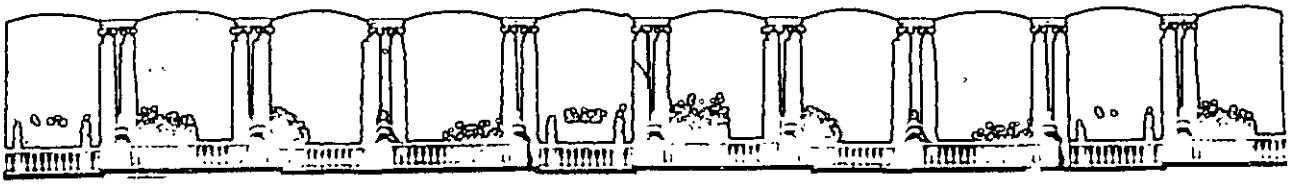


FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

CENTRO DE INFORMACION Y DOCUMENTACION
"ING. BRUNO MASCANZONI"

El Centro de Información y Documentación Ing. Bruno Mascanzoni tiene por objetivo satisfacer las necesidades de actualización y proporcionar una adecuada información que permita a los ingenieros, profesores y alumnos estar al tanto del estado actual del conocimiento sobre temas específicos, enfatizando las investigaciones de vanguardia de los campos de la ingeniería, tanto nacionales como extranjeras.

Es por ello que se pone a disposición de los asistentes a los cursos de la DECFI, así como del público en general los siguientes servicios:

- **Préstamo interno.**
- **Préstamo externo.**
- **Préstamo interbibliotecario.**
- **Servicio de fotocopiado.**
- **Consulta a los bancos de datos: librunam, seriunam en cd-rom.**

Los materiales a disposición son:

- **Libros.**
- **Tesis de posgrado.**
- **Noticias técnicas.**
- **Publicaciones periódicas.**
- **Publicaciones de la Academia Mexicana de Ingeniería.**
- **Notas de los cursos que se han impartido de 1980 a la fecha.**

En las áreas de ingeniería industrial, civil, electrónica, ciencias de la tierra, computación y, mecánica y eléctrica.

El CID se encuentra ubicado en el mezzanine del Palacio de Minería, lado oriente.

El horario de servicio es de 10:00 a 19:30 horas de lunes a viernes.

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO. Postal M-2285
Teléfonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 521-4020 AL 26

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

A LOS ASISTENTES A LOS CURSOS

Las autoridades de la Facultad de Ingeniería, por conducto del jefe de la División de Educación Continua, otorgan una constancia de asistencia a quienes cumplan con los requisitos establecidos para cada curso.

El control de asistencia se llevará a cabo a través de la persona que le entregó las notas. Las inasistencias serán computadas por las autoridades de la División, con el fin de entregarle constancia solamente a los alumnos que tengan un mínimo de 80% de asistencias.

Pedimos a los asistentes recoger su constancia el día de la clausura. Estas se retendrán por el periodo de un año, pasado este tiempo la DECFI no se hará responsable de este documento.

Se recomienda a los asistentes participar activamente con sus ideas y experiencias, pues los cursos que ofrece la División están planeados para que los profesores expongan una tesis, pero sobre todo, para que coordinen las opiniones de todos los interesados, constituyendo verdaderos seminarios.

Es muy importante que todos los asistentes llenen y entreguen su hoja de inscripción al inicio del curso, información que servirá para integrar un directorio de asistentes, que se entregará oportunamente.

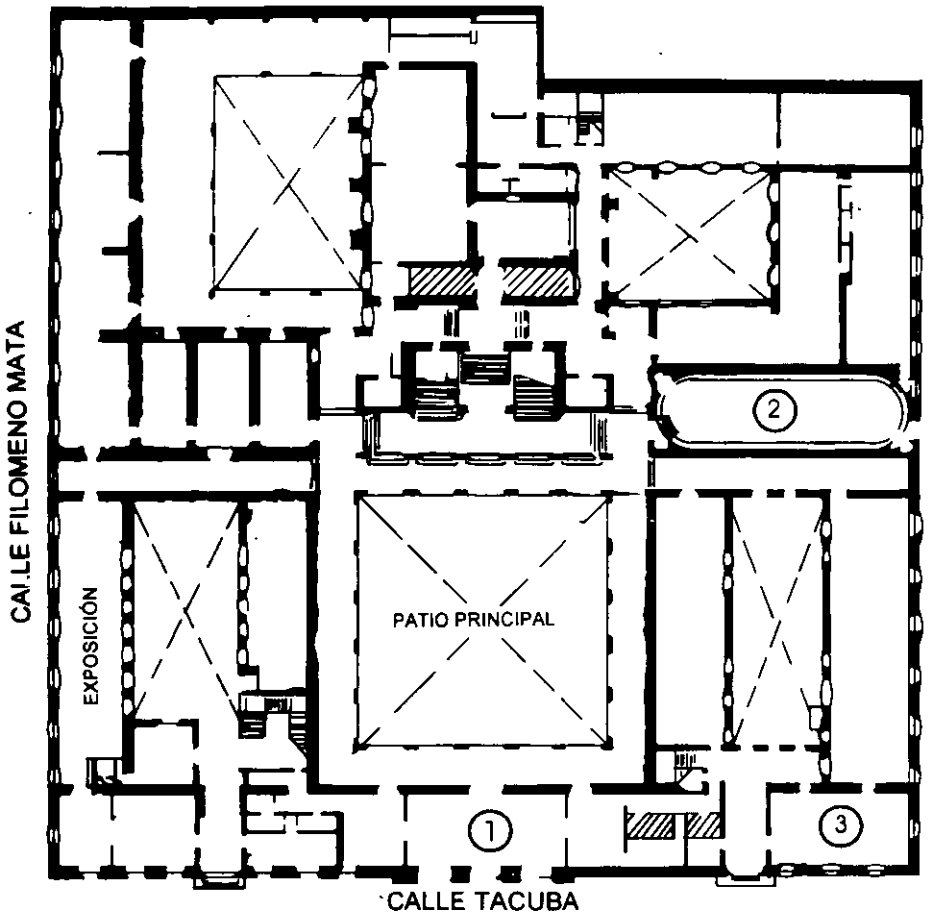
Con el objeto de mejorar los servicios que la División de Educación Continua ofrece, al final del curso deberán entregar la evaluación a través de un cuestionario diseñado para emitir juicios anónimos.

Se recomienda llenar dicha evaluación conforme los profesores impartan sus clases, a efecto de no llenar en la última sesión las evaluaciones y con esto sean más fehacientes sus apreciaciones.

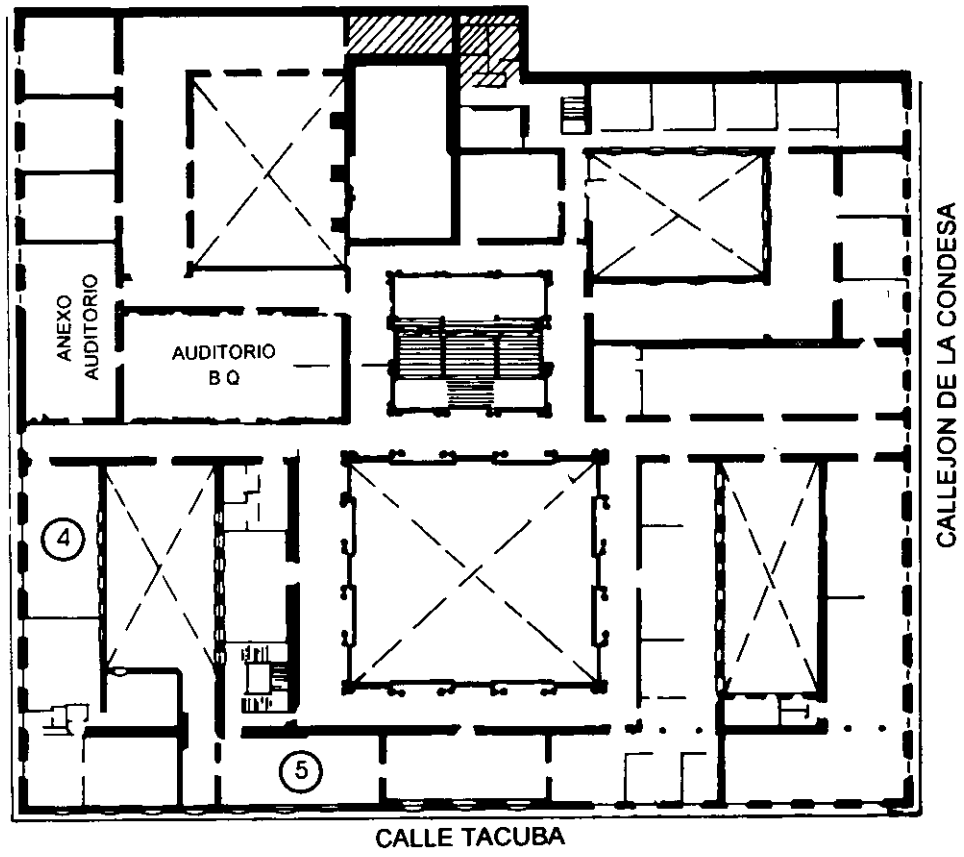
Atentamente
División de Educación Continua.

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO. Postal M-2285
Teléfonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 521-4020 AL 26

PALACIO DE MINERIA

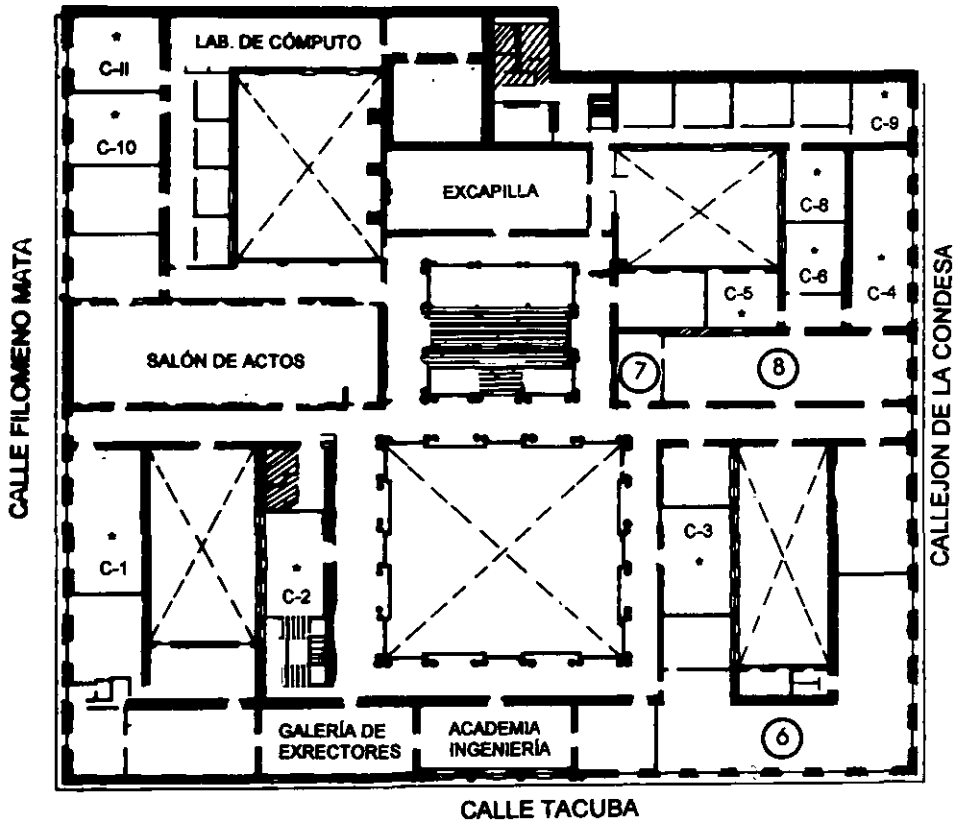


PLANTA BAJA



MEZZANINNE

PALACIO DE MINERIA



Ier. PISO

GUÍA DE LOCALIZACIÓN

1. ACCESO
2. BIBLIOTECA HISTÓRICA
3. LIBRERÍA UNAM
4. CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN "ING. BRUNO MASCANZONI"
5. PROGRAMA DE APOYO A LA TITULACIÓN
6. OFICINAS GENERALES
7. ENTREGA DE MATERIAL Y CONTROL DE ASISTENCIA
8. SALA DE DESCANSO

SANITARIOS

* AULAS

DIVISIÓN DE EDUCACIÓN CONTINUA
FACULTAD DE INGENIERÍA U.N.A.M.
CURSOS ABIERTOS

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

BASES DE DATOS Y SQL CON ORACLE

MATERIAL DIDACTICO

M.A.Y.O, 1996

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO. Postal M [illegible]
Teléfonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 512-5121 521-4020 AL 26

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE INGENIERÍA
UNIDAD DE SERVICIOS DE CÓMPUTO ACADÉMICO

BASES DE DATOS Y SQL

Elaborado por:
Jorge Barba Atilano
Miriam Bautista Angeles
Claudia A. Cordero Hidalgo
Gloria Morales Sánchez

Enero 1996
Revisión, mayo 1996

INTRODUCCION

En la actualidad el buen manejo de la información es primordial para el correcto desempeño de una institución o empresa.

La manera en que se manipula esta información en los equipos de cómputo ha sido diversa a través de la evolución de éstos, generando la necesidad de usar nuevas técnicas de optimización en la organización de la información.

Cuando el volumen de información que podía manejarse en una sola computadora empezó a ser grande, surgió la necesidad de desarrollar un software que fuera capaz de proveer un método eficaz de control y mantenimiento de tales volúmenes de información. Fué entonces cuando surgió la tecnología de manejo de grandes volúmenes de datos (Bases de Datos).

Esta tecnología proponía ver la información en forma global y no en forma particular o asociada a una aplicación o conjunto de programas, entonces fué necesario diseñar software ambiental que fuera capaz de llevar el control en forma global, el cual se denominó Sistema Manejador de Bases de Datos. Debido a esta característica se empezó a desplazar la convencional forma de trabajo que consistía en un conjunto de programas de aplicación y sus archivos maestros.

Ante tal necesidad comenzaron a emerger diversos Sistemas Manejadores de Bases de Datos, algunos desarrollados por proveedores líderes y otros desarrollados por prestigiadas casas de software. Estos DBMS (Sistemas Manejadores de Bases de Datos) fundamentaron su funcionamiento en diferentes modelos de datos, siendo los tres más aceptados y conocidos, los modelos Jerárquico, de Red y Relacional.

Cada uno de estos modelos tiene ventajas y desventajas, más las unas que las otras, dependiendo de la naturaleza de la información contenida en la Base de Datos correspondiente. Sin embargo el modelo relacional por su gran flexibilidad sin la necesidad de preestablecer "ligas" o "asociaciones" entre las entidades o componentes de la Base de Datos, por la simplicidad en el diseño (el cual se reduce a un conjunto de tablas) y por la ayuda de métodos de diseño como la normalización de datos, tomó mayor popularidad con respecto a los otros modelos.

Objetivos:

- Proporcionar al alumno los conceptos fundamentales de las bases de datos.
- Introducir al alumno en la construcción de bases de datos relacionales con la ayuda del modelo de datos entidad-relación.
- Dar al alumno las bases del lenguaje SQL para la construcción de Bases de Datos Relacionales.
- Mostrar al alumno la capacidad de SQL*Plus para la generación de reportes.

TEMA I. Introducción a las bases de datos relacionales.

Objetivo:

Al final del tema el alumno:

- Conocerá los conceptos básicos acerca de las bases de datos.
- Enumerará las diferencias entre un manejador de bases de datos y un manejador de archivos.
- Listará las ventajas que proporciona el modelo de bases de datos relacional.

CONCEPTOS BÁSICOS

Sistema

Conjunto u ordenación de elementos relacionados de tal manera que forman un todo orgánico.

Sistema basado en computación

Conjunto u ordenación de elementos organizados para llevar a cabo algún método, procedimiento o control mediante procesamiento de información.

Dato

Los datos son representaciones abstractas de algo.

Información

Conjunto de datos relacionados que nos reportan algo que es de nuestro interés.

Base de datos

Una colección grande de información a la que se accede mediante el software y que es una parte integral del funcionamiento del sistema.

- Cada persona y cada programa que estén **autorizados** para accesarla lo pueden hacer.
- Los datos pueden ser modificados por aquellos **autorizados** para hacerlo.
- Una base de datos diseñada propiamente debe minimizar la cantidad de información redundante.

Manejador de base de datos

Un manejador de bases de datos **DBMS** (Database Management System) es un módulo de programas que constituye la interfaz entre los datos de bajo nivel almacenados en la base de datos y los programas de aplicaciones y las consultas hechas al sistema. Y se responsabiliza de las siguientes tareas:

Interacción con el manejador de archivos.
Implantación de la integridad.
Almacena, permite obtener y modificar datos.
Puesta en práctica de la seguridad.
Respaldo y recuperación.
Control de consistencia.

Tabla
Es una representación en dos dimensiones con una o más columnas y cero o más renglones.

Campo
Columna de una tabla.

Registro
Renglón de una tabla.

DICCIONARIO DE DATOS

Es la colección de detalles de los contenidos de los flujos de datos, almacenamientos y procesos (descripción de entidades y atributos). En el diccionario de datos toda esta información se guarda en forma estructurada (características básicas). Sin embargo, el nombre de diccionario de datos es de un uso muy amplio, por lo que debería llamarse guía de proyecto.

Los datos que contendrá dicho documento son los siguientes:

CAMPOS

1. Nombre del Campo
2. Descripción de la entidad ó atributo
3. Tipo de dato almacenado
   - Carácter
   - Numérico
   - Fecha
   - Lógico
   - Calculado
4. Longitud
   - Si es numérico indicar la precisión.
5. Indexado
6. Observaciones o consideraciones
   - Permite datos nulos?
   - Es un campo único?
   - Es llave primaria?

Reseña histórica

Con el anvance de la tecnología de cómputo la necesidad de manejar la información en bases de datos aumentó, para poder hacerlo se crearon "modelos de datos" para describir los datos en los niveles conceptual y de visión.

1950 Inicio del procesamiento de datos. Segunda generación de computadoras.

1960 Surgimiento de COBOL (Por sí mismo define archivos)
Discos y cintas para transportar la información.
Primeros DBMS (Database Management System). Se basan en el modelo jerárquico y de red.

1970 Se publica la definición del modelo relacional desarrollado por el Dr. Codd, basado en la simplicidad matemática del Álgebra Relacional (manejo de conjuntos).
Aparece SQL Structured Query Language

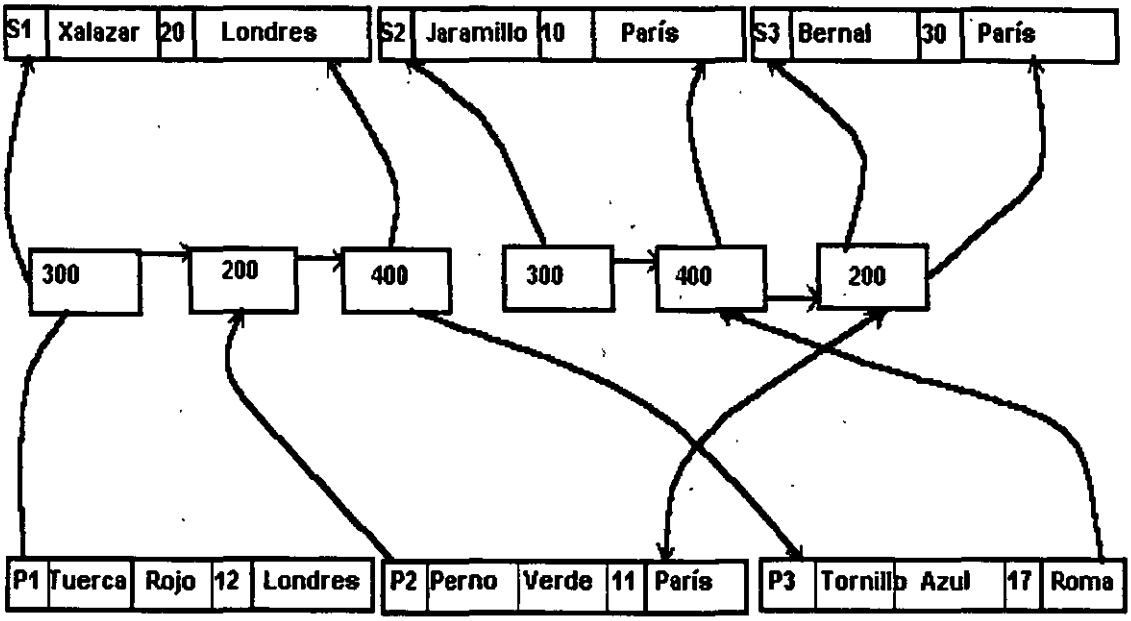
1978 ORACLE hace la primera implementación real del modelo relacional.

1986 El Dr. Date define las Bases de datos distribuidas.

1993 ORACLE implementa bases de datos distribuidas.

Modelo de red

Los datos en el modelo de red se representan por medio de "registros" y las relaciones entre los datos se representan con ligas, que pueden considerarse apuntadores. Los registros de la base de datos se organizan en forma de conjuntos de gráficas arbitrarias.

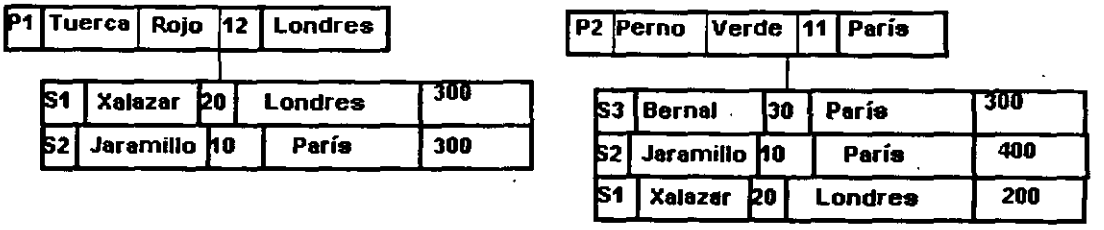


MODELO DE RED

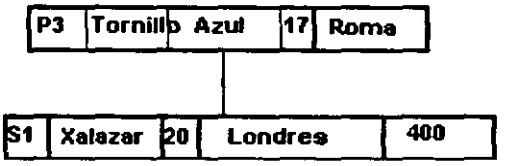
El modelo anterior nos muestra el problema de proveedores y partes, por cada proveedor existe un registro, lo mismo que por una parte y existe otro registro llamado conector, una ocurrencia en el conector representa una asociación entre un proveedor y una parte y contiene datos que describen esa asociación (la cantidad de la parte suminstrada). Cada ocurrencia del conector se guarda en dos cadenas (una de proveedores y una cadena de partes, la correspondencia entre el proveedor y los registros asociados del conector, es de uno a muchos, que es una de las mayores características del modelo de red, el modelar relaciones de muchos a muchos. Más adelante se explicará más detalladamente en qué consisten éstas relaciones.

Modelo jerárquico

El modelo jerárquico es similar al modelo de red en cuanto a que los datos y las relaciones entre los datos se representan por medio de registros y ligas, respectivamente.El modelo jerárquico difiere del de red en que los registros están organizados como conjuntos de árboles en vez de gráficas arbitrarias.



MODELO JERÁRQUICO



El mismo problema visto para el modelo jerárquico se representa a través de una estructura de árbol, en éste caso las partes van en un nivel superior que los proveedores. Cada árbol se compone de una ocurrencia de registro de proveedor subordinadas, uno para cada proveedor de la parte. Cada ocurrencia del proveedor incluye la cantidad correspondiente . El conjunto de ocurencias puede contener cualquier número de miembros, incluso cero. El tipo de registro en el tope del árbol se conoce como Raíz, en general, la raíz puede tener cualquier número de dependientes de nivel inferior, y así sucesivamente hasta cualquier número de niveles.

Modelo relacional

Los datos y las relaciones entre los datos se representan por medio de una serie de tablas, las cuales tienen varias columnas con nombres únicos.

Proveedor

Clave	Nombre	Ciudad
S1	Salazar	Londres
S2	Jaramillo	Paris
S3	Bernal	Paris

Parte

Clave	Nombre	Descripción	Ciudad
P1	Tuerca	Rojo	Londres
P2	Perno	Verde	Paris
P3	Tornillo	Azul	Roma

Venta

Proveedor	Parte	Cantidad
S1	P1	300
S1	P3	400
S2	P2	300
S2	P3	400
S3	P2	200

Para el modelo relacional los datos se guardan en tablas en las cuales se tiene un registro por cada elemento del problema (para proveedores y partes) y sus asociación se denota en este caso por ser una asociación de muchos a muchos en otra tabla.
Éste modelo es el que será motivo de nuestro estudio en los posteriores capítulos.

GENERACIONES DE LENGUAJES

Los lenguajes al igual que la tecnología de cómputo se ha ido desarrollando, tenemos aquí ejemplos de los distintos lenguajes de acuerdo a su generación.

PRIMERA GENERACIÓN

0110 1000 1111 0011

En esta generación los datos eran proporcionados a las máquinas a través de cables y proporcionando voltaje o no (un cero o un uno).

SEGUNDA GENERACIÓN

```
LOAD A,10
MOV A;B
SUM A,B;C
```

Estas son las instrucciones de un lenguaje "ensamblador" las cuales son llamadas *mnemónicos* o *directivas* que realizan operaciones directas sobre el microprocesador de las máquinas.

TERCERA GENERACIÓN

```
PROGRAM ALUMNOS
        NOMBRE CHAR(30)
        OPEN (1,'ALUMNOS')
        DO WHILE(.TRUE.)
            READ(1,10,END=100)NOMBRE
10 FORMAT(A,30)
            WRITE(*,*)NOMBRE
        ENDDO
100 CONTINUE
END
```

Para la tercera generación surgen los llamados lenguajes de alto nivel y los lenguajes estructurados, tal es el caso de Fortran, Basic, Pascal y C.

CUARTA GENERACIÓN

```
SQL> SELECT * FROM ALUMNOS;
```

Los lenguajes de cuarta generación son aquellos lenguajes que pretenden dar información a la máquina a través de instrucciones lo más cercanas a un lenguaje natural (es decir nuestro lenguaje).

MANEJADOR DE BASES DE DATOS VS. MANEJADOR DE ARCHIVOS.

Cuando el uso de la información se hizo más y más importante, la forma de manejarla tuvo que ser más complicada pero su complejidad ha tenido que ser eficiente, por lo que surgieron dos manejadores de información de los cuales a continuación hacemos una comparación:

MANEJADOR DE BASES DE DATOS.

VENTAJAS

Redundancia controlada.
Seguridad a nivel de campos.
Integridad.
Independencia Datos-Programas.
Recuperación de información.
Control de concurrencia.
Respaldos en línea.

DESVENTAJAS

Alto Costo
Bajo Rendimiento

MANEJADOR DE ARCHIVOS.

DESVENTAJAS
Redundancia.
Incongruencia.
Programas dependientes de los datos.
Seguridad sólo a nivel de archivo.

VENTAJAS
Menor costo
Alto rendimiento (Alta velocidad)

El enfoque relacional

- Un sistema de información de bases de datos relacional se organiza en forma de tablas.
- Las tablas se organizan en renglones y columnas.
- Cada renglón se denomina registro y es información referente a una instancia.
- Cada columna se denomina campo y es información de un solo tipo para todas las instancias.
- De esta forma tu puedes mediante la observación de las tablas entender y utilizar la información que te proporcionan.

Características de un RDBMS. (Relational Database Managment System)

- Representación de los datos por medio de tablas relacionadas.
- Utiliza lenguaje de cuarta generación.
- Flexibilidad:
  la modificación de los datos y
  los cambios a la estructura de la Base de Datos resulta muy sencillo.
- Contiene un Diccionario de Datos.

SOFTWARE DE ANALISIS Y DISEÑO DE SISTEMAS

Dentro de las técnicas para el análisis y diseño de sistemas, existen en el ámbito mundial paquetes de ayuda para el análisis y diseño de sistemas de información, dentro de los cuales destacan DESIGN AID, EASY CASE, ER/WIN, KASEVIP (KASEWORP),System Architect for Windows, The Visible Analyst Workbench V.5.2 , ADW/IEW, etc

Este tipo de software es conocido en forma genérica como herramientas CASE, y se divide en dos partes principales:

Upper Case, el cual utiliza principalmente la programación.
Lower Case, que es donde se lleva a cabo el diseño, estructuración y elaboración de manuales.

Cada CASE genera código sólo para algunas herramientas, por lo tanto, existe un CASE para cada equipo y software con el que se trabaje, por lo que en ocasiones representa un costo elevado para el diseño.

Componentes principales de éstas herramientas:

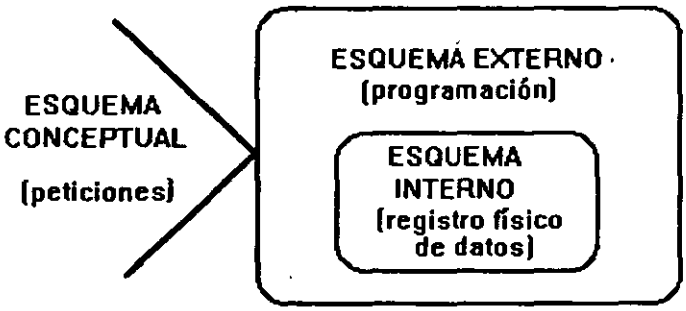
a) Grafitexto: grafica diversos tipos de diagramas.Gran ayuda para el profesional en el desarrollo de sistemas, en lo concerniente a Documentación, Modelado de Sistemas, Programación, Gráficas de Diseño, y además permite interpolar con Procesadores de Palabras.

b) Diccionario de Diseño: combina las ventajas de un Diccionario de Datos estándar con la habilidad de almacenar información sobre objetos de diseño tales como los Procesos, Subsistemas, Flujo de Datos y externos.

c) Analizador de Diseño: permite analizar tanto los Diagramas de Flujo de Datos como los programas escritos en un leguaje de programación, para su complemento y precisión. Además, algunos de ellos ofrecen la ventaja de la actualización automática del Diccionario de Diseño.

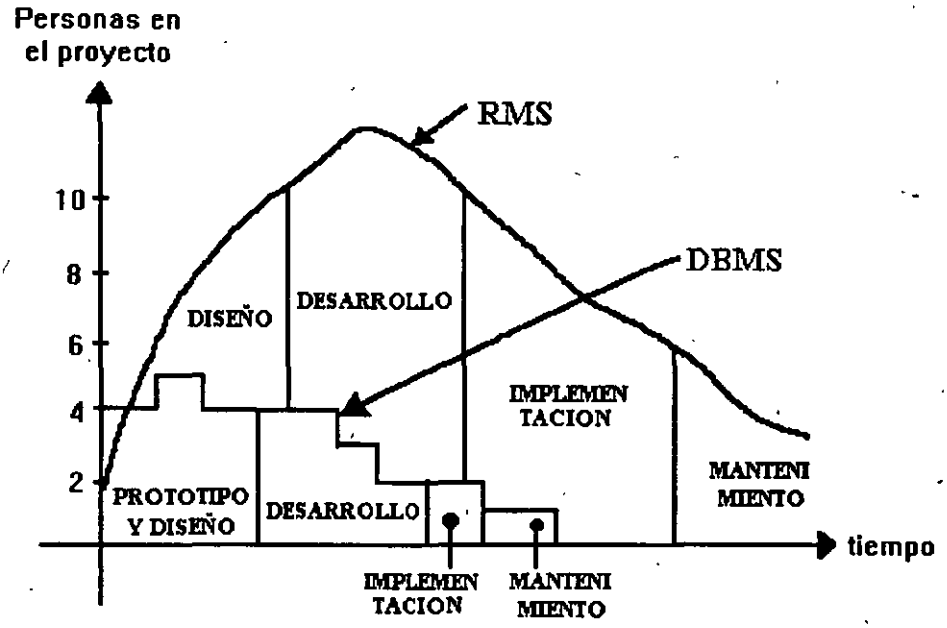
ARQUITECTURA DE TRES NIVELES

La forma de desarrollar un sistema de bases de datos parte de la forma en que cada participante del proyecto debe visualizar el problema y de lo que se supone debe hacer, así podemos ver un proyecto a través de su arquitectura:



El siguiente comportamiento no es otra cosa que una comparación entre el desarrollo de un sistema a través del método tradicional y a través del modelo relacional.

COMPORTAMIENTO DE PROYECTOS



Como podemos observar son claras las ventajas que el modelo relacional ofrece sobre métodos tradicionales, y donde se ve más clara la diferencia es principalmente en la etapa de mantenimiento en donde observamos que se reduce grandemente, lo que nos garantiza que un modelo relacional correctamente construido no proporcionará una vida más útil y duradera sin necesidad de hacerle constantes modificaciones.

Revisión de objetivos tema I

1.- Diga qué es una base de datos.
2.- Mencione los tipos de modelos de bases de datos que existen.
3.- Proporcione las ventajas de un manejador de bases de datos.
4.- Qué tareas debe realizar un Sistema Manejador de Bases de Datos Relacional (RDBMS).

Tema II.-MODELADO

El modelado de datos entidad-relación se basa en una percepción del mundo real que consiste en un conjunto de objetos básicos llamados entidades y relaciones entre estos objetos. Se desarrolló para facilitar el diseño de bases de datos permitiendo especificar un esquema empresarial. Este esquema representa la estructura lógica de los datos.

El método entidad relación es una metodología estructurada que puede sistemáticamente convertir los requerimientos del usuario en una base de datos bien diseñada.

Objetivo:

Al término del tema el alumno será capaz de:

- Listar las características del modelo de datos relacional.
- Analizar a través del enfoque relacional un problema que se le plantee.
- Dar solución al problema en base a las herramientas que proporciona el modelo entidad-relación.

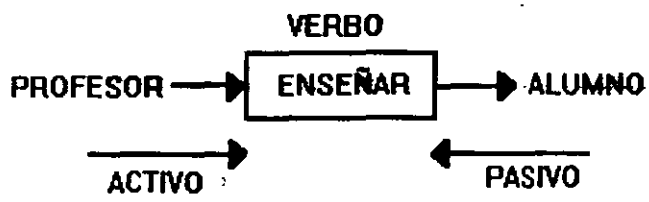
Definiciones:

Situación: Es un conjunto de circunstancias bien definidas que pueden ser descritas utilizando un lenguaje natural suficientemente completo.

Un lenguaje natural suficientemente completo incluye al menos los siguientes tres constructores gramaticales:

1. Sujetos. Es el nombre de una persona, de un animal, una planta, un lugar, una cosa, una sustancia o una idea. Un **sujeto propio** es el nombre de una ocurrencia particular o **instancia** de un sujeto. Un **pronombre** es una palabra utilizada como sustituto de un sujeto que hace referencia a un sujeto que ya ha sido nombrado o que se sobreentiende de acuerdo al contexto en el cual se utiliza. Los sujetos pueden ser tangibles o intangibles y es imposible describir alguna **situación** sin el uso al menos de un sujeto.

2. Verbos. Es una palabra que describe un modo de ser, una asociación, una acción o un evento. Los verbos describen el estado de los sujetos, y relacionan los sujetos dentro de una situación. Los verbos pueden ser activos o pasivos.

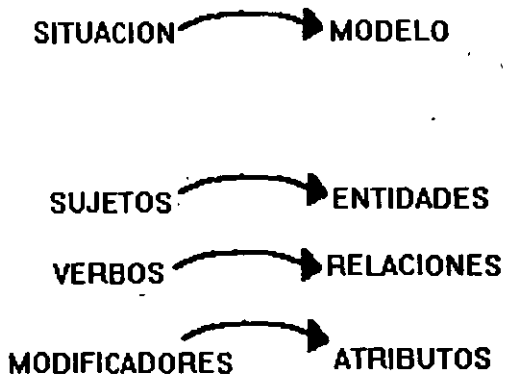


Esquema que nos muestra las formas en que puede actuar un verbo dentro de una situación descrita.

3. Modificadores. Es una palabra que califica a un sujeto o a un verbo de acuerdo a sus características, cantidad, extensión, etc. Los modificadores de los **sujetos** se llaman **adjetivos** y los modificadores de los **verbos** se llaman **adverbios**.

Las situaciones que se describen sin el uso de modificadores son demasiado triviales para ser de interés, sin embargo existen.

MODELADO



El esquema anterior nos muestra la relación que existe entre cada una de las partes que componen una situación y los elementos del modelo de datos relacional.

MODELO: Representación de una situación mediante la abstracción de un hecho real.

Modelo de datos Entidad-Relación

Se basa en una percepción del mundo real consistente en un conjunto de objetos básicos llamados entidades y de relaciones entre estos objetos.

MODELO DE DATOS RELACIONAL

Conexión finita de tablas (de dos dimensiones formadas por columnas y renglones) que representan una situación.

DEPARTAMENTO

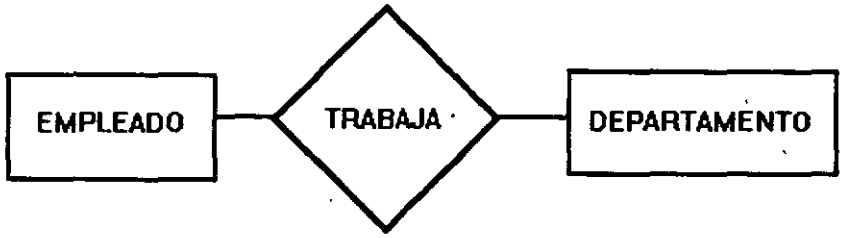
CLAVE	NOMBRE
VE	VENTAS
CO	COBRANZAS
IN	INVESTIGACION
ME	MERCADOTECNIA
ES	ESTADISTICA

EMPLEADO

NUMERO	NOMBRE	SALARIO	DEPTO	PUESTO
1	JUAN	2000	VE	VENDEDOR
2	RICARDO	3000	ES	PROGRAMADOR
3	ROSA	2700	IN	INVESTIGADORA
4	LOURDES	2000	VE	VENDEDOR

A cada sujeto le corresponde una tabla cuyas columnas son los adjetivos o características de ese sujeto.

Modelo Entidad-Relación



Representación del modelo relacional a través del modelo entidad-relación. Nótese la forma en la que cada tabla es representada por una entidad y las relaciones unen a éstas entidades.

CARACTERÍSTICAS DEL MODELO RELACIONAL

SIMPLICIDAD. Las tablas son de una forma familiar y explicables por sí mismas para representar los datos. La mayoría de la gente ha utilizado datos en forma de tabla, no se requiere un entrenamiento especial para entender y utilizar los datos que se representan en las tablas. En pocas palabras son amigables a los usuarios.

PRECISIÓN. Las tablas correctamente diseñadas mantienen un rigor matemático, dicen lo que significan y significan lo que dicen, pueden ser implementadas y procesadas por una variedad de configuraciones de software y hardware. En pocas palabras son amigables a la computadora.

FLEXIBILIDAD. Las tablas no solo muestran la estructura de los datos sino que pueden mostrar los datos también. Esto nos permite manejar el modelo antes de implementarlo. En otras palabras las tablas son apropiadas no solo para modelar datos sino para procesarlos también.

CARACTERÍSTICAS DE LAS TABLAS

El modelo de datos entidad-relación nos lleva a obtener un modelo de datos relacional que no es otra cosa que una colección finita de tablas de dos dimensiones formadas por columnas y renglones que representan una situación.

Llave primaria. Columna o grupo de columnas que identifica de manera única a cada renglón de la tabla. Símbolo: **PK**.
Cumple con las siguientes características:
NN : No debe aceptar valores nulos.
ND : No pueden existir 2 renglones con el mismo valor de llave primaria.
NC : No puede cambiarse este valor.
PK asignada por el usuario. (PK UA) De esta forma el usuario proporciona el valor para la llave primaria del registro insertado.
Ejemplo: RFC de cada trabajador.
PK asignada por el sistema. (PK SA) De esta forma el sistema proporciona automáticamente el valor de la llave primaria cuando el usuario inserta un registro.
Ejemplo: Número de factura.

Todas las tablas deben tener sólo una llave primaria, la cual puede estar compuesta de uno o más campos.

La llave primaria no es la única forma de tener acceso a los datos, se puede accesar a través de cualquier columna.

Llave foránea. Es una columna o grupo de columnas que es llave primaria en alguna otra tabla.

Nulo. Un nulo significa ausencia de dato.

DATO DERIVADO. (DD)
Se puede obtener de otros datos, ES REDUNDANTE, pero si se utiliza muy seguido esa información es recomendable utilizarlo para incrementar velocidad.

Relaciones de tablas.

Las relaciones de las tablas se logran a través de algunos de sus campos (columnas), los cuales reciben nombres especiales.

MODELADO DE ENTIDADES.

1. Descubrir entidades.
   Concentrarse en SUJETOS.
   Tener cuidado en no incluir atributos.

2. Definir el alcance de la entidad.
   Definir si ese entidad es de interés para el sistema.

3. Definir una llave primaria
   Verificar (No Nulos, No Duplicados, No Cambios), si hay varias que cumplen con lo anterior buscar la de más fácil manejo.
   Llaves primarias naturales al sistema (modelo).

4. Documentar.

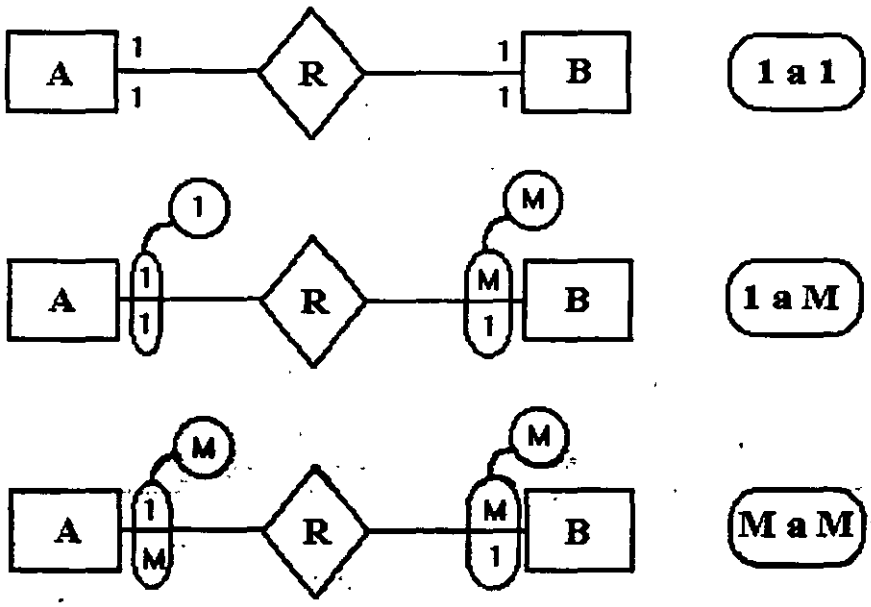
5. Incluir en el diagrama Entidad-Relación. (Con un rectángulo).

MODELADO DE RELACIONES.

1. Descubrir relaciones.
   Concentrarse en VERBOS.

2. Definir el alcance de la relación.
   Definir si la relación es de interés para el sistema.

3. Definir el tipo de relación.
(Verificar como se relacionan las entidades entre sí).



Los diagramas anteriores nos muestran los tipo de relaciones que podemos tener en un modelo de datos relacional, éstos tipos de relaciones se distinguen por su cardinalidad, es decir por el número de elementos de una entidad que están relacionados con otra entidad.
Así, tenemos relaciones uno a uno (1 a 1), relaciones uno a muchos (1 a M), o bien, muchos a muchos (M a M).

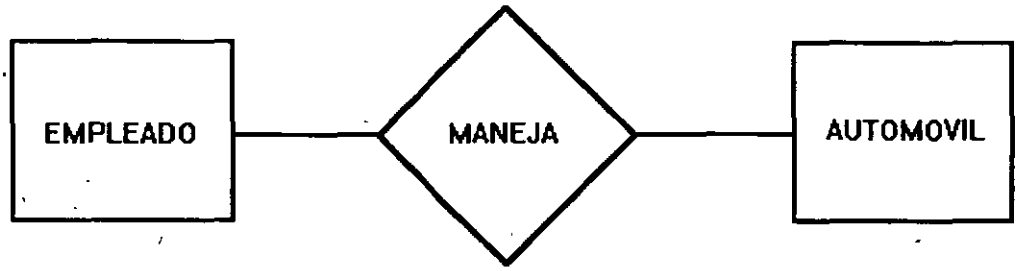
4. Documentarlas en el diagrama Entidad-Relación.

5. Documentarlo en tablas

Las relaciones sólo toman forma hasta el momento en que definimos al tipo al que corresponden e introducimos el concepto de migración de llaves (llave foránea) entre las entidades o tablas del modelo.

A continuación se describe lo que se debe hacer para cada tipo de relación que se nos presente en el momento de hacer el modelado de relaciones:

- Relación 1:1



EMPLEADO

NUM_EMP	NOMBRE

AUTOMOVIL

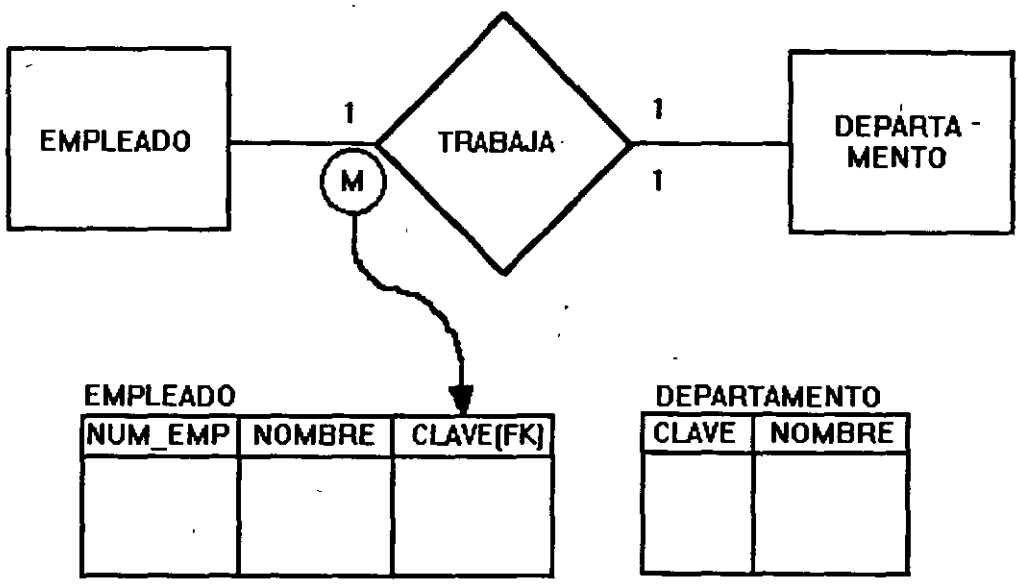
PLACAS	COLOR	NUM_EMP(FK)

Se puede colocar la llave foránea en cualquiera de las dos tablas pero siempre hay una tabla en donde es mejor. En este caso, se colocaría en la tabla de automóvil para tener los datos concentrados porque en la tabla de empleado habría varios nulos en caso de que hubiera empleados sin automóvil sin embargo un automóvil siempre estará relacionado con un empleado.

El espacio de almacenamiento es el mismo porque los nulos no ocupan espacio.

- Relación 1-M y M-1

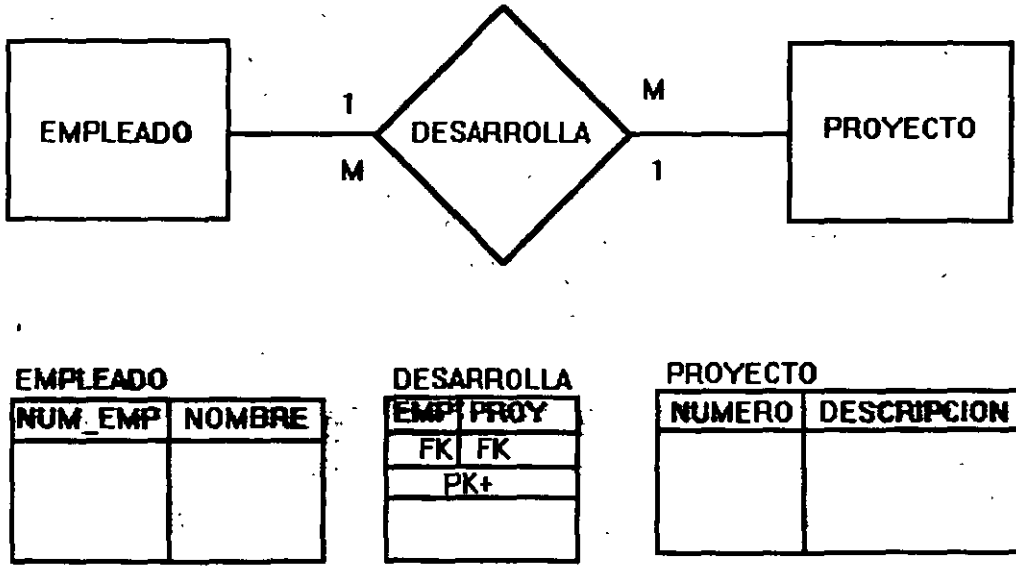
Este es un tipo de relación no simétrica, sólo puedo colocar la llave foránea FK en la tabla en donde la relación es M.



Como se puede ver, la llave foránea pasa a ser una columna más de la tabla a la cual está migrando.

Relación M-M

En este tipo de relación se concatenan las llaves foráneas y se crea una nueva tabla, esta tabla recibe el nombre de tabla COMPUESTA.



No importa el orden de las FK, de ahí se observa la simetría de la relación Muchos a Muchos.
A la tabla DESARROLLA se le conoce como TABLA COMPUESTA.

MODELADO DE ATRIBUTOS.

1. Describir atributos
   Concentrarse en adjetivos o adverbios.

2. Definir el alcance del atributo
   Decidir si el atributo es de interés al sistema.

3. Determinar una llave primaria para el atributo
   Determinar cual es la mejor opción para colocar el atributo en una tabla (eliminar duplicidad colocando los atributos de manera correcta.

4. Documentar el atributo en alguna tabla

MÉTODOS PARA COLOCAR ATRIBUTOS.

1. Método gramatical (por experiencia)
2. Normalización

La normalización es el método más común a través del cual podemos verificar si nuestro modelo es funcional y está listo para ser implementado. Está basado en una serie de reglas llamadas normas normales, se conocen muchísimas de estas reglas, sin embargo nosotros sólo nos basaremos en las tres principales.

NORMALIZACIÓN

Primera forma normal. (Dependencia entre la PK y un atributo)

Dada una tabla **T**, con una llave primaria **P** y un atributo **A**, se dice que **T** está en primera forma normal, si y sólo si el valor del atributo **A** en cualquier renglón depende del valor de la llave primaria **P** en ese renglón.

T

P	A
1	4
2	4
3	4
4	4

Si todos se repiten **No** está en primera forma Normal

Puede repetirse pero no todos iguales

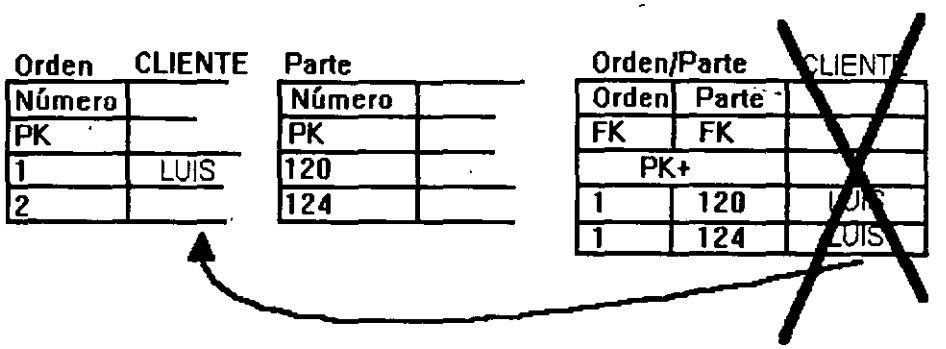
T_2

P	A
1	4
2	4
3	4
4	15

Si se tiene una tabla en la cual el valor de una columna (atributo) es una constante para *todos* los registros renglones) , entonces, ese atributo no debe considerarse para el modelo.

Segunda forma normal. (Dependencia entre un atributo y una FK en una tabla compuesta)

Dada una tabla T en primera forma normal, con una llave primaria P en varias columnas con componentes P1 y P2 y un atributo A se dice que T está en segunda forma normal, si y sólo si el valor del atributo A en cualquier renglón depende de los dos valores P1 y P2 en ese renglón, las tablas con llaves primarias en una sola columna siempre están en segunda forma normal.



Se aplica sólo en tablas compuestas.
Consideremos el ejemplo: Se tiene una entidad Orden la cual se refiere indudablemente a una orden de compra, en la cual se pueden incluir varias partes, es decir, un cliente lleva varios artículos en una compra, es necesario guardar el nombre del cliente por lo que se había considerado en un principio colocar la columna con el nombre del cliente en la tabla compuesta Orden/Parte, en la cual vemos claramente que existe redundancia de información, ya que por cada artículo que el cliente está comprando guardamos su nombre, además, vemos también que el nombre del cliente depende sólo del número de orden y como se trata de una tabla compuesta debería depender de las columnas que forman la llave primaria, por lo que es más apropiado colocar esa columna en la tabla orden, la cual sólo se guarda una vez, por cada compra que el cliente haga sin importar todos los artículos que lleve.

Tercera forma normal.

Dada una tabla T en segunda forma normal, con una llave primaria P y dos atributos A1 y A2, se dice que T está en tercera forma normal, si y solo si el valor del atributo A1 en cualquier renglón no depende del valor del atributo A2, a menos que A2 esté marcado como ND, y el valor del atributo A2 no dependa del valor del atributo A1 en ningún renglón, a menos que A1 esté marcado como ND.

Empleado

Número	Puesto	Salario
PK		
1	Secretaria	1000
2	Secretaria	1000
3	Vendedor	1400
4	Vendedor	1400

No está en tercera forma normal

Empleado

Número	Puesto
PK	
1	Secretaria
2	Secretaria
3	Vendedor
4	Vendedor

Puesto

Puesto	Salario
PK	
Secretaria	1000
Vendedor	1400

Está en tercera forma normal

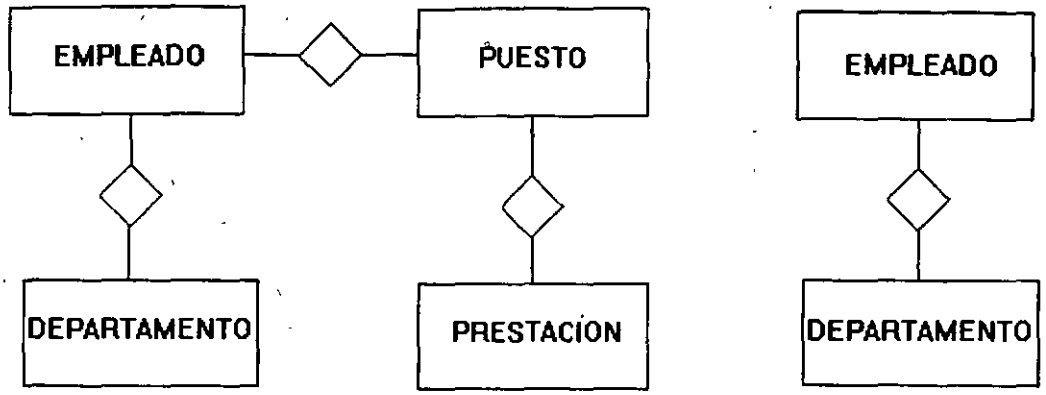
Observemos el primer modelo: en él vemos que existe una dependencia directa entre el atributo "puesto" y el de salario, lo cual no debe de ser, según la tercera forma normal, por lo cual se debe generar otra tabla , en este caso la tabla PUESTO en la que el campo salario debe depender sólo de la llave primaria y en éste caso la llave primaria es el campo "puesto".

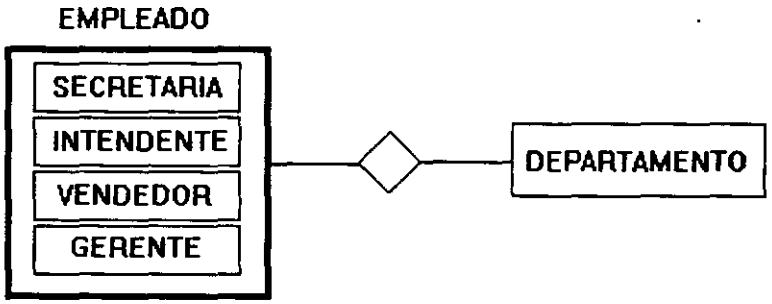
Al terminar nuestro modelo en tercera forma normal encontramos algunos datos duplicados por lo que a continuación se realiza un proceso de desnormalización. Éste proceso de desnormalización es un proceso que la experiencia en la creación de bases de datos no dará, ya que se realiza para facilitarnos el manejo de los datos al momento de generar consultas y reportes.

Superentidades

En el modelado podemos recurrir a las Superentidades, en el siguiente diagrama tenemos un modelo entidad-relación (figura del lado izquierdo), el cual describe lo siguiente: Un empleado labora en un departamento, cada empleado tiene un puesto y para cada puesto existen diferentes prestaciones.

El modelo visto desde el punto de vista de Superentidades podría reducirse como en el siguiente diagrama (figura del lado derecho)





La forma óptima de realizar el modelado depende de los datos que yo quiero obtener de la base de datos (reportes).

Ejercicio: **MODELADO DE SUPERENTIDADES**
El usuario dice:

Es mi responsabilidad dar seguimiento al inventario de la organización, se utiliza el término **equipo** de manera general, y se asigna un número de equipo único para cada compra que se realiza.

También existen códigos para clasificar equipo, por ejemplo **PC** para computadoras personales, **MP** para impresoras de matriz de puntos, **IL** para impresoras láser, **MD** para Modems.

Creo que es todo el equipo que existe.

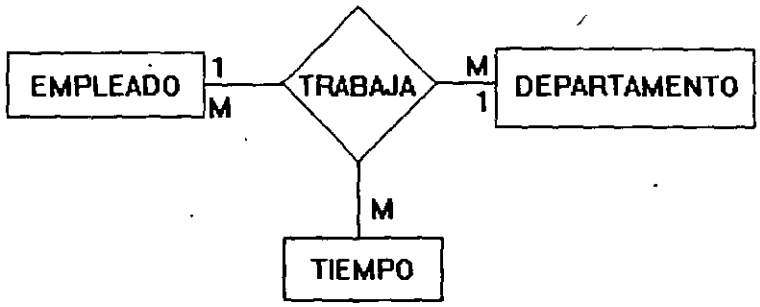
Tenemos características diferentes para cada tipo de equipo, por ejemplo, para las PC me interesa su procesador, su monitor y su capacidad de memoria, para los módem me interesa su velocidad de transmisión, para las impresoras, su resolución y el número de páginas por minuto. Además para cada equipo se requiere conocer el fabricante, por eso de las reparaciones.

Inclusión del tiempo

La variable Tiempo es muy importante en el modelado de Bases de Datos, en casi todos los modelos de magnitud media se incluye esta variable. La implementación de ésta por lo general es por medio de la utilización de fechas.

Ejemplo:

Quiero saber en cuánto tiempo ha trabajado un empleado en cada departamento durante su estancia en la empresa.



EMPLEADO
NUM_EMP
PK

TRABAJA		
NUM_EMP	NUM_DEPTO	FECHA_INICIO
FK	FK	FK
PK+		

DEPARTAMENTO
NUM_DEPTO
PK

Las referencias hacia el pasado o el futuro las obtengo mediante una entidad de TIEMPO, esta entidad no genera tabla, solo un atributo en otra tabla de relación, y se considera llave foránea, teniendo como dominio un período de tiempo. La llave primaria de esta tabla son tres columnas: NUM_EMP, NUM_DEPTO Y FECHA_INICIO.

Revisión de objetivos tema II

1.- Enumere las características del modelo relacional.

2.- Diga los elementos en los que nos debemos concentrar para hacer el modelado de un situación.

3.- Mencione los tipos de relaciones que pueden existir y en qué afectan al modelo relacional.

4.- Diga para qué se realiza la normalización del modelo.

5.- Realice el siguiente modelo:

Manejar tarjetas de crédito.

- Nacionales e Internacionales.
- Fecha de corte en cada tarjeta.
- Reporte de transacciones: Fecha de Operación, Costo, Fecha, Lugar.
- Un cliente puede tener varias tarjetas.
- Un reporte por tarjeta.
- Datos del cliente para mandarle cada mes por correo su estado de cuenta.
- Tarjetas adicionales.
- Tarjetas empresariales.

- Generar estados de cuenta.
- Hay varias sucursales.
- Cada número de tarjeta lo genera el sistema.

SUJETOS:	VERBOS:
CLIENTE	**TIENE**
CUENTA	**DEPENDE**
TARJETA	**REALIZA**
TRANSACCIÓN	**EXPIDE**
SUCURSAL	**LOCALIZA**
ESTADO	**PERTENECE**
PAÍS	**SER**
TIPO	

Tema III.- ÁLGEBRA RELACIONAL

El álgebra relacional fue definida por el Dr. Codd basada en el manejo del álgebra de conjuntos y que nos permite manejar nuestras tablas generadas a partir del modelo entidad-relación y manipular nuestra información de una manera sencilla, tal y como manejamos los elementos de un conjunto.

Objetivo:

El alumno una vez concluido el tema conocerá:

- Las operaciones en las que se basa el álgebra relacional.
- Realizar operaciones algebraicas con el modelo relacional, de manera que pruebe su consistencia antes de ser implementado.

Dados 2 conjuntos A y B, una relación R sobre A y B se define como:

$A\ R\ B=\{(a_1,b_1),(a_2,b_2)....\}$

donde $a_1,a_2...$e A y b_1,b_2 e B

A = { *,#,%,& }
B = { @,!,$,# }

$A \cup B$ = { *,#,%,&,@,!,$ } **Unión**

$A \cap B$ = { # } **Intersección**

Cardinalidad: Número de elementos de conjunto, por ejemplo :
A = { @,#,% } cardinalidad 3
B = { (a,b), (d,c) } cardinalidad 2
Las operaciones anteriores modifican la cardinalidad de los conjuntos.

Tomando en cuenta los conjuntos definidos al inicio
A x B = { (*,@),(*,!),(*,$),(*,&),
(#,@),(#,!),(#,$),(#,#),
....} **Producto Cartesiano**

En ésta operación se modifica el orden del conjunto resultante.
Orden: Número de elementos que componen un elemento atómico de un conjunto.
Como se puede apreciar los elementos atómicos del conjunto resultante es (*,@) que constituyen la unión de los elementos atómicos de los conjuntos que intervienen en el producto cartesiano.

Para el manejo de la información de nuestras tablas nos auxiliaremos de las operaciones anteriores.

Proyección (π)

Considerando una tabla llamada Empleado con los siguientes registros :

Empleado

Número	Nombre	Salario	No_depto
1	Juan	300	10
2	Luis	400	20

entonces:

π $Empleado_{Número}$
Resultado:

1
2

Es decir, la selección nos proporciona la información del campo que le indicamos y de la tabla indicada a continuación del símbolo de proyección.

Selección (σ)

Tenemos la tabla Departamento con la siguiente información:

Departamento

Clave	Nombre
10	Ventas
20	Contabilidad

entonces:

σ $Departamento_{Clave}$=20
Resultado:

20 Contabilidad

La selección nos trae la información de los registro que cumplan con la condición fijada en la instrucción de selección.

Join (θ)

El join equivale al producto cartesiano entre 2 tablas:

Tomando en cuenta las tablas de Empleado y Departamento ya mencionadas tenemos que:

Empleado θ Departamento =	Número	Nombre	Salario	Clave	Nombre
	1	Juan	300	10	Ventas
	1	Juan	300	20	Contabilidad
	2	Luis	400	10	Ventas
	2	Luis	400	20	Contabilidad

Si vemos la información que obtenemos de esto, es una información que no nos sirve mucho, tan solo es una combinación de todos los registros de la primer tabla con todos los registros de la segunda.

Por ello se introduce una condición a través de la cual se puedan relacionar.

Planteando nuevamente nuestras tablas considerando ya el concepto de llave primaria y foránea que ya mencionamos anteriormente:

Tenemos entonces:

Empleado θ Departamento =	Número	Nombre	Salario	No_depto	Clave	Nombre
No_depto=Clave	1	Juan	300	10	10	Ventas
	2	Luis	400	20	20	Contabilidad

Para la manipulación de nuestras tablas podemos hacer combinaciones con las operaciones mencionadas.

Revisión de objetivos tema III

- Enumere las operaciones en las que se basa el álgebra relacional.
- Mencione la diferencia entre el producto cartesiano y el join.

En base a las siguientes tablas hacer las operaciones que se piden:

Equipo

Clave	Nombre	Ciudad
1	Pumas	1
2	Chivas	2
3	Monterrey	3
4	UAG	2
5	América	1

Ciudad

Clave	Nombre
1	México
2	Guadalajara
3	Monterrey

Jugador

Clave	Nombre	Equipo	Fecha_nac	salario
1	J.Campos	1	03/jul/70	5000
2	Zague	5	04/may/65	4000
3	De la Torre	2	03/ene/63	3500
4	Garcia	5	10/feb/64	3000
5	Tibu	1	11/mar/63	3000

Jugador/Posición

Jugador	Posición
1	1
1	4
2	4
3	3
4	1
5	2

Posición

Clave	Nombre
1	Portero
2	Defensa
3	Medio
4	Delantero

Partido

Local	Visita	Goles_local	Goles_visita
1	2	3	1
2	4	1	0
5	1	1	4
3	2	2	3

a) Listar los nombres de los equipos.
b) Nombre de las ciudades con equipos registrados.
c) En qué ciudad juega cada equipo.
d) Dónde vive cada jugador.
e) Quiénes son porteros.
f) Listar a los jugadores del equipo américa.
g) En qué partidos los pumas jugando como local ganaron por diferencia de 2 goles.
h) Listar los resultados de los partidos.

TEMA IV. SQL

Consultas a la Base de Datos

SQL es un lenguaje de cuarta generación como ya se mencionó y se ha convertido en el lenguaje estándar para los manejadores de bases de datos relacionales, es por ello que ahora nos introduciremos en él.

Objetivo:

- AL final del tema el alumno será capaz de:
- Mencionar la estructura general de una consulta en SQL.
- Hacer consultas a la base de datos a través de las instrucciones de SQL.
- Estructurar una consulta que cumpla con múltiples condiciones.
- Mencionar el uso de los alias.
- Ordenar los resultados de las consultas.

Como iniciar una sesión SQL*PLUS.
Existen 2 formas:

Pasos para iniciar una sesión SQL*PLUS: (primera forma).

1. Entrar a Sistema Operativo con los correspondientes USERNAME y PASSWORD.
2. En el prompt del sistema operativo:
$ sqlplus [ENTER]
username: *username* [ENTER]
password:********* [ENTER] (el password no se muestra)

Pasos para iniciar una sesión SQL*PLUS: (segunda forma).
1. Entrar a Sistema Operativo con los correspondientes USERNAME y PASSWORD.
2. En el prompt del sistema operativo:
$ sqlplus *username/password*

Si existe algún problema o cualquier tipo de conflicto consulta al asesor en turno o al administrador de la base de datos.

Cómo obtener ayuda.

Para listar los comandos de SQL y SQL*PLUS.
SQL> help

Información de un comando en particular.
SQL> help select
SQL> help grant

Para ver una lista de tópicos.
SQL> help topics

El Diccionario de datos.

El Diccionario de datos es un grupo de tablas y vistas que contienen información descriptiva acerca de:

- Tablas.
- Privilegios de usuarios.
- Otras características de la Base de Datos.

A continuación se listan las tablas del diccionario de datos que se consultan con más frecuencia:

- TAB. Contiene una lista de las tablas, vistas y sinónimos que tú has creado.
- COL. Contiene una lista de la definición de columnas para las tablas que tú creaste.
- CAT. Contiene una lista de las tablas a las cuales tienes acceso.
- USER_TABLES. Información detallada de las tablas creadas por un usuario.

El comando DESCRIBE.

Se utiliza para obtener información acerca de la estructura de una tabla.
sintaxis:

```
SQL> describe
```

La información que se muestra es la siguiente:

Name. Nombre de la columna.
NULL? Esta columna contiene **NOT NULL** cuando el campo en la columna **NAME** no debe contener un dato NULO.
Type. Tipo de dato que contiene el campo.

Sintaxis general de una consulta

- Utilizando SQL puedes consultar la Base de Datos en un número ilimitado de formas.
- SQL es muy flexible y tiene una sintaxis muy específica.
- La orden de todas las cláusulas en el comando SELECT se muestra a continuación:

```
SELECT...
FROM...
[WHERE...]
GROUP BY...
[HAVING...]
[ORDER BY...]
```

Seleccionando todas las columnas.

```
SQL> select *
   from emp;
```

Se utiliza * para ver todas las columnas de una tabla.

Seleccionando columnas específicas.

```
SQL> select ename
     from emp;
```

Seleccionando un nombre de columna regresa los datos sólo de esa columna
En la consulta anterior nos proporcionará todos los datos contenidos en la columna **ename** de la tabla **emp**.

Seleccionando múltiples columnas.

```
SQL> select empno, ename, job
     from emp;
```

Se coloca una "*coma*" *(,)* entre cada nombre de columna.
Aparecen todos los registros para cada columna seleccionada.

Orden en la selección de columnas.

```
SQL> select empno, ename, job
     from emp;

SQL> select job,ename,empno
     from emp;
```

El orden de los resultados es el especificado en el comando select.

Selección de registros (Cláusula where)

```
SQL> select ename
     from emp;
     where deptno = 10;
```

Se obtienen los registros que cumplen con la condición especificada.

Ordenando registros

- En el modelo relacional los registros no tienen un orden en particular.
- El comando ORDER BY es la única forma de asegurar que los registros sean presentados de acuerdo a un criterio específico.

```
SQL> select ename
     from emp;

SQL> select ename
     from emp
     order by ename;
```

El primer caso no tiene ordenamiento; el segundo ordena por nombre de empleado.

Ordenando por múltiples criterios.

Cuando se ordena por múltiples criterios:

- El orden Primario es la primera columna que se lista
- El orden Secundario es la segunda columna que se lista
- y así sucesivamente.

- El default es la secuencia Ascendente (ASC) (De la A a la Z).
- Para ordenar en forma Descendente, se añade la palabra DESC después de que se especifica la columna por la cual ordenar.

```
SQL> select *
     from dept
     order by deptno desc;
```

Formato de las condiciones.

En la cláusula where se pueden comparar columnas con:
- una constante de carácter usando apóstrofos.

```
where ename = 'SMITH'
```

- Una expresión aritmética se expresa sin comillas.

```
where deptno = 20
```

- Otra columna

```
where emp.deptno = dept.deptno
```

(Esta construcción se conoce como JOIN)

Operadores

Operadores de igualdad y desigualdad.

Igual a	=
diferente a	!= ó <>
mayor que	>
mayor o igual a	>=
menor que	<
menor o igual a	<=

Otros operadores.

Igual a los miembros de la lista	**in(list)**
Mayor o igual a un valor, y menor o igual a otro	**between** low **and** high

Relaciona los siguientes patrones:	**like**
una cadena de cero ó más caracteres	**%**
una cadena de un carácter	**_**
Relaciona con NULOS	**is null**

Niega algunos de los comandos anteriores **not**
(ej.: not in, is not null)

Lista de valores: in and not in

- El operador **in** permite seleccionar registros que coincidan con uno de los valores en una lista (utiliza una evaluación OR).

¿Cuáles empleados son 'CLERK' o 'ANALYST'?

```
SQL> select ename, job
     from emp
     where job in ('CLERK','ANALYST');
```

- Con **not in** se seleccionan los registros que no concuerdan con la condición.

```
SQL> select ename, job
     from emp
     where job not in('CLERK','ANALYST');
```

Un rango de valores: between y not between.

- El operador **between** permite seleccionar registros que coincidan con un rango de valores.

¿Qué empleados ganan entre $2,000 y $3,000?

```
SQL> select ename, job, sal
     from emp
     where sal between 2000 and 3000;
```

- Con **not between** se seleccionan registros que no están dentro del rango.

```
SQL> select ename, job, sal
     from emp
     where sal not between 2000 and 3000;
```

Selección por medio de patrones: like y not like.

- Para hacer coincidir una cadena de caracteres se utiliza el operador **like** en la cláusula **where**.

Listar los empleados cuyo nombre inicia con 'S'.

```
SQL> select ename, deptno
     from emp
     where ename like 'S%';
```

Listar todos los empleados cuyo nombre termine con 'K'.

```
SQL> select ename, deptno
     from emp
     where ename like '%K';
```

Listar todos los empleados cuyos nombres inician con 'W' seguida por exactamente 3 caracteres.

```
SQL> select ename, deptno
     from emp
     where ename like 'W___';
```

Nota: el carácter '%' sustituye varios caracteres, el carácter '_' sustituye solo un carácter.
*Al utilizar **not like** se seleccionan todos menos los que coincidan con el patrón.*

Listar todos los empleados cuyo trabajo no inicie con la cadena 'SALES'.

```
SQL> select ename, job
     from emp
     where job not like 'SALES%';
```

Valores nulos: is null e is not null.

- Un valor **NULL** en un campo numérico no es lo mismo que un cero.
- El cero es un número, **NULL** no es un número.
- **NULL** significa ausencia de dato, no podría ser tratado como un cero.

Buscando valores nulos.

-Listar todos los empleados que no tienen comisión (columna de comisión con **NULL**)

```
SQL> select ename, job
     from emp
     where comm is null;
```

Buscando valores que no son nulos.

- Listar todos los empleados que tienen comisión

```
SQL> select ename, job
     from emp
     where comm is not null;
```

Nota: Los valores nulos no utilizan espacio de almacenamiento en el RDBMS de Oracle.

Condiciones múltiples.

La cláusula **where** puede evaluar más de una condición para satisfacer una consulta.

```
SQL> select ename, job
     from emp
     where deptno = 20
     and job != 'CLERK';

SQL> select ename, job
     from emp
     where deptno = 20
     or job != 'CLERK';
```

Precedencia de operadores

- En la cláusula **where** se puede utilizar and y or.
- Se puede establecer precedencia por medio de los paréntesis **( )**.
- Si no se especifica precedencia se evalúa como sigue:
  Primero and
  luego or.

- Por ejemplo:

```
where deptno = 30
and job = 'SALESMAN'
or sal > 2000
```

Esta se refiere a los 'SALESMAN' en el departamento 30 ó los empleados que ganan más de $2,000.
No se refiere a los 'SALESMAN' y a los que ganan más de $2,000 que están en el departamento 30.

Expresiones.

- Se puede utilizar SQL para calcular valores de expresiones con operadores aritméticos. Estos son:

\+ suma
\- resta
* multiplicación
/ división

Los operadores pueden ser usados en las cláusulas:
select
where
order by y otras

Nota: No se podrán utilizar en la cláusula **from**.

Expresiones numéricas

- Se puede utilizar más de una expresión en la misma consulta
- ¿Quiénes ganan una comisión mayor al 5% de su salario?

```
SQL> select ename, sal, comm, comm/sal
     from emp
     where comm > 0.5*sal
     order by comm/sal desc;
```

Expresiones numéricas con múltiples operadores.

- Orden de evaluación
Primero:
  Multiplicación *
  División /
Después:
  Suma +
  Resta -

La evaluación es de izquierda a derecha.

- Se puede controlar el orden de evaluación con el uso de paréntesis **( )**.

Por ejemplo:
12 * (SAL + COMM) != 12 * SAL + COMM

Utilización de expresiones en fechas.

- Se puede utilizar una expresión para especificar valores que involucren datos tipo fecha. La unidad básica utilizada es el día.
- Se pueden sumar y restar fechas
- Ejemplo de suma:
  Añadir 2 días al 6-mar-87
  6-MAR-87+2 = 8-MAR-87

  Añadir 2 horas al 6-mar-87
  6-MAR-87+2/24 = 8-MAR-87 y 2 horas

  Añadir 15 segundos al 6-mar-87
  6-MAR-87+15/(24*60*60) = 6-MAR-87 y 15 segundos

Utilización de alias para los nombres de la columnas.

- El encabezado de las columnas se despliega normalmente con el nombre del campo especificado cuando se crea la tabla.
- Se puede conseguir un despliegue diferente para el encabezado de la columna especificando un alias en la cláusula select.
- Para especificar un alias se coloca un espacio después del nombre de la columna y luego el alias.

Por ejemplo:
```
SQL> select ename employee
     from emp;
     where deptno=10
```

Expresiones y alias.

- Si la cláusula select contiene una expresión aritmética, esta expresión es utilizada para el despliegue del encabezado. Un alias puede ser utilizado temporalmente para renombrar el encabezado de la columna calculada y hacer de esta forma, más entendible la consulta.

-¿Quiénes ganan una comisión mayor que el 5% de su salario?

```
SQL> select ename, sal, comm, comm/sal "C/S RATIO"
     from emp
     where comm > 0.5 * sal
     order by comm/sal desc;
```

- La ordenación se realiza por medio de la columna calculada.
- Cuando un alias contenga espacios o caracteres especiales como el "/" debe encerrarse entre comillas.
- El alias únicamente afecta al comando **select** sobre el cual se utiliza. Este no tiene efectos sobre otras consultas.

Resumen.

Sintaxis completa:

```
select...
from...
where...
group by...
having...
order by...
```

Hasta ahora se han utilizado:

```
select col_1,col_2...col_n
from  nombre_de_la_tabla
where condición
order by col_1,col_2...col_n
```

Manejo del Buffer.

- El **buffer de SQL** mantiene el comando SQL actual, hasta que se ejecuta otro comando SQL ó exit.

- Comandos SQL*PLUS para edición:
* **list ó l.** Despliega el contenido del buffer.

* **list 4 ó 4**. Lista la cuarta línea del comando almacenado en el buffer y la hace la línea actual.

* **change ó c**. Cambia la primera ocurrencia de texto de la línea actual del buffer
nota: Con ... se relaciona una cadena de cualquier longitud. Ejemplo:

SQL> c/(...)/('ANALYST')

* **input ó i**. Añade una ó más líneas al comando SQL actual.

* **append ó a**. Añade texto al final de una línea

* **delete ó del**. Borra la línea actual del buffer.

* **run**. Lista y ejecuta el comando que se encuentra almacenado en el buffer.

/. Ejecuta el comando del buffer.

* **edit**. escribe el comando que está en el buffer a un archivo de texto y llama al editor especificado en la variable de ambiente EDITOR. Esta variable es de sistema operativo, no de SQL*PLUS.

Revisión de objetivos tema IV

1.- Cuál es la instrucción para generar consultas en SQL.
2.- Qué función tiene la cláusula *where.*
3.- Para que nos sirven los alias
4.- Cómo ordeno los resultados de una consulta
5.- Realice la práctica No. 1

Práctica No. 1.

1. Lista los pasos para entrar a una sesión de SQL*PLUS.
2. Despliega el contenido de la tabla **EMP**.
3. Despliega el nombre de los empleados que trabajan en el **departamento 20** y asegúrate que la **columna** que contiene al nombre de los empleados se llame **NOMBRE**.
4. Despliega el nombre de los empleados de los **departamentos 10 y 20**, en una lista **ordenada alfabéticamente** de manera **descendente**.
5. Despliega el nombre de todos los departamentos cuya letra inicial sea **diferente** de la letra '**N**'.
6. ¿Existen empleados que además de su salario perciban alguna comisión? de ser así, presenta una lista con los **salarios en forma descendente** y si dos empleados o más tienen el **mismo salario**, **ordénalos alfabéticamente** por **nombre**.
7. Presenta un reporte que contenga el **nombre del empleado**, y el **porcentaje de la comisión que recibe respecto de su salario**. Asegúrate que el reporte no tenga operaciones ni abreviaciones como nombres de columna. **No** se deben incluir los empleados que **no tengan comisión** y debe ser **ordenado por nombre**.

Tema V. Manipulación de los datos.

Hasta el momento hemos creado consultas a la base de datos, otra parte importante de SQL es que también nos permite la manipulación de estos datos, es decir, podemos cambiar sus valores, borrarlos, insertar nuevos datos entre otras cosas.

Objetivo:

El alumno al final del tema podrá:

- Insertar nuevos datos a la base de datos.
- Actualizará los datos ya existentes en la base de datos.
- Podrá borrar registros de acuerdo al cumplimiento de ciertos criterios o condiciones.

Inserción de datos.

Ejemplo:

```
SQL> insert into dept
     values (10,'ACCOUNTING','NEW YORK');
```

Antes de insertar datos a una tabla ésta debe estar creada.
Los valores de los campos se separan con comas.
Utiliza el comando DESCRIBE para mostrar el orden y tipo de las columnas.
Cada tipo de dato insertado debe relacionarse con el de la tabla.
Los datos de carácter y de fecha se encierran entre apóstrofos.

Al realizar una inserción se puede especificar el orden de los datos.
Ejemplo:

```
SQL> insert into dept(dname,deptno)
     values ('ACCOUNTING',10);
```

- Insertar un registro de otra tabla.

Se puede usar el comando insert con una consulta para seleccionar registros de una tabla e insertarlos en otra.
La consulta reemplaza la cláusula **values**.

Ejemplo:

```
SQL> insert into emp(empno,ename,deptno)
     select id,name,department
     from old_emp
     where department in(10,20,30,40);
```

- Utilización de parámetros en el comando insert.

Un comando insert puede contener parámetros cuyos valores se preguntarán cuando el comando se ejecute.
Cada parámetro consiste de un **&** el cual tradicionalmente es seguido por el nombre de la columna.

Ejemplo:
```
SQL> insert into dept
     values (&deptno,&dname,&loc);
```

La ejecución de esta sentencia provoca que SQL*PLUS pregunte por los valores de cada parámetro.
Al repetir la ejecución permite al usuario insertar múltiples registros rápidamente.

No se necesita poner apóstrofos a los datos tipo char y date si se colocan en el comando insert.
```
SQL> insert into dept
     values(&deptno,'&dname','&loc');
```

- Insertando Nulos.

Si alguna de las columnas de la tabla en la cual se insertan datos no se coloca en el comando insert, el valor que tendrá será NULL.
El valor NULL puede especificarse en el comando insert
```
SQL> insert into dept
     values (50,'EDUCATION',NULL);
```

- Insertando valores tipo fecha.

El formato de default para datos tipo fecha es:
'DD-MON-YY'

Introducir un nuevo empleado llamado 'STONE'
```
SQL> insert into emp
     (empno,ename,hiredate)
     values (7963,'STONE','07-APR-87');
```

Para introducir automáticamente la fecha y hora de hoy, se utiliza **SYSDATE.**

Introducir un nuevo empleado 'KOHN', el cual fue contratado hoy
```
SQL> insert into emp
     (empno,ename,hiredate)
     values (7600,'KOHN',SYSDATE);
```

- Actualizaciones.

Promover a 'MARTIN' a 'MANAGER'
```
SQL> update emp
     set job = 'MANAGER'
     where ename='MARTIN';
```
*Nota: Si se omite la cláusula where todos los valores en la columna **job** se cambian a MANAGER.*

- Actualizar múltiples registros.

Cambiar todos los 'SALESMAN' a 'MARKET REP'.

```
SQL> update emp
    set job = 'MARKET REP'
    where job = 'SALESMAN';
```

- Actualizar múltiples columnas.

Cambiar todos los 'SALESMAN' a 'MARKET REP' y transferirlos al departamento 40.

```
SQL> update emp
    set deptno=40, job='MARKET REP'
    where job = 'SALESMAN';
```

- Borrado de registros.

No se puede borrar registros parcialmente. Lo que se podría hacer es poner las columnas como NULL.
La cláusula where determina cuáles registros serán borrados.
Si se omite la cláusula where se borrarán todos los registros.

Despidieron a 'MARTIN', hay que quitarlo de la tabla.

```
SQL> delete from emp
    where empno=7654;
```

COMMIT y ROLLBACK.

Las acciones de inserción, borrado y actualización a tablas no son hechas permanentemente, es decir se guardan en un área temporal hasta que se haga COMMIT.
El commit es un comando que implica una aceptación de todas la modificaciones que se han hecho a la base de datos hasta antes de ser ejecutado.

Hasta que no se haga un COMMIT solo el usuario que ha hecho los cambios a las tablas puede ver esos cambios; los demás usuarios ven las tablas como hayan quedado después del último COMMIT.

Un COMMIT puede ser explícito, implícito o automático:

COMMIT explícito. Se utiliza el comando COMMIT para hacer permanentes todos los cambios pendientes.

```
SQL> commit
```

COMMIT implícito. Los siguientes comandos hacen COMMIT implícitamente:
alter, audit, comment, connect, create, disconnect, drop, exit, grant, noaudit, quit, revoke, y rename.

COMMIT automático. Está en automático cuando se utiliza la siguiente instrucción:
SQL> set autocommit on
Los cambios generados por los comandos insert, update o delete se hacen permanentes cuando está habilitado autocommit.

El comando rollback.

El comando ROLLBACK cancela todos los cambios que se han hecho hasta antes de dar un rollback, el rollback es el comando contrario al commit, por lo que si se quieren cancelar los cambios debe ejecutarse antes de dar un commit.

Cuando se ejecuta ROLLBACK la base de datos es restablecida al momento en que se dió el último COMMIT. .

Ejecución de rollback:
SQL> rollback

TRANSACCIONES LÓGICAS

Todos los cambios a la base de datos entre dos **commit** sucesivos son llamados una transacción.

Cuando una transacción es interrumpida por un problema serio, tal como una falla en el sistema, la transacción completa es automáticamente cancelada (rolled back).

Ejemplo de una transacción lógica:
. La creación de un nuevo departamento en una compañía:
La transacción consiste de actualizar la columna en la tabla EMP y de insertar un nuevo registro en la tabla DEPT.

Revisión de objetivos tema V

1.- Cómo se introduce un nuevo registro a una tabla.
2.- De qué forma puedo modificar los datos que existen en una tabla.
3.- Cómo borro los registros de una tabla.
4.- Qué es una transacción lógica
5.- Realiza la práctica No. 2

Práctica No. 2.

1. **Agrega** a la tabla **EMP** el nombre de **'JOE WILSON'** e incluye la información necesaria para completar su registro.
2. **Promueve** a **'JOE'** al puesto de **'MANAGER'**.
3. **'JOE'** fue despedido. **borra** su registro de la tabla correspondiente.
4. A los empleados de la oficina de Chicago **(DEPTO 30)** dales un 10% de aumento. Modificar la tabla apropiada, donde se refleja este cambio.
5. Crear un departamento de facilidades para **'HITECH CORP.'** y asignarle el número **50**. No se cuenta con más información de este departamento.

Tema VI. Creación de tablas y vistas.

Hasta el momento hemos visto como manipular los datos de las tablas y como hacer consultas, ahora veremos como construir estas tablas, que es la parte de la implementación del modelo, como ya se mencionó SQL es el estándar de los manejadores de bases de datos relacionales, por lo que las instrucciones para la creación de las tablas difiere muy poco dependiendo del manejador que se esté utilizando.

Objetivo:

Al término del tema el alumno podrá:

Implementar un modelo relacional:

- Seleccionará tipos de datos para las columnas las tablas.
- Dará longitudes y características a esas columnas (Llave primaria, NN, ND, NC, etc)
- Modificará la estructura de las tablas.
- Creará tablas a partir de otras.
- Generará vistas de las tablas.

Creación de una tabla.

* Cuando se crea una tabla se especifica lo siguiente:
  - Nombre de la tabla.
  - Nombres de las columnas.
  - Tipos de datos de las columnas.
  - Tamaño máximo de las columnas.

* El diccionario de datos es actualizado automáticamente.
* Una tabla puede llegar a tener hasta 254 columnas.

Reglas para nombrar tablas y columnas.

Restricciones en los nombres:

- El primer carácter debe ser A-Z ó a-z (pero todos son almacenados en MAYÚSCULAS).
- Los siguientes caracteres pueden ser números ó $, #, y _ ; (no se permiten comas)
- Los nombres pueden ser hasta de 30 caracteres.

Los nombres deben ser únicos:

- El nombre de la tabla debe ser único para el **userid** que la creó.
- No se deben utilizar palabras reservadas de ORACLE.
- Los nombres de las columnas deben ser únicos dentro de la tabla.

Se pueden utilizar comillas:

- Si se encierra el nombre de la tabla en comillas, las reglas antes mencionadas no se aplican.

- Los subsecuentes accesos a las tablas nombradas de esta forma requerirán el uso de comillas.

Tipos de datos.

*** char(n)**
n = Máxima longitud de la cadena. (hasta de 240)

*** number(n,d)**
n = máximo número de dígitos
d = máximo número de dígitos a la derecha del punto decimal.

*** date**
Es un dato que contiene una fecha y una hora.

*** long**
Hasta de 65,536 caracteres (64K) debe ser almacenado por campo.

*** raw**
Renglón de datos binarios.

- Control sobre las columnas con respecto a NULL.

Algunas veces se desea asegurar que una columna no contenga valores nulos.
Para hacer esto se introduce la cláusula NOT NULL al final de la definición de la columna.

Ejemplo:

```
SQL> create table dept
    (deptno number(2) not null,
    dname char(14),
    loc char(13));
```

Al tratar de insertar un registro sin valor para la columna **deptno**, se producirá un error.

- Alteración en la estructura de las tablas.

- Añadir una columna.

```
SQL> alter table dept
    add (headcnt number(3));
```

Cuando una columna es nueva sus datos son nulos.

- Alterando el tamaño de la columna.

```
SQL> alter table dept
```

```
modify dname char(20);
```

Se utiliza para agrandar el tamaño de columna.
No se puede reducir el tamaño a menos que la columna este vacía.
No se puede cambiar el tipo de dato a menos de que la columna este vacía.

Si la columna no tiene valores nulos se puede cambiár a **not null**.

```
SQL> alter table dept
   modify dname not null;
```

Vistas

Una vista es una ventana a través de la cual se puede ver o cambiar información de una tabla.
Una vista es una tabla virtual:

- Se ve como una tabla.
- Sus datos se derivan de tablas.

Sin embargo, no es una copia de los datos.
Una vista provee:

- Simplicidad. Para ver precisamente lo que se necesita.
- Seguridad. Para prevenir que usuarios sin autorización vean lo que no deben.

Se pueden tener muchas vistas de una sola tabla.

Creación, nombramiento y observación de vistas.

Creación de una vista:

```
SQL> create view managers as
   select ename, job, sal
   from emp
   where job='MANAGER';
```

La sentencia select no puede contener la cláusula **order by**.

Para ver la información contenida en una vista se hace igual que si fuera una tabla:

```
SQL> select *
   from managers;
```

Creación de vistas con alias para las columnas.

A menos de que otra cosa sea especificada, una vista hereda los nombres de columnas de la tabla sobre la cual se basa.
Para dar otros nombres de columna se utiliza la siguiente sintaxis:

```
create view viewname (alias, alias,...)
as query;
```

```
SQL> create view mydept
    (person, title, salary)
    as select ename, job, sal
    from emp;
    where deptno = 10;
```

Actualizando una vista:

Es posible borrar renglones de una tabla a través de una vista, si la consulta (QUERY) usada para definir la vista cumple que:
* Selecciona renglones de solo una tabla.
* No contiene una cláusula GROUP BY o DISTINCT, una función de grupo o una referencia a la pseudocolumna ROWID.

Se pueden actualizar renglones de una tabla a través de una vista, si la consulta usada para definirla observa las dos restricciones anteriores y la siguiente:
* No define alguna de las columnas que son actualizadas, con una expresión.

Se pueden insertar renglones en una tabla, a través de una vista, si la consulta usada cumple las tres restricciones anteriores y:
* Cualquier columna NO NULA (NOT NULL) definida en la tabla es representada en la vista.

La opción **with check option** especifica que las inserciones y actualizaciones realizadas a través de una vista no sean permitidas si se modifican los datos clave con que fué creada.

Ejemplo:

```
SQL> create view dept20 as
    select ename,,job,sal,deptno
    from emp
    where deptno = 20;
    with check option;
```

Ahora, la siguiente sentencia producirá un error:

```
SQL> update dept20
    set deptno = 30
    where ename = 'WARD';
```

Borrando tablas y vistas.

Borrado de tablas:
El comando para borrar tablas es **drop table**.
Si la tabla tiene datos, éstos serán borrados junto con la estructura de la tabla.
Una vez borrada no se puede recuperar.
Este comando tiene COMMIT implícito.

Borrado de vistas:
El comando es **drop view**
La(s) tabla(s) sobre las cuales está basada la vista no tendrán cambio alguno.

Revisión de objetivos tema VI

1.- Cómo creamos una tabla.
2.- Cómo indicamos que una columna será llave primaria.
3.- Cómo cambiamos la estructura de una tabla.
4.- Para qué nos sirven las vistas
5.- Se puede alterar una tabla a través de una vista.
6.- Realice la práctica No.3.

Práctica No. 3

1. Agregarle a la tabla EMP una columna que contendrá el primer nombre de cada empleado.
2. Cambiar la columna de apellidos para que acepte cadenas de hasta 24 caracteres de longitud.
3. Crear una vista con todos los empleados del departamento 30, incluyendo nombre, fecha, salario y comisión (si existe).
4. Crear una vista de todos los 'MANAGERS' incluyendo su nombre, número de empleado y número de departamento
5. Listar de la vista de 'MANAGERS', aquellos managers que tengan número de empleado mayor a 7650.
6. ¿Cuáles empleados de la vista del departamento 30, tienen una comisión superior a $100 por mes.
7. Crear una vista llamada empleado, que contenga para cada empleado las columnas:
   - Nombre del empleado.
   - Número del empleado.
   - Trabajo que realiza.

La vista no debe contener a los que sean 'CLERK'.

Tema VII. Consultas avanzadas.

Una parte importante del modelo relacional es precisamente la forma en la que relacionamos las tablas para obtener la información que necesitamos, para ello nos basamos en una operación del álgebra relacional que ya conocemos, el join.

Objetivo:

Al final del tema el alumno será capaz de:

- Crear consultas en base a más de una tabla.
- Generar consultas en base a subconsultas.

JOIN.

Se utiliza para combinar columnas de diferentes tablas.

Cuando se omite la cláusula where cada registro de la primer tabla se relaciona con todos los registros de la segunda tabla. El resultado es un producto cartesiano.
Ejemplo:

```
SQL> select ename,emp.deptno,loc
     from emp,dept;
```

Elaboración de un join simple (Equijoin).

```
SQL> select empno,ename,job,
     emp.deptno,dname
     from emp,dept
     where emp.deptno=dept.deptno;
```

En la cláusula **where** se especifican las columnas para realizar el join.

Outer Join

Si existe algún valor en dept.deptno que no se relacione con ningún valor en emp.deptno (tal como el departamento 40), los registros no aparecerán. Si se desea que éstos aparezcan se coloca el símbolo de outer join (+).
Ejemplo:

```
SQL> select ename,dept.deptno,loc
     from emp,dept
     where emp.depto(+)=dept.deptno;
```

Self Join

Cuando se desea realizar el join sobre una sola tabla se llama self join.
Ejemplo: Determinar el nombre del jefe de cada empleado.

```
SQL> select worker.ename,manager.ename manager
     from worker,emp manager
     where worker.mgr=manager.empno;
```

Se puede realizar el join de una tabla como si fueran dos tablas.
Es necesario usar un alias por lo menos para una ocurrencia de la tabla en la cláusula from, para distinguir los nombres de las columnas de una y otra.

Non equijoin.

Si la condición del join en la cláusula where especifica un =, es un **equijoin**, de lo contrario es un **non equijoin**.

Ejemplo:

```
SQL> select ename,sal
    from emp,salgrade
    where grade=3
    and sal between losal and hisal;
```

Creación de vistas para ejemplos.

Vista Account

```
SQL> create view account as
    select ename,sal,job
    from emp,dept
    where emp.deptno=dept.deptno
    and dept.dname='ACCOUNTING';
```

Vista Research

```
SQL> create view research as
    select ename,sal,job
    from emp,dept
    where emp.deptno=dept.deptno
    and dept.dname='RESEARCH';
```

Vista Sales

```
SQL> create view sales as
    select ename,sal,job
    from emp,dept
    where emp.deptno=dept.deptno
    and dept.dname='SALES';
```

Unión de registros.

Un conjunto de operadores combinan dos o más consultas en un resultado.

UNIÓN. Registros de la primera consulta más registros de la segunda consulta, menos los registros duplicados.

Ejemplo:

Listado de todos los departamentos

```
SQL> select ename,sal
    from account
    where sal>2000
    UNIÓN
    select ename,sal
    from research
    where sal>2000
    UNIÓN
    select ename,sal
    from sales
    where sal>2000;
```

INTERSECT. Registros en común.

Las columnas deben ser del mismo tipo.

Ejemplo:
¿Qué puestos tienen en común todos los departamentos?

```
SQL> select job
     from account
     INTERSECT
     select job
     from research
     INTERSECT
     select job
     from sales;
```

MINUS. Registros que se generan en la primera consulta menos los de la segunda consulta.
Ejemplo:
¿Existen puestos en el Departamento **Accounting** que no se encuentren en el Departamento de **Sales**?

```
SQL> select job
     from account
     MINUS
     select job
     from sales;
```

Subqueries. (subconsultas).

Un subquery es una consulta en la cláusula **where**.
Los resultados del subquery son utilizados por la consulta principal.
Ejemplo: ¿Qué empleados trabajan en el departamento de 'SMITH'?
Paso 1: **Subquery**. Obtener el departamento donde trabaja 'SMITH'
Paso 2: **Principal**. Selecciona los empleados que trabajan en ese departamento.

```
SQL> select ename,deptno
     from emp
     where deptno=
          (select deptno
          from emp
          where ename='SMITH');
```

- Las subconsultas pueden tener múltiples niveles:
- El anidamiento puede continuar indefinidamente
- Las subconsultas pueden utilizar tablas que no son usadas en la consulta principal.
- Una subconsulta no puede tener la cláusula ORDER BY.

Múltiples subconsultas.

Una subconsulta puede ser el operando de un operador relacional.

Ejemplo: ¿Qué empleados tienen el mismo puesto que 'CLARK' o tienen un salario mayor que el de él?

```
SQL> select ename,job,sal
     from emp
     where job=
        (select job
        from emp
        where ename='CLARK')
     or sal>
        (select sal
        from emp
        where ename='CLARK');
```

Revisión de objetivos tema VII

1.-Mencione las operaciones que podemos utilizar para ligar más de una consulta.

2.-.Indique que es una subconsulta y en qué casos se utiliza.

VIII. Índices.

Los datos incluidos en una tabla como hemos visto no tienen ningún orden de almacenamiento y tampoco tienen ningún orden de acceso, para que lo tengan se tiene que crear un índice, es decir, una columna mediante la cual se tenga acceso los registros de manera ordenada.

Objetivo:

- Al final del tema el alumno será capaz de decidir en que casos es recomendable la creación de índices y cómo se crean.

Indexación de tablas.

Los propósitos de una indexación son los siguientes:

- Ayuda a que las consultas a tablas muy grandes sean más rápidas.
- La búsqueda de los datos es más eficiente.

Para crear un índice:

Ejemplo:

```
SQL> create index emp_ename
     on emp (ename);
```

Para borrar un índice

```
SQL> drop index emp_ename;
```

Información acerca de los índices.

- Indexar únicamente tablas mayores de 50 registros.
- Insertar datos antes de indexar.
- Una tabla puede tener más de un índice.
- Usualmente se indexa por medio de la columna que identifica al registro (llave primaria).
- Las consultas en SQL no cambian para tablas indexadas.
- Los índices son actualizados automáticamente.

Utilización de índices para asegurar valores únicos.

- Además de la eficiencia de las consultas en tablas indexadas, los índices se pueden utilizar para asegurar que el valor de una columna es único.

Ejemplo: Asegurar que cada número de empleado en la tabla EMP aparece únicamente una vez.

```
SQL> create unique index emp_empno
     on emp (empno);
```

Una vez creado el índice único, al tratar de insertar o actualizar un **número de empleado** (columna EMPNO) con uno que ya existe enviará un mensaje de error.

Revisión de objetivos tema VIII

1.- Cómo y para qué creamos índices.
2.- Cómo garantizo un índice único.
3.- Qué sucede con las tablas indexadas.

Tema IX. Reportes con SQL*Plus

Una parte importante de la creación de un modelo es la parte en la que se generan reportes, los reportes son los resultados de las consultas, sólo que estos se presentan en formatos diferentes.

Objetivo:

El alumno tendrá los elementos para:

- Generar reportes con formatos definidos.
- Generar archivos para automatizar las consultas.
- Crear reportes que involucren operaciones sobre los datos obtenidos de una consulta.

A continuación se darán comandos para la personalización de las consultas en SQL; esto es para darles un formato más adecuado, es decir, para la creación de un reporte.

Ttitle (Top Title)

sintaxis:

SQL>Ttitle { Right / Center / Left } ´ Título ´

Lo que hace el Ttitle es poner un titulo en la parte superior del reporte que se va a desplegar, en cada página del reporte aparecerá el título.

ejemplo:

SQL> Ttitle Center ´Reporte de Prueba´

Desplegará de manera centrada el título Reporte de Prueba.

Btitle (Botton Title)

sintaxis:

SQL>Btitle { Rigth / Center / Left } ´Pie de página´

Pone un pie de página al final de cada hoja de despliegue del reporte.

Tanto para el Ttitle como para el Btitle, se pueden desplegar títulos en 2 o más renglones para lo cual se puede utilizar el separador de renglones que en este caso es el pipe ´|´ .

ejemplo:

SQL>Title Center ´Bases de Datos | Grupo 2 ´

y se desplegaría de manera centrada: Bases de Datos
Grupo 2

El Ttitle y el Btitle aparecerán en todas las consultas que se generen después de haber sido definidos, por lo que es conveniente desactivarlos después de cada reporte.
sintaxis:
SQL> Ttitle off
SQL> Btitle off

COLUMN

Para cambiar de nombres a los encabezados de las columnas a desplegar podemos utilizar a parte de los ya mencionados ´alias´ el comando COLUMN.

sintaxis:

SQL> Column {nombre del campo} **heading** ´Encabezado´

ejemplo:
Se quiere que al desplegar la información de los empleados de la tabla emp aparezca el encabezado ´Nombre´ en el campo ´ename´ .

SQL> Column ename heading ´Nombre´

El comando column también nos permite cambiar el formato de despliegue de la columna que le indicamos.

sintaxis:

SQL> Column { nombre del campo } **format** $9,999,999.999 (para campos numéricos el formato en pesos y con las comas de separación.
format a# (para campos alfanuméricos el # se sustituye por el número de espacios en los que queremos que se despliegue el campo)

ejemplo:

SQL> Column sal format $ 9,999.99 - desplegará el campo sal con ese formato
SQL> Column ename format a10 - el campo ename se desplegará en 10 espacios únicamente.

Al igual que la mayoría de los comandos de SQL*Plus al ser activados aparecerán en los subsecuentes reportes, para desactivar el comando COLUMN:

sintaxis:
SQL> Column { nombre del campo } **clear**

BREAK

El comando break es un agrupador de reportes, agrupa de acuerdo al campo que le indiquemos, y permite que el depliegue de la información de ese campo si es que aparece varias veces lo haga en solo un ocación al inicio de los datos que coinciden con ese campo.

Por ejemplo, se requiere desplegar la información de los empleados de la empresa y agruparlos de acuerdo al departamento en el cual laboran.
Para hacerlo bastaría con indicarlo en la cláusula GROUP BY de la siguiente forma:

```
SQL> Select *
     From emp
     Group by deptno;
```

Sin embargo, al desplegar la información el campo deptno aparecerá por cada registro que este incluido en la tabla.

Para evitar esto se utiliza el comando BREAK.

sintaxis:

SQL> Break on {campo a agrupar}

ejemplo:
considerando el ejemplo anterior:

```
SQL> Break on deptno
SQL> Select *
     From emp
     Group by deptno;
```

El Break siempre va a acompañado del Group By.

En un reporte se pueden hacer operaciones u ordenamientos sobre los datos que se estén desplegando por cada hoja de reporte o bien para todo el reporte por lo que existen dos cláusulas break muy utilizadas cuyo funcionamiento se comprenderá mejor al explicar el comando compute, estas cláusulas son:

Break on page
Break on report

COMPUTE

El comando compute es utilizado para realizar operaciones sobre las columnas que se están deplegando sobre un reporte.
sintaxis:
SQL> **Compute** {operación} **of** {campo} **on** {campo de agrupamiento} **{operación}**

Las operaciones que podemos utilizar son las siguientes:

- **sum** - Realiza las suma de los valores del campo indicado.
- **avg** - Obtiene el promedio de los valores que aparezcan en el campo indicado.
- **count** - Devuelve el número de valores no nulos que encontró en el campo indicado.
- **max** - Devuelve el valor máximo encontrado en el campo.
- **min** - Devuelve el valor mínimo encontrado en el campo.
- **std** - Realiza la desviación estándar de los valores del campo indicado.
- **var** - Encuentra la varianza sobre los datos del campo.
- **number** - Devuelve el número de registros incluyendo nulos que hay en la tabla.

{campo}

Campo de la tabla bajo el cual queremos que se realice alguna de las operaciones anteriores.

{campo de agrupamiento}

Este campo esta directamente relacionado con el campo indicado en el comando break, por lo cual, cada que se realiza un ***compute*** debe existir un ***break*** anterior.

Por ejemplo, se requiere hacer la suma del salario de los empleados por cada departamento y que aparezca la suma en el reporte.
con el comando compute quedaría de la siguiente forma:

```
SQL> break on deptno
SQL> compute sum of sal on deptno
SQL> select ename, deptno, sal
     from emp
     group by deptno;
```

SET

El comando set nos permite fijar los valores para las variables de ambiente que podemos manipular en SQL*Plus para automatizar nuestros reportes.
sintaxis:

SQL> **set** {variable de ambiente} valor

{variable de ambiente}

Las variables que podemos manipular son la siguientes:

pagesize {n} Donde n es el tamaño de renglones que queremos que tenga una página el valor por deafult es 14.

linesize {n} Donde n es el número de columnas que tenga la página el valor por default es 80.

heading {On/Off} Elimina o activa los encabezados que hayamos definido en el column.

feedback {On/Off} Elimina o activa el mensaje que aparece al final de cada consulta que indica el número de registros manipulados

echo {On/Off} Elimina o activa el eco, es decir permite que aparezca o no la instrucción que se esta ejecutando.

pause {On/Off/Texto} Hace o no pausa cuando termina una página, Texto lo podemos sustituir por un mensaje que aparecerá por cada pausa que se realice.

null `cadena´ Sustituye el nulo por una cadena sólo para cuestión de despliegue.

autocommit {On/Off} Activa y desactiva la ejecución permanente de la acciones que alteran la bases de datos.

Revisión de objetivos tema IX

1.- Qué se hace para cambiar los títulos de las columnas en los reportes.
2.- Para qué utilizamos el break.
3.- Cómo se cambia el tamaño a la página.
4.-Realizar la práctica 4.

Práctica No. 4

1.- Crear un reporte que incluya lo siguiente:

- Que el titulo sea ´Registros de Empleados ´
- Como subtítulo ´Hietch Corp´
- La fecha del día, en la parte superior.
- El número de la página en la parte superior.
- Como pie de pagina debe llevar ´Realizado por un estudiante de SQL*Plus´.
- Los empleados deben estar agrupados por departamento y cada departamento debe estar separado del siguiente por 2 líneas.
- Debe aparecer la suma de los salarios por departamento.

Tema X.-Funciones

Las funciones permiten modificar, combinar y cambiar formatos de los valores que se despliegan en un reporte.

Objetivo:
- El alumno utilizará las funciones para obtener resultados deseados en los reportes.

Las funciones las clasificamos de acuerdo al tipo de datos que manipulan y por el número de renglones que involucran, así tenemos funciones aritméticas grupales, funciones carácter individuales, etc.

Funciones tipo carácter:

1) **Initcap(cadena)** Pone en mayúscula la primera letra de la cadena que le indiquemos.
Ejemplo:

initcap('juan pérez) -----> 'Juan Pérez'

2) **Length(cadena)** Da el número de caracteres que componen la cadena(Longitud).
Ejemplo:

length('pedro')----------> 5

3) **Substr(cadena',inicio,longitud)** Extrae una subcadena a partir del carácter que le indiquemos, de la longitud indicada y a partir de la cadena proporcionada.
Ejemplo:

substr('juan pérez',7,5)-> 'pérez'

4) **Lower(cadena)** Convierte a minúsculas la cadena.
Ejemplo:

lower('Juan Pérez')------> 'juan pérez'

5) **Upper(cadena)** Convierte a mayúsculas la cadena.
Ejemplo:

upper('juan pérez')-----> 'JUAN PÉREZ'

6) **Least(cadena1,cadena2,cadena3...)** Devuelve la cadena menor alfabéticamente.
Ejemplo:

least('pedro','juan','ana')-----> 'ana'

7) **Greatest(cadena1,cadena2,cadena3...)** Devuelve la cadena mayor alfabéticamente.
Ejemplo:

greatest('ana','juan','pedro')-> 'pedro'

Las funciones las podemos incluir en las siguients cláusulas del comando select.

Select
Where
Order by
Having

Funciones tipo fecha:

El manejo de fechas en SQL se hace de manera especial, se consideran cadenas , pero tiene sus propias funciones para manipularlas.

1) **add_months(fecha,número de meses)** Suma los meses que le indicamos a una fecha dada.
Ejemplo:

add_months('13-jan-96',8)-------->'13-sep-96'

2) **months_between(fecha1.fecha2)** Devuelve el número de meses entra las dos fecha.
Ejemplo:

months_between('24-nov-95','24-jan-96')------->2

3) **next_day(fecha,día en inglés)**. Devuelve la fecha en la que caerá el día que le indicamos.
Ejemplo:

next_day('17-jan-96','saturday')---->20-jan-96

4) **to_char(fecha,formato)** Cambia el formato de la fecha al que le indiquemos de acuerdo a los formatos existentes:

DIAS		**MESES**	
dd	12	mm	3
dy	fri	mon	mar
day	friday	month	march
ddspth	twelfth		

AÑOS	
yy	96
yyyy	1996

El formato se indica entre apóstrofos.
Ejemplo:

to_char(sysdate,'month, day dd, yyyy')------> january, Monday 22, 1996

Funciones aritméticas:

Las funciones aritméticas son las mismas que se mencionaron para el comando compute y algunas más. Sólo que aquí veremos su aplicación como función no como operación.

1) **sum(campo)** - Realiza las suma de los valores del campo indicado.

2) **avg(campo)** - Obtiene el promedio de los valores que aparezcan en el campo indicado.

3) **count(campo)** - Devuelve el número de valores no nulos que encontró en el campo indicado.

4) **max(campo)** - Devuelve el valor máximo encontrado en el campo.

5) **min(campo)** - Devuelve el valor mínimo encontrado en el campo.

6) **std(campo)** - Realiza la desviación estándar de los valores del campo indicado.

7) **var(campo)** - Encuentra la varianza sobre los datos del campo

8) **number(campo)** - Devuelve el número de registros incluyendo nulos que hay en la tabla.

9) **abs(campo)** - Torna el campo a su valor absoluto.

10)**round(campo,decimales)** - Redondea el valor con el número de decimales que le indicamos.
Ejemplo:

round(12.966,2)------>12.97

11)**trunc(campo,decimales)** Pone el número en el número de decimales truncando el valor.
Ejemplo:

trunc(12.966,2)------>12.96

12)**nvl(campo con valores nulos,valor o cadena)** Cambia el valor de los nulos sólo por lo que durará consulta para poder operar con ellos o para desplegarlos en los reportes.

Como se pude observar entre las funciones anteriores tenemos funciones grupales e individuales, éstas no deben mezclarse en una consulta de manera normal, para ello debemos hacer uso de la cláusula GROUP BY, que ya había sido mencionada, ésto con el fin de agrupar los datos apartir del campo en el cual se está realizando la función grupal.
Ejemplo:
Mostrar los gastos que genera cada departamento en salarios para los trabajadores.

```
SQL> select deptno,sum(sal)
     from emp;
```

Esta sería nuestra lógica respuesta, sin embargo en el select estamos mezclando valores individuales (columna deptno) y valores grupales (suma de los salarios), que en este caso la esta haciendo de manera general y nos devuelve sólo un valor.
Es aquí donde interviene la cláusula Group By que nos permite realizar la operación grupal de acuerdo a cada valor diferente de un campo que le indiquemos:

```
SQL> select deptno,sum(sal)
     from emp
     group by deptno;
```

Así nos mostrará por cada departamento (deptno) la suma de salarios.

La cláusula Group by nos permite generar condiciones para los valores que se pueden obtener a través de funciones grupales, esto es, por ejemplo: se quiere saber qué departamentos gastan en salarios más de $9,000.
La consulta sería básicamente la misma, sin embargo, ahora nos ponen una condición, como hemos visto hasta el momento las condiciones las colocábamos en la cláusula Where, pero la cláusula where la utilizábamos para condiciones individuales, es decir, de registro a registro, es por ello que para condiciones de tipo grupal existe otra, la cláusula HAVING.

Para el ejemplo anterior tenemos ahora:

```
SQL> select deptno,sum(sal)
     from emp
     group by deptno
     having sum(sal)>9000;
```

El tener una condición grupal no implica que no se pueda tener una condición individual y viceversa.

Revisión de objetivos X

1.- Para qué se utilizan las funciones.
2.- Qué hace la cláusula having.
3.- Qué función me permite cambiar el formato de la fecha.
4.- Preparar un reporte de salarios que cumpla con los siguientes requerimientos:
- El reporte debe incluir el nombre del empleado, la antigüedad del empleado en semanas, el salario y el número de departamento en el que labora.

- Sólo aparecerá u departamento por página.
- Al final de cada página deberá ir la suma del de los salarios de loe empleados por departamento y la suma de las semanas de trabajo de todos los empleados.
- Todos los salarios deben de ir en dólares y la antigüedad de los empleados en semanas.
- El reporte debe llevar el título en cada página del reporte.

5.- Realizar las prácticas 5 y 6.

Práctica No. 5

Nota: Esta práctica involucra 3 nuevas tablas :
- Invention
- Nation
- Border
Pregunta a tu instructor como obtener estas tablas.

1.- Selecciona los nombres de todos los inventores que tienen más de dos inventos y ordena el resultado por la primera letra de sus nombres.
2.- Cuántos inventores diferentes hay.
3.- Encuentra la naciones cuya densidad de población (población dividida entre su área territorial) es menor que la longitud del nombre de la nación.
4.- Crea un reporte que incluya el nombre del empleado, el total de su salario (salario más comisión) y su fecha de ingreso a la compañía, el reporte debe cumplir con:

- Los nombres deben llevar la primer letra en mayúscula.
- La fecha de ingreso debe tener el formato MM/DD/YYYY
- El salario total de cada empleado incluyendo a aquellos que no perciban comisión.
- Los nombres ordenados alfabéticamente.

Práctica No. 6

1.- Despliega el nombre del departamento en el cual trabaja SMITH. Incluye el nombre de SMITH en el despliegue.
2.- Despliega el nombre de los inventores cuyos nombres empiecen con 'B'. Incluyendo sus países.
3.- Reporta el número de inventos por su país de origen, esto lo encontrarás en la tabla **invetion**. Asegúrate de incluir el nombre del país en el resultado.
4.- Selecciona todos los inventos del mismo año en el que el inventor BENZ hizo un invento.
5.- Lista el nombre y el tamaño de todas las naciones que tienen frontera con más de 7 fronteras con otros países.
6.- Encuentra los nombres y la población de todas las naciones que son islas y tienen una extensión territorial mayor o igual que el área de Japón.
7.- Despliega todos los países y sus fronteras con otros países desplegando los países con los que comparte su frontera, despliegalas ordenadas por país.

Tema XI.- Privilegios

Entre los objetos que componen una base de datos como ya se mencionó existen los usuarios, a estos usuarios se les brindan ciertos privilegios para que puedan crear y manipular sus propios objetos (tablas, vistas, índices, etc.).

Objetivo:

Al término del tema el alumno será capaz de:

- Dar privilegios a otros usuarios para accesar a sus tablas.
- Crear sinónimos sobre sus tablas para el acceso de otros usuarios.

Privilegios de sistema:

A nivel de sistema existen 3 tipos de privilegios, de mayor a menor privilegio son los siguientes:

- **DBA** (Administrador de la Base de Datos) Tiene todos los privilegios.
- **Resource.** Permisos para consultar la base de datos y de crear tablas.
- **Connect.** Permiso para consultar tablas.

Sólo el DBA tiene permiso para crear usuarios de la base de datos.

Sintaxis:

SQL> **Grant connect to** {username} **identified by** {password};

ejemplo:

```
SQL> Grant connect
     to manuel
     identified by tiger
```

{username} Aquí indicamos el username del usuario al que se le dará el permiso de conectarse a la base de datos para consultar tablas.

{password} Se le otorga un password al usario, en realidad es un identificador.

Privilegios de tabla:

Estos los pueden otorgar todos aquellos que tengan objetos propios en la base de datos, es decir, cada usuario puede dar privilegios a otros usuarios para manipular los objetos que él ha creado.

sintaxis:

SQL>**Grant** {privilegio} **on** {nombre de la tabla} **to** {username}

Ejemplo:

```
SQL> Grant select
     on emp
     to manuel;
```

{privilegio} Los privilegios que se pueden otorgar son los siguientes:

Insert, Update, Delete, Alter, Select.

{username} Se indica el username del usuario al que se le brindan los privilegios.

Si se desea dar todos los privilegios a un usuario se puede utilizar la siguiente instrucción:

SQL> **Grant all on** {nombre de la tabla} **to** {username}

En caso de querer dar todos los privilegios a todos los usuarios:

SQL> **Grant all on** {nombre de la tabla} **to public**

Sinómimos:

Los sinónimos son la forma en la que un usuario puede utilizar la tablas de otro usuario.

Sintaxis:

SQL> **Create synonym** {nombre _sinónimo}
for {username.tabla}

Ejemplo:

SQL> **Create synonym empleado**
for manuel.emp

{nombre_sinónimo} Nombre con el cual el otro usuario utilizará nuestra tabla.

{Username.tabla} La forma en que se identifica una tabla en una base de datos es a través del username del usuario que la creo, un punto, y el nombre de la tabla, así dariamos nuestro username, punto, el nombre de la tabla que tendrá un sinónimo.

Segundo nivel de privilegios:

Una vez que se dan privilegios a un usuario, éste a su vez puede dar privilegios a otros usuarios sobre éstas tablas.

sintaxis:

SQL> **Grant** {privilegio/all}
on {nombre_tabla}
to {username}
with grant option;

La cláusula with grant option es la que nos permite dar privilegios a otros usuarios.

Ejemplo:

```
SQL> Grant all
     on dept
     to manuel
     with grant option;
```

Borrado de privilegios:

En ocasiones otorgamos privilegios sólo de manera temporal por lo que debemos quitar los privilegios.

sintaxis:

```
SQL> Revoke {privilegio}
     on {tabla}
     from {username};
```

Ejemplo:

```
SQL> Revoke update
     on dept
     from manuel;
```

Revisión de objetivos tema XI

1.- Cómo se otorgan privilegios a otros usuarios para accesar a sus tablas.
2.- Para qué se crean sinónimos.
3.- Cuántos niveles de privilegios existen y cuáles son.