selección que es una expresión lógica.

```
SELECT nombre, apellido

FROM persona

WHERE fecha_nacimiento ='01-JAN-00'
```

En esta sentencia de SQL; permite recuperar todas las personas con fecha de nacimiento del primer día de enero del año 2000. Se puede notar que la sintaxis es muy sencilla; se indica solamente los campos a seleccionar, se le indica de que tabla y al final se ocupa un criterio de selección.
El lenguaje de manipulación puede actuar como huésped y como autocontenido, así ocurre, por ejemplo, con el SQL, que puede ser llamado desde un lenguaje anfitrión (por ejemplo C), o bien puede interaccionar directamente con la base de datos sin necesidad de depender en ningún lenguaje de programación. Esta es la razón por la cual se dice que el lenguaje SQL tiene la propiedad de dual o relacional uniforme. Se puede decir también que los lenguajes autocontenidos no son procedimentales o sea que el lenguaje tiene comandos que permite indicar lo que se quiere; pero el lenguaje oculta cómo lo hace. Por tanto SQL es poco procedimental, con uso en modo autocontenido o como lenguaje huésped desde un lenguaje anfitrión.

1.7.4 Características Importantes

1. Eliminan la redundancia de datos y aumentan la uniformidad de los mismos.
2. Aumenta en gran medida la flexibilidad para soportar los cambios y satisfacer las nuevas necesidades
3. Permite el compartimento de recursos entre varios usuarios
4. Permite una mejor estructuración de los archivos, lo cual hace posible un acceso más rápido a los archivos que se integran, relacionan y actualizan.
5. Da soporte a una gran diversidad de usuarios desde los más expertos en cómputo hasta los que no tienen mucho contacto con computadoras y que sólo requieren recuperar información de la base de datos para satisfacer sus necesidades.

Se mostró las características principales de las bases de datos para ubicar de manera correcta donde se hace necesario el lenguaje SQL; como se indicó este lenguaje opera en este entorno y por tanto es necesario conocerlo.
Las características principales de las bases de datos incluyendo su definición fueron manejadas en este capítulo y ya en este punto se puede abordar el uso del lenguaje SQL en un entorno que ya es más natural para el lector.

Capitulo 2 Selección de datos mediante SQL. Instrucciones DML.
Introducción
Se Inicia esta segunda sección de la tesis que corresponde al objetivo principal de ella que es mostrar el uso de las principales instrucciones del lenguaje SQL. Éstas están divididas en dos importantes categorías; instrucciones DML e instrucciones DDL.
Hay tres estándares del lenguaje SQL SQL-89, SQL-92, SQL3; en la tesis se abordaran principalmente el SQL-89 y SQL-92. Se manejara una misma base de datos para explicar cada instrucción con ejemplos y así hacer más fácil su comprensión y su seguimiento.
En este capítulo se hablará sobre el uso y la aplicación de las instrucciones DML(Data Manipulation Language ó Lenguaje de Manipulación de Datos).
El lenguaje de manipulación de datos es la estructura SQL usada para manipular datos de la base de datos. Es aquí donde se dice que se utiliza el lenguaje para hacer consultas; para obtener información que se necesita; con estas instrucciones no se modifica la estructura de la base; únicamente se trabaja con los datos. En un medio normal de trabajo los desarrolladores o programadores ocupan estas instrucciones para corroborar lo que van a programar.
Pero ¿Qué es el SQL?, ¿Qué significa?, ¿Cuáles son sus orígenes?.
SQL significa **L**enguaje de **C**onsulta **E**structurado. Fue desarrollado en los años setenta en IBM para permitir a los usuarios el uso de instrucciones estandarizadas en gran cantidad de bases de datos. El objetivo que se perseguía era crear un lenguaje que no estuviese basado en ningún otro lenguaje de programación, pero que realmente éste se pudiera utilizar en cualquiera de ellos para actualizar y consultar la información de la base de datos. El lenguaje SQL surge con el nombre de SEQUEL (**S**tructured **E**nglish **QUE**ry **L**anguage) implementado en un prototipo de IBM, el SEQUEL-XRM, durante los años 1974-1975. Este prototipo fue creciendo y evolucionó durante los años 1976 y 1977, cambiando el nombre al de SEQUEL/2, y cambiando su nombre por motivos legales al de SQL. El lenguaje continuo evolucionando; pero cada plataforma tenía el suyo; tenía que hacerse un estándar. Uno de los más importantes fue el SQL-89 que hasta la fecha se sigue utilizando. En 1992 se aprueba como norma internacional una nueva versión del SQL, conocida como SQL2 ó SQL-92. En la actualidad se acaban de aprobar nuevas propuestas para extender el SQL; el llamado SQL3 dotándolo de una mayor capacidad; donde se desea un soporte para bases de datos multimedia. En general el lenguaje continúa avanzando y adaptándose a las nuevas bases de datos que van saliendo al mercado. Es por esto que el lenguaje es potencialmente útil y es muy actual.
Las instrucciones SQL son sólo eso: instrucciones. Cada instrucción puede realizar la tarea, operaciones en las tablas, índices, etc. de diversas bases de datos. A pesar de todo el lenguaje SQL no es un lenguaje particularmente amigable al principio y se puede incrementar un poquito la dificultad si no se tiene conocimiento del idioma inglés. Por ello, muchos programas que permiten el uso de SQL intentan facilitar la generación de instrucciones mediante cuadros de diálogo, formularios y otras interfaces que facilitan el trabajo. Es importante saber que si los datos se encuentran en una base de datos relacional, se utilizará instrucciones de SQL ya sea de manera directa o indirecta.
Para verificar que la codificación que van a realizar les va a dar la información que necesitan. Tanto en reportes como en formularios de consulta deben de estar apoyados en este tipo de consultas a la base de datos. Algunas palabras comunes DML son **select, insert, update** y **delete**. Al terminar este capitulo se podrán utilizar instrucciones SQL para hacer consultas a la base de datos, así como reorganizar los datos y dado que SQL se utiliza

en casi todos los motores de bases de datos relacionales(SQL Server, Oracle, Informix, etcétera), se podrá aplicar en cualquier entorno de bases de datos.
En este capítulo se aprenderá a utilizar el comando SELECT para seleccionar los datos de una o más tablas y presentarlos en pantalla como si realmente fueran de una sola tabla. Se aprenderá el comando FROM para indicar el origen de los datos que vamos a manipular; el comando WHERE para indicar las condiciones que se tienen que hacer uso para limitar nuestro conjunto de registros y unir tablas con datos relacionados y se podrá ordenarlos con el comando ORDER BY. Se aprenderá a utilizar instrucciones que totalicen datos mediante el comando GROUP BY. Otra parte importante que se verá en este capítulo son las funciones que SQL utiliza para manejar cadenas y números. Es un capitulo con mucha información; pero con ejemplos y por supuesto en la práctica se aprenderá perfectamente todo esto. Es un capitulo muy interesante.

2.1 El comando SELECT y la cláusula FROM

Este es el comando más usado en SQL. Este comando permite obtener registros de una o más tablas. Este comando da la forma de recuperar los datos de una base de datos. Esta orden dice simplemente a la base de datos que información tiene que seleccionar para su recuperación. En concreto los campos de la tabla que se necesita que se muestren.
En su forma más simple, un comando SELECT tiene dos secciones:

1. Una lista de columnas(o campos) por seleccionar. Esta parte es obligatoria
2. Una lista de tablas de dónde obtener estos campos. Esta parte también es obligatoria

La sintaxis básica de una consulta de selección es la siguiente:

SELECT Campo1, Campo2, Campo 3,.....

FROM Tabla1, Tabla2,Tabla3,.....

Los campos van separados por comas; el último campo no la lleva. Sucede lo mismo con las tablas. Donde Campo(s) puede ser uno, dos o más campos de la tabla(s) que esta al lado derecho de la cláusula FROM. La cláusula FROM es la otra parte que conforma la forma más simple del comando SELECT; como se había dicho anteriormente estas dos partes son obligatorias.

Y se puede entender de la siguiente manera(Esta no es sintaxis es sólo una forma de explicar lo que esta haciendo con el comando SELECT):

SELECCIONA LOS SIGUIENTES CAMPOS Campo1,Campo2,Campo 3

QUE PROVIENEN DE LA TABLA(S) Tabla1, Tabla2, Tabla3

Ejemplo: Se va a recuperar información de una tabla llamada LIBRO[2]; se va a hacer uso del comando SELECT y la cláusula FROM. La tabla tiene campos que hacen referencia al título de varios libros.

El objetivo es recuperar todos los títulos de los libros, su ISBN y el año de publicación de cada uno de la tabla de Títulos. Con esto ya se tiene una pista, en el objetivo ya se tiene descritos los campos que se requieren; o sea el título, el año de la publicación y el ISBN; además se sabe que está en la tabla de Títulos. Se va a ver ahora como es una parte de la tabla(Tabla 3); para tener una mejor idea de lo que se va a hacer.

[2] Ésta tablas están al final de la tesis en el apéndice A para que las puedan ver; aun así aquí se van a describir en algunas ocasiones la tabla en el mismo ejemplo; para su mejor entendimiento. La base de datos se llama Libros.

libro_ISBN	libro_titulo	libro_subtitulo	libro_autorid	libro_ editorialid	libro_aniopublic	libro_descripción	libro_notas
0-1597536-4-4	Oracle	Guía de Aprendiz.	1	3	1986	Libro de Consulta	
1-2156486-5-5	Backs	Sample	5	5	1996	Samples	
1-4569853-5-0	El verano	Lo que se fué	6	8	1996	Novela	Novela
1-5025896-6-5	The cellar		3	1	1980		

Tabla 3. Parte de la tabla LIBRO de la base de datos Libros.

Entonces la instrucción SQL sería así:

SELECT libro_ISBN, libro_titulo, libro_aniopublic FROM LIBRO

En fondo gris en la tabla se tienen los campos que se requieren y también el nombre de la tabla.Como resultado de la consulta a los registros que son mostrados en la Tabla 3 se obtendrían los siguientes datos:

libro_ISBN	libro_título	libro_aniopublic
0-1597536-4-4	Oracle	1986
1-2156486-5-5	Backs	1996
1-4569853-5-0	El verano	1996
1-5025896-6-5	The cellar	1980

Aquí no importa si son tres registros o son mil todos van a ser mostrados; puede que los registros vengan en desorden; pero más adelante se aprenderá como controlar estas situaciones para darle mayor potencialidad a la consulta.

Se puede hacer uso de un comodín dentro la instrucción SELECT para poder mostrar todos los campos de la tabla, sin tener la necesidad de escribirlos después del SELECT.
Por ejemplo en este caso se puede poner así la instrucción:

SELECT *
FROM LIBRO

Que puede indicarse en palabras de esta manera
SELECCIONA TODOS LOS CAMPOS
QUE PROVIENEN DE LA TABLA LIBRO
Como resultado de esta consulta se obtendría toda la información de la tabla.
A pesar de no haber indicado ningún nombre en esta instrucción SELECT se mostraron todos los campos. Esto es de mucha utilidad por ejemplo cuando se desconocen los nombres de los campos de la tabla o tablas.

A pesar de que esta instrucción es muy útil también puede llegar a ser peligroso. Si la vista obtenida por SELECT contiene demasiados registros, podrá disminuir en mucho la eficiencia de la red o el entorno en donde se esté trabajando, abarcaría mucha memoria y podría hasta bloquearse. Es por esto que la instrucción SQL lleva otra cláusula más que va a ayudar a limitar el conjunto de registros, que va a filtrar la información en base a ciertas condiciones.

2.2 La Cláusula WHERE

Uno de los aspectos más importantes de la instrucción SELECT es el que se refiere al condicionamiento de los registros que se quieren mostrar, al filtrado de la información que se requiere, siempre se va a necesitar la información en base a condiciones. Por esto es que existe la cláusula WHERE. Es importa decir que esta cláusula es opcional pero cabe decir que en la mayoría de las consultas para recuperar información se va a requerir el uso de esta instrucción. Como decimos nosotros los universitarios "Allá afuera" se utiliza constantemente.

La sintaxis de la instrucción SELECT junto con la cláusula WHERE es así:

SELECT Campo1, Campo2, Campo 3,.....

FROM Tabla1, Tabla2,Tabla3,.....

WHERE Condición1 (operadores) Condición2.....

Y se puede interpretar de la siguiente manera(No es sintaxis):

SELECCIONA LOS SIGUIENTES CAMPOS Campo1,Campo2,Campo 3

QUE PROVIENEN DE LA TABLA(S) Tabla1, Tabla2, Tabla3

EN BASE A LAS SIGUIENTES CONDICIONES Condicion1 operador Condicion2

Las condiciones van unidas por operadores como AND y OR.[3]

La cláusula WHERE tiene dos vertientes o dos formas de utilizarla:

1.- Para limitar la cantidad de registros obtenidos
2.- Para vincular dos o más tablas

[3] En Álgebra Relacional son operaciones básicas.

2.2.1 Uso de WHERE para limitar la cantidad de registros

Como se ha dicho esta cláusula permite llevar acabo comparaciones lógicas en los datos de cualquier columna de la tabla. En la forma más sencilla que se puede apreciar la instrucción es en la siguiente:

WHERE columna = valor

En este caso columna representa el campo que va ser filtrado o condicionado.
Y valor representa el valor literal que se requiere como 22,'Casa', etc.
Como se ha mencionado es bien importante que valor sea correspondiente al campo[4]; si el campo es de valor entero; se espera del lado derecho del operador igual un valor entero.

Hay que dejar bien claro que esta instrucción siempre va precedida del comando SELECT.

Ahora se va a hacer un ejemplo donde se limita la cantidad de registros. Primeramente con una sola condición; después se aumentaran más.

Ejemplo: Se va a recuperar el título del libro **libro_titulo**, al año de publicación **libro_aniopublic** y el ISBN **libro_ISBN** de la tabla de **LIBRO**; pero donde sólo se recupera aquellos libros que fueron publicados entre 1995 y 1996. Nuevamente ya se tiene los campos que se quieren mostrar; la tabla y el nuevo elemento que es la condición para sólo mostrar libros que fueron publicados en un determinado rango de años. Entonces la consulta SQL quedaría así:

```
SELECT   libro_ISBN, libro_titulo, libro_aniopublic,
FROM     LIBRO
WHERE    libro_aniopublic BETWEEN 1995 AND 1996
```

Se puede entender así:

```
SELECCIONA LOS SIGUIENTES CAMPOS     libro_ISBN, libro_titulo,libro_aniopublic
QUE PROVIENEN DE LA TABLA            LIBRO
EN BASE A LA SIGUIENTE CONDICION     libro_aniopublic  ESTE ENTRE EL RANGO DE
                                     1995 Y 1996
```

En este consulta se esta filtrando la información en base a un rango y por esto es que utilizamos un operador nuevo llamado BETWEEN que va a servir en la cláusula WHERE y solo ahí, para establecer un rango de valores en el que se desea que el campo incurra.

Se podría haber hecho de esta manera también:

```
SELECT   libro_ISBN, libro_titulo, libro_aniopublic
FROM     LIBRO
WHERE    libro_aniopublic>=1995 AND libro_aniopublic<=1996
```

[4] Columna o campo se refieren al mismo termino

Y el resultado va ser exactamente igual. Hay dos opciones y depende ya del lector cual de las dos formas utilizar. Aunque entre más se evite escribir es mejor, porque el código se hace más legible y con el tiempo se hace una practica común. A veces se hace fácil realizar un código extremadamente rebuscado y difícil de entender. Cuando se hace un mantenimiento a un sistema; estas son las mayores limitantes que existen para realizar una reingeniería al sistema. Se pide al lector programar de manera sencilla y práctica.

Esta consulta devuelve un subconjunto de datos de la tabla LIBRO;

Se obtendrían los siguientes datos[5]:

libro_ISBN	libro_titulo	libro_aniopublic
1-2156486-5-5	Backs	1996
1-4569853-5-0	El Verano	1996
2-0123458-9-7	Chocolate Lovers paradise	1995
4-5689654-4-5	The New Vineyard of Virginia	1996
5-6987456-8-9	Training New Chefs	1996
7-5698563-5-8	Susan Teaches Chardonnay	1995

Hay que notar que fueron recuperados registros correspondientes a los años de publicación de 1995 y 1996 que es el rango que se establecio en la cláusula WHERE.
Ahora se va a hacer otro ejemplo donde se maneje más de una condición y se haga uso de los operadores para unir condiciones. La tabla para este ejemplo es EDITORIAL.
Ejemplo: El objetivo es recuperar la **Compañía, la Dirección, la Ciudad** y el **Teléfono** de las publicaciones en las editoriales que provienen de la ciudad de Indianápolis y que el código de publicación de la editorial se encuentre entre el 1 y el 5

Analizando una parte de la tabla EDITORIAL(Tabla 4) para dar una mejor idea de lo que se va a recuperar.

edit_id	edit_nombre	edit_compania	edit_dirección	edit_ciudad	edit_estado	edit_cp	edit_telefono	edit_fax
1	Sams.	Sams Publishin	103W. 103rd	Indianapolis	IN	52364	5125556	5125556
2	Exit		103W.102nd	Indianapolis	IN	55632	5125695	5125554
3	Bill	Bill's Publishin	201West99th	New York	NY	10028	2125468	
4	Benson	Ben..on Book	123Main Str.	Albany	NY	45678	6545698	6542225
5	Billiard	The Billiard	201.Inglenook	Sonoma	CA	98563	4445555	4445648
6	Cork	Cork Publisher	212Main Str.	Sn Francisco	CA	98765	9018965	9015555

Tabla 4. Parte de la tabla EDITORIAL de la base de datos Libros.

La consulta entonces quedaría así:

```
SELECT  edit_id, edit_direccion, edit_ciudad, edit_telefono
FROM    EDITORIAL
WHERE   edit_ciudad = 'Indinapolis'  AND
        edit_id BETWEEN 1 AND 5
```

Se puede entender así:

[5] No se coloca toda la información que recupera la consulta; se colocan algunos de los registros que recupera pero que poseen las características que se deseaban en la consulta. Es importante recurrir a las tablas al final de la tesis en el apéndice A para su mejor entendimiento y visualizacion completa.

SELECCIONA LOS SIGUIENTES CAMPOS	edit_id, edit_dirección, edit_ciudad, edit_telefono
PROVENIENTES DE LA TABLA	EDITORIAL
EN BASE A LAS SIGUIENTES CONDICIONES	edit_ciudad = Indianápolis Y
	edit_id ESTE ESTRE EL RANGO DE 1 y 5

Hay que tomar en cuenta que al usar el operador AND para unir las dos condiciones se esta indicando que ambas condiciones se tienen que cumplir en cada registro, para que éste sea recuperado. No va a ser suficiente con una condición. Si fuera el operador OR entonces con una sola que se cumpliera sería suficiente para que el registro fuera mostrado.[6]
Otro aspecto a recalcar en la cláusula WHERE es que como el campo Ciudad es de tipo carácter entonces del otro lado aparece una cadena que es delimitada entre comillas simples.[7]

Como resultado de esta consulta se obtienen registros de publicaciones de Indianápolis y con un código de publicación entre 1 y 5.

edit_id	Edit_direccion	edit_ciudad	edit_telefono
1	103W. 103rd	Indianapolis	5125556
2	103W. 102nd	Indianapolis	5125554

Estos registros fueron recuperados porque cumplen con ambas condiciones tanto en el campo de Ciudad y en el campo de ID.

Se va hacer otro ejemplo donde se va hacer uso del operador LIKE y el operador IN. Estos operadores son bastante utilizados en la practica y es conveniente que se inicie a practicar con ellos.
Ejemplo: Esta vez el objetivo es recuperar el **Nombre**, la **Compania**, la **Dirección** y la **Ciudad** de la publicación de la tabla de EDITORIAL. Pero se desea que el **ID** o el código de publicación de editorial este entre los siguientes valores: 1,2,3,7,8,9 y que el **Estado** de la publicación inicie con una letra "I"

Entonces la consulta quedaría de la siguiente forma:

```
SELECT   edit_nombre, edit_compania, edit_dirección, edit_ciudad
FROM     EDITORIAL
WHERE    edit_id IN(1,2,3,7,8,9)  AND
         edit_estado LIKE 'I%'
```

[6] Se pueden utilizar tantos operadores AND y OR como se necesiten dentro de la cláusula WHERE. La mayoría de las plataformas también soportan el uso de los operadores BETWEEN, IN, y LIKE
[7] En Visual Basic se acepta el uso de comillas simples o dobles; en Oracle se usan las comillas simples; este tipo de detalles son los que se tienen que consultar dependiendo de la base o el manejador con que se este trabajando.

Se puede entender así:

SELECCIONA LOS SIGUIENTES CAMPOS	edit_nombre, edit_compania, edit_dirección, edit_ciudad
PROVENIENTES DE LA TABLA	EDITORIAL
EN BASE A LAS SIGUIENTES CONDICIONES	edit_id ESTE DENTRO DE LA SIGUIENTE LISTA DE
VALORES(1,2,3,7,8,9) Y edit_estado INICIE CON LA LETRA I	

El operador LIKE se puede entender como las siguientes frases "como, parecido a ,que inicie con", aunque la mas correcta es "como". El porcentaje es un comodín para indicar que después de la I no importa lo que tenga. Se puede usar antes, después o antes y después de una letra o una cadena. En el caso del operador IN es para indicar que el campo pueda tener cualquiera de los valores que están dentro del paréntesis. Si no existiera este operador imagínense lo que se tendría que hacer; quizá poner rangos o poner una condición por cada valor que necesitaremos. Son dos operadores muy útiles.

La consulta recuperó entonces:

edit_nombre	edit_compania	edit_direccion	edit_ciudad
Sams	Sams publishing	103 W. 103 rd	Indianapolis
Que	Que Publishing	103W. 102 nd	Indianapolis
Promise	Primise Press	225698 Old Tipton Road	Springfield

Podemos dar cuenta que seleccionó los dos primeros registros ya que están dentro de la lista de números de código de publicación y en su campo Estado inician con la letra I. Es importante decirles que el campo filtrado no es necesario que esté dentro de los campos que queremos mostrar en el SELECT. En este ejemplo no se muestra el campo Estado; pero si fue afectado por las condiciones.

2.2.2 Uso de WHERE para unir o vincular dos o más tablas.

Se va a conocer otro aspecto que agrega potencialidad a la cláusula WHERE que es la capacidad para unir dos o más tablas. Esta instrucción puede aprovecharse para comparar los campos de tablas distintas; lo cual permite obtener una vista[8] donde se unan los campos.

La sintaxis es así:

```
SELECT   tabla1.campo1, tabla2.campo1
FROM     tabla1,tabla2
WHERE    tabla1.campo1= tabla2.campo1
```

tabla1 y tabla2 son tablas distintas de una base de datos.Pero es más conveniente ver un ejemplo para entender que la sintaxis aunque después de ver el ejemplo es recomendable volver a ver la sintaxis y ya va a tener otro sabor; va ser más entendible.

[8] Vista se le llama al conjunto de datos recuperados de una consulta.

Ejemplo: En el siguiente ejemplo el objetivo es recuperar el **ISBN**, el **Titulo** del libro y la **Compañía** que lo publica. Se sabe que los dos primeros campos que se piden recuperar están en la tabla de LIBRO; pero donde se encuentra la **Compañía** que los publica está en la tabla de EDITORIAL. Entonces se tiene que recuperar información de dos tablas; pero información relacionada. Se puede ver que cada tabla tiene un campo donde ambas tablas confluyen.
Entonces la consulta quedaría así:

```
SELECT    LIBRO.libro_ISBN, LIBRO.libro_titulo, EDITORIAL.edit_compania
FROM      LIBRO, EDITORIAL
WHERE     LIBRO.libro_editorialid = EDITORIAL.edit_id
```

Es importante decir que en la tabla de EDITORIAL edit_id es la llave primaria y en la tabla de LIBRO es una llave foránea. En otras palabras se esta vinculando a la tabla de EDITORIAL por medio de la llave foránea de la tabla de LIBRO.
Por eso es que se pueden relacionar; primero porque siempre estos valores no van a ser valores nulos y porque además siempre una llave foránea es la que permite relacionar con otra tabla. Es conveniente conocer la base de datos bien, conocer los campos que permiten vincularnos con otras tablas, las llaves foráneas.
También hay que decir que a veces aunque no sean llaves foráneas aun así se puede vincular a otra tabla; pero es conveniente vincular dos tablas que tengan un campo común que tenga la misma información; por ejemplo pueden ser campos que tengan el código del empleado, campos que tengan el código del estado de nacimiento de una persona, campos que tengan el ISBN del libro; etc.
En este caso el campo edit_id representa al código de cada publicación de una editorial única y por eso se encuentra en la tabla de EDITORIAL como llave primaria ya que siempre va a ver códigos de publicaciones de editoriales diferentes. En la tabla de LIBRO se hace necesario saber a que publicación pertenece por esto existe el campo aquí; gracias a esto si se desea podemos obtener gran cantidad de información del titulo del libro y agregarle información acerca de la publicación; este es el enfoque de bases de datos relacionales.
La consulta se puede entender de esta manera:

```
SELECCIONA                    LIBRO.libro_ISBN, LIBRO.libro_titulo, EDITORIAL.edit_compania
PROVENIENTES DE LAS TABLAS LIBRO, EDITORIAL
EN BASE A LA CONDICION    DE QUE EL CAMPO DE LA TABLA LIBRO libro_editorilid = CAMPO
edit_id DE LA TABLA EDITORIAL
```

Esta es parte de la información que se recupera

LIBRO.libro_ISBN	LIBRO.libro_titulo	EDITORIAL.edit_compania
0-1597536-4-4	ORACLE	Bill's Publishing House
1-2156486-5-5	Backs	The Billiard Press
1-4569853-5-0	El Verano	Baiton and Waite Publishing
1-5025896-6-5	The Cellar	Sams Publishing
2-0123458-9-7	Chocolate Lovers	The Billiard Press

Es importante decir que al seleccionar columnas de mas de una tabla, se recomienda anteceder el nombre de la columna con el de la tabla que corresponde; esto no es necesario pero como buena práctica de programación se recomienda. En la consulta se puede ver que los campos EDITORIAL.edit_id y LIBRO.libro_editorialid no están incluidas a continuación del SELECT en la instrucción. Así es, que no es necesario incluirlas allí para

poder utilizarlas. Lo único que se necesita es que estas columnas existan en las tablas indicadas para que el WHERE funcione.

2.2.3 Alias

Se puede notar en las consultas que se han realizado; SQL le asigna el nombre a las columnas de las vistas. Esto a veces puede parecer confuso porque no se entiende que esta recuperando en esa columna o campo. Es una buena idea entonces asignarle sobrenombres o alias a los campos para que en la vista de la recuperación se pueda entender mejor a que se refiere el campo. Tiene ciertas variaciones en cuanto a las plataformas; pero son mínimas. Por ejemplo en Visual Basic la consulta con alias se tendría que usar la cláusula AS. Ejemplo:

```
SELECT    LIBRO.libro_ISBN AS ISBN, LIBRO.libro_titulo AS Titulo,
          EDITORIAL.edit_compania AS Compania
FROM      LIBRO, EDITORIAL
WHERE     LIBRO.libro_editorialid = EDITORIAL.edit_id
```

En el caso de Oracle utilizando SQL *Plus se utiliza con la cláusula AS. También es posible utilizarla sin ella sólo colocando el nombre del alias adelante del campo. Ejemplo:

```
SELECT    LIBRO.libro_ISBN ISBN, LIBRO.libro_titulo Titulo,
          EDITORIAL.edit_compania Compania
FROM      LIBRO, EDITORIAL
WHERE     LIBRO.libro_editorialid = EDITORIAL.edit_id
```

De todos modos es necesario consultar este tipo de detalles dependiendo de la plataforma en que se va trabajar.

2.3 Funciones agregadas de SQL.

Existe un conjunto de funciones importantes que siguen los estándares de SQL y por tanto pueden ser utilizadas en todos los sistemas que sigan este estándar. Las funciones más importantes son las siguientes:

- AVG Esta función tiene el objetivo de calcular el promedio de los valores que se encuentren en una columna o campo.
- COUNT Esta función como objetivo tiene contabilizar el número total de registros que existen en una tabla. Esta es una de las funciones más utilizadas.
- SUM Esta función tiene el objetivo de calcular la sumatoria de los valores de una columna o campo.
- MAX Esta función tiene como objetivo encontrar el mayor valor encontrado en la columna o campo
- MIN Esta función tiene como objetivo encontrar el mínimo valor encontrado en la columna o campo.

Estas funciones son de gran utilidad. Por ejemplo la función de COUNT es muy útil cuando se desea insertar un registro en la tabla y no se sabe en que posición insertarlo. Entonces se recurre a la función COUNT para encontrar la máxima posición de registros insertados. Con esto ya se sabe donde se puede insertar un valor sin el temor de borrar otro registro u obtener algún error en la base de datos. La función AVG es muy utilizada en tablas estadísticas y a veces cuando se desean graficar ciertos datos. También en aplicaciones

contables es muy usada. La función de SUM ni que decir; es utilizada en aplicaciones contables, financieras, de inventarios; en todos lados es común encontrar formulas que se basaron en la utilización de una función de sumatoria. Y finalmente la función de MAX Y MIN son tan comunes cuando se desea saber; por ejemplo la máxima venta en el año ó el vendedor que tuvo la comisión más baja. Realmente son situaciones que son comunes y que son usuales en aplicaciones de bases de datos. Por esto es importante verlas pero con ejemplos para entenderlas mejor.

Ejemplo: El objetivo en la siguiente consulta es recuperar de la tabla VENTA el promedio de **Unidades** vendidas, la **Unidad** mínima vendida, la máxima **Unidad** vendida y el recuento de las **Unidades** vendidas.

Analizando parte de la tabla VENTA(Tabla 5) para dar una mejor idea de lo que se va a recuperar.

venta_semestre	venta_ventasrep	venta_libroid	venta_unidades
1	A	3-2356984-5-5	10
3	B	3-2356984-5-5	116
1	E	3-2356984-5-5	10
1	B	3-2356984-5-5	10
1	D	3-2356984-5-5	11
1	D	3-2356984-5-5	10

Tabla 5. Parte de la tabla VENTA de la base de datos Libros.

Y la consulta queda de la siguiente forma:

```
SELECT  COUNT(venta_unidades) AS RecuentoUnidades,
        AVG(venta_unidades) AS PromedioUnidades,
        SUM(venta_unidades) AS SumatoriaUnidades,
        MAX(venta_unidades) AS UnidadMax,
        MIN(venta_unidades) AS UnidadMin
FROM VENTA
```

La salida de esta consulta es la siguiente:

RecuentoUnidades	PromedioUnidades	SumatoriaUnidades	UnidadMin	UnidadMax
199	16.73	3330	5	116

Aquí también se puede utilizar la cláusula WHERE.
Ejemplo: El objetivo es igual que la anterior consulta. La única diferencia es que ahora sólo va ser con los **ISBN** de los libros que no inicien con el número 9.

La consulta sería la siguiente:

```
SELECT  COUNT(venta_unidades) AS RecuentoUnidades,
        AVG(venta_unidades) AS PromedioUnidades,
        SUM(venta_unidades) AS SumatoriaUnidades,
        MAX(venta_unidades) AS UnidadMax,
        MIN(venta_unidades) AS UnidadMin
FROM    VENTA
WHERE   venta_libroid NOT LIKE ('9%');
```

La salida de esta consulta varía en cuanto a que ahora se suma todo lo que no tenga un ISBN que inicia con un número 9.

2.4 ORDER BY

Es la parte que faltaba en un bloque de SELECT. Esta cláusula va después del WHERE y va a servir para establecer un criterio de ordenación de la información que recuperemos en la consulta. Esta parte es opcional. Pero en muchas consultas va a ser muy útil para tener ordenada la información que se esta recuperando.

La sintaxis del bloque completo entonces queda así:

```
SELECT      campo1, campo 2, .....
FROM        tabla1, tabla2,.....
WHERE       condición1,condición2.....
ORDER BY    criterio de ordenacion
```

Ejemplo: El objetivo en esta consulta es recuperar El **Titulo** del Libro, el **ISBN**, El **Año de Publicación** y el **ID del Autor** de la tabla de **LIBRO** donde el **ID del Autor** esté entre el 1 y el 10 y se van a ordenar los registros recuperados en base al año de publicación.

Una parte de la tabla se encuentra en (Tabla 6)

libro_ISBN	libro_titulo	libro_subtitulo	libro_autorid	Libro_aniopublic	libro_editorialid	libro_descripción
0-1597536-4-4	Oracle	Guia de aprendizaje	1	1986	3	Libro de Consulta
1-2156486-5-5	Backs	Sample	5	1996	5	Samples
1-4569853-5-0	El verano	Lo que se fue	6	1996	8	Novela
1-5025896-6-5	The cellar		3	1980	1	
2-0123458-9-7	Chocolate	Chocolate	4	1995	5	Candy making
3-2356984-5-5	Cakes,Pie		7	1992	2	
4-5698765-6-3	The New Virginia		21	1996	7	Virginia's new
4-5698765-6-3	Cooking wi		30	1989	11	
5-5465254-5-1	Pasta and		1	1955	6	Cooking pasta
5-5689756-2-5	Tom's Fish		16	1976	9	Fish and Lobst
5-5698741-0-0	The Cajun	Cajun	4	1990	6	Spyci Foods
5-5698753-9-5	Vegetarian		22	1986	9	

Tabla 6. Parte de la tabla LIBRO de la base de datos Libros

La consulta queda así:

```
SELECT    libro_titulo, libro_ISBN, libro_aniopublic AS Año_Publicacion,
          libro_autorid AS Codigo_Autor
FROM      LIBRO
WHERE     libro_autorid BETWEEN 1 AND 10
ORDER BY  libro_aniopublic
```

La consulta se puede entender así:

```
SELECCIONA                    libro_titulo, libro_ISBN, libro_aniopublic, libro_autorid
PROVENIENTES DE LA TABLA      LIBRO
EN BASE A LA CONDICIÓN        DE QUE EL CAMPO libro_autorid de la tabla LIBRO SE
ENCUENTRE ENTRE EL RANGO DE 1 Y 10 TOMANDO EN CUENTA  1 Y al 10
ORDENALO POR                  libro_aniopublic
```

Y la salida es:

Libro_titulo	libro_ISBN	Año_publicación	Codigo_Autor
Pasta and More	5-5465254-5-1	1955	1
The Cellars	1-5025896-6-5	1980	3
Norm's Guide to Beer	6-5698745-5-9	1984	8
ORACLE	0-1597536-4-4	1986	1
Secrets of the Master	6-7894567-7-9	1989	10
The Cajun Enciclopedy	5-5698741-0-0	1990	4
The Summer Book of Chop	9-6133518-4-1	1990	4
Sample Title	9-7632147-4-1	1990	3
Cakes, Pies and More	3-2356984-5-5	1992	7
Chocolate Lovers Paradise	2-0123458-9-7	1995	5
Backs	1-2156486-5-5	1996	5
El Verano	1-4569853-5-0	1996	6
The Cellars of France	9-6120517-3-2	1997	6
El caballo de Troya	9-4566575-8-9	1997	6

Se puede notar que los registros fueron recuperados en base al criterio y ordenados de una manera ascendente. Si no se especifica si es ASCendente o DESCendente SQL por default ordena ASCendente. Si se quisiera que sea DESCendente entonces la instrucción quedaría de esta forma:

```
SELECT    libro_titulo, libro_ISBN, libro_aniopublic AS Año_Publicacion,
          libro_autorid AS Codigo_Autor
FROM      LIBRO
WHERE     libro_autorid BETWEEN 1 AND 10
ORDER BY  libro_aniopublic DESC
```

Y la salida es:

libro_titulo	libro_ISBN	Año_Publicación	Codigo_Autor
The Cellars of France	9-6120517-3-2	1997	6
El caballo de Troya	9-4566575-8-9	1997	6
El verano	1-4569853-5-0	1996	6
Backs	1-2156486-5-5	1996	5
Chocolate Lovers Para	2-0123458-9-7	1995	5
Chocolate	2-0123458-9-7	1995	4
Cakes, Pies and More	3-2356984-5-5	1992	7
The Summer Book of	9-6133518-4-1	1990	4
Sample Title	9-7632147-4-1	1990	3
The Cajun Enciclopedy	5-5698741-0-0	1990	4
Secrets of the Master	6-7894567-7-9	1989	10
ORACLE	0-1597536-4-4	1986	1
Norm's Guide to Beer	6-5698745-5-9	1984	8
The Cellars	1-5025896-6-5	1980	3
Pasta and More	5-5465254-5-1	1955	1

2.5 El predicado DISTINCT

Esta instrucción de SQL tiene el objetivo de seleccionar distintos registros de una tabla. Por ejemplo en la siguiente tabla LIBRO se va a ver que en el campo libro_editorialid aparecen los distintos códigos de publicación correspondiente a cada título. Puede haber un código de publicación para dos distintos libros. Si se quisiera recuperar los distintos códigos de publicación no se podría hacer con un simple SELECT. Entonces se tendría que hacer uso de DISTINCT.

Ejemplo: Recuperar los distintos códigos de publicación que existen en la tabla LIBRO (Tabla 7).

libro_ISBN	libro_titulo	libro_subtitulo	libro_autorid	libro_aniopublic	libro_editorialid	libro_descripción
0-1597536-4-4	Oracle	Guía de aprend.	1	1986	3	Libro de Consulta
1-2156486-5-5	Backs	Sample	5	1996	5	Samples
1-4569853-5-0	El verano	Lo que se fue	6	1996	8	Novela
1-5025896-6-5	The cellar		3	1980	1	
2-0123458-9-7	Chocolate	Chocolate	4	1995	5	Candy making
3-2356984-5-5	Cakes,Pie		7	1992	2	
4-5698765-6-3	The New		21	1996	7	Virginia's new
4-5698765-6-3	Cooking wi		30	1989	11	
5-5465254-5-1	Pasta and		1	1955	6	Cooking pasta
5-5689756-2-5	Tom's Fish		16	1976	9	Fish and Lobst
5-5698741-0-0	The Cajun	Cajun	4	1990	6	Spycl Foods
5-5698753-9-5	Vegetarian		22	1986	9	

Tabla 7. Parte de la tabla LIBRO de la base de datos Libros.

Y la consulta queda de la siguiente forma:

```
SELECT  DISTINCT  libro_editorialid AS Codigo_Publicacion
FROM              LIBRO
ORDER BY          libro_editorialid
```

La consulta se puede entender de la siguiente manera:

SELECCIONA LOS DISTINTOS libro_editorialid COLOCANDO EL ALIAS DE 'Codigo_Publicacion'
PROVENIENTES DE LA TABLA LIBRO
ORDENADOS EN FORMA ASCENDENTE POR libro_editorialid

La salida entonces en base a la tabla es:

Codigo_Publicación
1
2
3
4
5
6
7
8
9
10
11
14

Si en la consulta SELECT se incluye más de un campo, todos estos campos van a ser afectados por el DISTINCT

Ejemplo: Se va a observar que ocurre al realizar esta consulta SQL. Se ocupa la instrucción DISTINCT pero con dos campos en el SELECT. La tabla es VENTA.

```
SELECT DISTINCT   venta_libroid AS ISBN, venta_unidades AS Unidades_Vendidas
FROM              VENTA
```

La tabla la pueden ver al final de la tesis en el apéndice A[9].

Parte de la salida de esta consulta es:

ISBN	Unidades_Vendidas
3-2356984-5-5	10
3-2356984-5-5	11
3-2356984-5-5	12
3-2356984-5-5	13
3-2356984-5-5	14
3-2356984-5-5	26
3-2356984-5-5	36
3-2356984-5-5	116
6-5689654-2-8	11
6-5689654-2-8	12
6-5689654-2-8	22
6-5689654-2-8	24
ISBN	Unidades_Vendidas

[9] En muchos ejemplos se ha colocado la tabla porque se trabaja sobre el conjunto de registros que se muestra en el ejemplo. Aquí se va a trabajar sobre todos los registros de la tabla. Por eso es mejor que se vea la tabla al final de la tesis en el apéndice A para corroborar que la recuperación de la información corresponde con lo que esta en la tabla. En este ejemplo se muestra parte de la salida debido a que mostrar toda la salida abarcaría más de dos hojas. Con la salida mostrada es suficiente para corroborar con las tablas.

6-5689654-2-8	35
4-5698765-6-3	10
4-5698765-6-3	11
4-5698765-6-3	16
4-5698765-6-3	26
4-5698765-6-3	28
2-0123458-9-7	10
2-0123458-9-7	11
2-0123458-9-7	22
2-0123458-9-7	24
9-3213454-7-8	5
9-3213454-7-8	10
9-3213454-7-8	13
5-8978965-5-6	8
5-8978965-5-6	11
5-8978965-5-6	18
5-8978965-5-6	30

Se puede creer que el DISTINCT no hizo su trabajo adecuadamente puesto que se observa que hay registros que se repiten y se supone que para eso sirve la instrucción; pero hay que tomar en cuenta que el DISTINCT tiene ahora 2 campos que tienen que ser diferentes en cada registro y viéndolo así no se llega a la conclusión que cada registro tiene esos dos campos diferentes en cada registro; por ejemplo el ISBN 9-3213454-7-8 aparece tres veces ; pero en la primera vez aparece con 5 unidades, la segunda con 10 y la última con 13 unidades; entonces si hay diferencias de un registro a otro. Hay que tomar en cuenta que se toma en conjunto la distinción de los registros.[10]

2.6 Las cláusulas GROUP BY y HAVING.

Estas cláusulas van a trabajar en conjunto con las funciones que se vieron anteriormente como: SUM, MIN, MAX, COUNT y AVG. La mejor manera de explicar estas cláusulas es con ejemplos.

Ejemplo. El objetivo es obtener de la tabla VENTA las Unidades Vendidas de cada libro. Por tanto se tiene que agrupar por ISBN las unidades que se vendieron.

Entonces la consulta quedaría de la siguiente manera:

```
SELECT      venta_libroid, SUM(venta_unidades)  AS Vendidas
FROM        VENTA
GROUP BY    venta_libroid
```

SELECCIONA venta_libroid , SUMATORIA DE venta_unidades CON EL ALIAS DE Vendidas
PROVENIENTES DE LA TABLA VENTA
AGRUPADOS POR venta_libroid

[10] Si lo que se desea es recuperar registros donde todos su campos sean diferentes en todos sus registros se puede consultar la instrucción DISTINCTROW que trabaja de manera similar y en la misma posición. Aunque es muy poco utilizada esta instrucción. La más común es el DISTINCT.

En esta caso la consulta va a agrupar en base al ISBN y va a hacer una sumatoria de las unidades.[11]
La cláusula GROUP BY requiere al menos una columna numérica establecida en el SELECT sea el resultado de una función agregada.
Un consejo importante que sirve de mucho en la práctica real; cuando se tiene que agrupar por varios campos; en el SELECT deben estar todos los campos excepto el campo que esta afectado por la función agrupada tiene que ir en la cláusula GROUP BY. Si se revisa en el ejemplo anterior el campo que no esta afectado por la función agrupada SUM; es el de Titulos; entonces es el que se encuentra en el GROUP BY. También es importante notar que el campo que esta afectado por la función agrupada nunca va ir en el GROUP BY. En este ejemplo si se hubiera tratado de agrupar por Unidades y hacer la función agrupada de SUM sobre el campo nos hubiera mandado un error de agrupación.

Parte de la salida de esta consulta es:

ISBN	**Vendidas**
3-2356984-5-5	411
6-5689654-2-8	159
4-5698765-6-3	144
2-0123458-9-7	128
9-3213454-7-8	28
5-8978965-5-6	155
9-7001327-0-1	191
5-5465254-5-1	354
9-6133518-4-1	14
6-7894567-7-9	8
6-7894561-5-2	44
5-5698741-0-0	310

Revisando la cláusula GROUP BY puede y debe ser una de las instrucciones más utilizas en el lenguaje SQL. Pero aún así se va crear la necesidad de poner condiciones al campo que fue afectado por la función agrupada; que en este caso fue SUM( ). Si se intenta poner un WHERE para manejar condiciones en el campo afectado no se va a obtener otra cosa que un grandioso error. Para esto se tiene que hacer uso de otra instrucción creada para este propósito llamada HAVING que se puede ver como un tipo de cláusula WHERE pero que sólo va a trabajar con el campo afectado por la función agregada. Con esto no se quiere decir que el WHERE se va a omitir. El WHERE puede ser utilizado pero no con campos afectados por funciones.

Ejemplo: El objetivo es el mismo de la consulta anterior pero donde se desea que muestre a los Títulos con más de 80 Unidades vendidas.

[11] La tabla se encuentra al final de la tesis; en el apéndice A. El resultado se basa en los registros que están en la tabla de VENTA al final.

Una consulta incorrecta sería así:

```
SELECT      venta_libroid, SUM(venta_unidades)  AS Vendidas
FROM        VENTA
GROUP BY    venta_libroid
WHERE       SUM(venta_unidades)>80
```

Una consulta correcta aplicando la cláusula que tiene la función de trabajar con funciones agrupadas. La función es HAVING.

```
SELECT      venta_libroid, SUM(venta_unidades)  AS Vendidas
FROM        VENTA
GROUP BY    venta_libroid
HAVING      SUM(venta_unidades)>80
```

Parte de la salida de esta consulta es:

ISBN	**Vendidas**
3-2356984-5-5	411
6-5689654-2-8	159
4-5698765-6-3	144
2-0123458-9-7	128

La cláusula HAVING puede trabajar con los operadores AND, OR y NOT.
Además no es necesario que las columnas indicadas en la cláusula HAVING se encuentren en SELECT. Revisando en la consulta anterior la función tenía la siguiente sintaxis:

```
HAVING      SUM(suma_unidades)>80
```

O sea que ocupa la función agrupada en su sintaxis; por lo tanto no va a ser siempre necesario que ésta aparezca en el SELECT ya que puede establecer condiciones usando las funciones agrupadas al lado de ella. Esto le da la cláusula una gran flexibilidad que va ayudar de mucho.

2.7 Consultas SELECT anidadas (Subconsultas).

Este tipo de consultas al principio parecen un tanto complejas; pero con un poco de práctica se va facilitando el uso de ellas. Se puede decir en pocas palabras que es utilizar el resultado de un SELECT para ocuparlo como condición dentro de otro SELECT que se encuentra más externo. O mejor dicho una consulta basada en los resultados de otra.
Para entender mejor este tipo de consultas se va a realizar un ejemplo.

Ejemplo: El objetivo en esta consulta es obtener la fecha de cumpleaños de los autores que publicaron su libro entre 1990 y 1996. Se sabe que año de publicación del autor esta en la tabla de LIBRO; así como su código de Autor que va a permitir relacionar con la tabla de AUTOR donde se tiene la fecha de cumpleaños.

Entonces se puede hacer de dos maneras; una ya se aprendió que es en base al WHERE igualando el campo AUID que es común en las dos tablas y la otra forma es en base a una subconsulta. Entonces la subconsulta quedaría de esta manera:

```
SELECT   autor_fechanac, autor_id  AS Autor
FROM     AUTOR
WHERE    autor_id   IN (SELECT  DISTINCT libro_autorid
                        FROM LIBRO
                        WHERE libro_aniopublic BETWEEN 1990 AND 1996)
```

Y se puede entender de esta manera:

SELECCIONA autor_fechanac, autor_id CON EL ALIAS DE AUTOR
PROVENIENTES DE LA TABLA AUTOR
EN BASE A LA CONDICION DE QUE ELCAMPO autor_id SE ENCUENTRE Y COINCIDA CON
(SELECCIONA LOS DISTINTOS libro_autorid
PROVENIENTE DE LA TABLA LIBRO
EN BASE A LA SIGUIENTE CONDICIÓN libro_aniopublic SE ENCUENTRE EN EL RANGO DE 1990 y 1996)

Al examinar la estructura de las dos tablas se ve que los datos que se requieren están en dos tablas diferentes y se tiene como opción realizar una subconsulta.
La primera subconsulta obtiene el código de Autor "libro_autorid" de las publicaciones hechas entre 1990 y 1996; de la tabla de LIBRO. La segunda consulta obtiene la fecha de nacimiento de los autores que aparecen como resultado de la primera consulta. El operador que une a ambas consultas es IN y el campo que hace posible que una consulta se base en la otra es autor_id.Por esto es que estas consultas son llamadas consultas anidadas o subconsultas; puesto que la consulta más externa depende de lo que recupere la primera consulta.Una subconsulta SQL entonces; se compone de tres elementos primordiales: la comparación, la expresión y la instrucción SQL.
La comparación en este ejemplo es la consulta SELECT FROM AUTOR. La expresión se encuentra en el operador IN. La instrucción SQL es la consulta SELECT que se encuentra entre paréntesis.
La instrucción SELECT que se encuentra entre paréntesis es la que primero se ejecuta para que su resultado pueda servir de comparación con la consulta más externa. El operador IN le dice a la instrucción más externa para que tome sólo aquellos registros obtenidos por la subconsulta.
La salida de la consulta anterior es:

autor_fechanac	AUTOR
02/02/1969	4
25/06/1965	5
11/11/1966	6
12/12/1960	7
15/07/1965	9
22/10/1963	11
10/02/1972	12
autor_fechanac	AUTOR
15/04/1950	15
21/04/1949	21
12/08/1955	23
29/03/1976	35
15/04/1961	36

Es importante decir que puede variar el formato en que aparezca la fecha de nacimiento.[12] El grado de anidamiento depende de cada uno; aunque es recomendable no hacer más de tres consultas anidadas; SQL fue hecho para ser un lenguaje de uso fácil y hay que evitar hacerlo más complejo.

2.8 Union

Es una operación que se utiliza bastante en SQL. Con esta instrucción se puede crear como su nombre lo indica la unión entre dos tablas que tengan datos parecidos pero no relacionados. Una operación de tipo UNION es muy útil cuando se desea corroborar información obtenida de dos consultas en una sola vista.Este tipo de consulta es muy útil si se desea cotejar información de dos tablas al mismo tiempo o incluso de una sola aplicándole operaciones; Por ejemplo se podría necesitar ver los títulos más vendidos(mayor a 3000 unidades) y los títulos menos vendidos(menor a 1000 unidades) todo en la misma consulta.

```
SELECT SUM(venta_unidades) AS Unidades_Vendidas, venta_libroid ISBN
FROM VENTA
GROUP BY venta_libroid
HAVING SUM(venta_unidades)>3000
UNION
SELECT SUM(venta_unidades) AS Unidades_Vendidas, venta_libroid ISBN
FROM VENTA
GROUP BY venta_libroid
HAVING SUM(venta_unidades)<1000
ORDER BY Unidades_Vendidos
```

La consulta se puede entender de la siguiente manera:

SELECCIONA DE LA SUMA DE TODO EL CAMPO(venta_unidades) CON EL ALIAS DE Unidades_Vendidas
PROVENIENTES DE LA TABLA VENTA
AGRUPÁNDOLO CADA OPERACIÓN POR venta_libroid
DONDE EL RESULTADO DE LA OPERACIÓN SUM(venta_unidades)>3000
UNION CON LA SIGUIENTE CONSULTA
SELECCIONA DE LA SUMA DE TODO EL CAMPO(venta_unidades) CON EL ALIAS DE Unidades_Vendidas
PROVENIENTES DE LA TABLA VENTA
AGRUPÁNDOLO CADA OPERACIÓN POR venta_libroid
DONDE EL RESULTADO DE LA OPERACIÓN SUM(venta_unidades)<1000
ORDENANDO TODA LA VISTA POR Unidades_vendidas

Es importante ver que son prácticamente dos SELECTS unidos por el comando UNION; en cada uno de los SELECTS están haciendo uso de una operación de SUM para contabilizar las unidades vendidas en base al ISBN; por eso es que se agrupa por titulo y con una condición aplicada al resultada de la suma con la utilización del comando HAVING que si se recuerda es utilizado para condicionar resultados de operaciones como SUM, MIN,

[12] Depende en mucho de la plataforma. Por Ejemplo en Oracle aparece de la siguiente manera: 02-AUG-78; en Visual Basic aparece como se mostró en la salida de la consulta. Si se desea cambiar el formato se pueden utilizar mascaras. Hay que buscar la documentación correspondiente a la plataforma en la sección de fechas. Muchas veces se puede cambiar la configuración de la base por medio de sus variables de sistema.

MAX, etc. Por último el ORDER BY va dar la pauta para saber el orden del mejor vendido al peor vendido.
Algo importante que hay que recordar con esta operación es que cada porción de la UNION deberá contener el mismo número de columnas. Si la primera consulta contiene 10 columnas. la segunda deberá contener 10.
Para mantener el mismo número de columnas, SQL sobrescribirá el tipo de datos de cada una para insertar los resultados en aquellas columnas que no tengan el mismo tipo de dato.

2.9 Adición de registros con la Instrucción INSERT INTO

Este es un tipo de instrucción DML ya que no se afecta la estructura interna de la base de datos.Con el tipo de instrucciones DML se puede manipular la información almacenada; se puede borrar, actualizar o ingresarla; que es el caso de esta instrucción. Muchas veces las aplicaciones tendrán pantallas de captura donde se va a poder realizar todas estas operaciones y donde van a resultar muy amigables para el usuario. Pero otras veces se va hacer necesario el uso de instrucciones DML para realizar estas operaciones. Esto debido a que en ciertos motores de bases de datos es el único medio para hacer movimientos con los datos; otras veces porque va a dar mayor velocidad al transferir una gran cantidad de datos con sólo una línea de instrucción. Es muy común transferir datos de una tabla a otra o de una base de datos a otra; este tipo de instrucciones son perfectas para este tipo de tareas.
La instrucción INSERT INTO se utiliza para insertar filas o registros en las tablas. Se puede utilizar par agregar datos automáticamente, sin necesidad de pantallas de captura. A su vez se puede ejecutar tal adición automática con muy poco código del programa.
Tiene entonces su porqué el utilizar esta instrucción, tiene una gran utilidad y flexibilidad. Además se sabe que utilizando este tipo de instrucciones se asegura una compatibilidad puesto que es el estándar de SQL en manejo masivo de datos. En grandes bases de datos se manejan muchos procesos a nivel de bases de datos en que el usuario ni se entera que se están llevando a cabo y es muy común utilizar instrucciones como INSERT INTO en gran cantidad de ellos. Por añadidura, con INSERT INTO siempre será posible agregar datos, incluso cuando no exista pantalla de captura alguna.. Así se puede aprovechar esta instrucción para establecer datos iniciales a alguna tabla que lo requiera, para procesos internos como ya se había mencionado y sobre todo para mejorar el rendimiento de las aplicaciones.
La sintaxis básica de la instrucción INSERT INTO es:

INSERT INTO NombreTabla(Campo1,Campo2,.....) VALUES (Valor1, Valor2,.....);
Esta instrucción tiene tres partes importantes:

- NombreTabla que es la tabla donde van a ser insertados los nuevos registros.
- (Campo1,Campo2,..) son los campos que van a recibir la información que va a ser insertada.
- (Valor1, Valor2,.....) son los valores propiamente que van a insertarse en los campos en el orden respectivo que aparecen. En este caso Valor 1 sería insertado al campo Campo1 y Valor2 sería insertado al campo Campo2.

Es posible indicar tantos o tan pocos campos como se desee, pero con el detalle que la cantidad y tipos de datos de los valores tendrán que coincidir exactamente. A su vez se deberá listar los valores en el mismo orden que los campos.. El primer valor se asignará al primer campo, el segundo valor al segundo campo y así sucesivamente. Otro aspecto importante a cuidar cuando se insertan valores es lo que respecta a los campos que son

requeridos como obligatorios para contener información; por ejemplo una llave primaria. Si se trata de insertar un registro y se olvida poner un valor correspondiente al campo de la llave primaria; se va a obtener un error concerniente a la integridad de la base de datos. La misma base de datos se protege de este tipo de errores; pero se tiene que ayudar para evitar rupturas e interrupciones en los procesos y aplicaciones.
Pero ahora se va a ver un ejemplo de cómo se pueden realizar inserciones de datos en una tabla.

Ejemplo: El objetivo es insertar un registro en la tabla de Publicaciones; esto es porque se tiene una nueva publicación que se quiere insertar en la base de datos. La llave primaria de esta tabla es el campo editorial_id; como se sabe en este campo no puede haber información duplicada; el campo llega hasta el valor 11, entonces se va a insertar el registro 12.

INSERT INTO EDITORIAL(editorial_id, editorial_nombre, editorial_compania, editorial_direccion, editorial_ciudad, editorial_estado, editorial_cp, editorial_telefono, editorial_fax) VALUES(12,'Morgan', 'Morgan ED','Roma 24','D.F.','Mexico D.F.',08500,'56789876','7890987');

Se puede entender de la siguiente manera (No es sintaxis)
INSERTA EN LA TABLA EDITORIAL en los siguientes campos (editorial_id, editorial_nombre, editorial_compania, editorial_direccion, editorial_ciudad, editorial_estado, editorial_cp, editorial_telefono, editorial_fax) LOS SIGUIENTES VALORES(12,'Morgan', 'Morgan ED','Roma 24','D.F.','Mexico D.F.',08500,'56789876','7890987');

En este ejemplo se pueden analizar varias cosas importantes; la primera es que en el caso de los valores; los que son de tipo carácter o cadena se encierran entre comillas simples; caso contrario con los valores numéricos que no llevan comillas. Los valores van en el orden correspondiente al orden que tienen los campos. Así el valor 12 va ser insertado al campo editorial_id, el valor Morgan va ser insertado al campo editorial_nombre, el valor Morgan ED va ser insertado al campo editorial_compania; y así sucesivamente con cada uno de los valores. Es importante que los campos y los valores coincidan en número y en tipo de dato.
Y por último el detalle más importante es que el campo PubID que es la llave primaria (Primary key en Inglés)fué llenada por un valor que no es nulo[13] y que además no es igual a los que ya tenía almacenados.
No se necesita asignar datos a todas las columnas o en el mismo orden en que fueron creadas. Si hay columnas que no sean requeridas y que pueden dejarse con valores nulos, pueden omitirse en INSERT INTO.
Ver el siguiente ejemplo donde se ilustra este último punto.

[13] Se ha hablado mucho del nulo; el valor nulo es simplemente una variable o algo que no posee ningún valor: nada. No confundir con el valor de cero que si es un valor y nunca va ser nulo. No olvidar nulo indica que no hay nada.

Ejemplo: El objetivo es insertar un registro en la tabla de LIBRO; su llave primaria es el ISBN y los campos que son requeridos a llenarse sólo son: libro_ISBN, libro_autorid y el libro_editorialid que son llaves foráneas[14](Foreign Key en Inglés) a otras tablas y que por esta razón se hacen obligatorias llenarlas si no se quiere tener un error.

```
INSERT INTO LIBRO(libro_ISBN,libro_titulo,libro_autorid,libro_editorialid)
VALUES('9-9050665-6-7','El Laberinto de la Soledad',5,8);
```

En este ejemplo no se están llenando todos los campos de la tabla; solamente los que se considera necesarios para el ejemplo: pero lo importante es que dentro de estos campos necesarios se incluyen los campos que se hacen obligatorios por su condición de llaves primarias o foráneas.
Es importante decir que tantos registros se quieren insertar tantas veces se tiene que teclear el comando. Pero existen los ciclos que nos permiten realizar tareas repetitivas con pocas líneas. Este tipo de casos ya le tocan a usted lector investigar dependiendo en la plataforma en la que va a trabajar; puesto que esto es dependiente del entorno de trabajo.

2.10 Creación de consultas de modificación con la instrucción UPDATE

Esta instrucción es la que se utiliza para actualizar información en la base de datos. La instrucción UPDATE permite modificar una gran cantidad de datos en una o más tablas de manera muy rápida y con poco código. Esta instrucción se utiliza para modificar datos existentes en tablas y una gran ventaja que puede tener que con sólo escribirla una sola vez, puede modificar varias columnas en una tabla.
Hay muchos casos en la vida real en que se hace necesario este tipo de instrucciones; puesto que sin ellas se tendrían programas gigantes en cuanto al código y lentos en cuanto a la velocidad. Por ejemplo es muy común que en las aplicaciones que corresponden al pago de la nomina de los empleados cada año se tengan variaciones en cuanto al sueldo, al impuesto, a sus prestaciones etc. Pero son a veces 1000 empleados; a veces son 15000; se puede imaginar teniendo que actualizar todos estos registros uno por uno; aunque el programa sea con ciclos la velocidad sería extremadamente lenta. Con instrucciones como UPDATE se trabaja con un conjunto de registros y con unas dos o tres líneas de código se lograría esto y con una gran velocidad.
La sintaxis de esta instrucción en su manera más simple es la siguiente:

```
UPDATE  NombreTabla SET
Campo1 = Valor1,
Campo2 = Valor2.....
```

Un punto importante es que pueden ser tantos campos como se desee; con sus correspondientes valores. Y en la actualización la asignación de valores se mantiene con el operador de igual '='. Para separar cada asignación se utiliza la coma.

En el siguiente ejemplo se analizará lo que se puede conseguir con este tipo de instrucciones.

---

[14] Al igual que las llaves primarias; las llaves foráneas se hacen obligatorias como campos a llenarse. Las llaves foráneas son las que nos permiten tener una relación con otras tablas; poder compartir información y es el campo que se va a hacer uso en la cláusula WHERE para obtener información de dos o más tablas.

Ejemplo: El objetivo es actualizar la tabla de LIBRO en el campo de libro_notas; se desea ponerle a todos estos libros que tienen una revisión del año 2000
La consulta quedaría de la siguiente manera:

```
UPDATE LIBRO SET
libro_notas = ' Revision correspondiente al 2000';
```

Se puede entender de la siguiente manera:

```
ACTUALIZA LA TABLA LIBRO  ASIGNANDO
Libro_notas = ' Revision correspondiente al 2000';
```

Y con esta sencilla consulta se logra actualizar todos los registros de la tabla LIBRO en el campo de libro_notas. Con estas dos líneas se tiene una gran potencialidad para manipular datos en una forma muy rápida y práctica.

En el siguiente ejemplo se van afectar tres columnas de una misma tabla.
Ejemplo: El objetivo es actualizar los campos libro_notas, libro_subtitulo y libro_comentario de la tabla LIBRO. En el campo de libro_notas se va a colocar el texto "Ultima Revision", en el campo de libro_subtitulo se va a colocar el texto "Sin Subtitulos" y en el campo libro_comentarios se va a colocar el texto "En Preventa"

La consulta queda de la siguiente manera:

```
UPDATE LIBRO SET
 libro_notas       = 'Ultima Revision' ,
 libro_subtitulo   = 'Sin Subtitulos',
 libro_comentario = 'En preventa';
```

Con esta consulta se consigue actualizar todos los registros de la tabla LIBRO en tres campos diferentes. Así sean mil ó diez mil registros: todos van a ser actualizados.

Aunado a estas grandes características de UPDATE; el comando posee otras más. Esta instrucción en su forma completa su sintaxis luce así:

```
UPDATE  NombreTabla SET
Campo1 = Valor1,
Campo2 = Valor2.....
WHERE
Campo 1 = Condición 1   (AND /OR)
Campo2  = Condición 2   (AND/OR).....
```

Hace uso del comando WHERE que va a permitir actualizar en base a las condiciones que ser requieran; así se puede afectar los registros que nos interesan sin afectar otros que no deben ser incluidos en la actualización.
Otro ejemplo con la instrucción completa de UPDATE .

Ejemplo: El objetivo es realizar una actualización a la tabla de LIBRO en el campo de libro_descripción; se desea colocarle la leyenda 'Ultima Publicacion'; pero solo a los Titulos de los libros donde el año de publicación sea mayor a 1990 y el identificador de la publicación pueda ser el 3,4,5,6,8,9,11,12.

La consulta quedaría de la siguiente manera:

```
UPDATE LIBRO SET
libro_descripcion = 'Ultima Publicacion'
WHERE libro_aniopublic > 1990 AND
        libro_editorialid IN(3,4,5,6,8,9,11,12);
```

Se puede entender de la siguiente manera:
ACTUALIZA LA TABLA LIBRO ASIGNANDO
libro_descripción = 'Ultima Publicación'
EN BASE A LAS SIGUIENTES CONDICIONES libro_aniopublic > 1990 Y
libro_editorialid ESTE DENTRO DE LA SIGUIENTE LISTA DE VALORES (3,4,5,6,8,9,11,12)

En este bloque de SQL se actualizó la tabla de LIBRO en un campo en base a dos condiciones utilizadas en el WHERE unidas por un operador AND. Con este pequeño bloque se pudo actualizar una gran cantidad de registros.
Y la pregunta es cómo se puedo saber si lo hizo correctamente?
Se ha aprendido a lo largo de la tesis como se puede hacer uso del SELECT para recuperar información de la base de datos. Entonces siempre que se busque verificar que la información ha sido utilizada correctamente se puede realizar una consulta para corroborar la información tomando el bloque de condición del UPDATE en el bloque de condición del SELECT de la consulta que se puede hacer para verificar.

2.11 Consultas para la eliminación de datos con DELETE

Este tipo de consultas se utiliza para eliminar una o más registros de la tabla. Esta operación como las dos anteriores es rápida y se puede apreciar mejor su potencialidad cuando trabaja con una gran cantidad de registros. Muchas veces se busca eliminar registros por ejemplo de empleados que ya fueron dados de baja hace mucho tiempo y ya no interesa tenerlos en la base de datos; o quizá para el mantenimiento de una tabla o tablas. Tiene mucha aplicación en un entorno de base de datos; casi a diario se realizan consultas de este tipo y es conveniente comenzar a utilizarla.

La sintaxis básica de esta instrucción es la siguiente:

```
DELETE   FROM      NOMBRE_TABLA;
```

La sintaxis completa de esta instrucción es así:

```
DELETE    FROM    NOMBRE_TABLA
WHERE  Campo1 =  Valor   (AND/OR)
          Campo2  = Valor   (AND/OR)
             ...              .....
```

En el primer tipo de sintaxis puede ser ocupada cuando se quiere borrar toda la tabla; en el segundo caso se ocupa cuando se desea borrar sólo determinados registros y no toda la tabla.
Uso de esta instrucción con un ejemplo.

Ejemplo: El objetivo es borrar los registros de la tabla de LIBRO donde el **Año de *Publicación*** *sea menor a* 1986.

La consulta quedaría de la siguiente manera:

```
DELETE   FROM LIBRO
WHERE   libro_aniopublic< 1986;
```

La consulta se puede entender de la siguiente manera:

```
BORRA LOS REGISTROS DE LA TABLA LIBRO
QUE CUMPLAN CON LAS SIGUIENTES CONDICIONES libro_aniopublic< 1986
```

Con esta sencilla instrucción se logra eliminar todos los registros que cumplan con esta condición. No importando si son diez o diez mil; todos son borrados. Se pueden utilizar tantas condiciones como se quiere; incluso subconsultas; todo va a depender de las necesidades de que se tengan.
Es importante ver en este tipo de instrucciones que es lo que se va a borrar o actualizar puesto que son instrucciones que afectan muchos registros. En el caso de DELETE en plataformas como Oracle se puede dar marcha atrás con una instrucción que se verá en el capitulo que le corresponde pero no en todas las plataformas es lo mismo así que es muy recomendable estudiar dos o más veces que es lo que se va hacer; es buena idea substituir el DELETE con un SELECT para ver que se va afectar y una vez que se esta seguro ahora sí ejecutar el DELETE.

Se han contemplado y cumplido los objetivos planteados en la introducción al capítulo; puesto que se analizaron los más importantes comandos de SQL correspondientes a las instrucciones DML o "Instrucciones de Manipulación de Datos."
Se vieron muchos ejemplos con la instrucción SELECT que es la que da pauta a todas las consultas agregándole más potencialidad con cláusulas como WHERE, ORDER BY.
Se manejo gran cantidad de operadores útiles en las consultas como LIKE, IN, BETWEEN.
Se utilizaron funciones agrupadas como SUM, MIN, MAX, AVG, etc. Se cubrió la parte correspondiente a como se puede agrupar conjuntos de datos trabajando en conjunto con las funciones agrupadas y condicionar los resultados de esas operaciones con HAVING.
Se cubrió el tema de las subconsultas, la forma de plantearlas y los resultados que entregaba. Instrucciones importantes concernientes a la actualización de los datos de la base de datos como lo fue la instrucción UPDATE, INSERT y DELETE y el manejo que se debe de tener con ellas. Obtención de información de dos o más tablas ya sea con datos relacionados (utilizando WHERE) o con datos no relacionados (UNION).
Con estos comandos y con práctica se puede desempeñar un buen papel en el entorno de la base de datos. Ahora viene el capitulo 3 ahora que se sabe como manipular datos; es conveniente saber como manejar la estructura de la base de datos con las instrucciones DDL; un capítulo muy interesante y complementario al estudio del lenguaje SQL.

Capitulo 3 Creación de Bases de datos con SQL. Instrucciones DDL.
Introducción.
En este capítulo se hablará del lenguaje de definición de datos que va a permitir realizar cambios a la estructura de la base de datos. En el capitulo anterior se mostró como seleccionar y recuperar información de la base de datos así como insertar o actualizarla; en pocas palabras el trabajo con los datos; por esto se les llamo instrucciones DML o "Instrucciones de Manipulación de Datos."

Con el lenguaje de definición de datos se usará el SQL para crear tablas, establecer relaciones y crear índices, con lo que se agregara más potencialidad a las aplicaciones. El lenguaje para la definición de datos, incluye diversos comandos para crear, organizar y modificar estructuras de tablas que más tarde formarán parte de la base de datos.

Las palabras clave de SQL que se mostraran en este capítulo pueden ser ocupadas en cualquier base de datos que cumpla con las normas de SQL. Con estas técnicas podrá crear bases de datos en casi cualquier motor disponible en la actualidad.

Por lo tanto los objetivos planteados en este capítulo son:

- Crear y eliminar Tablas con palabras clave como CREATE TABLE y DROP TABLE
- Crear y eliminar índices con CREATE INDEX y DROP INDEX
- Alterar tablas y otros elementos de la base.
- Manejo de restricciones
- Definir relaciones, entre ellas las claves externas, mediante PRIMARY KEY y FOREIGN KEY

Es importante recalcar que no se va a cubrir la totalidad de las instrucciones DDL; pero si se puede tener la seguridad que se van a ver las instrucciones más ocupadas en un entorno de bases de datos; instrucciones con las cuales se va a dar forma a una base de datos.

También hay que tomar en cuenta que cada base de datos le agrega ciertos comandos a estas instrucciones, para darle mayor potencialidad y flexibilidad cuando se trabaja con su base; sin embargo con las instrucciones que se verán se asegura portabilidad y ya si deseamos contribuir con las instrucciones que da el propio manejador de la base con un poco de investigación se lograra ya que ya se cuenta con la base.
Además de que en los dos últimos capítulos se verá una introducción a dos bases importantes como es la de Oracle y la de Access con Visual Basic con lo que se completara la tesis y así mismo el estudio básico de este importante lenguaje.

Con la intención de facilitar el aprendizaje de esta sección; este estudio se basó en el análisis de la base de datos de la librería a manera muy general utilizando tablas que van surgiendo de este análisis; donde se hacen necesario que las tablas como la de "EMPLEADO" y "CLIENTE" sean incluidas dentro de la estructura de la base para el mejor manejo de la librería y poder ampliar más la capacidad de la aplicación. Por supuesto el objetivo fundamental es aplicar las instrucciones en un medio que sea lo más real posible para ver el alcance, la potencialidad, los pro y contra de cada una.
Sin embargo es importante decir que cada ejemplo se debe tomar diferente al anterior; puesto que se puede llegar a pensar que a una misma tabla se le aplican todos los comandos y esto no es así; cada ejemplo es diferente y se debe tomar como tal aunque sea el nombre de una misma tabla.

3.1 Instrucciones DDL y la Administración de la Base de Datos

Como se ha mencionado las instrucciones DDL (Data Definition Language o Lenguaje de Definición de Datos) son las instrucciones que se van a emplear para manipular la estructura de la base de datos. Muchas de estas instrucciones suelen ocuparlas los Administradores de Bases de Datos(DBA) y algunas otras los programadores o desarrolladores con la autorización del DBA; puesto que una alteración en la estructura de la base representa cambios y ajustes que hay que tomar muy en cuenta. Lleva consigo un previo análisis que queda fuera del objetivo de la tesis; pero es importante saber que tiene base teórica y un análisis a fondo tomando en cuenta lo que ya esta hecho y lo que se va a realizar. Sin embargo en el capítulo 1 se hace un pequeño análisis de la base de datos de una librería que es el que se esta manejando para ocupar todas las instrucciones. Se indica de manera práctica como ejemplo real. Desde que se crea la necesidad de la librería hasta la creación de tablas. Es buena idea darle una revisada otra vez a esta sección para manejar de mejor manera esta sección.

El uso de estas instrucciones a través de SQL permite aprovechar dos importantes características llamadas legibilidad y portabilidad. La propia sintaxis de SQL permite analizar y asimilar con mayor facilidad la manera en que se diseño la base de datos y si se posee un poquito de vocabulario del idioma inglés todavía se hace más fácil el trabajo con estas instrucciones.

Es muy común incluso que con ver la secuencia de instrucciones para crear la estructura de una base de datos se pueda tener una idea general pero adecuada de cómo fue hecha la base. Una de las más importantes características de estas instrucciones además de las ya mencionadas es que se facilita generar y probar tablas con las instrucciones SQL. Si se trabaja en el diseño de una base de datos, y aún se trabaja en la forma como se va estructurar las tablas y sus relaciones; es fácil generar una secuencia de comandos DDL que en muchos lados se les llama "script", ejecutarlo y continuar con la depuración.
Este tipo de "script" es muchas veces un archivo de texto que contiene varios comandos DDL en el orden en que se deben ir ejecutando. Otras veces es una guía para la creación de la estructura de la base.
Muchas veces estas tablas se llenan de datos con instrucciones DML para probar si se trabaja adecuadamente con la estructura que se creo o se tiene que modificar. Es realmente muy común hacer cambios a la estructura de la base; a veces las necesidades del sistema exigen la creación de tablas; el borrado de otras, el crear más relaciones, el agregar campos, el quitar otros; entonces los "scripts" son muchas veces modificados y ajustados a las nuevas tablas o relaciones.

Este tipo de instrucciones SQL por tanto son para administrar tablas. Entonces este tipo de palabras clave permitirán crear nuevas tablas, alterar la estructura de tablas existentes y eliminarlas de la base de datos.
Es importante no olvidar que este tipo de instrucciones se podrán ocupar en cualquier base de datos que se apegue al estándar SQL.

3.2 Como crear una tabla. Diseño de nuevas tablas con CREATE TABLE

3.2.1 Creación de la tabla.

Para crear nuevas tablas en una base de datos existentes se utiliza el comando CREATE TABLE . En su forma más básica, este comando consta de tres partes: las palabras CREATE TABLE: un nombre de tabla y una lista con los nombres, tipos y tamaños de cada columna de la tabla.

La sintaxis es:

```
CREATE TABLE  Nombre_de_la_Tabla
( Nombre_Campo_1  Tipo_1  NOT NULL,
  Nombre_Campo_2  Tipo_2  NOT NULL,
  .....
  Nombre_Campo_n  Tipo_n  NOT NULL );
```

Con respecto a la anterior sintaxis se pueden citar algunas cosas importantes:

- Esta instrucción puede expresarse en una sola fila o en muchas; al igual que las instrucciones DML; esta instrucción no le afecta los espacios o la forma como se escriba la instrucción: mientras cumpla con el estándar SQL de que todas las instrucciones SQL terminen con punto y coma.
- CREATE TABLE es la palabra clave reservada para la creación de tablas.
- Nombre_Campo_1, Nombre_Campo_2, Nombre_Campo_n son los nombres de los campos que va a contener la tabla; es importante recordar que los nombres de los campos no pueden ir separados por espacios: es una buena idea utilizar el guión bajo como recurso para describir con detalle lo que representa el campo.
- Tipo_1 y Tipo_2 , etc., representan los tipos de datos procesados; en el punto 3.21 se verá algo al respecto.
- NOT NULL es una instrucción opcional que prohibe introducir un valor nulo en el campo seleccionado. Ello señala que el campo sólo debe contener un valor. Ya se había dado comentarios sobre esta característica; aun así mas adelante se dará una explicación más detallada.
- Esta instrucción hace uso del paréntesis después del nombre de la tabla y antes del punto y coma que indican finalización de la instrucción.

3.2.2 Tipos de datos

Esta es una breve descripción de los tipos de datos que soporta SQL; no olvidar que cada plataforma le agrega potencialidad a esta característica también teniendo otros tipos de datos con características muy especiales dedicadas a sus plataformas específicamente. Si se va a trabajar en plataformas como Oracle o Access; aparte de lo que se expondrá en los capítulos finales es bueno si se da una revisada a la documentación correspondiente a la plataforma para sacarle más provecho. Lo que aquí se expone corresponde al SQL estándar.

1. Cadenas de caracteres de longitud fija o variable. El tipo CHAR(n) denota una cadena de longitud fija de n caracteres.Ahora VARCHAR(n) indica hasta de n caracteres. El campo con tipo VARCHAR va almacenar cadenas de caracteres cuya extensión fluctúa entre 0 y n caracteres. Por ejemplo si la cadena 'foo' se transforma en el valor de un componente de un campo de tipo CHAR(5), entonces asume el valor 'foo' con dos espacios en blanco después de la segunda letra "o".
2. Cadenas de bits de longitud fija o variable. Se asemejan a las cadenas de caracteres de longitud fija o variable, pero sus valores son cadenas de bits y no de caracteres. El tipo BIT(n) denota cadenas de bits de longitud n, mientras que BIT VARYING(n) indica cadenas de bits con una longitud de hasta n.
3. El tipo INT o INTEGER (Son sinónimos) denota valores enteros típicos. El tipo SHORTINT también denota enteros, sólo que el número de bits permitidos puede ser menor según cada implementación(como el caso de los tipos int y short int en el lenguaje C).
4. Los números de punto flotante pueden representarse en diversas formas. Podemos emplear el tipo FLOAT o REAL para designar números flotantes comunes. El tipo DOUBLE PRECISION permite una mayor precisión; una vez más la distinción entre estos dos tipos es igual que en el lenguaje C. El lenguaje SQL cuenta demás con tipos que son números reales con un punto decimal fijo. Por ejemplo, DECIMAL(n,d) admite valores formados por n dígitos decimales; con el punto decimal a "d" posiciones a partir de la derecha. Así, 0123.45 es un valor posible del tipo DECIMAL(6,2).
5. Las fechas y las horas pueden representarse mediante los tipos DATE y TIME, respectivamente. Estos valores son esencialmente cadenas de caracteres de una forma especial. De hecho, se puede imponer a las fechas y a las horas el tipo de cadena; también lo contrario, si la cadena "parece adecuada" como fecha u hora.

3.2.3 Valores NULL y NOT NULL

Si se puede observar la definición de la sintaxis para crear tablas; se puede ver que algunos campos tienen "NOT NULL". Esto indica que la base de datos no aceptará una fila de datos en la tabla a menos que las columnas o campos así definidos contengan datos. En otras palabras, las columnas NOT NULL(no vacías) son campos obligatorios. Otra forma de interpretar el término NOT NULL es utilizar "obligatorio". Una columna NOT NULL significa que dicha columna o campo nunca puede estar vacía.
Respondiendo a la pregunta sobre que es un valor nulo (NULL) se puede decir que es un campo que no contiene ningún dato. Se puede pensar como una cadena de caracteres de longitud cero. Muchas veces, se introduce en una columna el valor nulo cuando éste es desconocido. El error más común que se comete es cargar el valor NULL en una columna numérica. El problema está en que 1 + NULL es NULL. Por tanto si accidentalmente carga valores NULL en un campo numérico, se puede muy fácilmente provocar un error.

3.2.4 Ejemplos

Como se menciono al inicio de la tesis los ejemplos son importantísimos para manejar un lenguaje como lo es SQL y las instrucciones DDL no son la excepción.

Ejemplo:
Debido a requerimientos de la librería se ha decidido tener una tabla con información de los empleados que laboran allí. Es importante para los dueños saber el personal que labora y las características de cada uno. Debido a esta situación la estructura de la base va a sufrir cambios. Gracias a un análisis de la información que requerían se determino crear la tabla con los campos que abajo se muestran. Había sido contemplada esta tabla en el diagrama entidad/relación como segunda fase y ahora tiene que ser construida la tabla.
Pero con la expectativa que puede cambiar. Se justifica el campo de fecha de nacimiento para determinar las edades de los empleados ya que es importante para la empresa tener gente joven: por supuesto que el nombre con apellidos es esencial por razones administrativas; así como medios para localizarlos que justifica la dirección y el teléfono; hay un código que indica un número único para cada empleado y finalmente el campo que indica la sucursal a la que pertenece puesto que la librería es grande y se compone de diversos establecimientos.
El objetivo en este primer ejemplo es crear una tabla llamada EMPLEADO que va almacenar el nombre del empleado, su código de empleado, el código de la sucursal a la que pertenece, su fecha de nacimiento, teléfono y dirección. Es importante recalcar que este ejemplo es simple para comenzar a entender las instrucciones DDL.

```
CREATE TABLE EMPLEADO
(
empl_id          INT(5),
empl_nombre      CHAR(50),
empl_paterno     CHAR(50),
empl_materno     CHAR(50),
empl_telefono    CHAR(10),
empl_sucurid     INT(2),
empl_fechanac    DATE,
empl_direccion   VARCHAR(200)
);
```

Esta es la forma más sencilla; pero ahora se va a indicar que el código del empleado así como la sucursal que pertenece al empleado son campos indispensables para la tabla; entonces la consulta quedaría así:

```
CREATE TABLE EMPLEADO
(
empl_id          INT(5) NOT NULL,
empl_nombre      CHAR(50),
empl_paterno     CHAR(50),
empl_materno     CHAR(50),
empl_telefono    CHAR(10),
empl_sucurid     INT(2) NOT NULL,
empl_fechanac    DATE,
empl_direccion   VARCHAR(200)
);
```

Analizando los dos ejemplos anteriores se ve la creación de una tabla en un entorno real. Este tipo de instrucciones como ya se ha mencionado viene respaldadas con un análisis

previo y sujeto a cambios conforme se avanza en la aplicación o en las presentaciones finales de los productos.
En el primer ejemplo todos los campos podían contener valores cero o nulos; en el segundo ya se esta obligando al usuario a colocar un valor en los campos empleado_id y empleado_sucursalid. Podría haber muchas razones para que se necesitara que este campos fueran digitados; una de ellas muy importante es porque los campos podrían conformar la llave primaria, quizá los campos fueran llaves foráneas o quizá es porque es valor clave para nuestra base de datos; situaciones como esta son las que vienen respaldadas por un previo análisis.
Ahora los campos empl_id y empl_sucursal fueron campos de tipo INTEGER ya que lo que se esperaba recibir en estos campos son valores numéricos o códigos. Es importante decir que esto es muy común cuando se trata de llaves foráneas ó primarias ya que con un número se puede ligar a otra tabla que podría contener tal vez la información de las distintas empresas en el caso del empl_sucursal.
Podría ser llave primaria empl_id porque además de ser NOT NULL u obligatorio con simple lógica se deduce que si la tabla se llama EMPLEADO y el campo se llama empl_id; entonces puede ser que este campo sea el que evite registros duplicados porque hay códigos de empleados únicos.

3.2.5 Eliminación de Tablas con la instrucción DROP TABLE

La instrucción DROP tiene la funcionalidad de borrar diversos elementos que conforman la base de datos. En este caso se va a ver la instrucción DROP TABLE. Esta instrucción es la que se utilizará para borrar tablas de la base de datos.
También se puede utilizar para eliminar tablas temporales.
Eliminar una tabla es una operación irreversible. Por tal razón, es importante verificar la estructura de la tabla y su contenido antes de suprimirla.

La sintaxis es la siguiente:

```
DROP TABLE  Nombre_Tabla  ;
```

La sintaxis es muy sencilla sólo hay que colocar el nombre de la tabla después de DROP TABLE y finaliza como todas las instrucciones en punto y coma. En etapas de análisis es muy común utilizarla; a veces hay tablas innecesarias, otras es porque se puede crear una tabla con diferentes características y que se hace necesario eliminar la anterior. Esta operación la realizan como parte de la administración y el análisis los DBA.

Ejemplo: Si se decidiera agregar otros campos a la tabla debido a requerimientos del análisis; una opción sería borrar la tabla de EMPLEADO que se creó en el ejemplo anterior para crear una nueva. Nuevamente es bueno recordar que al construir una aplicación es muy sujeto a cambios y no se debe dar por hecho que una tabla o estructura siempre se va a mantener así.
La instrucción en forma quedaría de la siguiente manera:

DROP TABLE Empleados:

Es muy importante no omitir la palabra reservada TABLE; es muy común que se intente borrar la tabla sólo con la instrucción DROP; se tiene que recordar que DROP sirve para borrar distintos elementos y no sólo tablas; por tanto es buena idea revisar dos veces la instrucción .

3.3 Definiciones de Vistas

3.3.1 Creación de Vistas

Las tablas que se crean con el uso de la instrucción CREATE TABLE realmente existen en la base de datos. Es decir, el sistema SQL almacena tablas en una organización física. Las tablas son persistentes: cabe suponer que duren tiempo indefinido y que no cambien, salvo que un comando INSERT, UPDTE o DELETE les ordenen explícitamente que lo hagan.

Hay otra clase de relaciones en SQL, llamadas vistas, que no existen físicamente. En realidad se definen con una expresión muy semejante a lo que es una consulta. Las vistas a su vez pueden consultarse como si existieran físicamente, y en algunos casos hasta es posible modificarlas.

El uso de vistas también es muy común en un entorno de bases de datos cuando no se desea crear una tabla. La realidad es que se gana espacio pero se pierde rapidez ya que la vista se tarda en cargar. La vista esta sin datos hasta que la llamamos: es entonces cuando se dispara la consulta y se comienzan a cargar datos en memoria. Como se había mencionado antes; se requiere de un previo análisis para establecer que nos conviene más; si la creación de una nueva tabla o la creación de una vista; se tiene que analizar las características actuales de la base y el soporte que pueda tener para la estructura que deseemos crear. Un uso común de las vistas es para dar soporte a los programas de reportes; muchas veces se desea que el reporte sea bastante complejo y es aconsejable generar una vista con múltiples "joins" o tablas ligadas para generar este tipo de reportes y evitar así construir demasiadas tablas y demasiados procedimientos para llenar estas tablas.

La forma simple de definir una vista es

1. Las palabras clave CREATE VIEW
2. El Nombre de la vista
3. La palabra clave AS
4. Una consulta. Esta es la definición de la vista. Siempre que se consulte la vista, el lenguaje SQL se comporta como si la consulta se ejecutara en ese momento y como si se aplicase la consulta a la tabla producida por ella.

La sintaxis de la vista sería la siguiente:

```
CREATE VIEW Nombre_Vista AS Definición_de_la_Vista
```

Ejemplo: Los dueños de la tabla desean tener determinados reportes de su personal para darle un pequeño presente el día de su santo. Para esto necesitan reportes diarios por nombre de persona; donde indique su nombre y su teléfono. Se van crear unas vistas para cumplir con este objetivo y servir de base para el reporteador que más adelante se va a definir.
El objetivo es construir una vista que simule la tabla que anteriormente construimos; la de empleados del inciso 3.7. de todas las personas que se llaman "Francisco".
La definición de la vista queda así:

```
CREATE VIEW  VISTA_EMPLEADOS AS
SELECT  empl_nombre, empl_telefono, empl_paterno
FROM    EMPLEADO
WHERE  empl_nombre='Fernando';
```

En el análisis que se puede hacer del ejemplo anterior; primero que nada observamos que:

- La vista se basa en tablas que ya están construidas.
- Crear una vista significa crear un espacio de memoria que va almacenar una consulta con SELECT, sencilla o compleja.
- El nombre de la vista es VISTA_EMPLEADOS.
- Los campos de la vista son definidos en la segunda línea de la consulta enseguida de la palabra reservada SELECT
- La definición de la vista completa se da a partir del SELECT hasta finalizar con punto y coma.

Los datos de la vista pueden ser consultados de la misma manera de que son consultados en una tabla.

Ejemplo: El objetivo es recuperar el Nombre y el Teléfono del empleado de la vista VISTA_EMPLEADO.

```
SELECT  empl_nombre AS Nombre, empl_telefono AS Telefono
FROM    VISTA_EMPLEADO
```

Si se recuerda la vista, tenía una condición en que sólo iba a almacenar los empleados con nombre "Fernando". Al recuperar los datos se va a obtener el mismo nombre pero con diferentes teléfonos.
Si los datos de la tabla original son actualizados, borrados, etc.; entonces por consecuencia los datos de la vista cambiaran; recordar que la vista se puede ver como un SELECT que se ejecuta cada que se carga la vista.
Es importante decir que se puede trabajar con la vista igual que una tabla; en ciertos casos va ser posible relacionarlas y obtener un conjunto de datos de una tabla o muchas tablas con una vista o muchas vistas.

3.3.2 Cambio de nombre (renombramiento) de los campos de la vista.

Algunas veces se podría preferir asignarles nombres más adecuados acorde a la información que tienen a los campos de una vista, en vez de usar los nombres que provienen de la definición de ella por la consulta.
Se puede especificar los campos de la vista listándolos. encerrados entre paréntesis. después del nombre de la vista en la proposición CREATE VIEW

Ejemplo: El objetivo es crear la misma vista pero con la definición de campos que se adecue más a las necesidades de la librería o a los conceptos de ella; entonces la creación de la vista queda así:

```
CREATE VIEW  VISTA_EMPLEADOS(
Nombre,Telefono,Paterno) AS
SELECT  empl_nombre, empl_telefono, empl_paterno
FROM    EPLEADO
WHERE empl_nombre='Fernando';
```

Los nombres van en la correspondencia del orden de los campos.

3.3.3 Borra Vista

Para borrar una vista sigue el mismo procedimiento que cuando se borra una tabla:

```
DROP VIEW Nombre_Vista;
```

3.4 Llaves en SQL

El tipo más importante de restricción de una base de datos es una declaración de que cierto campo o conjunto de campos constituyen una llave de una relación. Es decir, se prohibe que dos registros de la tabla concuerden en un campo que ha sido declarado como llave o todo el conjunto de campos que se declaran que juntos forman una llave. Una restricción de llaves, como tantas otras, se declara dentro del comando CREATE TABLE de SQL.

3.4.1 Llaves Primarias

Hay dos formas parecidas de declarar llaves; por medio de la palabra reservada PRIMARY KEY o de la palabra clave UNIQUE. Es importante recalcar que una tabla puede tener sólo una llave primaria.

La llave primaria puede constar de uno o más campos de la tabla. Hay dos formas de declarar una llave primaria en la proposición CREATE TABLE.
La primera de estas declaraciones indica que se puede declarar que un atributo es una llave primaria, cuando está listado en el esquema relacional. En este caso se coloca la palabra PRIMARY KEY después del campo y su tipo.

En el segundo tipo de declaración se puede agregar a la lista de elementos declarados en el esquema; que hasta ahora sólo han sido exclusivamente campos, una declaración más que diga que un atributo particular o conjunto de atributos constituyen la llave primaria. En este caso la declaración se compone de la palabra clave PRIMARY KEY y una lista entre paréntesis del atributo o atributos de que se compone la llave.

Se puede reflejar estas dos situaciones en los siguientes ejemplos:
El objetivo es crear una tabla llamada EMPLEADO donde la llave primaria es el código del empleado.

Este ejemplo ilustra la primera definición.

```
CREATE TABLE  EMPLEADO (
empl_id  INT(5) PRIMARY KEY.
empl_nombre  CHAR(50),
empl_paterno  CHAR(50),
empl_materno  CHAR(50),
empl_sucurid  INT(2),
empl_telefono  CHAR(10),
empl_fechanac VARCHAR(200));
```

Este ejemplo ilustra la segunda definición.

```
CREATE TABLE EMPLEADO(
empl_id  INT(5),
empl_nombre  CHAR(50),
empl_paterno  CHAR(50),
empl_materno  CHAR(50),
empl_telefono  CHAR(10),
empl_sucurid  INT(2),
empl_fechanac VARCHAR(200)
);
PRIMARY KEY (empl_id)
);
```

La otra manera de declarar una llave que consiste en emplear la palabra UNIQUE. Esta palabra puede aparecer exactamente donde puede hacerlo PRIMARY KEY; tras un atributo y su tipo o como un elemento independiente dentro de una proposición CREATE TABLE.

3.4.2 Integridad Referencial y Llaves Exteriores o Foráneas.

Después de la llave primaria; viene otro importante tipo de restricción en el esquema de la base de datos donde los valores de algunos campos deben de ser lógicos o valores que ya se esperan de alguna manera.
La restricción implícita de "integridad referencial" establece que, sí un registro tiene un campo de una tabla A que esta relacionado con otro campo de una tabla B; entonces el

valor que espera el campo de la tabla A es un valor que sea de los que tiene la tabla B para que tengan una relación correcta.
En SQL entonces se puede declarar que un campo o campos de una tabla es una llave foráneo exterior, al hacer referencia a un campo o campos de una segunda tabla; esto lleva a dos consideraciones importantísimas.

1. El campo o campos referenciados de la segunda tabla ha de declarase como llave primaria de su relación.
2. Cualquier valor que aparezca en un campo de la llave foránea de la primera tabla también habrá de hacerlo en el campo correspondiente de la segunda. En otras palabras, se da una restricción de integridad referencial que conecta los dos campos o conjuntos de campos.

Hay dos formas de declarar una llave foránea y son similares a las formas de declarar una llave primaria.
En el caso de que la llave es un sólo campo, se puede poner después de su nombre y tipo una declaración que "referencia" algunos campos (los cuales han de ser la llave primaria) de alguna tabla. La sintaxis de la declaración es:

```
REFERENCES  Nombre_Tabla   (Campos)
```

En el otro caso consiste en anexar a la lista de campos en una proposición CREATE TABLE una o más declaraciones donde se establezca que un conjunto de campos es una llave foránea. Se indica que después de la tabla y sus campos (que han de ser la llave primaria) a los que se refiere ella. La forma de la sintaxis es:

```
FOREING KEY Campos REFERENCES  Nombre_Tabla (Campos)
```

Ejemplo: Debido a requerimientos de la librería se ha decidido poner una tabla más para el manejo de los clientes; el plan de la librería es extenderse en poco tiempo a más sucursales y llegar a tener clientes importantes y frecuentes y para ellos es importante tener una tabla histórica con todos los clientes y así brindarles diferentes tipos de servicios a los clientes más frecuentes y hacer que los clientes nuevos regresen par pasar a ser clientes distinguidos.

Este análisis ha llevado a la conclusión de generar una tabla donde sean incluidos los datos personales del cliente como lo es el nombre, la fecha de nacimiento, el empleado que atendió al cliente y por supuesto un código único para cada cliente.

Esta tabla ya había sido tomada en cuenta en el diagrama entidad/relación en la segunda fase de construcción y ahora se tiene que construir.

Aunado a esto se decidió crear una tabla de pedidos donde se va a indicar los pedidos del cliente con sus características e incluir al empleado que facturó el pedido. Esta tabla esta a prueba y va ser creada; posiblemente más adelante sufra cambios.

El objetivo entonces es crear la tabla y establecer una llave foránea a la tabla de CLIENTE con la tabla de EMPLEADO.

```
CREATE TABLE CLIENTE(
clien_id            INT(5) PRIMARY KEY,
clien_nombre        CHAR(50),
clien_paterno       CHAR(50),
clien_materno       CHAR(50),
clien_telefono      CHAR(10),
clien_empresa       INT(2),
clien_dirección     VARCHAR(200),
clien_empatid       INT  REFERENCES  EMPLEADO (empl_id));

CREATE TABLE PEDIDO(
pedid_id                INT(5),
pedid_clienteid         INT REFERENCES CLIENTE(clien_id),
pedid_sucursalid        INT(2) REFERENCES SUCURSAL(sucur_id),
pedid_nombreprod        VARCHAR(50),
pedid_descriprod        VARCHAR(50),
pedid_empatid           INT REFERENCES EMPLEADO(empl_id));
```

Por supuesto que la tabla de EMPLEADO; en este caso ya debe de existir y la llave primaria debe ser el campo (empl_id). Para la tabla de pedidos ya deben de estar creadas las tablas de CLIENTE, SUCURSAL y EMPLEADO.
En la creación de la primera tabla se busca tener una llave foránea que permita saber que empleado atendió al cliente y obtener todos los datos de él.
En la segunda se añade el hecho de saber cual fue el cliente que solicitó el pedido.

La forma alterna es de la siguiente manera:

```
CREATE TABLE CLIENTE(
clien_id            INT(5) PRIMARY KEY,
clien_nombre        CHAR(50),
clien_paterno       CHAR(50),
clien_materno       CHAR(50),
clien_telefono      CHAR(10),
clien_empresa       INT(2),
clien_dirección     VARCHAR(200),
clien_empatid       INT,
FOREIGN KEY clien_empatid REFERENCES  EMPLEADO (empl_id));
```

Entonces en cualquiera de las dos declaraciones se puede decir que el significado es: siempre que un valor aparece en el campo clien_empatid de un registro Clientes, ese valor ha de aparecer también en el campo empl_id de la tabla de EMPLEADO. La tabla de PEDIDO puede ser también declarada de esta misma manera.

3.4.3 Mantenimiento de la Integridad Referencial.

Política de Omisión
Por omisión el lenguaje SQL tiene la política de que el sistema rechace toda actualización que viole la restricción de integridad referencial. Basándose en el ejemplo anterior se pueden analizar algunas situaciones:

Si se intenta insertar un nuevo registro en CLIENTE cuyo valor clien_empatid no es NULL y es un valor que no aparece en el campo empl_id de la tabla EMPLEADO. El sistema rechazará la inserción y el registro por consecuencia nunca será insertado.

Si se intenta actualizar un registro de la tabla de CLIENTE para convertir el campo clien_empatid en un valor NULO. Se rechaza la actualización y el registro permanece igual.
Si se intenta eliminar un registro de la tabla EMPLEADO y su campo empl_id aparece como el componente clien_empatid de la tabla de CLIENTE. Se rechaza eliminación y el registro se conserva en CLIENTE.

Si se intenta actualizar un registro de la tabla de EMPLEADO en una forma que cambie el valor de empl_id y que el valor sea el de clien_empatid. Se rechaza el cambio y el registro permanece intacto.

Política en cascada
Con esta política, cuando se elimina un registro de EMPLEADO, para mantener la integridad referencial el sistema eliminará los registros referenciados en CLIENTE.

3.5 Restricciones en los campos.
Este tipo de restricciones son las que limitan los valores que pueden aparecer en los componentes de algunos campos.

3.5.1 Restricciones no nulas.
Una restricción simple a asociar con un campo es la NOT NULL. Su efecto es rechazar los registros donde el campo es NULL.
Se le declara con las palabras clave NOT NULL colocadas después de la declaración del campo en una proposición CREATE TABLE.

3.5.2 Restricciones con el uso de la palabra reservada CHECK.
Hay restricciones más completas y que hacen tener mejor control de lo que se recibe como dato en un campo. Una de ellas es basándose en la palabra reservada CHECK, seguida de una condición entre paréntesis que ha de cumplirse con todos los valores del campo.

Es práctica común el uso de esta instrucción como una limitación simple sobre los valores; pero también se puede ocupar en la definición de llaves externas por ejemplo.

```
CREATE TABLE CLIENTE(
clien_id            INT(5) PRIMARY KEY,
clien_nombre        CHAR(50),
clien_paterno       CHAR(50),
clien_materno       CHAR(50),
clien_telefono      CHAR(10),
clien_empresa       INT(2),
clien_dirección     VARCHAR(200),
clien_empatid       INT REFERENCES EMPLEADO (empl_id)
CHECK(clien_empatid >100));
```

O en la misma declaración de los campos.

```
CREATE TABLE EMPLEADO(
empl_id  INT(5),
empl_nombre CHAR(50) CHECK(empl_nombre IN('Alejandra','Javier','Isidro','Omar')),
empl_paterno  CHAR(50),
empl_materno CHAR(50),
empl_telefono CHAR(10),
empl_sucurid  INT(2),
empl_fechanac DATE,
PRIMARY KEY (empl_id));
```

3.5.3 Restricciones de Dominios.

Es posible también restringir los valores de un campo declarando un dominio(última sección) con una restricción similar y luego declarándolo como el tipo de dato del campo. La única diferencia importante es que, cuando se intenta escribir una restricción acerca de una valor en un dominio, no se tiene nada para llamar a ese valor; en el estándar SQL2 se da como opción el uso de la palabra reservada VALUE que se refiere a un valor del dominio.

Ejemplo

```
CREATE DOMAIN DominioSexo CHAR(1) CHECK (VALUE IN('F','M'));
```

3.6 Creación de Índices

Un índice de un campo de una tabla es una estructura de datos que permite encontrar rápidamente los registros que poseen un valor fijo en ese campo. Los índices generalmente facilitan las consultas en las que el campo se compara con una constante. Cuando las tablas son muy grandes, cuesta mucho trabajo explorar todos los registros de una tabla para encontrar los que corresponden a una condición determinada (quizá unas cuantas).

El índice se puede ver como una tabla virtual que almacena los distintos valores que puede contener un campo; cuando se hace una búsqueda; está se realiza en la tabla virtual y no en la tabla física; gracias a esto se incrementa en gran medida la velocidad de búsqueda.

Aunque la creación de índices no forma parte de los estándares de SQL, incluido el estándar SQL2, en general los sistemas comerciales permiten al diseñador de la base de datos decir que el sistema debiera de crear un índice sobre cierto campo para una tabla determinada.

La sintaxis para crear un índice en una tabla ya definida en la siguiente:

```
CREATE [ UNIQUE ] INDEX índice
   ON tabla (campo [ASC|DESC][, campo [ASC|DESC], ...])
   [WITH { PRIMARY | DISALLOW NULL | IGNORE NULL }]
```

En donde:

Parte	Descripción
Índice	Es el nombre del índice a crear.
Tabla	Es el nombre de una tabla existentes en la que se creará el índice.
Campo	Es el nombre del campo o lista de campos que constituyen el índice.
ASC|DESC	Indica el orden de los valores de los campos ASC indica un orden ascendente (valor predeterminado) y DESC un orden descendente.
UNIQUE	Indica que el índice no puede contener valores duplicados.
DISALLOW NULL	No permite valores nulos en el índice
IGNORE NULL	Excluye del índice los valores nulos incluidos en los campos que lo componen.
PRIMARY	Asigna al índice la categoría de clave principal, en cada tabla sólo puede existir un único índice que sea "Clave Principal". Si un índice es clave principal implica que no puede contener valores nulos ni duplicados.

Se puede utilizar CREATE INDEX para crear un pseudo índice sobre una tabla adjunta en una fuente de datos ODBC tal como SQL Server que no tenga todavía un índice. No necesita permiso o tener acceso a un servidor remoto para crear un pseudo índice, además la base de datos remota no es consciente y no es afectada por el pseudo índice. Se utiliza la misma sintaxis para las tabla adjunta que para las originales. Esto es especialmente útil para crear un índice en una tabla que sería de sólo lectura debido a la falta de un índice.

```
CREATE INDEX MiIndice ON EMPLEADO (empl_id, empl_telefono);
```

En el ejemplo anterior se crea un índice llamado MiIndice en la tabla EMPLEADO con los campos empl_id y empl_telefono.
Se busca tener un índice en esta tabla porque va a ser muy consultada por la gente de la librería; el aspecto empleado para ellos es muy importante y por tal razón se crea el índice en la tabla para incrementar la velocidad de su aplicación.

En este ejemplo a partir de la creación del índice, las consultas en SQL que especifiquen un código y un teléfono serán ejecutadas por el procesador de consultas de modo que sólo se examinen los registros de empleados con el prefijo y el teléfono especificado, con un decremento concomitante en el tiempo necesario para contestar la consulta.

A menudo se cuenta con un índice multiatributos o multicampos. Este adopta valores para varios campos y localiza eficientemente los registros con esos valores para ellos. A primera vista parecería que los índices multiatributos son menos útiles que los de uno solo, ya que los segundos podrían usarse aun cuando no se le asignen valores a todos los atributos del primero. No obstante índices multiatributos identifican más eficientemente los registros deseados cuando es posible utilizarlos. En el ejemplo de arriba es un ejemplo de este tipo de índice.

CREATE UNIQUE INDEX MiIndice ON EMPLEADO (Empleado_Codigo) WITH DISALLOW NULL;
En el ejemplo anterior se crea un índice en la tabla EMPLEADO utilizando el campo empl_id, obligando que el campo empl_id no contenga valores nulos ni repetidos.
Si se quiere eliminar el índice, basta incluir su nombre en una proposición como la siguiente:

DROP INDEX MiIndice;

La existencia de un índice en un campo agiliza enormemente las consultas en que se especifica un valor de él.
La selección de índices es una de las partes más trabajosas del diseño de bases de datos, pues exige estimar cual será la combinación normal de consultas y otras operaciones sobre la base de datos. Si una tabla se consulta con mucha frecuencia en un determinado campo, conviene utilizar índices sobre los campos especificados. Si las modificaciones son la operación predominante, se debería ser muy conservador respecto a la creación de índices. Incluso entonces tal vez se mejore la eficiencia al crear un índice en un campo de uso constante. De hecho, como algunos comandos de modificación requieran consultar la base de datos (por ejemplo, el comando INSERT con una subconsulta seleccionar-de-donde o el comando DELETE con una condición), hay que calcular con mucho cuidado la frecuencia relativa de las modificaciones y las consultas.

3.7 Dominios
Hasta hora se han definido un tipo de datos para cada campo. Una alternativa consiste en definir primero un dominio, que es un nuevo nombre para un tipo de datos. También puede declararse en un dominio otro tipo de información; por ejemplo, un valor por omisión o las restricciones de valores.
Entonces cuando se declare el campo, se podrá poner después de su nombre el del dominio en vez de un tipo de datos. Varios campos pueden usar el mismo dominio integrándolos de un modo útil. Por ejemplo, si recurrimos al mismo dominio con dos atributos, se sabe que el valor de uno siempre puede ser el del otro.
La forma de una definición de dominio son las palabras clave CREATE DOMAIN, el nombre del dominio, la palabra clave AS y el tipo de datos.

La sintaxis es la siguiente:

```
CREATE DOMAIN  Nombre del Dominio  AS  tipo de datos ;
```

Ejemplo: El objetivo es crear un dominio llamado Tipos_doc que va a utilizar una descripción de tipo de dato CHAR(1) y con una restricción por medio de la cláusula CHECK.

```
CREATE DOMAIN Tipos_Doc CHAR(1)
CONSTRAINT Articulos_o_Libros
CHECK (VALUE IN('A','L'));
```

En este ejemplo se esta definiendo un dominio que consiste en la creación de un nuevo tipo de dato donde los campos que se declaren de este tipo podrán aceptar sólo un carácter y este carácter puede ser únicamente la letra "L" o la letra "A".
Las cláusula CHECK es muy ocupada en SQL para limitar a un cierto rango de valores específicos un campo y así poder manipular y saber que esperar en ese campo. Como se sabe la cláusula CONSTRAINT lleva un nombre y da una restricción alterando de cierta manera las condiciones de la tabla.

Por ejemplo si se quisiera usar este dominio como un tipo de dato; se podría utilizar cuando creo una nueva tabla; en este caso si se hiciera una derivación de la tabla EMPLEADO.
Ejemplo:

```
CREATE TABLE  EMPLEADO(
empl_id              INT(5),
empl_nombre          CHAR(50) NOT NULL,
empl_paterno         CHAR(50),
empl_materno         CHAR(50),
empl_telefono        CHAR(10) NOT NULL,
empl_suscurid        INT(2)      NOT NULL,
empl_categoría       Tipos_Doc,
PRIMARY KEY(empl_id),
UNIQUE(empl_sucurid));
```

En esta creación de tabla ya más en forma hay varias cosas que analizar; la primera de ellas que corresponde al objetivo de esta sección es donde se define a un campo empl_categoría como un tipo de dato Tipos_Doc que fue el dominio que se creo previamente; también hay que hacer notar el uso de NOT NULL en campos que se requieren que sean obligatorios, el uso de PRIMARY KEY que es una llave primaria que va a permitir que no haya registros duplicados y UNIQUE para un valor único, aunque no sea llave primaria.

3.8 La Cláusula CONSTRAINT

La cláusula CONSTRAINT va a permitir dar diferentes características a las tablas y restricciones para un adecuado manejo de los datos.

Se utiliza la cláusula CONSTRAINT en las instrucciones ALTER TABLE y CREATE TABLE para crear o eliminar índices. Existen dos sintaxis para esta cláusula dependiendo si se desea crear o eliminar un índice de un único campo o si se trata de un campo multiíndice. Como consejo importante si se va a utilizar el motor de datos de Microsoft, sólo podrá utilizar esta cláusula con las bases de datos propias de dicho motor.

Para los índices de campos únicos la sintaxis es la siguiente:

```
CONSTRAINT nombre {PRIMARY KEY | UNIQUE | REFERENCES tabla
externa
  [(campo externo1, campo externo2)]}
```

Para los índices de campos múltiples la sintaxis es así:

```
CONSTRAINT nombre {PRIMARY KEY (primario1[, primario2 [, ...]]) |
  UNIQUE (único1[, único2 [, ...]]) |
  FOREIGN KEY (ref1[, ref2 [, ...]]) REFERENCES tabla externa [(campo externo1
  [,campo externo2 [, ...]])]}
```

Parte	Descripción
Nombre	Es el nombre del índice que se va a crear.
PrimarioN	Es el nombre del campo o de los campos que forman el índice primario.
UnicoN	Es el nombre del campo o de los campos que forman el índice de clave única.
RefN	Es el nombre del campo o de los campos que forman el índice externo (hacen referencia a campos de otra tabla).
Tabla externa	Es el nombre de la tabla que contiene el campo o los campos referenciados en refN
Campos externos	Es el nombre del campo o de los campos de la tabla externa especificados por ref1, ref2, ..., refN

Si se desea crear un índice para un campo cuando se esta utilizando las instrucciones ALTER TABLE ó CREATE TABLE la cláusula CONSTRAINT debe aparecer inmediatamente después de la especificación del campo indexado.

Si se desea crear un índice con múltiples campos cuando se está utilizando las instrucciones ALTER TABLE o CREATE TABLE la cláusula CONSTRAINT debe aparecer fuera de la cláusula de creación de tabla.

Tipo de índice	Descripción
UNIQUE	Genera un índice de clave única. Lo que implica que los registros de la tabla no pueden contener el mismo valor en los campos indexados.
PRIMARY KEY	Genera un índice primario el campo o los campos especificados. Todos los campos de la clave principal deben ser únicos y no nulos, cada tabla sólo puede contener una única clave principal.
FOREIGN KEY	Genera un índice externo (toma como valor del índice campos contenidos en otras tablas). Si la clave principal de la tabla externa consta de más de un campo, se debe utilizar una definición de índice de múltiples campos, listando todos los campos de referencia, el nombre de la tabla externa, y los nombres de los campos referenciados en la tabla externa en el mismo orden que los campos de referencia listados. Si los campos referenciados son la clave principal de la tabla externa, no tiene que especificar los campos referenciados, predeterminado por valor, el motor Jet se comporta como si la clave principal de la tabla externa fueran los campos referenciados.

Si se revisan los ejemplos de esta sección se puede entenderlos mejor con esta ayuda teórica; se pueden visualizar el uso de palabras como UNIQUE para generar un índice de clave única o PRIMARY KEY para generar índices primarios y por supuesto la palabra reservada más importante CONSTRAINT.

Ejemplo: El objetivo es crear la tabla de EMPLEADO con las características y mejoras de un índice único (no permite valores repetidos) formado por los tres campos.

```
CREATE TABLE EMPLEADO(
empl_id   INT(5)
empl_nombre  CHAR (50),
empl_paterno  CHAR (50),
empl_materno CHAR (50),
empl_telefono CHAR (10),
empl_sucurid  INT(2),
empl_fechanac DATE
)
CONSTRAINT IndiceGeneral UNIQUE  ([empl_id], [empl_paterno], [empl_nombre]);
```

Ejemplo: El objetivo es crear una tabla llamada EMPLEADO; el campo empl_id de tipo entero el que establece como clave principal.

```
CREATE TABLE EMPLEADO (empl_id INTEGER CONSTRAINT IndicePrimario
PRIMARY,
empl_nombre   CHAR (50),
empl_paterno   CHAR (50),
empl_materno   CHAR (50),
empl_telefono   CHAR (10),
empl_sucurid   INT(2),
empl_fechanac  DATE
);
```

Parece un poco complejo; pero sólo es al principio; después con práctica se hace más fácil y se maneja de manera más automática. Es muy común alterar tablas e índices. Y no es un requisito aprender de memoria las instrucciones; el punto es saberlas aplicar.

Las restricciones de columna pueden indicar la obligatoriedad de un valor –escribiendo NOT NULL-, si una columna es clave primaria de la tabla –mediante PRIMARY KEY-, si es clave alternativa, esto es, que sus valores no pueden repetirse en toda la tabla –especificando UNIQUE-, o cualquier otra condición. En el caso de que la clave primaria o las claves alternativas estuviesen compuestas por varios atributos, habría que establecer restricciones de tabla de forma análoga a las de columna. En general se puede considerar las restricciones de columnas como un caso especial de restricciones de tablas, con lo que, por ejemplo, la definición de clave primaria –cuando está compuesta sólo por un atributo- puede realizarse tanto al lado de la columna correspondiente como al finalizar la descripción de atributos.

3.9 Modificar el diseño de una tabla

3.9.1 Alteración de las restricciones sobre la tabla

Tal vez el programador o el DBA en muchas ocasiones se vean obligados a modificar el esquema de una relación con mayor frecuencia que con la que se elimina una tabla que forma parte de una base de datos perdurable. Las modificaciones se realizan mediante una proposición que comienza con las palabras clave ALTER TABLE y el nombre de una tabla. Se disponen entonces de varias opciones, siendo las más importantes

1. ADD seguida del nombre de un campo y su tipo de datos
2. DROP seguida del nombre de un campo.

con la cláusula ALTER se modifica el diseño de una tabla ya existente, se pude modificar los campos o los índices existentes. Su sintaxis es:

ALTER TABLE tabla {ADD {COLUMN tipo de campo[(tamaño)] [CONSTRAINT índice]
CONSTRAINT índice multicampo} |
DROP {COLUMN campo I CONSTRAINT nombre del índice} }

En donde:

Parte	Descripción
Tabla	Es el nombre de la tabla que se desea modificar.
Campo	Es el nombre del campo que se va a añadir o eliminar.
Tipo	Es el tipo de campo que se va a añadir.
Tamaño	El tamaño del campo que se va a añadir (sólo para campos de texto).
Índice	Es el nombre del índice del campo (cuando se crean campos) o el nombre del índice de la tabla que se desea eliminar.
Índice multicampo	Es el nombre del índice del campo multicampo (cuando se crean campos) o el nombre del índice de la tabla que se desea eliminar.

Operación	Descripción
ADD COLUMN	Se utiliza para añadir un nuevo campo a la tabla, indicando el nombre, el tipo de campo y opcionalmente el tamaño (para campos de tipo texto).
ADD	Se utiliza para agregar un índice de multicampos o de un único campo.
DROP COLUMN	Se utiliza para borrar un campo. Se especifica únicamente el nombre del campo.
DROP	Se utiliza para eliminar un índice. Se especifica únicamente el nombre del índice a continuación de la palabra reservada CONSTRAINT

Ejemplo: El objetivo es generar un campo más para la tabla de EMPLEADO; se ha decidido agregar el campo que muestre el salario del empleado para darles una categoría y saber que recibe cada uno. Es de gran importancia para la librería cubrir aspectos básicos de los empleados y tomar en cuenta su salario para futuros aumentos o ajustes. Para esto la tabla ya existe; entonces se hace necesario el uso de la instrucción ALTER TABLE.

ALTER TABLE EMPLEADO ADD empl_salario INT;
Agrega un campo empl_salario de tipo INT a la tabla EMPLEADO.

Si al platicar con los dueños de la librería se decidiera no tomar en cuenta este campo y no incluir todo lo referente a los salarios de los empleados entonces; se tendría que utilizar la siguiente instrucción.

ALTER TABLE EMPLEADO DROP COLUMN empl_salario;
Elimina el campo salario de la tabla EMPLEADO.

ALTER TABLE PEDIDO ADD CONSTRAINT RelacionPedidos FOREIGN KEY
(pedid_empleadofact) REFERENCES EMPLEADO (empl_id);

Agrega un índice externo a la tabla PEDIDO. El índice externo se basa en el campo pedido_empleadofact y se refiere al campo empl_id de la tabla EMPLEADO. En este ejemplo no es necesario indicar el campo junto al nombre de la tabla en la cláusula REFERENCES, pues empl_id es la clave principal de la tabla EMPLEADO.

Si se deseara borrar el índice en la tabla de pedidos entonces la instrucción queda así:
ALTER TABLE PEDIDO DROP CONSTRAINT RelacionPedidos;

3.9.2 Alteración de las restricciones sobre dominios.

Para suprimir una restricción de dominio se debe hacer uso de la proposición ALTER, donde la palabra clave DROP antecede al nombre de la restricción. Para incorporar una restricción en un dominio:

- La proposición ALTER tiene la palabra clave ADD,
- El nombre de la restricción
- Una condición CHECK que define la restricción.

Ejemplo: El objetivo es eliminar la restricción de dominio que fue creada en la sección 3.8 con el CONSTRAINT Articulos_o_Libros.

La sentencia quedaría de la siguiente manera:

```
ALTER DOMAIN Tipos_Doc DROP CONSTRAINT Articulos_o_Libros;
```

Pero también se puede reinsertar la restricción con:

```
ALTER DOMAIN Tipos_Doc ADD CONSTRAINT Articulos_o_Libros
CHECK (VALUE IN('A','L'));
```

3.9.3 Alteración de las Aserciones

Se han podido analizar hasta ahora restricciones tanto para un campo como para todo un registro para mantener la integridad de la base de datos; ahora se va a ver un tipo de restricción con gran potencialidad que viene en el estándar de SQL2.
En muchas ocasiones se necesitará de una restricción que incluya una tabla entera y a más de una tabla. Se sabe que las restricciones de llaves foráneas constituyen un ejemplo que conecta dos tablas, pero su forma es limitada.
Las aserciones en SQL2(denominadas también "restricciones generales") permiten imponer cualquier condición(expresión que puede venir después de WHERE).

De la misma manera de otros elementos de este tipo, se declara una aserción con la proposición CREATE. La forma de una aserción es:

1. Las palabras clave CREATE ASSERTION
2. El nombre de la aserción
3. La palabra clave CHECK
4. Una condición entre paréntesis

Entonces la sintaxis quedaría de la siguiente manera:
CREATE ASSERTION Nombre_Aserción CHECK (Condición)

La condición en una aserción siempre ha de ser verdadera, y se rechazará cualquier modificación de una base de datos que la haga falsa. No se tiene que olvidar que los otros tipos de las restricciones CHECK que se vieron antes pueden ser violados en determinadas condiciones, si incluyen subconsultas.
Existen una importante diferencia entre la forma en que se escribe las restricciones del CHECK basadas en registros y la forma en que se escriben en las aserciones. Las verificaciones basadas en registros pueden referirse a los campos de la tabla en cuya declaración aparecen.
La condición de una aserción no puede ser así. Cualquier campo que se mencione en la condición tiene que ser introducido en la aserción, generalmente con sólo señalar su relación en una expresión SELECT WHERE . Puesto que la condición tiene que tener un valor boleano; se acostumbra a consolidar de algún modo los resultados de la condición

para efectuar una sola elección verdadero/falso. Por ejemplo, se podría escribir la condición como una expresión que produzca una relación a la cual se aplique NOT EXISTS; en otras palabras, la restricción impone que esta relación siempre este vacía. Otra opción consiste en aplicar un operador de agregación como SUM a una columna de una tabla y compararlo después con una constante; de ese modo se podría establecer que una suma siempre sea menor que algún valor limite.

Ejemplo: El objetivo es evitar que un empleado pueda tener un número de teléfono que empiece con los dígitos 044 para que trate con los clientes que se encuentran en las tablas de clientes.
La restricción quedaría de la siguiente manera:

```
CREATE ASSERTION Tel_Empleado CHECK
(NOT EXISTS
      (SELECT  *
       FROM    EMPLEADO, CLIENTE
       WHERE  empl_id = clien_empatid  AND
              empl_telefono LIKE '044%'
      )
);
```

En la librería se tienen ciertas medidas que aplican a empleados.
En este ejemplo de restricción se evita que el empleado que atendió al cliente no tenga un número telefónico que empiece con 044; si se nota usamos el operador LIKE para indicarle que muestre todo lo que empiece con 044; después se aplica la negación con NOT EXISTS. Sigue un formato parecido a una subconsulta pero obviamente no son lo mismo.

3.10 El ambiente del lenguaje SQL

Esta sección al igual que el 3.12 incluyen instrucciones DDL más orientadas a la administración de bases de datos; son dos secciones adicionales para entender un poco mejor el ambiente de SQL; donde esta trabajando y saber interactuar con el administrador de la base de datos (DBA) cuando sea necesario. Es muy común que se trabaje con estas personas a la par y es conveniente empezar a conocer algo de lo que ellos hacen y de lo que en dado momento puede tocar realizar en un ambiente de trabajo; donde muchas veces se realizan tareas que no corresponden totalmente al perfil de desarrolladores; pero por las mismas necesidades de una empresa; se tienen que realizar. Además que va a dar una introducción al siguiente capítulo; donde se verá ya SQL en un entorno de trabajo.

3.10.1 Ambientes

Un ambiente es el marco donde pueden existir los datos y ejecutarse las operaciones de SQL sobre los datos. Esto en la práctica se entiende como un sistema de administración de bases de datos que se corre en alguna instalación. Este sistema al correr en ciertas maquinas

constituye un ambiente SQL. En estos capítulos se ha hablado de varios elementos de la base de datos como lo son: tablas, vistas, dominios y aserciones; estos elementos forman parte del ambiente de SQL. Estos elementos se organizan dentro de una jerarquía de estructuras, cada una de las cuales cumple una función específica dentro de la empresa. La organización consta de las siguientes estructuras:

1. Esquemas. Son colecciones de tablas, vistas, aserciones, dominios de manera general. Los esquemas son las unidades básicas de organización, casi lo podríamos llamar una "base de datos", pero en realidad un poco menos.

2. Catálogos. Son colecciones de esquemas. Constituyen la unidad básica de soporte de una nomenclatura única y accesible. Cada catálogo tiene uno o más esquemas; los nombres de esquemas dentro de un catálogo deben de ser únicos.

3. Conglomerados. Son colecciones de catálogos. Un conglomerado es el ámbito máximo sobre el cual puede emitirse una consulta; por eso, en cierto modo, es la "base de datos" vista por un usuario en particular.

Un ambiente de SQL es más que una mera colección de catálogos y esquemas. Contiene elementos cuyo propósito es soportar las operaciones hechas en la base de datos o en las bases representadas por esos catálogos y esquemas. Dentro de un ambiente de SQL se dan dos tipos especiales de procesos: clientes y servidores de SQL. Un servidor soporta las operaciones efectuadas en los elementos de la base de datos y un cliente le permite al usuario conectarse al servidor. Se supone que el servidor se ejecuta en un anfitrión grande que alberga la base de datos y que el cliente lo hace en otro anfitrión, quizá en una estación personal de trabajo alejada del servidor. Pero también es posible que el cliente y el servidor se ejecuten en el mismo anfitrión.

3.10.2 Conexiones

Si se quiere correr un programa que incluya el SQL en un anfitrión donde exista un cliente del lenguaje, se puede abrir una conexión entre los dos ejecutando una proposición del SQL

CONNECT TO <Nombre_del_Servidor> AS <Nombre_de_Conexion>

El nombre del servidor es algo que depende de la instalación. Con la palabra DEFAULT puede sustituirse un nombre y conectar al usuario a cualquier servidor que la instalación trate como "servidor por omisión". Con el nombre de la conexión se puede designar más adelante. La razón por la cual posiblemente se deba referir a ella es que el programa SQL2 le permite al usuario abrir varias conexiones, pero sólo una puerta estar activa en todo momento. Para pasar de una a otra, se puede convertir a *conn1* en la conexión activa mediante la proposición

SET CONNECTION conn1;

La conexión que estaba activa se vuelve pasiva, mientras no sea reactivada con otra proposición SET CONNECTION.

También se usa el nombre cuando se elimina la conexión:

DISCONNECT conn1;

3.10.3 Sesiones

Una sesión constituye o se refiere a las operaciones de SQL que se llevan a cabo mientras una conexión esta activa. La sesión se realiza simultáneamente con la conexión que la produjo. Por ejemplo, cuando una conexión se vuelve pasiva, lo mismo sucede con sesión; la reactivación de la conexión por medio de la proposición SET CONNECTION también activa la sesión.

3.11 La seguridad y la autorización de usuarios en el lenguaje SQL2

En el estándar de SQL2 existen ciertos elementos llamados identificadores de autorización, los cuales son esencialmente nombres de usuarios. El SQL también tiene una identificación de autorización especial PUBLIC, que incluye a cualquier usuario. A las identificaciones de autorización se les puede conceder privilegios, como por ejemplo UNÍX generalmente controla tres tipos de privilegios: lectura, escritura y ejecución. Esta lista es adecuada, porque los objetos protegidos del sistema son archivos.

3.11.1 Privilegios.

En el estándar de SQL2 hay definidos seis privilegios importantes. Se puede entender como privilegios al permiso que tiene una persona o varias para poder trabajar con la base de datos. Se han visto que podemos interactuar de muchas maneras con la base de datos; ya sea con instrucciones DML o DDL. Se puede interactuar de esta manera con la base porque se tienen permisos para poder realizar todas estas operaciones; se tienen privilegios que se permita interactuar con elementos de la base de datos.[15]

Estos privilegios son:

- SELECT
- INSERT
- DELETE
- UPDATE
- REFERENCES
- USAGE

Los cuatro primeros permisos (SELECT, INSERT, DELETE, UPDATE) se aplican a una relación que puede ser una tabla o una vista. Con estos permisos se otorga al tenedor del privilegio de consultar, insertar, eliminar y actualizar registros.

[15] En ocasiones que se desea modificar una tabla, se desea seleccionar ciertos datos, insertar ciertos valores; operaciones comunes.; pero se impide y manda mensajes con respecto a privilegios; esto es porque se carece de permisos para ejecutar estas operaciones. Como desarrollador o programador se debe de tomar en cuenta y estar conscientes a que se tiene derecho y a que no. Es una buena idea siempre preguntar al DBA que privilegios se tienen; y si se desea hacer operaciones con una tabla a la que no se tienen permisos; se tienen que solicitarle al DBA que nos otorgue esos permisos.

El privilegio REFERENCES es el derecho de referirse a la relación de una restricción de integridad. Esta clase de restricciones se enfocan en aserción y restricciones de integridad referencial. Una restricción no puede verificarse si el esquema donde aparece no tiene el privilegio de REFERENCES sobre todos los datos concernientes a ella.

El privilegio USAGE sobre un dominio y otra clase de elementos de esquema que no sean relaciones y aserciones es el derecho de usuario de utilizar este elemento en sus declaraciones.

A los tres privilegios –INSERT, UPDATE y REFERENCES- también puede asignárseles un solo campo como argumento. En este caso, el privilegio se refiere exclusivamente al campo mencionado. Es posible tener varios privilegios, cada uno de los cuales menciona un campo; de ese modo podemos autorizar el acceso a cualquier subconjunto de las columnas de una relación.

3.11.2 Concesión de privilegios

Esta sección explica como obtener los privilegios que se requieren para efectuar las operaciones que se requieren en base a las necesidades en la base de datos. La concesión de privilegios presenta dos aspectos:

1) Como se crean inicialmente
2) Como se transmiten de un usuario a otro

1)Los elementos de SQL como esquemas o módulos tienen un propietario. Este propietario de un elemento goza de todos los privilegios relacionados con ese elemento; por ejemplo cuando se crea un esquema, se supone que éste y todas sus tablas y elementos son propiedad del usuario que lo creó

2)No sólo se obtienen privilegios cuando se es el creador y propietario del elemento de la base. El SQL ofrece una proposición GRANT que permite a un usuario otorgarle un privilegio a otro. El primero también conserva el privilegio concedido.
Cabe aclarar que hay una importante diferencia entre otorgar privilegios y copiarlos. Todo privilegio conlleva a una opción de concesión.

Por ejemplo; un usuario puede tener un privilegio como SELECT sobre la tabla EMPLEADO "con la opción de concesión", mientras que un segundo usuario puede tener el mismo privilegio pero sin la opción. En el caso del primer usuario este podrá otorgar el privilegio SELECT sobre EMPLEADO a un tercer usuario y además la concesión puede acompañarse o no de la opción. Para el caso del segundo usuario; no podrá otorgar a nadie el privilegio el privilegio SELECT sobre la tabla EMPLEADO. Si el tercer usuario obtiene mas tarde este mismo privilegio con la opción, entonces podrá concederlo a un cuarto usuario, una vez más con la opción, entonces podrá concederlo a un cuarto usuario, una vez más con la opción o sin ella y así sucesivamente.

Una proposición de concesión consta de los siguientes elementos.

Elemento 1) La palabra clave GRANT

Elemento 2) Los privilegios que puede ser uno o más en forma de lista; si se desean otorgar todos los privilegios se coloca la palabra clave ALL PRIVILEGES

Elemento 3) La palabra clave ON

Elemento 4) El elemento sobre el cual se otorga estos privilegios; que es un elemento de la base de datos. Puede ser una tabla, un dominio, etc.

Elemento 5) La palabra clave TO

Elemento 6) Una lista de uno o más usuarios

Elemento 7) El elemento que puede ser opcional; que es la palabra clave WHIT GRANT OPTION; este elemento es para darle la posibilidad al usuario que se otorga los privilegios de que el tenga la capacidad de otorgar esos privilegios a otros usuarios. Si se omite esta palabra clave el no podrá heredar estos privilegios a nadie más

La sintaxis quedaría de la siguiente manera:

GRANT <privilegio o lista de privilegios> ON <elemento de la base de datos> TO <usuario> [WHIT GRANT OPTION]
Los privilegios pueden ser para consultar o actualizar (borrar, insertar o modificar) tablas, así como para referenciarlas o para usar dominios (Domain), etc.

Ejemplo: Se desea otorgar privilegios al usuario Erikca para que pueda borrar registros, insertar o actualizar en la tabla de EMPLEADO de la base de datos de la librería; ya que ella va ser la encargada de manipular las computadoras para dar de alta o de baja al personal interno que se tiene capturado en la base de datos.

GRANT DELETE ON EMPLEADO TO Erikca
GRANT SELECT ON EMPLEADO TO Ericka WITH GRANT OPTION
GRANT INSERT ON EMPLEADO TO Ericka
GRANT UPDATE ON EMPLEADO TO Ericka

En este ejemplo se nota que se han otorgado privilegios al usuario Erikca a través de un usuario que puede conceder privilegios y ella va atener la capacidad de heredar el privilegio de SELECT pero sólo éste. Esto suena lógico si ella va ser la responsable del manejo de las computadoras de la biblioteca y que puede dar la capacidad a otros empleados de sólo hacer determinadas consultas pero sin alterar la información.

Es importante saber que se puede otorgar a privilegios a múltiples usuarios en lugar de otorgar los privilegios cada uno por su nombre; esto mediante la utilización de "PUBLIC"

3.11.3 Revocación de privilegios

Así como es posible otorgar privilegios; también es posible quitarlos o revocarlos. Incluso puede requerirse una revocación en cascada. La revocación de un privilegio mediante la opción de concesión que ha sido transmitida a otros usuarios va exigir la cancelación de esos otros privilegios. La revocación consta de los siguientes elementos:

Elemento 1) La palabra reservada REVOKE

Elemento 2) Los privilegios que pueden ser uno o una lista de privilegios.

Elemento 3) La palabra reservada ON

Elemento 4) Un elemento de la base de datos

Elemento 5) La palabra clave FROM

Elemento 6) La lista de uno o más usuarios.

La sintaxis quedaría de la siguiente manera:
REVOKE <privilegio> ON <elemento de la base> FROM <usuario>

Debido a la forma de conceder privilegios hay elementos o palabras reservadas que se pueden añadir a la revocación como son:

La terminación de la proposición con la palabra CASCADE. De esta manera cuando se revoquen los privilegios especificados, también se puede hacer lo mismo con los que fueron otorgados sólo a causa de los privilegios revocados.

La proposición también puede terminar con RESTRICT, palabra que significa que no puede ejecutarse la proposición de revocación, si la regla en cascada descrita en el inciso anterior hacía cancelar privilegios, porque los privilegios revocados fueron transferidos a otros.

Entonces en la revocación de privilegios en la que si se usa el comportamiento restringido (RESTRICT) no dejara de revocar privilegios que hayan sido concedidos a otros usuarios, mientras que si se emplea la opción en cascada, se borrarán todos los privilegios.

Ejemplo : Se ha decidido que el usuario que manipule el sistema sea Yessica y no Ericka; entonces se tendrán que revocar todos los permisos otorgados a Ericka para insertar o borrar registros

REVOKE INSERT, UPDATE ON EMPLEADO FROM Ericka;

Es importante tomar en cuenta que a pesar de revocar privilegios a un usuario; el usuario podría tenerlos a través de otro usuario(que les haya concedido los mismos privilegios). Otro aspecto a verificar es que cualquier borrado de un objeto (tabla, dominio, vista) causa la revocación de todos los privilegios sobre el objeto a todos los usuarios.

Se han contemplado y cumplido los objetivos planteados en la introducción al capítulo; puesto que se analizaron los más importantes comandos de SQL correspondientes a las instrucciones DDL o Instrucciones de Definición de Datos.
Se vieron instrucciones para la creación de tablas así como su borrado, se analizaron los índices respecto a su creación y diferentes formas de plantearlos de acuerdo a las necesidades. También se manejó la alteración de tablas con la instrucción ALTER TABLE para agregar varios elementos a la base como campos a una tabla.
Un aspecto importante que no se debe de olvidar es el manejo de restricciones que existen y deben de existir en la base y que fueron tocados en este capítulo.
Finalmente se manejo el uso de las llaves primarias y foráneas para mantener la integridad de la base.
Como un apartado final se manejó el tema de privilegios y ambientes de SQL para dar una visión completa de SQL en este capítulo.
En los siguientes capítulos se verá SQL ya en acción con entornos en Visual Basic por medio de dos herramientas gráficas y en Oracle por medio de SQL *Plus.

Capítulo 4 SQL con Visual Basic para base de datos Access

Introducción

Ahora se va a ver la parte que le corresponde a Visual Basic 6.0; como es que se puede utilizar el lenguaje SQL para manipular la información de una base de datos Access por medio de herramientas gráficas que acompañan a Visual Basic 6.0; estas dos herramientas son: SQLVB6 y Visdata. También se verá una pequeña sección de cómo implementar código SQL con código de Visual Basic. Gracias a estas dos herramientas se facilita el poder trabajar con bases de datos Access; y se hace muy claro como interviene o que papel toma SQL en todo esto. El manejo de las instrucciones básicas ya se estudió en capítulos anteriores y lo que se puede lograr con ellas; pero ahora hay que saber como aplicarlas en bases de datos Access para extraer la información que importa de esta base; para crear nuevos elementos y entender perfectamente el entorno en que se va a desenvolver. Por esto SQL es el lenguaje básico que se debe de saber para trabajar en un entorno de bases de datos. Precisamente este es el objetivo de este capítulo; saber como aplicar las instrucciones de SQL para extraer información de una base de datos Access por medio de herramientas gráficas que acompañan a Visual Basic 6.0; además de conocer el entorno de estas herramientas. Es obvio que no se va enseñar el uso o el manejo de Visual Basic; puesto que el tema requeriría por lo menos un libro; sólo se van a conocer algunas instrucciones que Visual Basic 6.0 añade a SQL para explotar más su funcionalidad. Cabe recordar que en los anteriores dos capítulos se hablaba de las extensiones que cada manejador de base de datos le agrega para explotar más el lenguaje SQL; aquí se van analizar unos ejemplos de la extensión de estas instrucciones; por esto era importante conocer el lenguaje SQL estándar; para saber distinguir de las extensiones. Es bien importante que si las aplicaciones que se creen van estar permanentemente utilizadas por el mismo manejador; es bueno colocar extensiones; pero si hay la posibilidad de migrar a otros manejadores o bases; es bueno tratar de utilizar lo más posible el lenguaje estándar de SQL. Con este capítulo y esta sección se cubre el objetivo de la tesis; de conocer los principales comandos de SQL y su aplicación en dos bases de datos importantes. En las empresas el tiempo es vital; por esto la tesis esta enfocada a ese objetivo; tomar las cosas rápido y empezar a trabajar. La profundidad que se puede requerir en ciertos aspectos queda al lector. Pero con esto es más que suficiente para hacer un buen papel en los entonos de bases de datos.

4.1 Uso del intérprete SQLVB6

El conocer esta herramienta gráfica va a proporcionar suficientes elementos para manejar una base de datos Access. Los objetivos de esta sección son:

- Crear y eliminar tablas con las instrucciones ya vistas de CREATE TABLE y DROP TABLE.
- Agregar o eliminar índices en una tabla mediante CREATE INDEX y DROP INDEX
- Conocimiento de código adicional al SQL Estándar.
- Agregar o eliminar campos en una tabla por medio de ADD COLUMN y DROP COLUMN
- También se manejara las relaciones y lo que implican mediante PRIMARY KEY y FOREIGN KEY

El programa puede quedar instalado en Programas|Sqlvb6 o en otra dirección. Es independiente del directorio de Visual Basic 6.0.
Al dar doble clic; se va a cargar el programa que muestra la pantalla principal dividida en 2 zonas verticalmente; en la parte de arriba muestra el menú principal y una barra de tareas de once íconos(Fig. 9). Ya en esta pantalla es donde se ejecutan las instrucciones de SQL.
De izquierda a derecha de la barra se encuentra el icono de: New, Open, Save, Delete, Print, Properties, Execute, Generate, DB Tables, Views y Help.
Los más importantes para el objetivo de este capítulo son:

- New; sirve para crear nuevos archivos con comandos SQL para que sean procesados y actúen directamente con la base de datos.
- Open; para abrir scripts que se hayan almacenado para futuros cambios a la base
- Save; para salvar scripts o archivos que puedan ser utilizados en varias ocasiones; ya que es una actividad común realizar determinadas tareas a la base
- Execute; para ejecutar un script y realizar alguna tarea directamente sobre la base.

Este programa tiene una forma de trabajo muy particular; procesa secuencias de comandos de SQL. Las secuencias de comandos son archivos de texto con líneas de comandos SQL enlazadas que se pueden utilizar en este programa o transportarse a cualquier otro editor de bases de datos. La extensión de estos archivos es sqv.

4.1.1 Creación de un archivo de secuencia de comandos

La creación de un archivo de secuencia de comandos es en su forma básica simple. Se presiona el botón de New para hacerle saber al SQLVB6 que se va a crear un nuevo script; una vez teniendo la pantalla en blanco; se puede comenzar a capturar el código que realice la tarea que se ajuste a las necesidades del usuario.
Una vez capturado se debe Guardar con el botón Save.

4.1.2 Modificación de una secuencia de comandos

Si ya hay archivos de secuencia de comandos(script) guardados; se pueden editar, cambiarlos, ejecutarlos y volverlos a almacenar.

Para cambiar uno de estos script se tienen que seguir los siguientes pasos:

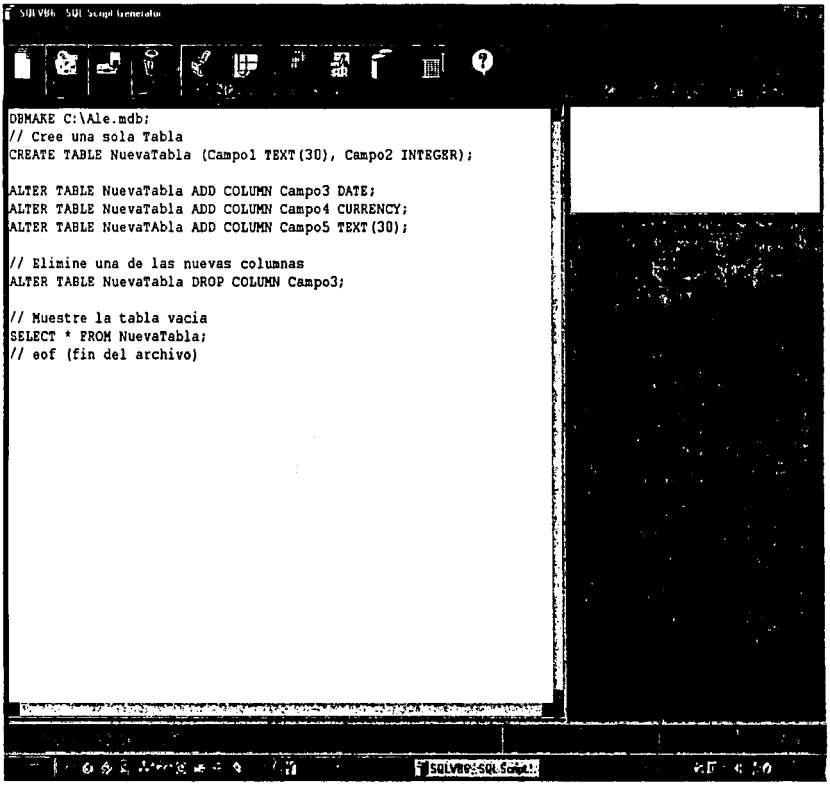

Fig. 9 Pantalla principal de SQLVB6

Ir al menú File
2. En la opción Open del menú
3. Seleccionar el script a modificar
4. Modificar las líneas
5. Ejecutar el archivo para ver que realice las acciones que tiene que realizar
6. Volverlo a almacenar

Siempre es importante ver un archivo de estos para entender a lo que se refiere; entonces veremos un script; después se irán aclarando puntos sobre algunos comandos, conforme se avance en el capítulo.

Ejemplo: Modificar el siguiente script Ejemplo_1.csv; para que la tabla de clientes muestre solo a los clientes que su primera letra inicie con la letra "P"

```
// ***************************************************
// Ejercicio de prueba de una secuencia de comandos SQL
// para el interpretador SQLVB
// ***************************************************

// Abrir la base de datos
dbOpen C:\Archivos de Programa\FUENTES\DATOS\TIENDA.MDB;

// Abra algunas tablas para su visualización
  SELECT * FROM EMPLEADO WHERE emple_nombre='Fernando';
  SELECT * FROM CLIENTE;
//
```

Paso 1. Ir al menú File
Paso 2. Seleccionar Open y seleccionar el script llamado Ejemplo_1.csv
Paso 3. Modificar el script

```
// *****************************************************
// Modificación al Ejercicio de prueba en el SELECT
//de Clientes
// *****************************************************

// Abrir la base de datos
dbOpen C:\Archivos de Programa\FUENTES\DATOS\TIENDA.MDB;

// Abra algunas tablas para su visualización
  SELECT * FROM EMPLEADO WHERE empl_nombre='Fernando';
  SELECT * FROM CLIENTE WHERE clien_nombre LIKE 'P%';

//
// eof
```

Paso 4. Dar clic en el icono con la leyenda "Execute" y se va a presentar una pantalla pequeña donde va a mostrar el resultado de la consulta o consultas de SQL.[16]

Paso 5. Si el resultado es el que se espera; entonces dar clic en el icono con la leyenda Save.

4.1.3 Carga de un archivo de secuencias de comandos(sqv).

Este archivo de secuencia de comandos comúnmente llamado script; es fácilmente cargado; si se tiene almacenado con el botón de Open se puede buscar y abrir el archivo; una vez abierto para cargarlo y hacer que realice su tarea dependiendo del objetivo del script; sólo hay presionar el botón Execute y si no hay errores el script realizará su tarea. Si hay un error en el código éste no se ejecutará y mandará un mensaje de error. Es necesario quitar todos los errores para que sea ejecutado. No olvidar que todo es por código; desde la conexión y demás modificaciones.

4.1.4 Diseño de nuevas tablas

Se conoce perfectamente el comando CREATE TABLE de SQL para crear nuevas tablas; aquí en este entorno se ocupara la misma instrucción para las nuevas tablas. En su forma básica este comando consta de tres partes: las palabras CREATE TABLE; el nombre de la tabla; y la lista de los nombres, tipos y tamaños de cada columna en la tabla; una instrucción en Visual Basic como ejemplo; tendría la siguiente forma:

CREATE TABLE PROVEEDOR (prove_ID INTEGER ,prove_nombre STRING(40), prove_edad INTEGER);

Quizá puede confundir el hecho de desconocer el tipo de campo; pero hay que recordar que se esta en un entorno de Visual Basic y por tanto se tiene que adaptar a los tipos de datos que se manejan en Visual Basic.

El script para crear la nueva tabla se llama Tabla_Nueva.SQV; es de la siguiente manera:

```
//*******************************************************
//Creación de la tabla de PROVEEDOR; en la base de datos TIENDA
//*******************************************************

//Abro una base de datos existente
dbOpen C:\Archivos de Programa\FUENTES\DATOS\TIENDA.MDB;

//Creo una sola tabla
CREATE TABLE PROVEEDOR (prove_ID INTEGER ,prove_Nombre
STRING(40), prove_edad INTEGER);

//Muestro la tabla vacia
SELECT * FROM PROVEEDOR
```

[16] La instrucción dbOpen y dbMake son instrucciones propias de Visual Basic y no forman parte de los comandos de SQL estándar; dbOpen sirve para abrir una base de datos existente, dbMake para crear una nueva base de datos.

Se utiliza el SQLVB6 para ejecutarlo mediante la opción Actions | Execute. Al lado derecho de la pantalla en la parte inferior aparecerán los dos campos sin datos; en el mismo lado derecho en la parte superior aparecerá un icono mostrando la leyenda Recordset.
Con esto se indica que se ha creado una nueva tabla en una base de datos Access; que por el momento esta sin datos; pero que se puede manipular tanto en Visual Basic como en Access.
Se puede completar la creación de la tabla agregando los ingredientes que hacen falta; por ejemplo una llave primaria para evitar la duplicación de registros. Entonces se puede editar el archivo Tabla_Nueva.SQL y establecer el campo prove_ID como la llave primaria.

El script quedaría de la siguiente manera:

```
//*********************************************************
//Creación de la tabla de PROVEEDOR; en la base de datos TIENDA
//agregando una llave primaria al campo de identificador de empleado.
//*********************************************************
//Abrir una base de datos existente
dbOpen C:\Archivos de Programa\FUENTES\DATOS\TIENDA.MDB;

//Creación una sola tabla
CREATE TABLE  PROVEEDOR (prove_ID INTEGER  cpProveedores
PRIMARY KEY ,prove_nombre STRING(40), prove_edad INTEGER);

//Mostrar la tabla vacia
SELECT * FROM PROVEEDOR
```

Como se puede ver en el ejemplo; todo sigue las reglas del SQL estándar con unas pocas líneas de Visual Basic.

4.1.5 Modificación de tablas

Se verá ahora un ejemplo de la alteración de tablas; ya se estudió el código para alterar tablas; entonces se puede aplicar a este entorno. Algunas variantes que se vieron con este comando se van aplicar aquí.
Uso de ADD COLUMN. Con esta alteración se pueden agregar columnas a una tabla sin perder ningún dato de los campos existentes.[17]
El script para agregar columnas a la tabla de PROVEEDOR queda de la siguiente manera:

```
//*********************************************************
//Script para crear dos campos a las tablas de PROVEEDOR
//*********************************************************
dbOpen C:\Archivos de Programa\FUENTES\DATOS\TIENDA.MDB;
//Se agregan dos columnas
ALTER TABLE PROVEEDOR ADD COLUMN prove_tipo STRING(10);
ALTER TABLE PROVEEDOR ADD COLUMN prove_dir STRING(100);
//Para ver la tabla
SELECT * FROM PROVEEDOR:
```

[17] Hay que tomar en cuenta que ADD COLUMN siempre agregará los campos a la izquierda de las demás en la tabla. Se puede controlar el orden en que aparecen los campos mediante SELECT; pero el orden físico de los campos; sólo se puede hacer con CREATE TABLE.

4.1.6 Manejo de Índices

Como se analizó en el anterior capítulo; el comando CREATE INDEX se utiliza para crear un índice en una tabla. La forma más básica es:
CREATE INDEX indice_uno ON PROVEEDOR (prove_nombre);
En los anteriores capítulos se vieron las derivaciones para aplicar los índices; en el siguiente script se verá la aplicación de los índices en la tabla de PROVEEDOR.

```
//*********************************************************
//Aplicación de las instrucciones para crear índices
//Se va crear la tabla de proveedores ; pero se va crear una nueva base
//de datos
//*********************************************************

//Se crea la base de datos
dbMake TIENDA_DOS.MDB ;

//Se crea la tabla de Proveedores
CREATE TABLE PROVEEDOR
(prove_ID        INTEGER,
 prove_nombre    STRING(40),
 prove_edad      INTEGER,
 prove_dir       STRING(100),
 prove_tipo      STRING(10));

//Se crea la clave principal
CREATE INDEX indice_id_empleado ON PROVEEDOR(prove_ID) WITH
PRIMARY;

//Crea una columna sin nulos
CREATE INDEX indice_tipo ON PROVEEDOR(prove_tipo) WITH
DISALLOW NULL;

//Crea una clave para la clasificación de varias columnas
CREATE INDEX indice_nomdirtip ON PROVEEDOR(Nombre, Edad, Tipo);

//Finalmente se muestra la tabla
SELECT * FROM PROVEEDOR;
```

Este ya es un script más en forma; llevando consigo los índices; la creación de una base de datos en Visual Basic, la creación de una tabla y la consulta a una tabla.
Con respecto a las instrucciones de Visual Basic no entran en el objetivo de esta tesis; pero es bueno ver como se trabaja en Visual Basic para interactuar con Access.

4.1.7 Administración de Relaciones.

Por el estudio que se ha llevado se ha visto la gran importancia de la llave primaria y la llave foránea para administrar adecuadamente la base de datos; ahora se analizará como se hace en Visual Basic.

El uso más común de la cláusula CONSTRAINT es con PRIMARY KEY. Se utiliza para definir el campo (o conjunto de campos) que contenga el índice principal de la tabla.

Ejemplo de script con llave primaria.

```
//Se crea una base de datos con código de VB
dbMake Libros.MDB

//Se crea la tabla de PUESTO
CREATE TABLE PUESTO(
puest_nombre        STRING(20) CONSTRAINT llavePuesto PRIMARY KEY,
puest_descripción    MEMO);

//Se muestra la tabla
SELECT * FROM PUESTO;
```

Otro tipo de constraint y que es realmente de gran potencialidad es el que hace uso de la restricción FOREIGN KEY. Esta restricción se utiliza para establecer relaciones entre tablas. Entonces se ocupa FOREIGN KEY para relacionar una tabla base con otra que tenga una lista válida de valores de columnas; la otra tabla por tanto y casi como estándar debe tener una columna definida con el mismo nombre de la columna utilizada como índice principal. Cuando se establece una relación con un índice o clave externo, se puede establecer una regla que indique que sólo se deben proporcionar datos validos en un campo, de acuerdo a aquellos que existen en el índice principal.

Ahora se creará un script que trabajará con la base de datos Libros.MDB y con la tabla de Puestos que se le puede llamar un catálogo. Se creara una tabla de Empleados con la que se ha venido trabajando.

```
//*************************************************************
//Uso del Foreign Key
//*************************************************************
dbOpen C:\Libros.MDB;
//Suponiendo que ya existe la tabla de Puestos; se crea la de Empleados
CREATE TABLE EMPLEADO
(empl_id    INTEGER,
 emple_nombre    STRING(50),
 emple_telefono   STRING(10)
 emple_dirección  STRING(50),
 emple_puesto      STRING(20) CONSTRAINT cnpuesto REFERENCES
PUESTO(puest_nombre));
//Se muestran las tablas
SELECT * FROM  PUESTO;
SELECT * FROM EMPLEADO;
```

Al ejecutar el script ya sea por el icono o por el menú; en la parte derecha superior se van a mostrar los dos iconos por el uso de las dos tablas; y en la parte inferior los campos.

4.2 Uso del Administrador Visual de datos(Visdata).
Esta herramienta es muy completa para construir y manejar de manera adecuada bases de datos. Esta herramienta posee diversas utilidades para la administración de las bases; por esta razón sólo vamos a cubrir los aspectos que intervienen directamente con el lenguaje SQL para manejar una base de datos Access. Los otros aspectos quedan al lector. Por tanto los objetivos de esta sección son:

- Conocer el entorno del Visdata
- Conocer la funcionalidad con respecto a SQL
- Manejar instrucciones DML en este entorno

Esta herramienta además de proveer una interface para el manejo del lenguaje SQL; se puede utilizar para crear bases de datos, agregar o modificar tablas e índices, establecer relaciones, asignar derechos de grupos o usuarios, verificar o almacenar instrucciones SQL y llevar a cabo la captura en las tablas.

4.2.1 Que és el Visdata.
El Administrador Visual de Datos(Fig. 10) es una herramienta gráfica que va a permitir realizar varias operaciones en una base de datos. Esta herramienta acompaña a Visual Basic 6.0. Con esta herramienta se puede hacer consultas SQL y guardarlas en la base de datos como procedimientos almacenados que se podrán acceder desde diversas aplicaciones. Aquí mismo se pueden copiar registros de una tabla a otra y tablas de una base a otra. Aumenta la funcionalidad de Visdata al saber que se puede conectar a base de datos de Microsoft Jet como es el caso de Access, dBase, FoxPro y Parados. Incluso se puede conectar a paginas de Excell, archivos de texto delimitados y bases conectadas por ODBC. Para ejecutar el Visdata es necesario ir al menú Complementos | Administrador visual de datos(Add-Ins | Visual data Manager).
El paso inicial siempre es abrir una base de datos; para realizar esto es necesario al menú Archivo| Abrir base de datos | Microsoft Access (File | Open Database | Microsoft Access) y buscar la base de datos que me interesa. Todos los elementos de la base de datos que abra se mostraran en la Ventana de Bases de datos o Database Window.

4.2.2 Conocimiento de su entorno
El administrador Visual de Datos se compone de dos ventanas principales; una barra de botones y una barra de menú.

La ventana izquierda; es la ventana de base de datos o Database Window; en esta ventana se pueden apreciar ciertos elementos de la base de datos como son:

- Tablas
- Querys o consultas almacenadas
- Las propiedades o valor de ciertas variables de la base de datos.

Esta ventana se puede agregar tablas y modificarlas, se puede agregar registros a las tablas entre otras muchas funciones.

La ventana derecha; es la ventana de instrucción SQL; en esta ventana se puede escribir y ejecutar todas las consultas SQL que se aplicarán a la base de datos abierta. Esta es la ventana principal con la que se va a trabajar para hacer uso de las instrucciones DML.

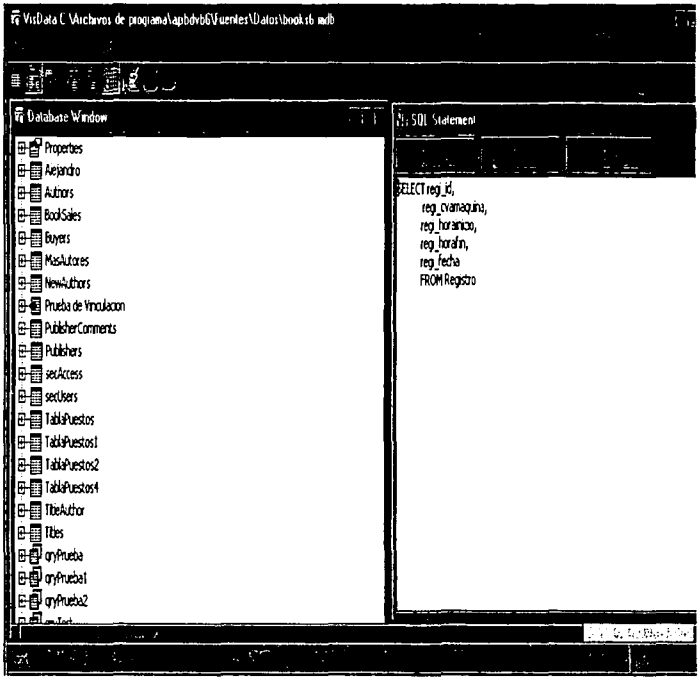

Fig 10. Pantalla principal del Administrador Visual de datos

La barra de menú va a permitir acceder a todas las características de la aplicación; tiene cuatro opciones principales que son: Archivo, Utilidad, Ventana y Ayuda.
En el Opción Archivo (File) se tienen opciones para:

- Abrir la base de datos
- Para crear una base de datos
- Para cerrar una base de datos
- Para migrar información
- Compactar y reparar una Base

En la opción Utilidad (Utility) se tienen opciones de:

- Generador de consultas; que es una herramienta gráfica para realizar consultas tipo SQL; pero sin teclear código, sólo llenado un cuadro con datos.
- Generador de formularios en base a una tabla o a un query; con esto ya no se tiene que construir un formulario.
- Vincular a una tabla externa y tratarle como si fuera parte de la base
- Manejo de usuarios y grupos.

Las otras dos opciones del menú principal son iguales alas de otras aplicaciones de Windows de ayuda y ventana.
La barra de botones se utilizará para determinar el tipo de objetos con el que se desee trabajar.

4.2.3 Trabajo con SQL

El trabajo directo con código de SQL se realiza en la ventana Instrucción SQL; esta ventana permite capturar y ejecutar consultas SQL estándar en la base de datos abierta.

Además de que se pueden almacenar las consultas SQL para utilizarlas posteriormente o generar un formulario. Con respecto a las instrucciones DDL; estas se ejecutan de manera indirecta; por medio de menús y comandos; que evitan al usuario capturar código; pero si se pudiera ver el código de cada menú se puede dar cuenta que son instrucciones de SQL estándar DDL. Por esta razón en esta herramienta la manera directa de ocupar código SQL es por medio de las instrucciones DML.

Para generar una consulta se siguen estos pasos:

1. Ir al menú File | Open
2. Como es una base Access seleccionar la opción Microsoft Access
3. Buscar la base de datos Access sobre la cual deseamos abrir y hacer la consulta
4. Hacer clic en la ventana Instrucción SQL
5. Capturar la consulta en código estándar
6. Presionar el botón Ejecutar(Execute) para llevar cabo la consulta
7. El resultado de la consulta si es correcto se presentara en una tercera ventana que trae como titulo la consulta; en un Grid o cuadricula tipo hoja de Excell; si es incorrecta se muestra un mensaje con el posible error.

Ejemplo: El objetivo es generar una consulta en el Visdata y nos muestre todos los campos de la tabla de Empleados de la base de datos Libros, donde los teléfonos de los empleados inicien con el 763

Solución:

1. Una vez abierto el Visdata; ir al menú File | Open
2. Como Librería es una base de datos Access se selecciona la opción Microsoft Access.
3. Se busca la ubicación de la base y se selecciona
4. Se da un clic en la ventana de instrucción SQL.
5. Capturar la siguiente consulta: SELECT * FROM EMPLEADO WHERE emple_telefono LIKE '763%'
6. Presionar el botón Execute
7. La consulta se presenta en una ventana aparte superpuesta a las demás con el resultado de la consulta.

4.3 SQL en Visual Basic por medio de código.

Se han analizado dos herramientas gráficas para poder hacer uso del lenguaje SQL dentro de Visual Basic para manipular una base de datos Access; ahora se verá como se podría ocupar SQL dentro de código de Visual Basic; código de aplicaciones. Es importante saber esta parte porque muchas veces se va a requerir ocupar el código de SQL sin herramientas graficas; basándose en lo que nos regrese una consulta para dar pauta a la aplicación. Nuevamente se hace la observación que no se va a enseñar como programar en Visual Basic; es bueno que el usuario tenga el conocimiento de Visual Basic para manipular adecuadamente SQL. Términos como recordset, dynaset y funciones de Visual Basic, tanto para manipular cadenas como para manipular la base de datos quedan al lector por investigar.

Existen dos tipos de consultas SQL: las consultas de selección (devuelven datos) que son por medio de las instrucciones DML y las consultas de acción (aquellas que no devuelven ningún registro) que por lo general aparecen las instrucciones DDL. Ambas pueden ser tratadas en Visual Basic pero de forma diferente.

Las consultas de selección se ejecutan recogiendo la información en un recordset previamente definido mediante la instrucción openrecordset(), por ejemplo:

```
Dim SQL as String
Dim RS as recordset

  SQL   = "SELECT * FROM EMPLEADO;"
  Set RS = Nombre_Base.OpenRecordSet(SQL)
```

Si la consulta de selección se encuentra almacenada en una consulta de la base de datos:

```
  Set RS = Nombre_Base.OpenRecordset("Nombre_Consulta")
```

Las consultas de acción, al no devolver ningún registro, no se pueden asignar a ningún recordset, en este caso la forma de ejecutarlas es mediante los métodos Execute y ExecuteSQL (para bases de datos ODBC), por ejemplo:

Dim SQL as string

```
SQL = "DELETE * FROM EMPLEADO WHERE emple_nombre = 'Francisco';"
Nombre_Base.Execute SQL
```

4.3.1 Funciones de Visual Basic utilizables en una Instrucción SQL

Función	Sintaxis	Descripción
Now	Variable= Now	Devuelve la fecha y la hora actual del sistema
Date	Variable=Date	Devuelve la fecha actual del sistema
Time	Variable=Time	Devuelve la hora actual del sistema
Year	Variable=Year(Fecha)	Devuelve los cuatro dígitos correspondientes al año de Fecha
Month	Variable=Month(Fecha)	Devuelve el número del mes del parámetro fecha.
Day	Variable=Day(Fecha)	Devuelve el número del día del mes del parámetro fecha.
Weekday	Variable=Weekday(Fecha)	Devuelve un número entero que representa el día de la semana del parámetro fecha.
Hour	Variable=Hour(Hora)	Devuelve un número entre 0 y 23 que representa la hora del parámetro Hora.
Minute	Variable=Minute(Hora)	Devuelve un número entre 0 y 59 que representa los minutos del parámetro hora.
Second	Variable=Second(Hora)	Devuelve un número entre 0 y 59 que representa los segundos del parámetro hora.

DatePart

Esta función devuelve una parte señalada de una fecha concreta. Su sintaxis es:
DatePart(Parte, Fecha, ComienzoSemana, ComienzoAño)

Parte representa a la porción de fecha que se desea obtener, los posibles valores son:

Valor	Descripción
Yyyy	Año
Q	Trimestre
M	Mes
Y	Día del año
D	Día del mes
W	Día de la semana
Ww	Semana del año
H	Hora
M	Minutos
S	Segundos

ComienzoSemana indica el primer día de la semana. Los posibles valores son:

Valor	Descripción
0	Utiliza el valor pode efecto del sistema
1	Domingo (Valor predeterminado)
2	Lunes
3	Martes
4	Miércoles
5	Jueves
6	Viernes
7	Sábado

ComienzoAño indica cual es la primera semana del año; los posibles valores son:

Valor	Descripción
0	Valor del sistema
1	Comienza el año el 1 de enero (valor predeterminado).
2	Empieza con la semana que tenga al memos cuatro días en el nuevo año.
3	Empieza con la semana que esté contenida completamente en el nuevo año.

Se han cubierto los objetivos planteados en la introducción del capítulo respecto al uso de instrucciones DDL como la creación y borrado de tablas, la creación de índices, llaves primarias y foráneas, así como el agregar columnas , también se vieron algunas instrucciones adicionales que proporciona Visual Basic para añadir potencialidad al código; en general el manejo de las instrucciones que se han visto a través de los dos anteriores capítulos a través de las dos herramientas gráficas por medio de scripts o en su propio entorno. Se manejaron también las instrucciones DML y al ejecutarlos el ver el resultado en forma gráfica.
Se cubrió la herramienta Visdata y el SQLVB6 respecto a SQL.
En el siguiente capítulo se verá el uso del lenguaje SQL en un entorno llamada SQL *Plus que proporciona Oracle para el manejo de la información de su base y así cumplir con los objetivos de la tesis al ver dos plataformas interactuando con SQL.

Capitulo 5 SQL en Oracle con SQL *Plus.

Introducción

En este capítulo se hablará del uso y aplicación del lenguaje SQL dentro de un entorno llamado SQL *Plus que es un programa que viene integrado en las herramientas que da Oracle para manejar la información de sus bases de datos. Se puede usar el programa en conjunción con el lenguaje de bases de datos SQL.

Oracle es un sistema de administración de bases de datos relacionales y complejas.

Cabe mencionar que no se trata de enseñar Oracle ni su lenguaje propio que es PL/SQL; el objetivo principal de este capítulo es mostrar como trabajar el lenguaje SQL en este entorno; así como conocer el ambiente; las ventajas y desventajas que brinda; algunas funciones que se pueden agregar al lenguaje; pero en general se va a conocer el ambiente y lo que se puede aplicar en base a las instrucciones que se ha aprendido.

Por lo tanto los objetivos planteados en este capítulo son:

- Conocer el entorno SQL *Plus
- Conocer los comandos que integra el entorno SQL *Plus
- Saber que instrucciones se pueden aplicar en este entorno

Tiene una importante justificación este capítulo; muchas veces en trabajos se ha observado; se explica un lenguaje; sus características, pero no se explica como ocupar las instrucciones; como manejar el entorno; que se puede y no se puede aplicar. En un medio como es el de sistemas de cualquier empresa muchas veces se pide que se tome un manual y se comience a aplicar en dos días lo que se ha aprendido; puesto que la empresa requiere de la adaptabilidad a las cosas muy rápido; es preciso tener información clara, concreta y que de las bases importantes para saber manejar un software; esto es lo que complementa mi tesis para llegar al objetivo de utilizar este trabajo como un medio para comenzar a trabajar rápidamente con las ideas básicas; con las principales instrucciones del lenguaje y con conocimiento del entorno, pero no con eso sin fundamentación y erróneas; sino con seguridad de que lo que se esta haciendo es correcto y que se va ir adaptando al medio rápidamente. Por esto estos últimos capítulos son cortos pero con la idea de complementar este trabajo para que justifique el objetivo de enseñar lo principal y dejar al lector los detalles según las necesidades que se le presenten.

5.1 Introducción a SQL *Plus

SQL o Standard Query Languaje (Lenguaje estructurado de consulta) es un lenguaje de cuarta generación (L4G); el cual es útil en la definición, consulta y manejo de tablas relacionales. SQL *Plus, además de integrar todas las funciones de SQL y PL/SQL, permite llevar a cabo otro tipo de tareas.[18]

5.2 Cómo entrar a SQL *Plus

Antes de entrar a SQL *Plus, hay que asegurarse de que este instalado el software. Es común que se instale en Programs/ Oracle for WidowsXX / SQL Plus XX
Cuando se entra a SQL *Plus la pantalla desplegará líneas[19] como las siguientes:

```
SQL *Plus 3.x.3.x2.1 –Production on Wed Jun 22 22:53:53 1994
Copyright © ORACLE Corporation 1992, 19xx. All rights reserved
```

Después de estas líneas; va a mostrar las líneas que piden el nombre de usuario y la contraseña que el administrador de la base de datos da a cada usuario de la base. La contraseña no aparece en la pantalla mientras se escribe.

```
Enter user-name:

Enter password:
```

Cuando se escribe el user-name o nombre de usuario; se debe dar la tecla Enter para pasar al área del Password o contraseña; una vez tecleada la contraseña se debe dar Enter para acceder al entorno.

Si el nombre de usuario y la contraseña son correctos; que indica que es un usuario registrado para la base de datos; se mostrara las siguientes líneas:

```
Connected to:

ORACLE v7.0.x.xx x.x – Production
PL/SQL Release 2.0.xx.x.x – Production

SQL>
```

[18] En este capitulo sólo se analizará la parte correspondiente a SQL. Este programa puede realizar funciones de reporteador y tiene comandos específicos para estas funciones.
[19] Los mensajes de conexión pueden variar según las versiones

Una vez que se ve el prompt **SQL>** indica que ya se esta conectado y listos para trabajar con la base de datos.

5.3 Final de una sesión en SQL *Plus

Para salir de Oracle; escriba el siguiente comando en el prompt o indicador del programa y después de la tecla de Enter:

```
SQL>EXIT
```

5.4 Tipos de datos

La Tabla 8 presenta los tipos de datos más utilizados en Oracle con SQL:

Tipo	Descripción
CHAR()	Cadena entre 1 y 255 caracteres
DATE	Formato predefinido por el sistema; por ejemplo: DD-MMM-AAA o 12-NOV-98
LONG	Cadena extensa, con una longitud máxima de 65 535 caracteres
NUMBER(x,y)	Valor numérico con un máximo de 38 cifras. El usuario puede definir la longitud máxima (x) y la precisión (y); esta última es opcional
RAW	Dato binario que puede contener un máximo de 255 bytes. Se utiliza a menudo en el lenguaje de control
VARCHAR() o VARCHAR2()	Cadena con una longitud máxima de 2000 caracteres

Tabla 8. Tipos de datos

5.5 Definición de Datos en SQL *Plus

SQL *Plus soporta las instrucciones DDL del lenguaje SQL. Se verá a continuación algunos ejemplos sobre estas instrucciones; los ejemplos son cortos y sólo de algunas instrucciones puesto que este tipo de instrucciones ya se cubrieron en anteriores capítulos y además se va a notar que no hay prácticamente ningún cambio al aplicar este tipo de instrucciones en SQL *Plus.

5.5.1 Cómo crear una tabla

La sintaxis para crear una tabla dentro de este entorno queda de esta forma en base a un ejemplo[20]:

```
SQL> CREATE TABLE EMPLEADO(
1       emple_id        NUMBER(5)  NOT NULL,
2       emple_nombre     CHAR(50),
3       emple_paterno    CHAR(50),
4       emple_materno    CHAR(50),
5       emple_telefono   CHAR(10),
6       emple_sucurid    NUMBER(2)  NOT NULL
```

Si no se cometió ningún error en la sintaxis, ORACLE confirmará la creación de la tabla[21] con el siguiente mensaje:

```
Table created.
SQL>
```

Si se comete el error desplegara una pantalla con el código de error. En estos casos es bueno consultar la ayuda que viene en el menú HELP; donde vienen los códigos de error, la posible causa que lo origino y algunas veces la solución.

Si se desea conocer todas las tablas que se tienen en la actual base de datos; se puede realizar con la siguiente instrucción:

```
SQL>SELECT TABLE_NAME FROM TABS;
```

Como resultado de esta instrucción va a dar un listado de todas las tablas que están almacenadas en la base de datos.

5.5.2 Modificar la estructura de una tabla

Antes de querer modificar la estructura de la tabla; puede ser útil ver la estructura que actualmente tiene; esto se realiza con la instrucción DESCRIBE.

Sintaxis basada en un ejemplo:

```
SQL>DESCRIBE EMPLEADO        ó        SQL>DES EMPLEADO
```

[20] Para separar una sola instrucción en varias líneas consecutivas, presione la tecla Enter al final de cada renglón; de este modo, SQL*Plus numerará las líneas en forma automática. Es importante recordar que las instrucciones al terminarlas es necesario colocar un signo de punto y coma, de lo contrario se considerarían incompletas.

[21] El número máximo de tablas permitido en una base de datos es 65535, con 254 campos como máximo

Esta instrucción entregaría como resultado algo como esto:

```
SQL>DESC EMPLEADO

Nombre                    Null          Tipo
------------------------  ------------  --------------------
emple_id                  NOT NULL      NUMBER(5)
emple_nombre                            CHAR(50)
emple_paterno                           CHAR(50)
emple_materno                           CHAR(50)
emple_telefono                          CHAR(10)
emple_sucurid             NOT NULL      INT(2)
```

5.5.2.1 Para agregar un campo

Aplicado a la misma tabla quedaría de la siguiente manera:

```
SQL> ALTER TABLE EMPLEADO
1       ADD (emple_tipo_sangre  VARCHAR2(10)) ;
```

Oracle confirmará la adición de la columna con la siguiente línea:

```
Table altered

SQL>
```

Con "DESCRIBE" se puede verificar la adición del nuevo campo en la estructura de la tabla.

5.5.2.2 Modificar un campo

El campo[22] no puede ser eliminado; pero si puede ser modificado en su definición de tipo y en su anchura.

Aplicado a la misma tabla quedaría un ejemplo de modificación de columna al nuevo campo:

```
SQL> ALTER TABLE EMPLEADO
1       MODIFY(emple_tipo_sangre NUMBER(2) NOT NULL);
```

[22] Cuando un campo esta vacío no es posible modificar ni el ancho ni el tipo de datos. La instrucción NOT NULL sólo se aplica en las columnas con algún valor nulo.

Con "DESCRIBE" se puede verificar la modificación del campo en la estructura de la tabla.

5.5.3 Eliminación de una tabla

Eliminar una tabla es una operación irreversible.

Aplicando a la tabla de EMPLEADO:

```
SQL> DROP TABLE EMPLEADO
```

ORACLE informará de la tabla borrada

```
Table dropped

SQL>
```

Si la tabla no existiera; el mensaje de error sería similar a este:

```
SQL> DROP TABLE EMPLEADO
ERROR at line 1
ORA-00942: table or view does not exist

SQL>
```

5.5.4 Creación de vistas

Aplicado al ejemplo; se quiere construir una vista que se base en la tabla de EMPLEADO:

```
SQL>CREATE VIEW Empleados_View
  1       Codigo,
  2       Nombre,
  3       Materno
  4       AS
  5       SELECT  emple_id ,emple_nombre,emple_paterno
  6       FROM    EMPLEADO
  7       WHERE   emple_tipo_sangre=1 and
  8               emple_paterno='Lopez';
          .
```

Oracle informará de la vista creada

```
View Created

SQL>
```

5.5.5 Borrado de una vista

El comando que borra las vistas es DROP VIEW:

```
SQL>DROP VIEW Empleados_View
```

5.5.6 Transacciones

Una transacción es un elemento importante que se tiene que tomar en cuenta cuando se trabaja con una base datos Oracle. Una transacción es una serie de comandos SQL (DELETE, INSERT, UPDATE) ejecutados en forma sucesiva. Las transacciones se pueden establecer por medio de la instrucción COMMIT o anularse en forma parcial o total con el comando ROLLBACK.

El comando COMMIT fija una serie de acciones; ejecuta varios comandos en forma consecutiva.

```
SQL>COMANDO1;
SQL>COMANDO2;
SQL>COMANDO3;
SQL>.....
SQL>COMMIT;
```

5.6 Instrucciones DML

En esta parte no hay mucho que decir respecto a lo que se vio en el capitulo de las instrucciones DML. Es igual. Por ejemplo para el uso de un bloque SQL básico sería de la siguiente manera:

```
SQL>SELECT  emple_nombre, emple_paterno, emple_materno
  1      FROM EMPLEADO
  2      WHERE emple_paterno LIKE 'Lop$'
  3      ORDER BY emple_paterno;
```

El resultado lo va a mostrar de la siguiente manera:

```
emple_nombre          emple_paterno      emple_materno
--------------------- ------------------ ------------------------------------
Rojas                 Lopez              Jorge
Sánchez               Lopez              Hugo
Perez                 Lopez              Omar

3 rows selected.
```

Todas las instrucciones aprendidas respecto al bloque de SELECT van a poder ser aplicadas en este entorno de la misma manera.
Si se desea cambiar la consulta se ocupan los comandos como ED o EDIT para acceder al editor y modificarla de acuerdo a las necesidades u ocupar los comandos como LIST o R o L que se tocaron en este capítulo en el siguiente apartado.

5.7 Comandos de SQL Plus

Muchas veces ocurre que al querer hacer uso de un comando; se cometió un error y se desea verificar que es lo que ocurrió; en esta sección se verán una lista de comandos que ayudan a la tarea de visualizar que se hizo y la probable causa de los errores. Esto es útil para ahorrar tiempo en las búsquedas de errores.

5.7.1 Mostrar el último comando utilizado

```
SQL>LIST (Opción)
```

Porque se puede visualizar este último comando; la razón es muy sencilla; la última instrucción es guardada en un buffer y es reemplazada cuando se escribe el siguiente comando.[23] En la siguiente tabla se pueden visualizar las opciones:

Opción	Descripción	Ejemplo de sintaxis
(no hay opción)	Despliega el comando completo	LIST
n m	Despliega las líneas de la n a la m en el comando	LIST 2 6
N	Despliega la línea n del comando	LIST 2
n LAST	Despliega desde la línea n hasta la última línea del comando	LIST 2 LAST
n*	Muestra desde la línea n hasta la actual, que por lo general es la última línea	LIST 2 *
*	Despliega la línea actual	LIST *
* LAST	Empieza por la línea actual y termina con la última del comando	LIST * LAST
LAST	Muestra la última línea del comando	LIST LAST

[23] Al ser LIST un comando de SQL*Plus no necesita la terminación de punto y coma. Sólo se requiere teclear la instrucción con la opción y dar Enter.

5.7.2 Modificación de un comando

Una de las formas de corregir un comando es:

```
SQL> <número de línea que contiene el error> <nuevo texto>
```

Pero si el comando es muy grande es mejor trabajar de esta manera:

```
SQL>L <numero de la línea que contiene el error>
1       CHANGE / <texto erróneo /<nuevo texto>/
```

Es importante tomar en cuenta algunas consideraciones en este comando:

- Siendo comandos de SQL*Plus no se les coloca punto y coma.
- El uso de diagonales es para separar y puede utilizarse también el signo de admiración(¡)
- Se debe evitar tener espacios entre los separadores y el texto.

Ejemplo del uso de esta instrucción:

Se obtuvo un error al querer cambiar un campo de NULL a NOT NULL ; esto se realizó con una instrucción ALTER TABLE. Pero la instrucción no se escribió correctamente y este error nos mandó Oracle:

```
ALTER TABE EMPLEADO
           *
ERROR at line 1:
ORA-00940: invalid ALTER command
```

El error es porque en vez de poner ALTER TABLE se colocó ALTER TABE; la forma en que se debe corregir el error es entonces:

1. Escribir L1 para visualizar la línea 1; después se debe escribir C/TABE/TABLE/ y se desplegarán los cambios al momento ya corregidos

```
SQL>L1
      1*ALTER TABE DIREC
SQL>
       CHANGE/TABE/TABLE/

      1*ALTER TABLE DIREC
SQL>
```

Lo que queda por hacer es ejecutar nuevamente la línea para terminar la tarea que se tenía de alterar el campo.

5.7.3 Ejecución de un comando residente en memoria

La sintaxis para correr un comando o instrucción es la siguiente:

SQL>RUN	ó	SQL>/

5.7.4 Agregar otra línea de comando

La instrucción que ocupamos para agregar líneas aun comando que esta en el buffer es INPUT.
La sintaxis para agregar otra línea de comando es la siguiente:

```
SQL>INPUT <texto>
```

Otros ejemplos:
SE realiza una instrucción SELECT para visualizar los Nombres y Apellidos de los empleados de la siguiente manera:

```
SELECT emple_nombre, emple_paterno
FROM    EMPLEADO
```

Pero se desea agregar a este comando que esta residente en memoria un ORDER BY; la instrucción para agregar esta línea es:

```
SQL>INPUT ORDER BY emple_paterno;
```

Con un comando LIST o ejecutando la instrucción con RUN se puede visualizar la línea agregada al comando.

Otro ejemplo clásico que se puede presentar es que se omita algún campo que se deseaba mostrar en el SELECT; por ejemplo se deseaba colocar el campo emple_materno en la anterior instrucción.[24] Esta línea se puede agregar entre dos líneas del comando residente en memoria. Entonces se haría de la siguiente manera la adicción:

```
SQL>LIST 1
  1  SELECT emple_nombre, emple_paterno

SQL>INPUT ,emple_materno
```

Con una instrucción LIST se puede visualizar que fue agregada la línea.

[24] Se colocó una coma antes del campo debido a que forma parte de la sección del SELECT que va separado por comas los campos.

```
SQL>LIST
  1  SELECT emple_nombre, emple_paterno
  2  ,emple materno
  3  FROM    EMPLEADO
```

5.7.5 Eliminación de una línea de comando

La sintaxis para eliminar una línea de comando es la siguiente:

```
SQL>DEL <texto>
```

La forma de trabajo es la misma que cuando se agrega una línea. Por medio del comando DEL se elimina la línea actual del comando residente en memoria. Si se coloca el número de línea después de Del se borra la línea especificada.

5.7.6 Borrado del buffer

La sintaxis es la siguiente:

```
SQL>CLEAR BUFFER
      buffer cleared
SQL>
```

Oracle confirmará el borrado de los datos almacenados en el buffer con la leyenda "buffer cleared" [25]

Para verificar esto; se puede utilizar el comando LIST. Que nos va indicar "No lines in SQL bufer"

5.7.7 Forma de Guardar un comando SQL con SQL*Plus

En este entorno se tiene la facilidad de poder almacenar los comandos dentro de un archivo tipo SQL con la ventaja que pueden ser utilizados más adelante. La sintaxis es la siguiente:

```
SQL>INPUT
      <líneas de comando sin terminación en punto y coma>
     SAVE <nombre de archivo>
```

[25] Dependiendo de la versión de base de datos; varían los mensajes en español o en inglés según el caso. Como ejemplo cuando es borrado un buffer; el mensaje puede ser "buffer cleared" o "buffer suprimido".

Un ejemplo puede ser cuando se insertan valores a una tabla y se desea seguir guardando este comando para futuras inserciones:

Lo primero que debe hacerse es limpiar el buffer

```
SQL>CLEAR BUFFER
      buffer cleared
SQL>
```

El siguiente paso es ya la sintaxis

```
SQL> INPUT
      INSERT INTO EMPLEADO(emple_nombre, emple_paterno)
      VALUES
                              (&Nombre,&Paterno)
      SAVE Inserccion
```

Oracle una vez más va a confirmar el archivo creado con la leyenda "Created File Inserción"

En este caso gracias al símbolo de "&" este comando siempre va esperar un valor para cada campo en la tabla; en este caso esperará valor para el Nombre y para el Apellido Paterno. Ya no es importante lo que contiene el buffer porque el comando ha sido guardado; la extensión dada al archivo es SQL. Si se desea ejecuta el comando sin colocar el path o ruta donde esta guardado es preferible almacenar este archivo en la siguiente Ruta:
ORAWIN95/BIN
En esta ruta el comando se puede ejecutar desde SQL*Plus de esta manera:

```
SQL>START Insercion          o          SQL>@Inserccion
```

Cualquiera de las dos instrucciones nos va a permitir ejecutar este comando.

5.7.8 Comandos residentes en memoria

Como se vio en el anterior ejemplo; se hizo uso del comando START que sirve para invocar los comandos que están residentes en memoria o almacenados; la sintaxis de este comando es:

```
SQL>START <Nombre del arhivo>
```

La modificación de los comandos residentes se puede realizar gracias a los comandos que se muestran en la siguiente tabla. Para almacenar el contenido el archivo en el buffer, se debe de escribir GET <Nombre del arhivo>

Instrucción	*Funcionalidad*
APPEND <texto>	Agrega un texto al final de una línea
CHANGE/ anterior/nuevo/	Reemplaza el texto anterior por uno nuevo
CHANGE/texto	Elimina el texto señalado en la línea actual
CLEAR BUFFER	Borra el contenido del buffer
DEL	Suprime la línea actual
INPUT	Agrega una línea después de la línea actual
INPUT <texto>	Agrega texto después de la línea actual

5.8 Escritura en un archivo de salida de SQL*Plus

Gran cantidad de veces se va a necesitar que se almacene el resultado de una consulta en un archivo plano; puede servir como archivo para migrar información quizá de una base de datos a otra o para utilizar esta información como reportes para análisis; en fin representa una gran utilidad poder almacenar esta información.
La orden spool indica a Oracle que guarde la salida de SQL*Plus en un archivo de datos. Es necesario indicarle la ruta y el nombre del archivo donde se va a almacenar la salida. Una vez colocada la instrucción y un Enter cualquier cosa que se escriba en el SQL*Plus va a ser almacenada; por tanto estará esperando el query o consulta que va a traer la información.

Por ejemplo usando Windows; la orden quedaría de la siguiente manera:

```
SQL>spool c:\Ejemplo1.lst
SQL>SELECT.....
      FROM.....
```

Casi todos los sistemas operativos añaden toda la extensión **.lst** al nombre que se especifique. Esta extensión puede diferir de unos sistemas operativos a otros.

Una vez que la consulta terminó de entregar toda la información es necesario dar la siguiente instrucción:

```
SQL>spool off
```

El archivo obtenido se puede buscar en la ruta especificada y se puede ver con el bloc de notas de Windows o WordPad.

5.9 Manejo del diccionario de datos

Esta información es propia de la base de datos y da una gran visión respecto a todos los objetos que se tienen almacenados en la base. El diccionario de datos contiene información referente a los objetos, usuarios y eventos de la base. Esta información se despliega en forma de vistas y tablas, a las cuales se tiene acceso mediante el comando SELECT ...FROM.....WHERE

Ejemplos:

La siguiente instrucción nos muestra los objetos(tablas, vistas, índices, clusters, secuencias y demás elementos) que están disponibles para el usuario:

```
SQL>SELECT * FROM TAB;
```

Si se desea ver todas las vistas:

```
SQL>SELECT * FROM USER_VIEWS;
```

Si deseo ver la lista de índices junto con las tablas correspondientes y su tipo (único o no)

```
SQL>SELECT INDEX_NAME, TABLE_NAME, ÚNICOS
      FROM USER_INDEXES;
```

Si deseo ver la lista de índices junto con las tablas correspondientes y el campo donde se aplica cada índice:

```
SQL>SELECT INDEX_NAME, TABLE_NAME, COLUMN_NAME
      FROM USER_IND_COLUMNS;
```

Hay más tablas a del diccionario de datos; estas son las más comunes y a las que un usuario común tiene derecho; hay algunas otras propias del DBA a las cuales la mayoría de los usuarios no van tener acceso; es recomendable saber los privilegios que se tienen como usuarios para saber a que tenemos derecho y a que no.

5.10 Como modificar el ambiente SQL *Plus

El comando "SET system variable value" permite modificar el ambiente SQL *Plus para una sesión de trabajo.

La sintaxis es la siguiente:

SET variable_sistema valor

Algunas de las principales variables del sistema involucradas en esta tare son: PAUSE, ECHO, FEEDBACK, HEADING, LINESIZE, TERMOUT, ARRAYSIZE, AUTOCOMMIT, MAXDATA, DEFINE, LONG, NULL, SPACE, SQLNUMBER,NEWPAGE,COMPACTIBILITY, etc.

Estas variables pueden ser modificadas para personalizar la forma de trabajo en una sesión de SQL. La ventaja es que pueden ser cargadas estas instrucciones en un archivo o script para llamarlo cada vez que accedemos a una sesión.

Ejemplo de una sesión personalizada guardada en un script llamado Configuración.sql para utilizarla cada vez que se accede a usa sesión de SQL.

Configuración.sql

```
SET LINESIZE 1000;
SET TIME ON;
SET SERVEROUTPUT ON;
SET SQLPROMPT PRODUCCION>
```

Este script coloca un prompt o indicador que en vez de decir SQL va decir PRODUCCIÓN, va a colocar la hora del sistema, va a activar la modalidad del servidor para desplegar todos los mensajes que pueden ser programados y se van poder ver mayor número de caracteres en una línea.

La sesión quedaría de la siguiente manera:

```
12:24:33 PRODUCCION>
```

Se cubrieron los objetivos propuestos en la introducción del capítulo respecto a conocer el ambiente de SQL *Plus que es la herramienta que proporciona Oracle para manejar sus bases. También se vieron instrucciones DDL y DML en menor escala a lo que se ha estado manejando puesto que la intención es ver los entornos y como se aplica el lenguaje. Se manejaron varios comandos para tener una mayor facilidad a la hora de interactuar con SQL *Plus; se cubrieron algunos aspectos del manejo de diccionario de datos y el cambio en algunos casos del ambiente de SQL *Plus.
Con este capítulo se cubre la totalidad de la tesis al contemplar las principales instrucciones de SQL, y la aplicación de estas en dos plataformas como lo son Oracle y Access.

Conclusión general

Como se manifestó en la introducción; se cubrió el objetivo principal de mostrar los aspectos más importantes del lenguaje SQL; cubriendo las instrucciones DDL y las instrucciones DML para interactuar con una base de datos.

El conocimiento del lenguaje SQL con el uso adecuado nos proporciona las herramientas necesarias para interactuar con las bases de datos más importantes como es el caso de la base Oracle y la base Access que son actualmente dos de las bases de datos más comerciales.

En el caso de las base de datos Oracle, éstas hacen uso del lenguaje PL/SQL para el desarrollo de sus stores procedures(procedimientos almacenados en la base de datos) que dan a los desarrollos en Oracle el grado de potencialidad y eficiencia que se necesita en el manejo de grandes conjuntos de datos. Este lenguaje es basado en SQL.Tiene además el entorno SQL*Plus que permite el monitoreo y manejo de las bases de datos Oracle. El lenguaje para operar Oracle desde este entorno es SQL.

La base de datos Access vimos que es muy usado con Visual Basic, este usa instrucciones SQL para el manejo de sus datos. No siempre se usa el lenguaje SQL de manera tan clara, los creadores de manejadores han creado interfaces para facilitar el uso del lenguaje, pero va implicito el lenguaje SQL.

SQL estándar facilita el trabajo con plataformas y estas a su vez añaden característicasal lenguaje para dar mayor potencialidad; el lenguaje pierde la portabilidad pero gana profundidad.Cada plataforma maneja diferentes interfaces gráficas con una gran funcionalidad ocultando dentro de cada función una instrucción SQL.

En general se mostró el papel tan preponderante del lenguaje SQL que da la pauta de la necesidad de conocer el lenguaje y saberlo aplicar en las bases de datos.
SQL es un estándar y es una obligación para la gente involucrada en las bases de datos manejarlo adecuadamente.

Queda finalmente a la persona que lea este trabajo seguir extendiendo el conocimiento de este lenguaje en sus aspectos más detallados ya que este lenguaje es muy grande y conforme avanza en sus estandares sigue incrementando su potencialidad.

Apéndice A. Base de datos "Libros"

La base de datos Libros se conforma de las siguientes tablas que sirven de base para el capitulo de instrucciones DML.

Tabla: AUTOR

autor_id	autor_nombre	autor_fechanac	autor_vigente
2	Alejandro Chavez		S
3	Carolina Chavez Rodriguez	15/12/1955	S
4	Jorge Ubaldo Cruz Gomez	02/02/1969	N
5	Javier Chavez Rodriguez	25/06/1965	S
6	Burns, Timothy R.	11/11/1966	S
7	Bishop, Diane	12/12/1960	S
8	Kennedy, Martin Jr.	14/07/1975	S
9	Victor, George	15/07/1965	N
10	DeSilvia, Carl	31/12/1949	S
11	Jose Manuel Molina	22/10/1963	N
12	Roberto Villagran		N
13	Thompson, James E.	30/06/1966	N
14	Valley, Margo	14/08/1955	S
15	Cruis, Jonathon	15/04/1950	S
16	Engle, Martin	16/04/1932	S
17	Patrick, Elliot	21/09/1965	S
18	Wilso, Kathy		N
19	Osbern, Scott	12/09/1980	S
20	Marisol Maria Magdalena	23/01/1953	S
21	Jenkins, Peter	21/04/1949	S
22	Longworth, Carol	25/04/1962	S
23	Millerson, Robert T.	12/08/1955	N
24	Jackson, Marlo	12/04/1963	N
25	Person, Joen	28/02/1960	N
26	Omar, Carl	30/06/1933	S
27	Leon Uribe Miguel Angel	31/03/2027	S
28	Bilks, Tom		S
29	Chris, Rutherford	12/04/2022	S
30	Norman, Chas	12/07/1933	S
31	Greenwall, Valerie	31/05/1955	S
32	Milan Kundera	21/05/1956	S
33	Billerie, Mathew	21/06/1935	S
34	Magdalena Rodriguez Ramirez	07/06/1960	S
35	Smith, John		S
36	Jones, Thomas	15/04/1961	S
37	Fred, Ferd	02/02/2002	S
38	Alejandra Mercado Diep	12/03/2000	S

Tabla: VENTA

venta_semestre	venta_sucursalid	venta_ventasrep	venta_libroid	venta_unidades
1	1	A	3-2356984-5-5	10
3	1	B	3-2356984-5-5	116
1	1	E	3-2356984-5-5	10
1	1	B	3-2356984-5-5	10
1	2	D	3-2356984-5-5	11
1	1	D	3-2356984-5-5	10
1	1	A	3-2356984-5-5	13
2	1	B	3-2356984-5-5	11
2	1	E	3-2356984-5-5	11
2	1	B	3-2356984-5-5	11
3	1	D	3-2356984-5-5	11
3	1		3-2356984-5-5	12
3	2	A	3-2356984-5-5	11
3	1	B	3-2356984-5-5	11
3	1	D	3-2356984-5-5	11
3	2	A	3-2356984-5-5	11
3	1	D	3-2356984-5-5	11
3	1	B	3-2356984-5-5	36
4	1	A	3-2356984-5-5	14
4	1	A	3-2356984-5-5	11
4	1	C	3-2356984-5-5	11
4	1	D	3-2356984-5-5	26
4	1	F	3-2356984-5-5	11
4	1	E	3-2356984-5-5	11
2	1	C	6-5689654-2-8	11
3	1	C	6-5689654-2-8	22
3	1	B	6-5689654-2-8	35
4	1	D	6-5689654-2-8	22
4	1	A	6-5689654-2-8	12
4	1	B	6-5689654-2-8	24
4	1	E	6-5689654-2-8	11
4	1	B	6-5689654-2-8	11
4	1	F	6-5689654-2-8	11
1	1	F	4-5698765-6-3	10
1	1	D	4-5698765-6-3	10
1	2	B	4-5698765-6-3	16
1	1	F	4-5698765-6-3	10
1	1	E	4-5698765-6-3	28
3	2	B	4-5698765-6-3	26
3	1	C	4-5698765-6-3	11
4	1	E	4-5698765-6-3	11
4	1	B	4-5698765-6-3	11
4	1	B	4-5698765-6-3	11
1	1	B	2-0123458-9-7	10
1	2	F	2-0123458-9-7	22
1	1	C	2-0123458-9-7	10

venta_semestre	venta_sucursalid	venta_ventasrep	venta_libroid	venta_unidades
1	2	D	2-0123458-9-7	10
1	1	C	2-0123458-9-7	10
1	1	B	2-0123458-9-7	10
1	1	D	2-0123458-9-7	10
2	1	A	2-0123458-9-7	11
2	2	A	2-0123458-9-7	11
2	1	E	2-0123458-9-7	24
1	1	C	9-3213454-7-8	13
1	1	E	9-3213454-7-8	10
1	1	C	9-3213454-7-8	5
2	1	E	5-8978965-5-6	11
2	1	C	5-8978965-5-6	11
2	1	F	5-8978965-5-6	11
2	2	F	5-8978965-5-6	11
3	1	D	5-8978965-5-6	11
3	2	C	5-8978965-5-6	11
3	1	F	5-8978965-5-6	11
4	1	E	5-8978965-5-6	11
4	1	F	5-8978965-5-6	18
3	1	E	9-7896532-2-6	8
3	1	E	9-7896532-2-6	30
3	1	A	9-7896532-2-6	11
1	1	F	9-7001327-0-1	104
1	1	A	9-7001327-0-1	10
1	2	C	9-7001327-0-1	10
1	2	E	9-7001327-0-1	10
1	2	E	9-7001327-0-1	18
1	1	D	9-7001327-0-1	10
1	1	A	9-7001327-0-1	10
2	1	B	9-7001327-0-1	8
2	2	B	9-7001327-0-1	11
1	1	C	5-5465254-5-1	10
1	1	F	5-5465254-5-1	21
1	1	B	5-5465254-5-1	10
1	1	B	5-5465254-5-1	24
1	1	E	5-5465254-5-1	29
1	2	C	5-5465254-5-1	10
1	1	F	5-5465254-5-1	10
1	1	F	5-5465254-5-1	27
2	1	D	5-5465254-5-1	11
2	2	E	5-5465254-5-1	23
2	1	D	5-5465254-5-1	11
2	1	E	5-5465254-5-1	11
2	1	D	5-5465254-5-1	30
2	1	B	5-5465254-5-1	11
3	1	B	5-5465254-5-1	23
3	2	E	5-5465254-5-1	5

venta_semestre	venta_sucursalid	venta_ventasrep	venta_libroid	venta_unidades
3	1	C	5-5465254-5-1	26
3	1	A	5-5465254-5-1	11
3	1	E	5-5465254-5-1	20
3	1	E	5-5465254-5-1	31
2	1		9-6133518-4-1	14
1	1	B	6-7894567-7-9	8
4	1	F	6-7894561-5-2	8
4	1	E	6-7894561-5-2	11
4	1	C	6-7894561-5-2	14
4	1	C	6-7894561-5-2	11
1	2	D	5-5698741-0-0	17
1	1	A	5-5698741-0-0	10
1	1	A	5-5698741-0-0	10
1	2	B	5-5698741-0-0	32
2	1	A	5-5698741-0-0	11
2	1	E	5-5698741-0-0	114
2	2	C	5-5698741-0-0	12
2	1	B	5-5698741-0-0	11
2	2	C	5-5698741-0-0	11
2	1	A	5-5698741-0-0	17
2	1	C	5-5698741-0-0	15
3	2	F	5-5698741-0-0	11
3	2	D	5-5698741-0-0	11
3	1		5-5698741-0-0	11
3	1	B	5-5698741-0-0	17
2	1	A	9-6120517-3-2	11
2	2	F	9-6120517-3-2	14
2	1	F	9-6120517-3-2	11
2	1	F	9-6120517-3-2	11
2	1	A	9-6120517-3-2	26
2	2	E	9-6120517-3-2	11
3	1	A	9-6120517-3-2	11
3	2	D	9-6120517-3-2	11
3	2	B	9-6120517-3-2	14
3	1	F	9-6120517-3-2	11
3	1		9-6120517-3-2	32
3	1	C	9-6120517-3-2	11
3	2	D	9-6120517-3-2	11
4	1	D	9-6120517-3-2	11
4	1	C	9-6120517-3-2	11
4	1	E	9-6120517-3-2	11
4	1	F	9-6120517-3-2	11
4	1	B	9-6120517-3-2	11
4	1	F	9-6120517-3-2	15
2	1	E	5-8956231-2-5	11
2	2	C	5-8956231-2-5	18
2	1	D	5-8956231-2-5	31

venta_semestre	venta_sucursalid	venta_ventasrep	venta_libroid	venta_unidades
2	1	D	5-8956231-2-5	19
2	1	A	5-8956231-2-5	35
1	1	A	5-5689654-4-5	20
1	2	A	5-5689654-4-5	10
2	1	C	5-5689654-4-5	11
2	1	D	5-5689654-4-5	29
3	1	F	5-5689654-4-5	14
3	1	E	5-5689654-4-5	11
3	1	E	5-5689654-4-5	19
3	1	D	5-5689654-4-5	11
3	1	F	5-5689654-4-5	11
3	1	C	5-5689654-4-5	11
4	1	C	5-5689654-4-5	114
4	1	E	5-5689654-4-5	11
4	1	C	5-5689654-4-5	23
4	1	F	5-5689654-4-5	5
4	1	A	5-5689654-4-5	11
4	1	F	5-5689654-4-5	11
4	1	D	5-5689654-4-5	17
4	1	B	5-5689654-4-5	11
4	1	E	5-5689654-4-5	11
4	2	D	5-5689654-4-5	35
4	1	C	5-5689654-4-5	11
4	1	D	5-5689654-4-5	36
1	1	C	9-6133518-4-1	33
2	1	B	9-6133518-4-1	11
2	1	D	9-6133518-4-1	11
2	2	B	9-6133518-4-1	11
2	1	A	9-6133518-4-1	26
2	1	F	9-6133518-4-1	11
2	1	B	9-6133518-4-1	11
2	1	A	9-6133518-4-1	36
3	1	A	9-6133518-4-1	11
3	1	C	9-6133518-4-1	11
3	1	E	9-6133518-4-1	11
3	2	C	9-6133518-4-1	11
3	1	D	9-6133518-4-1	15
4	1	D	9-6133518-4-1	11
4	1	D	9-6133518-4-1	26
1		F	5-6987456-8-9	10
1	1	B	5-6987456-8-9	24
1	1	A	5-6987456-8-9	10
1	2	C	5-6987456-8-9	10
1	1	E	5-6987456-8-9	10
1	1	E	5-6987456-8-9	14
2	1	F	5-6987456-8-9	22
2	2	D	5-6987456-8-9	11

venta_semestre	venta_sucursalid	Venta_ventasrep	venta_sucursalid	venta_unidades
2	1	C	5-6987456-8-9	5
4	1	A	5-6987456-8-9	30
4	1	B	5-6987456-8-9	11
4	2	E	5-6987456-8-9	11
3	1	A	5-5698753-9-5	24
3	1	A	5-5698753-9-5	11
3	1	F	5-5698753-9-5	11
4	1	F	5-5698753-9-5	11
4	1	B	5-5698753-9-5	11
4	1	A	5-5698753-9-5	31
4	1	A	5-5698753-9-5	19
4	1	C	5-5698753-9-5	11
4	1	A	5-5698753-9-5	29

Tabla: LIBRO

libro_ISBN	libro_titulo	libro_autorid	libro_aniopublic	libro_editorialid	libro_descripcion	libro_subtitulo	libro_comentarios	libro_notas
0-1597536-4-4	ORACLE	1	1986	3	La mas completa introd			
1-2156486-5-5	Backs	5	1996	5	Sample - Please Delete	Sample VB Program		
1-4569853-5-0	El Verano	6	1996	8	Novela	Lo que se fue	Excelente Novela	Novela Dramática
1-5025896-6-5	The Cellars	3	1980	1				
2-0123458-9-7	Chocolate Lovers Paradise	4	1995	5	Candy making with choc	Chocolate		Recetas
3-2356984-5-5	Cakes, Pies, and More	7	1992	2				Recetas
4-5689654-4-5	The New Vineyard of Virginia	21	1996	7	Virginia's new industry			Geography
4-5698765-6-3	Cooking with Spices	30	1989	11				Recetas
5-5465254-5-1	Pasta and More	1	1955	6	Cooking pasta and sauces			Third edition
5-5689756-2-5	Tom's Fish Recipes	16	1976	9	Fish and Lobster Recipes	Fish		
5-5698741-0-0	The Cajun Encyclopedia	4	1990	6	Spicy Foods	Cajun		
5-5698753-9-5	Vegetarian Cooking	22	1986	9				
5-6987456-8-9	Training New Chefs	23	1996	14				
5-8956231-2-5	The Joy of Eating	17	1956	5	Epicurean Delights	Cooking		Out of print
5-8978965-5-6	Ice Cream Dream Treats	16	1985	11				
6-1234568-9-5	Cooking with Zest	23	1977	10		Spicy foods		
6-5689654-2-8	Cookies, Cookies, Cookies	11	1992	2	What else, Cookies!			
6-5698745-5-9	Norm's Guide to Beer	8	1984	3	Beer	Beer		
6-7894561-5-2	The ABCs of Wine Tasting	9	1993	2				
6-7894567-7-9	Secrets of the Masters	10	1989	2	Conversational guide on Beverages	Wine		
7-5698563-5-8	Susan Teaches Chardonnay	12	1995	5		Wine		
7-8956234-1-5	The Vineyard of California	13	1987	1	Visits to local California Vineyards			
8-9874563-5-6	The Great Wines of California	14	1986	5	California Wines	Wine	the best	
9-4569875-5-6	The Winter Book of Stews	36	1991	7				
97896532-2-6	Kitchen Layout and Design	35	1994	4	Construction of the kitchen			

9-8956125-5-6	The Caviar Journal	15	1994	3		Food		
libro_ISBN	**libro_titulo**	**libro_autorid**	**libro_aniopublic**	**libro_editorialid**	**libro_descripcion**	**libro_subtitulo**	**libro_comentarios**	**libro_notas**
9-6133518-4-1	The Summer Book of Chops	4	1990	3				
9-7632147-4-1	Sample Title	3	1990	4				
9-6120517-3-2	The Cellars of France	6	1997	4				
9-3213454-7-8	Food Preparation Tips	11	1994	5	Guide, only guide			
9-4566575-8-9	El caballo de Troya	6	1997	8	Novela historica			
9-7001327-0-1	Oregon State Pinot Noirs	23	1979	14				

Tabla EDITORIAL

edit_id	edit_nombre	edit_compania	Edit_direccion	edit_ciudad	edit_estado	edit_cp	edit_telefono	edit_fax	edit_comentarios
1	Sams	Sams Publishing	103 W. 103rd Street	Indianapolis	IN	52364	512-555-6598	512-555-6685	
2	Que	Que Publishing	103 W. 102nd Street	Indianapolis	IN	55632	512-569-5698	512-555-4512	
3	Bill-	Bill's Publishing House	201 West 99th Street	New York	NY	10028	212-546-8975		
4	Benson	Benson Bookhouse	123 Main Street	Albany	NY	45678	654-569-8899	654-222-5555	
5	Billiard	The Billiard Press	201 Inglenook Lane	Sonoma	CA	98563	444-555-3333	444-564-8965	
6	Cork	Cork Publishers	212 Main Street	San Francisco	CA	98765	901-896-5645	901-555-5463	
7	Converse	Converse and James Pre	225 Chateau	Providence	MX	08564	555-666-8965	555-666-7800	
8	Bastion	Bastion and Waite Publis	222 East Fifth	Cincinnati	OH	45202	513-564-8956	513-623-5695	
9	Promise	Promise Press	225698 Old Tipton Road	Springfield	IL	69875	521-562-5453		
10	Cornstone	Cornstone Press	21 Lake Street	Baltimore	MD	23658			
11	Homer	Homer Press	22256 Elliot Drive	Oxnard	CA	90028	952-254-5456		

Tabla: SUCURSAL

sucur_id	sucur_nombre	sucur_direccion	sucur_telefono
1	suc. zona centro	Av. 20 de Noviembre 223	57890657
2	suc. zona sur	Coapa 23	54678906

Apéndice B Glosario de Terminos

Administrador de Base de datos(DBA): Usuario de una base de datos autorizado para signar privilegios de acceso a los usuarios de la base de datos o para modificar las opciones de datos que afectan a todos los usuarios.

Alias: Nombre temporal asignado a una tabla o una columna de una instrucción SQL.

Base de datos: Conjunto de tablas, vistas u objetos almacenados formando parte de una estructura.

Buffer: Zona de almacenamiento temporal que utliza SQL*Plus para almacenar los comandos utilizados.

Campo: Esta representado por las columnas de una tabla

Comando: Declaración de SQL

Consulta: Declaración SQL que permite extraer datos de una o más tablas. Es también llamada Query.

COMMIT: Valida y ejecuta la transacción actual

DBMS: El SGBD o DBMS son dos términos que se refieren a lo mismo, al sistema de gestión o administración de la base de datos, Agrupación de varios programas, procedimientos y lenguajes que permiten manipular y describir los datos almacenados en una base de datos.

DDL: Instrucciones SQL de definición de datos. Instrucciones que afectan la estructura de la base de datos.

DML: Instrucciones de manipulación de datos. Instrucciones para ejecutar consultas

Foreign key: Llave foránea; es el conjunto de campo o campos que van a tener relación con el campo o campos de otra tabla.

GRANT: Definición de privilegios que permite al usuario realizar ciertas operaciones con las tablas de datos.

Instrucción: Palabra reservada dentro de una declaración de SQL que en ocasiones aparece seguida de un parámetro.

NULL: Permite introducir valores nulos en una tabla. Las consultas pueden seleccionar o ignorar registros donde aparecen los valores nulos (NULL) en los campos seleccionados.

NOT NULL: Prohíbe la instroducción de un valor nulo en la tabla.

Primary key: Llave primaria; campo o campos identificados para definir en forma única cada registro de una tabla

Privilegio: Derecho que el autor de una tabla concede a un usuario para realizar operaciones en la misma.

Registro: Representa las filas de una tabla. Elemento individual de una base de datos.

Relación: Vínculo compuesto de entidades y atributos

ROLLBACK: Anula las últimas modificaciones realizadas en la base de datos durante una transacción

Sintaxis: Conjunto de instrucciones que determinanla forma de crear declaraciones vá-+lidas en SQL.

Spool: Espacio de memoria asignado en forma dinámica para contener un archivo que se habra de imprimir

SQL: Lenguaje Estructurado de Consulta. Lenguaje de base de datos.

Apéndice C. Palabras Reservadas de SQL

Las palabras reservadas que están en este apéndice son palabras reservadas de SQL que tienen un significado sintáctico importante en el lenguaje. No pueden utilizarse como identificadores de nombres de variables, nombres de procedimientos, de tablas o campos.

ACCESS	IS	SESSION
ADD	INSERT	TRIGGER
ALTER	INDEX	THEN
AUDIT	INCREMENT	TO
ALL	LEVEL	UNIQUE
AND	LIKE	USER
ANY	LOCK	UPDATE
AS	MAXEXTENTS	UID
ASC	MODE	UNION
BETWEEN	NOCOMPRESS	VALIDATE
BY	NUMBER	VARCHAR2
CHAR	NOWAIT	VIEW
CLUSTER	NOT	VALUES
COMPRESS	NOAUDIT	VARCHAR
CREATE	NULL	WHENEVER
CHECK	OF	WITH
COLUMN	ON	WHERE
CONNECT	OPTION	
COMMENT	ORDER	
DESC	ONLINE	
DROP	OFFLINE	
DEFAULT	OR	
DELETE	PRIOR	
DISTINCT	PRIVILEGES	
ESCLUSIVE	PCTFREE	
ELSE	PUBLIC	
EXISTS	ROWLABEL	
FORM	ROWS	
FROM	RESOURCE	
FILE	REVOKE	
FOR	ROW	
FLOAT	ROWID	
GROUP	RAW	
GRANT	RENAME	
HAVING	SHARE	
IDENTIFIED	STATE	
IMMEDIATE	SMALLINT	
INITIAL	SYSDATE	
INTEGER	SET	
IN	SYNONYM	
INTERSECT	SUCCESSFUL	

Bibliografía

Jeffrey D. Ullman, Jennifer Widom
Introducción a los Sistemas de Bases de Datos
PRENTICE HALL, México 1999

Michael Abbey, Michael J. Corey
Oracle 8 Guía de Aprendizaje
McGRAW- HILL/INTERAMERICANA DE ESPAÑA S.A.U. 1998

Thao Vo, Armand St-Pierre
SQL *PLUS bajo Oracle Guía práctica con ejercicios.
1997, Editorial Trillas S.A. de C.V.

Apuntes de Bases de Datos de la Enep Aragon

Kevin Loney
Oracle 8 Manual del administrador
McGRAW- HILL/INTERAMERICANA DE ESPAÑA S.A.U. 1999

Artículo de Internet; SQL FAQ: SQL Standard
http://epoch.cs.berkeley.edu:800/sequoia/dba/montage/FAQ/SQL_TOC.html

Artículo de Internet; SQL Tutorial
http://www.1keydata.com/sql/sql.html

Artículo de Internet; Oracle: SQL*Plus Tutorial
http://www.vwp.edu/baldwin/sqlplus.htm

Artículo de Internet; Introduction to Structured Query Language
http://w3.one.net/~ihoffman/sqltut.htm

Artículo de Internet; SQL Interpreter & Tutorial with live practice database
http://sqlcourse2.com/

Artículo de Internet; SQL Tutorial
http://www.cs.pdx.edu/~len/SQLtut.html

Adoración de Miguel Castaño, Mario G. Piattini Velthuuis
Fundamentos y modelos de bases de datos
1999 ALFAOMREGA GRUPO EDITOR S.A. de C.V.

Martin Rinehart
Desarrollo de bases de datos en Java
McGRAW- HILL/INTERAMERICANA DE ESPAÑA S.A.U. 1999

Santiago Zorrilla A. ,Miguel Torres X.
Guía para elaborar la tesis
1992 McGRAW-HILL INTERAMERICANA DE MÉXICO, S.A. de C.V

Softtek UdeN ORACLE
Diseño Relacional de Bases de Datos.
Manual de curso.
Octubre 1997

DDEMESIS
Desarrollo de una Aplicación bajo Arquitectura Cliente/Servidor con Microsoft Visual basic 4.0 y Microsoft SQL Server 6.5
Manual de Curso

Microsoft
Implementing a Database Design SQL Server 6.5
Manual de Curso

Dorsey Hudicka
Oracle 8 Diseño de bases de datos UML
McGRAW- HILL/INTERAMERICANA DE ESPAÑA S.A.U. 1999

Kroenke, D. M.
Procesamiento de Bases de datos
Prentice Hall 1997 2da. Edición

Abraham Silverchatz
Fundamentos de Bases de Datos
Mc Graw Hill