10

UNIVERSIDAD NACIONAL AUTONOMA
DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
CAMPUS ARAGON.

"CONECTIVIDAD Y FUENTES DE SERVICIOS INTERACTIVOS ORACLE"

2972

TESIS PROFESIONAL

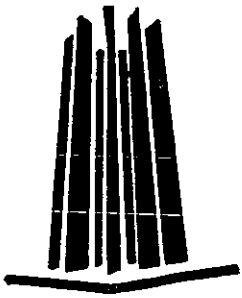
QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A :

SALOMON RODRIGO BERNAL JIMENEZ

ASESOR: ING. SILVIA VEGA MUYTOY



MEXICO, D. F. 2001

CONECTIVIDAD Y FUENTES DE SERVICIOS INTERACTIVOS ORACLE

A Xochitl Martinéz H.
A mis padres y hermanos por sus bendiciones
A mi gran amigo Sady, y al mejor entrenador
de futbol soccer Juan Carlos R.

Indice

		Página
Introducción		**1**
Capítulo I, Protocolos de comunicación		
1.1 I	TCP/IP	2
	1.1.1 Interconexiones de sistemas abiertos	3
	1.1.2 Jerarquías del protocolo ISO	4
	1.1.3 Protocolos asíncronos	7
1.2 I	Sistemas de nombres de dominio	7
	1.2.1 Direcciones IP, redes y subredes	9
	1.2.2 Números IP	9
	1.2.3 Tipos de clase	9
1.3 I	Observaciones	10
Capítulo II, Structured Query Languaje		**11**
2.1 II	Introducción	11
2.2 II	Conceptos básicos	12
2.3 II	Tipos de RDBMS(Relational Data Base Management System)	12
	2.3.1 Jerárquicos	12
	2.3.2 Red	13
	2.3.3 Relacional	13
2.4 II	Características de un RDBMS	13
2.5 II	Tablas, renglones y columnas	14
	2.5.1 Tablas relacionadas	14
2.6 II	Comandos de SQL	15
	2.6.1 Cómo entrar a SQL/Plus	15
	2.6.2 Pidiendo ayuda	16
	2.6.3 Diccionario de datos	16
	2.6.4 Comando Describe	17
2.7 II	Sintaxis general de un query	17
2.8 II	Manipulación de columnas en una tabla	17
	2.8.1 Selección de una columna específica	18
	2.8.2 Selección de múltiples columnas	19
	2.8.3 Selección de renglones	20
	2.8.4 Ordenado de renglones	20
	2.8.5 Onden de renglones con criterio multiple	21

2.9 II	Selección de renglones específicos	21
2.10 II	Operadores lógicos	21
2.11 II	Condiciones múltiples	24
	2.11.1 Presedencia de operadores multiples	25
2.12 II	Expresiones numéricas	26
	2.12.1 Expresiones numéricas con operadores múltiples	26
	2.12.1 Uso de expresiones numéricas en fechas	27
2.13 II	Uso de alias en columnas	27
2.14 II	Ambiente para edición de sentencias SQl	28
	2.14.1 Como insertar datos en una tabla	29
	2.14.2 Insertar renglones de otra tabla	39
	2.14.3 Uso de parámetros en el comando insert	39
	2.14.4 Insersión de valores	30
	2.14.5 Insersión de valores de tipo date	30
2.15 II	Actualización de datos	31
	2.15.1 Actualización de múltiples renglones	31
	2.15.2 Actualización de múltiples columnas	32
2.16 II	Borrado de renglones	32
2.17 II	Control de los efectos de cambios	33
2.18 II	Como crear una tabla	34
	2.18.1 Reglas para nombrar tablas y columnas	34
2.19 II	Tipos de datos mas comunes	34
2.20 II	Vistas	36
	2.20.1 Una tabla múltiples vistas	36
	2.20.2 Como crear, nombrar y consultar vistas	37
	2.20.3 Crear vistas con nombres de alias de columnas	37
2.21 II	Copiando tablas y columnas	38
	2.21.1 Borrar tablas y vistas	48
2.22 II	Encabezados y pies de pagina	40
	2.22.1 Nombres de columnas	40
	2.22.2 Manejo de cortes	41
2.23.II	Desplegar cálculos por grupo	42
2.24.II	Salvar comandos de un archivo	44
2.25 II	Recuperación de comandos almacenados	45
	2.25.1 Mandar la salida a un archivo	46

2.26 II Propósito de las funciones 46
2.26.1 Funciones de tipo char mas comunes 47
2.26.2 Funciones tipo DATE 47
2.26.3 Función TO_CHAR 48
2.26.4 Funciones numéricas 49
2.26.5 Función NVL 50
2.26.6 Funciones en grupo 51
2.27 II Clausula GRUP BY 53
2.27.1 Criterios múltiples para agrupar renglones 53
2.28 II Joins 55
2.28.1 Como hacer un join simple 56
2.28.2 Joins externos 56
2.28.3 Joins de tablas consigo mismas 57
2.28.4 Joins de no equivalencia 57
2.29 II Operadores de conjunto 58
2.29.1 Operador UNION 58
2.29.2 Operador INTERSECT 59
2.30 II Introducción a subquerys 61
2.30.1 Múltiples tablas y subquerys 62
2.30.2 Funciones de grupo en subquerys 62
2.31 II Seguridad de los datos 63
2.32 II Privilegios del sistema 63
2.32.1 Privilegios sobre tablas 64
2.33 II Sinónimos 64
2.34 II Como eliminar privilegios del sistema 65
2.35 II Indices 65
2.36 II Observaciones 66

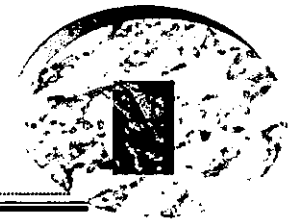
Capítulo III, Diseño del modelo relaciónal de la base de datos 67
3.1 III Introducción al análisis y diseño de bases de datos 67
3.2.1 Requerimientos del negocio 68
3.2 III Proceso de desarrollo de bases de datos 68
3.3 III Modelo de datos conceptual básico 70
3.4 III Componentes del modelo de entidad relacional 71
3.4.1 Sintaxis 71
3.4.2 Comunicación del usuario 71
3.4.3 Estándares para la diagramación de entidades 73

	3.4.4 Relaciones	74
	3.4.5 Sintaxis de una relación	74
	3.4.6 Estándares de diagramación	74
	3.5.6 Grados de relación	76
3.5 III	Como usar una matriz de relaciones	77
	3.5.1 Estándares de matrices de relaciones	78
3.6 III	Normalización de un modelo de datos	78
	3.6.1 Regla principal de la forma normal	79
	3.6.2 Validación	79
	3.6.3 Regla de la segunda forma normal	79
	3.6.4 Validación	79
	3.6.5 Regla para la tercera forma normal	80
	3.6.6 Validación	80
3.7 III	Llaves primarias	81
3.8 III	Introducción al diseño inicial de la base de datos	82
	3.8.1 Pasos para el diseño inicial de la base de datos	83
3.9 III	Observaciones	88
Capitulo IV,	**Implantación del sistema**	**89**
4.1 IV	Diseño de una base de datos oracle	89
4.2 IV	Introducción al ciclo vital de desarrollo de un sistema	89
	4.2.1 Conocimiento de los requerimientos de los usuarios	89
	4.2.2 Introducción al análisis	89
	4.2.3 Modelo conceptual de datos	89
4.3 IV	Asignación de atributos a los elementos	91
4.4 IV	Terminología relaciónal	93
4.5 IV	Normalización	94
	4.5.1 Primara forma normal	94
	4.5.2 Segunda forma normal	95
	4.5.3 Tercera forma normal	96
4.6 IV	Afinación de la cardinalidad	97
4.7 IV	Creación unicial de la base de datos oracle	100
	4.7.1 Creación de un tablespace	101
	4.7.2 Creación de un segmento de rollback	101
	4.7.3 Modificación de las usuarios SYS y SYSTEM	101

4.8 IV Parámetros temporales del archivo INIT.ORA 104
4.9 IV Parámetros del producto(Archivo INITCC1.ORA) 104
4.10 IV Configuración de la base de datos(Archivo CONFIGCC1.ORA) 104
4.11 IV Observaciones 106

Capitulo V, Tips de administración oracle 107
5.1 V Proceso de funcionamiento 108
5.1.1 Tablespace 108
5.2.2 Instancia 110
5.2 V Elementos internos de la base de datos oracle 110
5.2.1 Tablas y columnas 110
5.2.2 Esquemas 112
5.2.3 Indices 112
5.2.4 Cluster 113
5.3 V Pasos para crear un usuario 114
5.3.2 Borrar un usuario 115
5.4 V Grupos 115
5.4.1 Grupos hash 115
5.5 V Vistas 115
5.6 V Secuencias 116
5.7 V Paquetes 116
5.8 V Privilegios y roles 116
5.9 V Enlaces de bases de datos 118
5.9.1 Forma en que se puede utilizar un enlace 118
5.10 V Segmentos, Extensiones y Bloques 118
5.11 V Segmentos de rollback 119
5.11.1 Modificación de un segmento de rollback 120
5.11.2 Borrar un segmento de rollback 120
5.12 V Estructura interna de la base de datos oracle 120
5.12.1 Constraints 120
5.12.2 Triggers 121
5.12 V Procedimientos almacenados 123
5.12.1 Estructura general de un PL/SQL 124
5.12.2 Cursores 125
5.13 V Conclusiones 127
5.14 V Bibliograía 128

Capítulo



PROTOCOLOS DE COMUNICACIÓN

INTRODUCCIÓN

En los últimos años se ha visto notablemente incrementada la tendencia a construir e implementar sistemas de información que ayuden a la toma de decisiones; esto se debe a la rapidez con la que ahora las empresas deben responder a los cambios en los que operan. Afortunadamente existen en los mercados muchos tipos de software y herramientas para la construcción de los mismos, por ello es importante conocerlos con el fin de ayudar a las empresas a determinar el tipo de sistema que le sea mas rentable, para que de esta manera se pueda desarrollar un software integral de aplicaciones que pueda mejorar la forma en que las empresas realizan sus negocios.

Aún cuando en una empresa las proyecciones son prometedoras y los márgenes de rentabilidad se mantienen satisfactorios ó los clientes parecen estar contentos, aún en estas condiciones se debe pensar en un cambio con la idea de que los negocios podrían ir mejor. A través de este material podrá encontrar los mecanismos de apoyo para la construcción de cualquier aplicación en un ambiente **cliente/servidor**, incluso, se proporciona una interfaz gráfica fácil de usar, por medio de la cual el usuario puede consultar e interpretar la información y su correlación. Determinar las reglas del negocio y su perfecta validación de campos es uno de los principales objetivos de este trabajo, así como saber almacenar información en una base de datos múltidimencional de manera tal que el usuario pueda consultar a través de múltiples vistas. Para aquellos que se inician en el tema de las base de datos, este material les ayudara a convertirse en verdaderos conocedores del desarrollo de aplicaciones **cliente/servidor**, y para los que ya son expertos será un material de respaldo valioso para sus futuras aplicaciones.

Un ejemplo de este tipo de aplicaciones podría ser la implementación de un sistema para el monitoreo de las ventas de una empresa dedicada a la comercialización de artículos electrónicos con varias tiendas ubicadas en diferentes partes del país. La información base para el sistema sería la generada por las facturas o recibos de los clientes, la cual podría estar almacenada en una base de datos relacional en cada sucursal, o bien, en una base de datos centralizada en el corporativo.

Por sus características, estas aplicaciones ofrecen múltiples ventajas y beneficios que han probado ser un éxito en las empresas de grupos financieros, comerciales, tecnológicos y de investigación con presencia a nivel mundial. Estimado lector esperamos que sean cubiertas todas sus espectativas que desea de este material, y mucho éxito.

1.1 I PROTOCOLOS DE COMUNICACIÓN DE DATOS TCP/IP

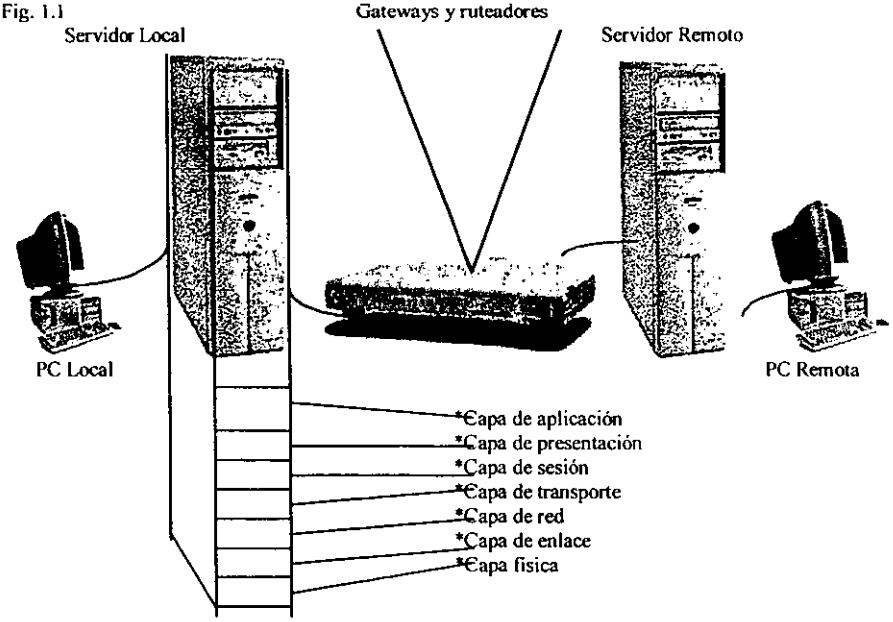
INTRODUCCION

La meta principal de este tema es brindar a los usuarios las herramientas necesarias para establecer una red para el control de flujo de operaciones. Una arquitectura de red deliniando la manera de como la red de comunicaciones de datos esta arreglada y estructurada. Generalmente incluye el concepto de niveles y capas de la arquitectura del modelo **OSI (open system interconection)**. Cada capa dentro de la red consta de *protocolos* específicos o reglas para comunicación de datos. Los protocolos son convenios entre los países del mundo para la transferencia de datos. Esencialmente, un protocolo es un conjunto de reglamentos para el comportamiento formal y ordenado de datos, así como un protocolo diplomático o un protocolo militar. Un protocolo de una red de datos es un conjunto de reglas que gobiernan el intercambio ordenado de datos, la principal función es controlar el flujo de información entre el programa de aplicaciones y las terminales remotas. Por lo que debe haber un conjunto de reglas que indiquen como un **LCU(Unidad de Control de Línea)** reacciona a diferentes tipos de transmisiones. Este conjunto de reglas se conoce como *protocolo de enlace de datos*, a su vez, este protocolo de enlace incluye las secuencias precisas de caracteres que aseguran un intercambio ordenado de datos entre diferentes fuentes de servicios.

Arquitectura

La correspondencia entre las capas de TCP/IP y las capas OSI se muestran de manera clara en la fig 1.1 , a continuación un ejemplo del uso de estos protocolos. Supóngase que un usuario desea transmitir un archivo a un servidor remoto, el servidor local le pedirá a la capa de transporte que cree una conexión de punto a punto con el servidor remoto, a su vez la capa de transporte utilizara los servicios de la capa de red (que se encarga esencialmente de enviar y recibir paquetes de información como se vera mas adelante) una vez solicitados los servicios de red, esta misma capa utiliza los servicios de la capa de enlace, que se encarga de enviar a través de la red local y de un ruteador la información de un servidor a otro, para hacerla disponible al usuario que hizo la solicitud. Este proceso es representado en el siguiente diagrama.

Fig. 1.1



Aquí se muestra el flujo de información y relación entre las capas, Cada capa define funciones especificas que permiten que el proceso completo se lleve a cabo.

La meta principal de la arquitectura de red es darle a los usuarios las herramientas necesarias para establecer la red y para el control del flujo de operaciones. Una arquitectura de red delinea la manera como la red de comunicaciones de datos está arreglada ó estructurada y generalmente incluye el concepto de niveles o capas dentro de la arquitectura. Cada capa dentro de la red consta de protocolos especificos o reglas para comunicarse en funciones comúnes.

En un circuito de comunicaciones de datos, la estación que esta transmitiendo actualmente se llama maestro, mientras que la estación receptora se llama esclavo. En una red centralizada la estación primaria controla el momento en que puede transmitir la estación secundaría. Cuando una estación secundaria esta transmitiendo el maestro y la estación primaria son ahora los esclavos. El papel del maestro es temporal y la estación primaria delega a cada estación la función del maestro. Inicialmente la primaria es el maestro, la estación primaria solicita a cada estación secundaria el turno para transmitir.
Los protocolos de enlace de datos generalmente se catalogan como sincronos y asincronos. Como regla los primeros utilizan un formato de datos sincronos y modems sincronos, mientras que los segundos utilizan formatos y modems asincronos.

INTERCONEXIONES DE SISTEMAS ABIERTOS

El termino **Interconexión de sistemas abiertos(OSI)**, es el nombre dado a un conjunto de estándares para las comunicaciones entre las computadoras. El propósito principal de los estándares **OSI** es servir como un guía estructural para intercambiar información entre computadoras, terminales y redes. El **OSI** se apoya por el **ISO** y **CCITT**, las cuales han trabajado juntas para establecer un conjunto de estándares **ISO** y recomendaciones **CCITT** que son esencialmente idénticas. En 1983, la **ISO** y **CCITT** adoptaron un modelo de referencia en la arquitectura de comunicación con siete capas, cada capa consta de protocolos especificos para comunicarse.

JERARQUIA DEL PROTOCOLO ISO

El modelo de siete capas de interconexión de sistemas abiertos ISO se muestra de la siguiente forma.

Fig. 1.2

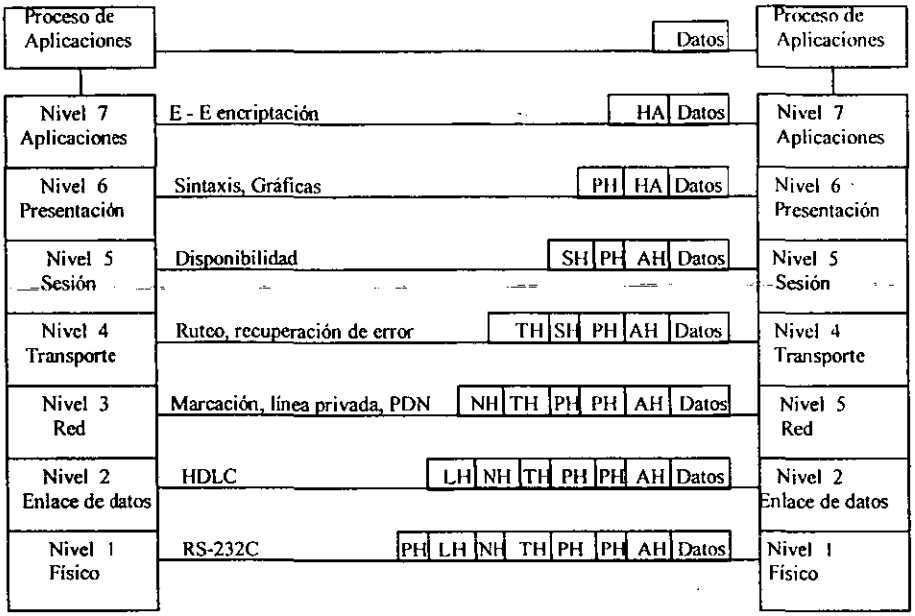


Fig . 1.1 Jerarquía del protocolo internacional **ISO**, **AH**, es el encabezado de aplicaciones; **PH**, Encabezado de presentación; **SH**, Encabezado de sesión; TH, encabezado de transporte; **NH**, encabezado de red; **LH**, encabezado de enlace; **PH**, encabezado de físico.

Esta jerarquía fue desarrollada para facilitar las intercomunicaciones de equipos para el procesamiento de datos al separar las responsabilidades de la red en siete capas diferentes. El concepto básico de las responsabilidades de cada capa es que individualmente agregan valor a los servicios proporcionados por el conjunto de capas diferentes, de esta manera para el nivel mas alto se ofrecerá el conjunto completo de los servicios necesarios para correr una aplicación de datos distribuida.
Existen varias ventajas de usar una arquitectura en capas para el modelo **OSI**. Las diferentes capas permiten que diversas computadoras se comuniquen en diferentes niveles. Además, conforme ocurren los avances tecnológicos, es mas fácil modificar el protocolo de una capa sin tener que modificar las demás capas, cada capas es esencialmente independiente de cada una de las otras capas; Por lo tanto muchas de las funciones realizadas en las capas inferiores se removieron completamente de las tareas de soflware para remplazarlas con hartware. La desventaja principal de la arquitectura de siete capas es la cantidad de sobrecarga requerida al agregar encabezados a la información que se transmite en las diferentes capas.
Los niveles 4, 5, 6 y 7 permiten que se comuniquen directamente las computadoras al **HOST (huésped)**; El host es un termino muy común en la interconexión de redes, donde una computadora se convierte en las huésped de otra al facilitar información, y al intercambiar los papeles de solicitud ésta se convierte en el host de aquella. Las tres capas inferiores se ocupan de la mecánica del movimiento de datos (a nivel de bit) de una maquina a otra.

1.- La capa física: Esta capa es el nivel mas bajo de la jerarquía y especifica los estándares físicos, eléctricos, funcionales y de procedimiento para ingresar a la red de comunicaciones. Las definiciones como niveles de voltaje máximo y mínimo e independencia entre circuitos a través de la interfaz serial 232C de EIA

2.- Capa de enlace de datos: Esta capa es responsable de las comunicaciones entre los nodos primario y secundario dentro de las red, esta capa proporciona un medio de activar, mantener y desactivar el enlace de datos, Pues esta misma facilita la trama final de la envolvente de información, y facilita el flujo ordenado de datos entre los nodos, a demás permite la detección y corrección de errores. Un ejemplo de los protocolos de enlace de datos, son las comunicaciones bisincronas(bisync) de IBM y el control de enlace de datos sincronos(SDLC).
Esta capa esta implementada en el device driver del sistema operativo y en la targeta de interfaz del servidor que esta conectado a la red. Esta capa tiene a su cargo los detalles de la comunicación en la parte física(hartware) así como garantizar la confiabilidad de esta. La capa de red le entrega a la capa de enlace la información llamados datagramas. Cada datagrma contiene el numero **IP(el cual es un numero de 32 bits)** de su destinatario, esta capa tiene las funciones de:

- Convertir los datagramas en tramas. Esto se debe a que las tarjetas de red requieren que la información que estas envíen estén encapsuladas en forma de tramas.
- Convertir el numero IP del destinatario en su dirección física, cuando un servidor desea enviar una trama a otro, es necesario que conozca la dirección física del servidor destinatario(cada tarjeta de red contiene una dirección física única, esto se debe a que á ese nivel las direcciones IP son significativas, la traducción del numero IP a direcciones físicas se realiza mediante el protocolo de resolución **CSLIP(Compressed SLIP, SLIP Comprimido)** ó **PPP(Point to Point Protocol)**, si se trata de una línea telefónica.
- Convertir de regreso las tramas recibidas en datagramas para entregarlas a la capa de enlace en el lado del receptor.

3.- Capa de red: La capa de red determina que configuración de red es la mas apropiada para la función que da la misma. Esta capa define también el mecanismo en el cual los mensajes se dividen en paquetes de datos y son enrrutados desde un nodo de salida a un nodo receptor dentro de la red de comunicaciones, su función principal es la entrega de paquetes(llamados datagramas) de un servidor fuente a un servidor destino, también implementa algoritmos para ruteo para evitar congestionamientos el las interconexión de redes(getways y ruteadores). Toda la información que se transmite son datagramas IP. Esta capa no verifica que un datagrama haya sido recibido ó volverlo a mandar en caso de existir algún error. El protocolo central de esta capa es IP, y se encarga de realizar las siguientes funciones únicamente:

- Recibe la capa de transporte de información a enviar(en paquetes llamados segmentos) que incluye la dirección IP del destinatario.
- Encapsula dichos segmentos de datagramas.
- Determina cual es la ruta que debe seguirse para entregar cada datagrama. IP solo es capaz de entregar paquetes a servidores físicamente conectados en la misma red, Asi, si se desea enviar un datagrama a otra red, será necesario que IP determine cual es el ruteador o getway al que debe enviarle la información. Una vez conocida la dirección del siguiente servidor a contactar, le entrega a la capa de enlace el datagrama que incluye la dirección IP de destino.
- Cuando el servidor recibe un datagrama, verifica si esta destinado para el, si es así, lo reensambla en segmentos y lo pasa a la capa de transporte, de lo contrario efectúa nuevamente la operación del punto anterior. El protocolo importante de esta capa, es el protocolo de mensajes **ICMP**(Internet Control Message Protocol), por lo cual es posible realizar la siguientes funciones:

    - Control de flojo: Evita que otro servidor envíe mas datagramas de los que el receptor puede procesar.
    - Detección de errores en las rutas que siguen los datagramas: En ocasiones, algunas rutas no están disponibles, y si alguien desea comunicarse, este se encarga de notificar el error.

4.- Capa de transporte: Esta capa controla la integridad de un extremo a otro del mensaje, la cual incluye mensaje de ruteo, segmentación y recuperación de errores. La capa de transporte es la capa mas alta en términos de comunicaciones. Las capas por encima de la capa de transporte no consideran los aspectos tecnológicos de las red. Las tres capas superiores se dirigen a las características de las aplicaciones en la red, mientras que las tres capas inferiores se orientan a la transferencia de mensajes. Por lo tanto la capa de transporte actúa como interfaz entre la red y las capas de sesión, pero además permite la comunicación directa del remitente a los destinatarios. Con ayuda de dos protocolos TCP, cuya función es permitir la comunicación libre de errores tipo de conexión; y **UDP**(User Datagrama Protocol), cuyo papel es permitir el uso directo de datagramas IP. Las funciones de TCP son las siguientes:

- Dividir la información que recibe de la capa de aplicación en segmentos que pasaran a la capa de red.
- Al enviar un segmento inicializa un reloj, en espera de una contraseña(indicando que el mensaje o archivo se recibió); si el reloj expira antes que esta ultima se reciba, reenvia el segmento suponiendo que el segmento se ha perdido.
- Cuando TCP envía el mensaje, envia al remitente una contraseña confirmando la recepción
- Implementa algoritmos algoritmos para verificar que la información recibida fue la misma que la enviada; en caso de que el segmento llegue dañado a su destino, se indica al remitente del hecho, y este untimo lo rrenvia.
- Puesto que IP no garantiza el orden de llegada a los segmentos que envia, TCP debe reordenarlos si es necesario.
- Implementa algoritmos de control de flujo.

TCP otorga a la capa de aplicación una comunicación libre de errores punto a punto(de fuente a destino) que aparenta ser orientada a conexión TCP, además, TCP define el nivel de direccionamiento llamado puerto, que permite distinguir entre diferentes conexiones que se estén efectuando simultaneamente. Cada puerto es identificado con un numero de 16 bits. Su uso es claramente ejemplificado por el modelo cliente/servidor(en el cual desarrollaremos nuestra aplicación), para que el cliente pueda conectarse con el servidor, es necesario que el primero sepa donde encontrar al segundo; Para resolver esta problema varios números de puertos esta reservados para algunas aplicaciones(correo electrónico, telnet, ftp, web, etc.) los números de puertos son asignados por **IANA(Internet Assigned Number Authority)**. Esta agencia reserva números a los servicios que puede ofrecer un servidor, por ejemplo el numero de puerto del servidor ftp, es el 21, el de telnet es 23, el de web es 80, en general los números de puerto entre 1 y 255 los asigna la IANA, un cliente de telnet sabe que para conectarse a un servidor, debe establecer una conexión TCP al puerto 23 de la maquina en cuestión.

5.- Capa de sesión: Esta se encarga de la disponibilidad de la red(es decir, almacenamiento del buffer y capacidad de almacenamiento). Las responsabilidades de la sesión incluyen procedimientos para ingreso y el abandono de la red, así como verificar la autenticidad del usuario. Una sesión es una condición temporal que existe cuando los datos están en el proceso de ser transferidos y no incluye procedimientos tales como el establecimiento de llamadas, instalación o procedimientos de desconexión.
En esta capa es donde se encuentran las aplicaciones utilizadas por el usuario, algunas aplicaciones son tan comunes que se decidió estandarizarlas, entre ellas se encuentra el acceso remoto(telnet y rlogin), transferencia de archivos(ftp), correo electrónico (SMTP), Web (HTTP), etc. Para una descripción detallada de los protocolos de aplicación de TCP/IP, ver la fig.1.2

6.- Capa de presentación: Esta se dirige a cualquier conversión necesaria de códigos o sintaxis para presentar los datos a la red en formato común para la comunicación. Las funciones de presentación incluyen el formato de archivos de datos, codificación(ASCII, EBCDIC, etc.), encriptación y desencritación, Esta capa también realiza la translación del conjunto de códigos y caracteres, además determinan también el mecanismo para el desplegado de los mensajes.

7.- Capa de aplicación: Es la capa mas alta de la jerarquía y es análoga al administrador general de la red. La capa de aplicaciones controla la secuencia de actividades dentro de una aplicación. Estas capa se comunica directamente con el programa de aplicación del usuario.

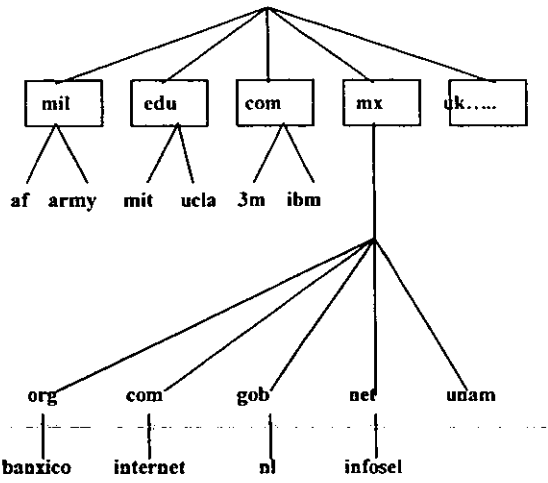
PROTOCOLOS ASINCRONOS

Dos de los protocolos de datos asincronos mas comúnmente utilizados, son el sistema de llamada selectiva(8 a 1/8 b 1) de Wertern Electric y el protocolo de enlace asincrono(83b) de IBM, estos protocolos tienen el mismo conjunto de procedimientos.
Los protocolos asincronos están orientados a caracteres. Es decir, los caracteres de control de enlace de datos único tal como el **fin de transmisión (EOT)** y **comienzo de texto(STX)** no importa en donde aparezca en una transmisión, garantizan la misma acción. Por ejemplo el carácter de fin de transmisión usado con ASII es 04H. No importa cuando 04H sea recibido por una secundaria, la LCU se borrara y se colocara en el modo de monitor de línea. Consecuentemente, debe tenerse cuidado al asegurar que las secuencias de bits para los caracteres de control de enlace de datos no ocurran dentro de un mensaje, al menos que estos sean intencionados para que realizan sus funciones de enlace de datos designada. La verificación de redundancia vertical(paridad) es el único tipo de detección de error utilizado con los protocolos asincronos, y la sustitución de símbolos y ARQ(retransmisión) se usan para la corrección de errores.
Con los protocolos asincronos, cada estación secundaria generalmente se limita a un solo par de terminal/impresora, a este arreglo de estación sele llama autónomo, con la configuración autónoma todos los mensajes transmitidos o recibidos en la terminal CTR también se escriben en la impresora de todas las transmisiones. Además de la línea de modo de monitoreo, una estación remota puede estar en cualquiera de los tres modos de operación: Transmisión, Recepción y Local. Una estación secundaria esta en el modo de transmisión siempre que se designe como maestro. En el modo de transmisión, la secundaria puede enviar mensajes formateados, una secundaria esta en el modo de recepción siempre que haya sido seleccionada por la primaria. En el modo de recepción, la secundaria puede recibir mensajes formateados de la primaria. Para que un operador de una terminal introduzca información en su terminal de computadora, la terminal debe estar en modo local. Una terminal puede colocarse en el modo local por medio de comandos de software enviados por la primaria, o el operador puede hacerlo directamente desde el teclado

1.2 I SISTEMA DE NOMBRES DE DOMINIO

En un servidor normalmente cada computadora tiene una dirección IP que la distingue de manera única en la red, esta direcciones son siempre traducida a nombres que nosotros podemos recordar fácilmente; a estos nombres seles llama nombres de dominio, este sistema(DNS Domain Name System) es una base de datos distribuida cuyo protocolo indica como convertir números IP a nombres de dominio de maquina en la red y viceversa, el acceso a la base de datos DNS se hace con las funciones GetHostByName y GetHostByAddr, las cuales tienen el nombre de la computadora y el numero IP. Las computadoras conectadas al servidor responden a dichas requisiciones. Normalmente toda computadora se comunica con uno o mas servidores, una de las características mas sobresalientes de DNS es que no existe una computadora que contenga información sobre todos los nombre de las direcciones IP de la red.
La tarea de convertir el nombre de dominio a número IP ó viceversa, esta a cargo de la capa de aplicación. DNS Tiene un esquema jerárquico representado en el sig. árbol.

Fig. 1.3



Cada nodo del árbol tiene una etiqueta, estas son la secuencia de los nodos necesarios para ir desde una hoja hasta el nodo raíz. A cada subfijo de un nombre sele conoce también como el nombre de dominio. Por ejemplo la computadora 129.97.208.5 tiene como nombre de dominio *csg.uwaterloo.ca*, cada nombre de dominio es administrado por una organización que tiene autoridad sobre sus subdominios, por ejemplo la universidad de waterloo es quien tiene autoridad sobre *awaterloo.ca* , esta puede crear cualquier subdominio en ella como así lo crea conveniente, los dominios de primer nivel son de tres tipos.

- **arpa** -. Dominio especial usado para conversiones de números IP a nombre y viceversa
- **generico** .- Fundamentalmente para el uso de EUA, esta constituido por los siguientes 7 dominios de primer nivel del árbol DNS : ***com*** que abrevia organizaciones comerciales, ***edu*** para instituciones educativas, ***gov***, para instituciones gubernamentales, ***int***, para organizaciones internas, ***mil***, para organizaciones militares, ***net***, para redes, ***org***, para otras organizaciones.
- países .- O dominios geográficos, basado en el nombre de los paises conectados a la red internet. Estas abreviaturas están especificadas en el estándar ISO. Por ejemplo : ***ae*** para Arabia, ***mx***, para México zx, para Zimbawe, y así sucesivamente para cada país.

La administración de nombres de dominio de primer nivel esta a cargo de InterNIC , en el caso de México, cuatro de los dominios genéricos forman el segundo nivel del árbol : ***com, gov, net***, y org. Por ejemplo el banco de México: *banxico.org.mx*; La empresa internet de México: *internet.com.mx*; El gobierno del estado de Nuevo León *nl.gob.mx*; La red infosel: *infosel.net.mx*. El dominio ***edu*** no existe, *v.gr. unam.mx, ipn.mx*. Existen algunas organizaciones que en lugar de registrarse con el NIC de México, deciden hacerlo en la de E.U. y por lo tanto tienen nombres de dominio como *sar.net, bravo.net*, etc. La administración del dominio mx esta acargo del NIC Mexicano nic.mx.
Una zona es el subárbol del árbol de DNS que se administra por separado, cada zona esta dividida por subzonas. Por ejemplo una universidad puede dividir la administración de la red por departamentos. La organización responsable de cada zona esta a cargo de proveer los servidores de nombres para dicha zona. Cada zona requiere de, por lo menos dos servidores de nombre que respondan preguntas sobre la zona, uno de ellos debe estar localizado fuera de la red local, con lo que aunque la red local no funcione, el DNS de la misma si lo haga.

Un servidor de nombre no necesita saber mas que la información de zona de la que es responsable, y las direcciones de los servidores de las subzonas que define y las de los servidores de la zona padre de la que depende. Otra propiedad importante del DNS es el cache de nombres, el cual ayuda disminuir el trafico de la red. Esto es cuando una computadora solicita el nombre de un servidor lo almacena de manera temporal.

DIRECCIONES IP, REDES, SUBREDES

NUMEROS IP

Como ya se dijo antes, el número IP o dirección IP, es una dirección empleada principalmente para distinguir de manera única a una computadora en un servidor. Los números IP constan de 32 bits, normalmente expresados con cuatro números decimales separados por puntos. Cada uno de estos números representan un byte de número IP; Por ejemplo: el número IP en notación punto decimal 255.97.18.1, representa al número IP 11111111 01100001 00111010 0000001, en notación binaria. Un número IP es compuesto de 3 partes:

Identificador de tipo clase.
Número de la red de servidores.
Número de computadora dentro de la red local.

TIPO DE CLASE

El tipo de clase permite identificar cuantos bits se utilizan para el número de la red de servidores, y cuantos para identificar a cada computadora, si la dirección comienza con un cero, entonces se trata de una dirección de clase A, que reserva 7 bits para el numero de red y 24 para el de la pc, si inicia con los primeros bits igual a 10, entonces se trata de una dirección clase B, con 14 y 16 bits respectivamente para el número de red y de pc, finalmente si inicia con los tres primeros bits igual a 110, se trata de una red tipo C, que reserva 21 bits para el numero de red y ocho para pc.

Red Clase A

	7 bits	**24 bits**
0	identificador de red	identificados de computadora

Red Clase B

		14 bits	**16 bits**
1	0	identificador de red	identificados de computadora

Red Clase C

			21 bits	**8 bits**
1	1	0	identificador de red	identificados de computadora

La clase **"A"** es para redes que pueden tener hasta 16,777,214 computadoras, las redes clase **"B"**, pueden tener hasta 65,534 computadoras, mientras que las de clase **"C"** tienen menos de 254 computadoras, todo número IP debe pertenecer alguna de estas clases, el rango de los números IP que pertenece a cada clase es de:

Clase A	0.0.0.0 - 127.255.255.255
Clase B	128.0.0.0 - 91.155.255.255
Clase C	192.0.0.0 - 255.255.255.255

CONCLUSIONES

Cuando un usuario desde su computadora(conectada a un servidor ó a una red local), necesita enviar uno o mas archivos, se realizan un sin numero de pequeños procesos que tienen como objetivo transmitir la información deseada y asegurar que dicha transmisión se realice libre de errores; Durante esta transmisión se utilizan varios protocolos, el mas común: **TCP/IP(Transmissión Control Protocol / Internet Protocol)** En este se agrupan decenas de protocolos que implementan funciones a todos los niveles de capas OSI. El objetivo ha sido, describir los protocolos de TCP/IP, y conocer la manera de como interactuan entre ellos para crear lo que conocemos como TCP/IP, así como algunos conceptos básicos necesarios para la administración de redes TCP/IP.

La correspondencia entre las capas de TCP/IP y las capas OSI se han mostrado de manera clara. Un ejemplo que ilustra el uso de estos protocolos, sería cuando un usuario desea transmitir un archivo a un servidor local o remoto, el servidor local le pedirá a la capa de transporte que cree una conexión de punto a punto con el servidor remoto, a su vez la capa de transporte utilizara los servicios de la capa de red (que se encarga esencialmente de enviar y recibir paquetes de información como se vera mas adelante) una vez solicitados los servicios de red, esta misma capa utiliza los servicios de la capa de enlace, que se encarga de enviarlos a través de la red local ó de un ruteador la información de un servidor a otro, para hacerla disponible al usuario que hizo la solicitud. Un protocolo de una red de datos es un conjunto de reglas que gobiernan el intercambio ordenado de datos. Sé ha mencionado ya que la principal función de **TCP/IP**, es controlar el flujo de datos entre el programa de aplicaciones y las terminales remotas, entonces es conveniente decir también que la fuente de desarrollo para la interfaz, juega también un papel super importante para la extracción de datos. **TCP/IP** es el canal de transmisión, pero también necesitamos conocer el lenguaje que mantendrá la comunicación con los datos, este lenguaje será la forma en como queremos vestir nuestra información para que pueda viajar por **TCP/IP.** El lenguaje que utilizaremos en nuestro propósito es **SQL(Strutured Query Languaje)** puesto que es un lenguaje de consulta a bases de datos tales como: **ORACLE, SYBASE, INFORMIX** etc. Bases de datos con un alto grado de aceptación en la industria de los negocios por su alta capacidad de información, y de ello trata el siguiente tema.

Capitulo

STRUCTURED QUERY LANGUAJE

2.1 II SQL

INTRODUCCION

Hoy en día el mundo de los sistemas de información modernos requieren contar con equipos apropiados y potentes que soporten cantidades fuertes de información, brindando así, la seguridad, la eficiencia, y la rapidez de procesamiento que las necesidades requieren, por ello es la importancia de ilustrar de manera mas profunda los medios que manipulen cantidades masivas de información, es necesario también conocer que tipo de plataforma soporta las características de diseño apropiadas para nuestro sistema.

Los **RDBMS(Relacional Data Base Manipulation System)** actuales, requieren de **SQL** para poder hacer uso de sus productos contenidos en bases de datos, para mostrar el uso de un RDBMS citemos ORACLE e INFORMIX, cada uno tiene sus propias características y plataformas, lo común en estos es que ambos trabajan con herramientas SQL, eso permite que además de hacer consultas directas a través de querys(consultas) sobre la base de datos, exista la posibilidad también de obtener información de ambos RDBMS en una sola consulta, estos es posible gracias a que dentro de sus múltiples herramientas existen sublenguajes basados en el lenguaje de programación "C" que permiten un mejor uso de SQL en la obtención de información mas particular de la base de datos. Estas herramientas conocidas como: PRO*C y SQL/C de ORACLE e INFORMIX respectivamente, resultan de gran ventaja en el aprovechamiento de los recursos. Lo primero que debemos hacer es conocer las herramientas SQL, Para ello explicaremos cada uno de sus componentes.

2.2 11 CONCEPTOS BASICOS

SQL*PLUS

Es un canal de consultas a la base de datos de manera directa, sqlplus permite que el usuario pueda interactuar con la base de datos realizando las siguientes tareas por mencionar solo algunas.

- Crear tablas, actualizar tablas y crear vistas.
- Preguntar a la base de datos cualquier información.
- Formatear reportes profesionales.

BASE DE DATOS

Es una colección de archivos relacionados entre si, de la cual los usuarios pueden extraer información sin considerar las fronteras fisicas de los archivos, su finalidad es de servir a una ó más aplicaciones, los datos son independientes de los programas que los usan, estos datos por lo general son compartidos.

CARACTERISTICAS

- Controlan la redundancia de datos
- Mantiene la consistencia de datos
- Logran la integración de los datos
- Comparten los datos entre las diferentes aplicaciones
- Compartimiento de standares
- Tienen facilidad en el desarrollo de aplicaciones
- Uniforman controles de seguridad, privacia e integridad
- Mantienen una independencia entre datos y programas
- Reducen el mantenimiento a los programas

QUE ES UN DBMS

Un DBMS(Sistema Manejador de Bases de Datos) Es un sistema de manejo de bases de datos, esta constituido básicamente de un conjunto de datos y programas relacionados relacionados entre si. Los usos mas comunes son los siguiente

OBJETIVOS DE UN DBMS

- Almacenar, recuperar y modifica datos.
- Mantener la consistencia de los datos.
- Resolver problemas de concurrencia.
- Permitir una interfaz universal con los datos.
- Regular el acceso a los datos.
- Minimizar la redundancia de datos.
- Proteger los datos contra fallas del sistema.

TIPOS DE RDBMS(Relationa Data Base Management System)

JERARQUICOS

Los datos se representan como estructura de árbol, el árbol representa una jerarquía de registros de datos, el procesamiento es top-down (un padre, múltiples hijos). La característica sobresaliente de este modeló el la conexión uno a muchos, cada hijo sólo tiene un padre.

RED
Se pueden tener relaciones de un hijo a varios padres, en una red un hijo puede tener varios hijos y varios padres a la vez, los datos se representan como registros ligados formando un conjunto de datos interceptados, este modelo además permite cualquier conexión entre entidades(relaciones de muchos a muchos). En una red, un hijo puede tener varios padres y varios hijos a la vez.

RELACIONAL
Representación de datos en forma de tablas que se componen de renglones y columnas, las relaciones se modelan a través de columnas en común. Esta estructura esta basada en la representación de entidades mediante tablas. La ventaja de los sistemas relaciónales es de poder modificar la información sin la preocupación de especificar las combinaciones entre registros.
El sistema establece automáticamente todas las relaciones posibles; Originado con el modelo de IBM (E.F.Codd,1970); ORACLE V2 Primer RDBMS comercial (1979).

CARACTERISTICAS DE UN RDBMS

- Representación de datos en forma de tablas
- Lenguaje de 4a Generación (4GL)
- Capacidades relaciónales completas
- Flexibilidad
- Diccionario de datos integrado

Es importante aclarar, que este proyecto esta basado por requerimientos necesarios en un diseño relacional

ENFOQUE RELACIONAL

En una base de datos relaciónal, la información esta organizada en forma de tablas como se muestra en la figura 1.1, DENON es el nombre de la tabla.

Las categorías de información están listadas a lo largo de la parte superior de cada tabla.**(EMPNO,ENAME JOB , MGR , HIREDATE , SAL , COMM , DEPTNO)**

Los casos individuales están listados hacía abajo.(cada categoría es una columna).En esta forma, se puede visualizar, entender y utilizar la información de una clara y sencilla.

Tabla DENON

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
8001	ANDRUS	CLERK	7902	17-DEC-80	800		20
8002	TEBAS	SALESMAN	7698	20-FEB-81	1,600	300	30
8003	ARGOS	SALESMAN	7698	22-FEB-81	1,250	500	30
8004	MENFIS	MANAGER	7839	02-APR-90	2,875		20
8005	ANDROS	SALESMAN	7698	28-SEP-81	1,250	1,400	30
8006	ORION	MANAGER	7839	01-MAY-93	2,850		30
8007	BABILON	MANAGER	7839	09-JUN-81	2,450		10
8008	TROYA	CLERK	7566	09-NOV-82	3,000		20
8009	BURNEO	CLERK	7902	17-DEC-80	4,000		10
8010	CORSEGA	SALESMAN	7698	08-SEP-81	1,500	0	30
8011	LESBOS	CLERK	7788	23-SEP-81	1,100		20
8012	ACRES	CLERK	7698	03-SEC-81	950		30
8013	ATENAS	ANALYST	7566	03-DEC-80	3,00		20
8014	MALTA	ANALYST	7782	23-JAN-82	3,000		10

TABLAS, RENGLONES Y COLUMNAS

- Cada columna contiene un tipo de información
- Cada renglón esta compuesto de varias columnas que contienen , cada una , un valor como ejemplo, la columna SAL en el renglón de MENFIS tiene un valor de 1,600

Tabla DENON

EMPNO	ENAME	JOBS	MGR	HIREDATE	SAL	COMM	DEPTNO
8001	ANDRUS	CLERK	9001	17-DEC-80	800		20
8002	TEBAS	SALESMAN	9002	20-FEB-81	1,600	300	30
8003	ARGOS	SALESMAN	9003	22-FEB-81	1,700	500	30
8004	MENFIS	MANAGER	9004	02-APR-90	1,600		20
8005	MARTIN	SALESMAN	9005	28-FEB-90	1,800	1400	30
8006	ANDROS	MANAGER	9006	13-MAR-72	1,700		30

TABLAS RELACIONADAS

Tabla RHAM

DEPTNO	DNAME	LOC
40	ENGINEER	BOSTON
30	DOCTOR	CHICAGO
20	RESEARCH	DALLAS
10	CHEMICAL	NEW YORK

Tabla DENON

EMPNO	ENAME	JOBS	MGR	HIREDATE	SAL	COMM	DEPTNO
8001	ANDRUS	CLERK	9001	17-DEC-80	800		20
8002	TEBAS	SALESMAN	9002	20-FEB-81	1,600	300	30
8003	ARGOS	SALESMAN	9003	22-FEB-81	1,700	500	30
8004	MENFIS	MANAGER	9004	02-APR-90	1,600		20
8005	MARTIN	SALESMAN	9005	28-FEB-90	1,800	1400	30
8006	ANDROS	MANAGER	9006	13-MAR-72	1,700		30

CARACTERISTICAS

- La información de una tabla esta relacionada con la información de otra tabla
- En ambas tablas, debe existir un campo en común
- La habilidad para relacionar información de una tabla con otra, permite organizar la información en unidades independientes y fáciles de manejar

- Se puede mantener la información de los empleados lógicamente independiente de linformación de los departamentos almacenándola en diferentes tablas

- Se puede relacionar información de 2 ó más tablas (hacer un join de tablas) por ejemplo, para encontrar la localidad (**LOC**) en la tabla **DEPT** de cualquier empleado en **DENON**

- Cuándo las tablas tienen columnas relaciónadas, los valores comunes en estas columnas permiten operaciones de join

COMANDOS DE SQL

Los comandos de SQL se utiliza para crear, almacenar, cambiar, recuperar y mantener la informaciónde una base de datos ORACLE. Un comando de SQL se guarda en una parte de la memoria SQL buffer, en donde se queda hasta que un nuevo comando es introducido. SQL tiene 17 comandos que inician con la siguientes sentencias.

ALTER	**AUDIT**	**COMMIT**	**COMMENT**	**CREATE**
DELETE	**DROP**	**GRANT**	**INSERT**	**LOCK**
NO AUDIT	**RENAME**	**REVOKE**	**SELECT**	**UPDATE**
VALIDATE				

COMANDOS DE SQL*PLUS

Además de los comandos estándar de SQL, el lenguaje SQL*PLUS de oracle incluye comandos adicionales, llamados comandos de SQL*PLUS. Éstos comandos no se almacenan en el buffer de SQL. Los comandos de SQL*plus se utilizan para generar reportes sofisticados, editar sentencias de SQL, proveer una facilidad de ayuda y mantener variables del sistema; Son los siguientes.

ACEPT	**APPEND**	**BREAK**	**BTITLE**	**CHANGE**	**UNDEFINE**
COLUMN	**COMPUTE**	**CONNECT**	**COPY**	**DEFINE**	**DESCRIBE**
EDIT	**EXIT**	**GET**	**HELP**	**JOST**	**DISCONNECT**
INPUT	**LIST**	**NEWPAGE**	**PAUSE**	**QUIT**	**DOCUMENT**
REMARK	**RUN**	**SAVE**	**SET**	**SHOW**	**SQLPLUS**
START	**TTITLE**	**TIMING**	**SPOOL**	**CLEAR**	

COMO ENTRAR A SQL*plus

Acceder al systema operativo unix a través de una cuenta, y un password

USERED: cuenta
PASSWORD: password

En el prompt del sistema operativo ejecutamos sqlplus

$ sqlplus usuario/password

SQLPLUS <return>

```
ENTER USERNAME : SCOTT <return>
ENTER PASSWORD:          <return>
```

COMO PEDIR AYUDA

Lista de todos los comandos de SQL Y SQL*plus

```
SQL> HELP COMMANDS
```

Información de comandos individuales.

```
SQL> HELP SELECT        SQL> HELP CREATE
```

-Para ver los tópicos de ayuda.

```
SQL> HELP TOPICS
```

DICCIONARIO DE DATOS

El diccionario de datos es un conjunto de tablas y vistas que contienen información descrita acerca de:

. Tablas
. Privilegios de acceso
. Otras cualidades de la base de datos

Las siguientes son las tablas mas utilizadas al acceder al diccionario de datos.

TAB

Una lista de tablas, vistas y sinónimos que usted ha creado.

DTAB

Tablas que componen en diccionario de datos.

COL

Una lista de las definiciones de columnas de las tablas que usted ha creado.

CATALOG

Lista de todas las tablas a las que usted tiene acceso

COMMANDO DESCRIBE

Si usted desea obtener la descripción de una tabla especifica, utilice el comando DESCRIBE.
La descripción va a contener.

NAME
Nombre de la columna.

NULL
Si se permiten valores nulos en esta columna.

TYPE
char (w)
Numeric (w,d)
Date
Row
w=longitud
d=número de decimales

```
SQL> DESCRIBE RHAM
```

Name	Null	Type
DEPTNO		NUMBER(2)
DNAME		CHAR(14)
LOC		CHAR(13)

SINTAXIS GENERAL DE UN QUERY

Utilizando SQL, usted puede consultar la base de datos de un número interminable de formas.A pesar de que SQL es muy flexible, su sintaxis es muy especifica. Es posible hacer una pregunta errónea en SQL y que resulte ya sea ningún resultado, o que la respuesta sea a una pregunta que no era la que quería preguntar. Las sintaxis general es muy simple:

```
SELECT.....
FROM....

[WHERE.....]

[ GROUP BY

  [HAVING.....] ]

[ORDER BY.....]
```

MANIPULACION DE COLUMNAS EN UNA TABLA

La información contenida en alguna tabla puede ser utilizada aplicando querys de consultas, un query es el siguiente.

Datos existentes en una tabla

```
SQL>SELECT * FROM DENON;
```

El asterisco significa que seleccione todo.

Tabla DENON

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
8001	ANDRUS	CLERK	7902	17-DEC-80	800		20
8002	TEBAS	SALESMAN	7698	20-FEB-81	1,600	300	30
8003	ARGOS	SALESMAN	7698	22-FEB-81	1,250	500	30
8004	MENFIS	MANAGER	7839	02-APR-90	2,875		20
8005	ANDROS	SALESMAN	7698	28-SEP-81	1,250	1,400	30
8006	ORION	MANAGER	7839	01-MAY-93	2,850		30
8007	BABILON	MANAGER	7839	09-JUN-81	2,450		10
8008	TROYA	CLERK	7566	09-NOV-82	3,000		20
8009	BURNEO	CLERK	7902	17-DEC-80	4,000		10
8010	CORSEGA	SALESMAN	7698	08-SEP-81	1,500	0	30
8011	LESBOS	CLERK	7788	23-SEP-81	1,100		20
8012	ACRES	CLERK	7698	03-SEC-81	950		30
8013	ATENAS	ANALYST	7566	03-DEC-80	3,00		20
8014	MALTA	ANALYST	7782	23-JAN-82	3,000		10

El query únicamente selecciona el nombre de la tabla, columnas y renglones con sus respectivos registros

SELECCION DE UNA COLUMNA ESPECIFICA

El siguiente query hace una selección especificando una sola columna, la columna es ENAME de la tabla DENON.

```
SQL>SELECT  ENAME  FROM  DENON;
```

Tabla DENON

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
8001	ANDRUS	CLERK	7902	17-DEC-80	800		20
8002	TEBAS	SALESMAN	7698	20-FEB-81	1,600	300	30
8003	ARGOS	SALESMAN	7698	22-FEB-81	1,250	500	30
8004	MENFIS	MANAGER	7839	02-APR-90	2,875		20
8005	ANDROS	SALESMAN	7698	28-SEP-81	1,250	1,400	30
8006	ORION	MANAGER	7839	01-MAY-93	2,850		30
8007	BABILON	MANAGER	7839	09-JUN-81	2,450		10
8008	TROYA	CLERK	7566	09-NOV-82	3,000		20
8009	BURNEO	CLERK	7902	17-DEC-80	4,000		10
8010	CORSEGA	SALESMAN	7698	08-SEP-81	1,500	0	30
8011	LESBOS	CLERK	7788	23-SEP-81	1,100		20
8012	ACRES	CLERK	7698	03-SEC-81	950		30
8013	ATENAS	ANALYST	7566	03-DEC-80	3,00		20
8014	MALTA	ANALYST	7782	23-JAN-82	3,000		10

ENAME
ANDRUS
TEBAS
ARGOS
MENFIS
ANDROS
ORION

```
BABILON
TROYA
BURNEO
CORSEGA
LESBOS
ACRES
ATENAS
MALTA
```

Es de gran ventaja utilizar únicamente los campos que necesitamos de nuestra tabla

SELECCION DE MULTIPLES COLUMNAS

```
SQL>SELECT EMPNO , ENAME , JOB
      FROM DENON;
```

Tabla DENON

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
8001	ANDRUS	CLERK	7902	17-DEC-80	800		20
8002	TEBAS	SALESMAN	7698	20-FEB-81	1,600	300	30
8003	ARGOS	SALESMAN	7698	22-FEB-81	1,250	500	30
8004	MENFIS	MANAGER	7839	02-APR-90	2,875		20
8005	ANDROS	SALESMAN	7698	28-SEP-81	1,250	1,400	30
8006	ORION	MANAGER	7839	01-MAY-93	2,850		30
8007	BABILON	MANAGER	7839	09-JUN-81	2,450		10
8008	TROYA	CLERK	7566	09-NOV-82	3,000		20
8009	BURNEO	CLERK	7902	17-DEC-80	4,000		10
8010	CORSEGA	SALESMAN	7698	08-SEP-81	1,500	0	30
8011	LESBOS	CLERK	7788	23-SEP-81	1,100		20
8012	ACRES	CLERK	7698	03-SEC-81	950		30
8013	ATENAS	ANALYST	7566	03-DEC-80	3,00		20
8014	MALTA	ANALYST	7782	23-JAN-82	3,000		10

Tabla DENON

EMPNO	ENAME	JOB
8001	ANDRUS	CLERK
8002	TEBAS	SALESMAN
8003	ARGOS	SALESMAN
8004	MENFIS	MANAGER
8005	ANDROS	SALESMAN
8006	ORION	MANAGER
8007	BABILON	MANAGER
8008	TROYA	CLERK
8009	BURNEO	CLERK
8010	CORSEGA	SALESMAN
8011	LESBOS	CLERK
8012	ACRES	CLERK
8013	ATENAS	ANALYST
8014	MALTA	ANALYST

Poner una coma entre cada nombre de columna, y aparecen todos los renglones para cada columna seleccionada.

SELECCION DE RENGLONES

Si usted no especifica una cláusula WHERE, se seleccionan todos los renglones. Especificando una cláusula WHERE, usted puede seleccionar ciertos renglones que cumplan con alguna condición.

```
SQL> SELECT ENAME
     FROM   DENON
     WHERE  DEPTNO=10;
```

EMPNO	ENAME	JOBS	MGR	HIREDATE	SAL	COMM	DEPTNO
8001	ANDRUS	CLERK	9001	17-DEC-80	800		20
8002	TEBAS	SALESMAN	9002	20-FEB-8	1,600	300	30
8003	ARGOS	SALESMAN	9003	22-FEB-81	1,700	500	30
8004	MENFIS	MANAGER	9004	02-APR-90	1,600		20
8005	MARTIN	SALESMAN	9005	28-FEB-90	1,800	1400	30
8006	ANDROS	MANAGER	9006	13-MAR-72	1,700		30
8007	ORION	MANAGER	9007	13-MAR-70	1,900		10
8008	BABILON	ANALIST	9008	09-NOV-90	1,600		30
8009	TROYA	PRESIDENT	9009	02-JAN-89	900		10
8010	BURNEO	SALESMAN	9010	23-SEP-82	1,400		40

ENAME
ORION
TROYA

ORDENADO DE RENGLONES

En el modelo relacional, los renglones no tienen un orden particular. El comando ORDER BY es la única como usted puede asegurar que los renglones van ser desplegados de acuerdo a cierto criterio.

```
SQL>SELECT ENAME
    FROM  DENON;
```

ENAME
ANDRUS
TEBAS
ARGOS
MENFIS
MARTIN
ANDROS
ORION
BABILON
TROYA
BURNEO

DESORDENADO

```
SQL>SELECT ENAME
    FROM  EMP
    ORDER  BY ENAME
```

ENAME
ANDRUS
ANDROS
ARGOS
BABILON
BURNEO
MARTIN
MENFIS
ORION
TEBAS
TROYA

ORDENADO

ORDEN DE RENGLONES CON CRITERIO MULTIPLE

Cuando se ordena por criterio múltiple. El default es orden ascendente (ASC) (de A a Z, no de Z a A). Para ordenar en forma descendente, se agrega la palabra DESC después del nombre de la columna.

```
SQL>SELECT * FROM RHAM
    ORDER BY DESC;
```

Tabla RHAM

DEPTNO	DNAME	LOC
40	ENGINEER	BOSTON
30	DOCTOR	CHICAGO
20	RESEARCH	DALLAS
10	CHEMICAL	NEW YORK

SELECCION DE RENGLONES ESPECIFICOS

UNA SOLA CONDICION

Con la cláusula **WHERE**, usted puede comparar el valor de una columna con:

```
WHERE ENAME ='ANDRUS';
```

Se debe usar apóstrofes en una constante de caracteres, además se debe respetar en dato de la manera en que se encuentra guardado, es diferente mayúsculas y minúsculas.
Las expresiones aritméticas no usan apóstrofes.

```
where deptnp = 20
```

Los comandos de SQL pueden ser escritos con mayúsculas o minúsculas, en este caso el comando WHERE también puede ser escrito con minúsculas, así como el nombre de la columna.

OPERADORES LÓGICOS

igual a	=	desigual	!= ó <>	mayor que	>
mayor ó igual	>=	menor que	<	menor ó igual	<=

OTROS OPERADORES

Igual a cualquier elemento de la siguiente lista **IN**(lista).

Mayor ó igual a un valor y menor ó igual que otro **BETWEEN** menor **AND** mayor.

Coincide con un patrón, una cadena de 0 ó más caracteres ó también con una cadena de un carácter.
LIKE %

No aplicable / valor inexistente. **IS NULL**

Negados de algunos operadores **NOT** (ejem. **NOT IN , IS NOT NULL**).

LISTAS DE VALORES
IN y NOT IN

El operador **IN** le permite seleccionar valores que coincidan con uno de los valores en la lista (mediante una evaluación **OR**).

Ejemplo:

Que empleados son , ya sea , CLERKS o ANALYST ?

```
SQL> SELECT ENAME , JOB
     FROM DENON
     WHERE JOB IN('CLERK','ANALYST');
```

ENAME	JOB
ANDRUS	CLERK
BABILON	ANALYST
CORSEGA	CLERK
LESBOS	CLERK
ACRES	ANALYST
ATENAS	CLERK

El operador **NOT IN** le permite seleccionar los renglones que no caen en la lista.

Ejemplo:

```
SQL> SELECT ENAME , JOB
     FROM DENON
     WHERE JOB NOT IN('CLERK','ANALYST');
```

RANGO DE VALORES
BETWEEN y NOT BETWEEN

El operador **BETWEEN** le permite seleccionar renglones que contengan valores dentro de un rango.

Ejemplo:

Que empleados ganan entre $ 1,000 y $ 1,900 ?

```
SQL> SELECT ENAME , JOB , SAL
     FROM DENON
     WHERE BETWEEN 1,000 AND 1,900;
```

Contrariamente, el operador **NOT BETWEEN** selecciona renglones fuera de un rango.

```
SQL> SELECT ENAME , JOB
     FROM  DENON
     SAL  NOT  BETWEEN 1,000  AND  1,900;
```

BUSQUEDA DE PATRONES
LIKE y NOT LIKE

Para buscar una cadena de caracteres, se utiliza el operador **LIKE** en la cláusula **WHERE**.

Ejemplo:

Listar a todos los empleados cuyos nombres comiencen con "A".

```
SQL> SELECT  ENAME , DEPTNO
     FROM  DENON
     WHERE  ENAME  LIKE 'A%';
```

Lista todos los empleados cuyos nombres terminen con "S"

```
SQL> SELECT  ENAME , DEPTNO
     FROM  DENON
     WHERE  ENAME LIKE '%S';
```

Lista todos los empleados cuyos nombres empiecen con "T" seguida por exactamente 3 caracteres

```
SQL> SELECT  ENAME  DEPTNO
     FROM  DENON
     WHERE ENAME  LIKE 'T---';
```

Observe que el símbolo "%" va a coincidir con una cadena de cualquier longitud, mientras que coincide con un sólo carácter.

Se utiliza **NOT LIKE** para seleccionar renglones que no coincidan con un patrón.

Lista a todos los empleados cuyos puestos no empiecen con SALES.

```
SQL> SELECT  ENAME , JOB
     FROM  DENON
     WHERE  JOB  NOT  LIKE 'SALES%';
```

VALORES NULOS
IS NULL e IS NOT NULL

Un valor NULO es una columna, no es lo mismo que un cero. Cero es un número, **NULL** no es un número. **NULL** significa que el valor es desconocido, faltante o no aplicable; no debe ser tratado como cero. Para buscar valores NULOS.

Lista a todos los empleados que no son candidatos para recibir comisión (cuyo valor en la comisión se NULO).

```
SQL> SELECT ENAME , JOB
       FROM  DENON
       WHERE COMM  IS  NULL;
```

Para buscar valores NO-NULOS:

Lista a todos los empleados que son candidatos para recibir comisión.

```
SQL> SELECT ENAME , JOB
       FROM  DENON
       WHERE COMM IS  NOT NULL;
```

Nota: Los valores NULOS no consumen espacio de almacenamiento en la base de datos ORACLE.

CONDICIONES MULTIPLES

La cláusula WHERE puede calificar y seleccionar renglones de las tablas especificando más de una condición de búsqueda.

Condición única:

```
SQL> SELECT  ENAME  , JOB
       FROM   DENON
       WHERE  RHAM = 20;
```

ENAME	JOB
ANDRUS	CLERK
MENFIS	MANGER

Condición múltiple.

```
SQL> SELECT ENAME,JOB
       FROM  DENON
       WHERE DEPTNO=30
       AND JOB !='CLERK';
```

ENAME	JOB
TEBAS	SALESMAN
ARGOS	SALESMAN
MARTIN	SALESMAN
ANDROS	MANGER
BABILON	ANALYST

El operador **AND** agrega criterio adicional que las columnas seleccionadas deben cumplir. En contraste, el operador **OR** especifica selección de renglones que cumplan cualquiera de las condiciones.

```
SQL>SELECT ENAME, JOB
     FROM DENON
     WHERE DEPTNO=30
      AND JOB != 'CLERK';
```

ENAME	JOB
TEBAS	SALESMAN
ARGOS	SALESMAN
MARTIN	SALESMAN
ANDROS	MANGER
BURNEO	SALESMAN

```
SQL>SELECT ENAME,JOB
     FROM DENON
      WHERE DEPTNO=30
       OR JOB !='CLERK';
```

ENAME	JOB
ANDRUS	CLERK
TEBAS	SALESMAN
ARGOS	SALESMAN
MENFIS	MANAGER
ANDROS	MANAGER
ORION	MANAGER
BABILON	ANALYST
TROYA	PRESIDENT
BURNEO	SALESMAN
CORSEGA	CLERK
ACRES	ANALYST
ATENAS	CLERK

PRECEDENCIA DE OPERADORES MULTIPLES

Se puede utilizar **AND** Y **OR** en una misma cláusula **WHERE**. Sé debe usar () para establecer precedencia. Orden de evaluación si no se especifica n ()

```
Primero  AND
Después  OR
```

Ejemplo:

```
WHERE DEPTNO  = 30
AND   JOB     ='SALESMAN'
OR    SAL     > 1500
```

Esto se refiere a los vendedores del departamento 30 ó cualquier empleados que gana más de 1500, no se refiere a cualquier persona del departamento 30 que sea vendedor o que gane mas de 1500.

INTRODUCCION A EXPRESIONES

Sé puede utilizar SQL como una calculadora para expresar valores con operadores aritméticos.

+	suma
-	resta
*	multiplicación
/	división

-Éstos operadores pueden ser utilizados en cualquiera de las siguientes cláusulas

SELECT
WHERE
ORDER BY

y otras cláusulas que se verán más adelante.

Nota : No tiene ningún sentido utilizar operadores aritméticos en la cláusula FROM.

EXPRESIONES NUMERICAS

Usted puede utilizar más de una expresión aritmética en el mismo query.

Quien gana más del 05% de su salario?

```
SQL>SELECT,SAL,COMM,COMM/SAL
    FROM DENON
    WHERE COMM>.05*SAL
    ORDER BY COMM/SAL DESC;
```

ENAME	SAL	COMM	COMM/SAL

EXPRESIONES NUMERICAS CON OPERADORES MULTIPLES

Es posible anidar varias expresiones. A continuación se presenta el orden de evaluación:

Se evalúa primero:

Multiplicación	*
División	/

Después se evalúa:

Suma	+
Resta	-

La evaluación de expresiones se efectúa de izquierda a derecha en donde el orden de evaluación anterior no se aplique. Sé puede controlar el orden de evaluación utilizando ().

Ejemplo:

12*(SAL+COMM) != 12*SAL+COMM

USO DE EXPRESIONES ARITMETICAS EN FECHAS

Sé pueden utilizar expresiones para especificar valores que involucren datos tipo FECHA, Esta aritmética utiliza los días como unidad básica.

Es posible sumar y restar fechas.

Algunos ejemplos utilizando suma de fechas:

SUMAR DOS DIAS AL 06-MAR-92

06-MAR-92 + 2 = 08-MAR-92

SUMAR 2 HORAS AL 06-MAR-92

06-MAR-92 + 2/24 = 06-MAR-92 Y 2 Horas

SUMAR 15 SEGUNDOS AL 06-MAR-92

06-MAR-93 + 15/(24*60*60) = 06-MAR-92 Y 15 SEG.

USO DE ALIAS EN COLUMNAS

El encabezado de las columnas que se despliega normalmente refleja el nombre de las columnas que se especifican al crear la tabla.
Se puede cambiar el encabezado de la columna que se despliega especificando un ALIAS de columna en la cláusula **SELECT**.
Para especificar un alias, se debe dejar un espacio en blanco después del nombre de la columna y después el ALIAS que se quiera colocar.

```
SQL>SELECT ENAME EMPLOYEE
     FROM DENON
     WHERE DEPTNO = 10;
```

EMPLOYEE
ORION
TROYA
ATENAS

ALIAS DE COLUMNAS Y EXPRESIONES

Si una cláusula **SELECT** tiene una expresión aritmética, esta expresión se utiliza como el encabezado de la columna. Un ALIAS se puede utilizar para renombrar temporalmente una columna calculada para hacer que los resultados de un query sean más legibles.

Ejemplo:

Quien gana comisiones que representan más del 5% de su salario. ?

```
SQL>SELECT ENAME,SAL,COMM,COMM/SAL,"C/S RATIO"
     FROM   DENON
     WHERE  COMM>.05*SAL
     ORDER BY COMM/SAL  DESC;
```

ENAME	SAL	COMM	C/S RATIO
MARTIN	1,250	1,400	1.12
WARD	1,250	500	0.4
ALLEN	1,600	300	0.1875

Noté que la expresión, no el ALIAS es la que se referencia en la cláusula **ORDER BY**. Esta regla se aplica también para todas las otras cláusulas en el **SELECT**.

Noté también que un ALIAS que contenga caracteres especiales como espacios ó / debe ponerse entre comillas.
El ALIAS afecta únicamente el **SELECT** en que se usa, no afecta otros querys.

AMBIENTE PARA ADICION DE SENTENCIAS DE SQL

BUFFER DE SQL

Este buffer contiene el comando activo de SQL, hasta que se introduce otro comando de SQL ó termina la sesión (exit).

Comandos de adición SQL*plus:

LIST o L Despliega el contenido del buffer de SQL

LIST 4 Lista la cuarta línea del comando SQL que esté activo

CHANGE o C Cambia la primera ocurrencia de cierto texto en una línea
Ejemplo :

```
C / (...) / ('ANALYST')
        C/'SCOTT/''ANDRUS'
```

INPUT o I Agrega una Línea al comando de SQL

APPEND o A Agrega texto a una línea.

DEL Borra una linea.

RUN Lista y corre las instrucciones que se encuentran en el buffer.

/ Corre la instrucción que se encuentra en el buffer.

EDIT Escribe el comando del buffer a un archivo de texto del sistema operativo y llama al editor del sistema para poder hacerle modificaciones.

COMO INSERTAR DATOS EN UNA TABLA

```
SQL>INSERT INTO DEPT (DNAME,DEPTNO)
    VALUES ('CHEMICAL',10);
```

Tabla RHAM

DEPTNO	DNAME	LOC
10	CHEMICAL	

Sé listan los nombres de las columnas en el **INSERT** para:
- Insertar datos solamente en algunas columnas de la tabla.
- Introducir los datos en alguna secuencia especifica.

Los valores se deben en el orden especificado en la cláusula **INSERT**.

INSERTAR RENGLONES DE OTRA TABLA

Sé puede utilizar el comando **INSERT** con un query para seleccionar renglones de una tabla e insertarlos en otra.

Él query sustituye la cláusula **VALUES**.

```
SQL>INSERT INTO DENON(EMPNO,ENAME,DEPTNO)
    SELECT ID,NAME,DEPARTMENT
    FROM   OLD_DENON
    WHERE  DEPARTAMENT IN (10,20,30,40);
```

USO DE PARAMETROS EN EL COMANDO INSERT

Un comando insert puede contener parámetros representando valores que van a ser provistos por el usuario al momento de correr el comando.

Cada parámetro consiste de un & seguido por el nombre de la columna.

```
SQL> INSERT INTO RHAM
     VALUES (&DEPTNO,&DNAME,&LOC)
```

La ejecución de este comando hace que SQL*plus pida al usuario los valores para cada uno del los parámetros.

La ejecución repetida del comando permite al usuario insertar varios renglones a una tabla rápidamente y en forma interactiva.

No es necesario poner apóstrofes en datos tipo **CHAR, DATE** si se ponen en el parámetro.

```
SQL> INSERT INTO RHAM
     VALUES (&DEPTNO,'&DNAME','&LOC');
```

INSERSIÓN DE VALORES

Si no se incluye una columna de la tabla en la cláusula **INSERT**, el valor de esa columna queda como NULO por default. Sé puede especificar **NULL** en la cláusula **VALUES** (a menos que **NO NULL** esté especificado para esa columna.

```
SQL> INSERT INTO RHAM
     VALUES (50,'EDUCATION',NULL);
```

INSERTAR VALORES DE TIPO DATE

Formato default para introducir fecha (valores tipo **DATE**).

'DD-MON-YY'

Ejemplo:

Introducir un nuevo empleado llamado 'STONE'

```
SQL> INSERT INTO DENON(EMPNO,ENAME,HIREDATE)
     VALUES (7963,'STONE','07-APRI-87);
```

Para introducir automáticamente la fecha y la hora actual.

SYSDATE

Introducir a un nuevo empleado 'KOHN' que fue encontrado hoy.

```
SQL>INSERT INTO DENON(EMPNO,ENAME,HIREDATE)
    VALUES(7600,'KOHN',SYSDATE);
```

ACTUALIZACION DE DATOS

Promover a MARTIN a MANAGER (gerente)

```
SQL> UPDATE DENON
     SET JOB='MANAGER'
     WHERE ENAME ='MARTIN';
```

Tabla Denon

8001	ANDRUS	CLERK
8002	TEBAS	SALESMAN
8003	ARGOS	SALESMAN
8004	MENFIS	MANGER
8005	MARTIN	SALESMAN
8006	ANDROS	MANAGER
8007	ORION	MANAGER
8008	BABILON	ANALYST
8009	TROYA	PRESIDENT
8010	BURNEO	SALESMAN

TABLA ORIGINAL

Tabla Denon

8001	ANDRUS	CLERK
8002	TEBAS	SALESMAN
8003	ARGOS	SALESMAN
8004	MENFIS	MANAGER
8005	MARTIN	MANAGER
8006	ANDROS	MANAGER
8007	ORION	MANAGER
8008	BABILON	ANALYST
8009	TROYA	PRESIDENT
8010	BURNEO	SALESMAN

TABLA ACTUALIZADA

ACTUALIZACION DE MULTIPLES RENGLONES

Cambiar los puestos de todos vendedores (SALESMAN) por Market Rep

```
SQL>UPDATE DONON
     SET JOB='MARKET REP'
     WHERE JOB='SALESMAN';
```

Tabla Denon

EMPNO	ENAME	JOB
8001	ANDRUS	CLERK
8002	TEBAS	SALESMAN
8003	ARGOS	SALESMAN
8004	MENFIS	MANGER
8005	MARTIN	SALESMAN
8006	ANDROS	MANAGER
8007	ORION	MANAGER
8008	BABILON	ANALYST
8009	TROYA	PRESIDENT
8010	BURNEO	SALESMAN

TABLA ORIGINAL

Tabla Denon

EMPNO	ENAME	JOB
8001	ANDRUS	CLERK
8002	TEBAS	MARKET RET
8003	ARGOS	MARKET RET
8004	MENFIS	MANAGER
8005	MARTIN	MARKET RET
8006	ANDROS	MANAGER
8007	ORION	MANAGER
8008	BABILON	ANALYST
8009	TROYA	PRESIDENT
8010	BURNEO	MARKET RET

TABLA ACTUALIZADA

ACTUALIZANDO MULTIPLES COLUMNAS

Cambiar los puestos de todos los vendedores por MARKET RET
y transferibles al departamento 40.

Tabla denon

EMPNO	ENAME	DEPTNO
CLERK	ANDRUS	20
SALESMAN	TEBAS	30
SALESMAN	ARGOS	30
MANAGER	MENFIS	20
SALESMAN	MARTIN	30
MANAGER	ANDROS	30
MANAGER	ORION	10
ANALYST	BABILON	30
PRESIDENT	TROYA	10
SALESMAN	BURNEO	40

TABLA ORIGINAL

Tabla denon

EMPNO	ENAME	DEPTNO
CLERK	ANDRUS	20
SALESMAN	TEBAS	30
SALESMAN	ARGOS	30
MANAGER	MENFIS	20
SALESMAN	MARTIN	30
MANAGER	ANDROS	30
MANAGER	ORION	10
ANALYST	BABILON	30
PRESIDENT	TROYA	10
SALESMAN	MARKET RET	40

TABLA ACTUALIZADA

Sé pueden actualizar múltiples columnas especificando los cambios en la cláusula SET.

BORRADO DE RENGLONES

- No se pueden borrar renglones parciales si se desea hacer esto, actualice la columna por NULL.
- La cláusula WHERE determina los renglones a borrar de la tabla.
- Si omitimos la cláusula WHERE se borran todos los renglones de la tabla.

Ejemplo:

Martín renunció. Se debe remover la base de datos de la empresa.

```
SQL> DELETE FROM DENON
     WHERE  EMPNO =8005;
```

Tabla DENON

EMPNO	ENAME	JOBS	MGR	HIREDATE	SAL	COMM	DEPTNO
8001	ANDRUS	CLERK	9001	17-DEC-80	800		20
8002	TEBAS	SALESMAN	9002	20-FEB-8	1,600	300	30
8003	ARGOS	SALESMAN	9003	22-FEB-81	1,700	500	30
8004	MENFIS	MANAGER	9004	02-APR-90	1,600		20
8006	ANDROS	MANAGER	9006	13-MAR-72	1,700		30
8007	ORION	MANAGER	9007	13-MAR-70	1,900		10
8008	BABILON	ANALIST	9008	09-NOV-90	1,600		30
8009	TROYA	PRESIDENT	9009	02-JAN-89	900		10
8010	BURNEO	SALESMAN	9010	23-SEP-82	1,400		40

CONTROL DE LOS EFECTOS DE CAMBIOS

Inserciones (altas), borrados (bajas) y actualizaciones (cambios) a tablas no se hacen permanentes hasta que el trabajo es salvado en la base de datos con **COMMIT**

Hasta que el trabajo es salvado, únicamente el usuario que hizo los cambios los puede ver todos los demás usuarios ven los datos como estaban al momento del último **COMMIT**.

Un **COMMIT** puede ser implícito, explícito ó automático

COMMIT explícito
Se debe dar el comando de SQL **COMMIT** para que los cambios se hagan permanentes.

```
SQL> COMMIT;
```

COMMIT implícito
Los siguientes comandos de SQL causan un COMMIT implícito.

ALTER,AUDIT,COMMENT,CONNECT,CREATE
DISCONNECT,DROP,EXIT,GRANT,NOAUDIT,QUIT
REVOKE,RENAME.

COMMIT automático
Los cambios tienen efecto inmediatamente después de un **INSERT**, **UPDATE**, ó **DELETE** si el **AUTOCOMMIT** se encuentra habilitado.

Para ello se utiliza el comando SET de SQL*plus.

```
SQL> SET AUTOCOMMIT ON
```

Él comando **ROLLBACK** cancela todos los cambios pendientes regresando al estado en que estaba la información al momento del último **COMMIT**.

```
SQL> ROLLBACK
```

TRANSACCIONES LOGICAS

Todos los cambios a la base de datos entre un **COMMIT** y otro se conocen como una transacción.

Cuándo una transacción es interrumpida por un error serio, como una falla del sistema, se hace un **ROLLBACK** de toda la transacción.
Esto previene el error de que sólo una parte de la transacción lógica quede salvada en la base de datos.
Ejemplo:

- Retiró de fondos de una cuenta de ahorros seguido por un depósito de estos fondos a una cuenta de cheques.

- Crear un nuevo departamento en una compañía. La transacción consiste en actualizar el número de departamento en la tabla DENON e insertar un nuevo renglón en la tabla RHAM.

COMO CREAR UNA TABLA

```
SQL> CREATE TABLE ROCK
       (DEPTNO  NUMBER(2),
        DNAME   CHAR(14),
        LOC     CHAR(13));
```

```
Tabla rock

DEPTNO    DNAME    LOC
```

Cuándo se crea una tabla se debe especificar :

Nombre de la tabla
Nombres de las columnas
Tipos de valores de las columnas
Longitud máxima de las columnas

Él diccionario de datos se actualiza automáticamente

Una tabla puede tener hasta 254 columnas

REGLAS PARA NOMBRAR TABLAS Y COLUMNAS

RESTRICCIONES EN TABLAS.

- El primer carácter debe ser alfabético (A-Z ó a-z), pero todos se guardan como mayúsculas.
- Los caracteres subsiguientes pueden ser numéricos $,#, y - (no se permiten comas).
- Los nombres deben ser únicos
- El nombre de su tabla debe ser único dentro de su cuenta
- No puede utilizar palabras reservadas de ORACLE.
- Los nombres de las columnas deben ser únicos dentro de una tabla.
- Se pueden utilizar comillas

Si ponemos en nombre de la tabla entre comillas, no se aplican la reglas anteriores en cuanto a caracteres. El utilizar mayúsculas o minúsculas sólo hacen diferencia cuando usamos comillas. Todos los accesos subsecuentes a los objetos nombrados de esta forma, requieren también, del uso de comillas.

TIPOS DE DATOS MAS COMUNES

CHAR (n)
n= máxima longitud de las cadenas de caracteres, n puede ser cualquier longitud hasta 240 caracteres. No permite a n, el espacio de almacenamiento no se ve afectado por esto.

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

DATE
Consiste de la fecha más la hora del día.

LONG
Hasta 65,536 caracteres (64K) pueden almacenarse por campo.

ROW
Datos en binario .

PROHIBIR VALORES NULOS

A veces alguien se quiere asegurar de que una columna no tenga valores nulos .

Por ejemplo :

Puede querer evitar que se inserte un renglón a la tabla de empleados (DENON) si no se conoce el número de empleados (EMPNO).

Para prohibir que se inserten valores nulos en una columna, se debe poner la cláusula **NOT NULL** después de la información de la columna.

A partir de entonces, cada vez que se quiera insertar un renglón sin valor para la columna que se definió como **NOT NULL**, va a haber un mensaje de error y la operación va a fallar.

```
SQL> CREATE TABLE ROCK
       (DEPTNO NUMBER(2) NOT NULL,
       DNAME   CHAR(14),
       LOC     CHAR(13));
```

AGREGAR UNA NUEVA COLUMNA A LA TABLA

```
SQL> ALTER TABLE ROCK
       ADD   (HEADCNT NUMBER(3));
```

Tabla ROCK

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATION	MEXICO

TABLA ORIGINAL

Tabla ROCK

DEPTNO	DNAME	LOC	HEADCNT
10	ACCOUNTING	NEW YORK	
20	RESEARCH	DALLAS	
30	SALES	CHICAGO	
40	OPERATION	MEXICO	

TABLA ALTERADA

*Cuando una columna es agregada, todos sus renglones son NULL.

ALARGANDO UNA COLUMNA
ALTER, TABLE Y MODIFY

```
SQL> ALTER TABLE ROCK
     MODIFY (DNAME CHAR(20));
```

Tabla ROCK

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW
YORK		
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATION	MEXICO

TABLA ORIGINAL

Tabla ROCK

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATION	MEXICO

TABLA ALTERADA

- Sé utiliza ALTER para cambiar la longitud de la columna.
- No se puede reducir a menos que la columna este vacia.
- Tampoco se puede cambiar el tipo de la columna a menos que esté vacia.
- No se puede modificar la columna y hacerla NOT NULL a menos que todos los renglones tengan
  valores para esa columna.

VISTAS

Una vista es como una ventana a través de la cual se puede consultar ó modificar información de una tabla.

Una vista es una tabla virtual

- Se ve como una tabla, pero no existe como tal.
- Sus datos se derivan de las tablas.

A pesar de esto, no hay copia de los datos.

Una vista provee:

- Simplicidad para ver exactamente lo que necesita.
- Seguridad para prevenir que usuarios no autorizados vean ciertas columnas ó renglónes de la tablas.

UNA TABLA, MULTIPLES VISTAS

Sé puede tener muchas vista de la misma tabla.

Tabla DENON

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
8001	ANDRUS	CLERK	7902	17-DEC-80	800		20
8002	TEBAS	SALESMAN	7698	20-FEB-81	1,600	300	30
8003	ARGOS	SALESMAN	7698	22-FEB-81	1,250	500	30
8004	MENFIS	MANAGER	7839	02-APR-90	2,875		20
8005	ANDROS	SALESMAN	7698	28-SEP-81	1,250	1,400	30
8006	ORION	MANAGER	7839	01-MAY-93	2,850		30
8007	BABILON	MANAGER	7839	09-JUN-81	2,450		10
8008	TROYA	CLERK	7566	09-NOV-82	3,000		20
8009	BURNEO	CLERK	7902	17-DEC-80	4,000		10
8010	CORSEGA	SALESMAN	7698	08-SEP-81	1,500	0	30
8011	LESBOS	CLERK	7788	23-SEP-81	1,100		20

8012	ACRES	CLERK	7698	03-SEC-81	950	30
8013	ATENAS	ANALYST	7566	03-DEC-80	3,00	20
8014	MALTA	ANALYST	7782	23-JAN-82	3,000	10

ENAME	JOB
ANDRUS	CLERK
LESBOS	CLERK
ACRES	CLERK

VISTA DE "CLERKS"

ENAME	JOB	SAL
MENFIS	MANAGER	2,975
ANDROS	MANAGER	2,850
ORION	MANAGER	2,450

VISTA DE "MANAGERS"

COMO CREAR, NOMBRAR Y CONSULTAR VISTAS

Crear una VISTA es diferente que crear una tabla.

```
SQL > CREATE VIEW MANAGER AS
      SELECT ENAME,JOB,SAL
      FROM   DENON
      WHERE  JOB='MANAGER';
```

Sé puede utilizar cualquier **SELECT** que no contenga un **ORDER BY** cuando se crea una VISTA.

Para consultar una vista, se hace un **SELECT** como si la vista fuera una tabla.

```
SQL> SELECT * FROM MANAGER;
```

ENAME	JOB	SAL
MENFIS	MANAGER	2,975
ANDROS	MANAGER	2,850
ORION	MANAGER	2,450

CREAR VISTAS CON NOMBRES DE ALIAS DE COLUMNAS

A menos que se especifique lo contrario la vista hereda los nombres de las columnas de la tabla de la que se deriva.
Para dar a la vista nombres de columnas diferentes a los de la tabla base, se utiliza la siguiente sintaxis.

```
CREATE VIEW nombre _vista (alias , alias,....)
AS query;
```

```
SQL> CREATE VIEW  RIMBROS
     (PERSON,TITLE,SALRY)
     AS SELECT   ENAME,JOB,SAL
     FROM        DENON
     WHERE       DEPTNO=10;
```

Más acerca de las vista con : WITH CHECK OPTION

WITH CHECK OPTION especifica que las inserciones y actualizaciones hechas a la base de datos a partir de la vista, no pueden manipular datos que la vista no pueda seleccionar.

```
SQL> CREATE VIEW RODERICK AS
     SELECT ENAME ,JOB,SAL,DEPTNO
     FROM DENON
     WHERE DEPTNO= 20
     WITH CHECK OPTION;
```

El siguiente comando de actualización genera un error.

```
SQL> UPDATE RODERICK
     SET DEPTNO =30
     WHERE ENAME ='ARGOS'
```

COPIANDO TABLAS Y COLUMNAS

Razones para copiar tablas y vistas.

- Respaldar tablas y vistas originales
- Dar una copia a alguien más
- Hacer cambios tentativos
- Archivarlas antes de borrar

Para copiar una tabla

- Utilice el comando CREATE TABLE con la cláusula AS seguida por un subquery
- Cuando la tabla es copiada, la copia incluye la definición de las columnas u los datos.

BORRAR TABLAS Y VISTAS

Para borrar tablas

El comando de SQL es `DROP TABLE nombre_tabla`

Si la tabla tiene datos, éstos van a ser borrados permanentemente. Una vez que la tabla se borra, no se puede recuperar.

Para borrar una vista

El comando de SQL es `DROP VIEW nombre_vista`

Las tablas en las que se basa la vista no se alteran.

LIMITACIONES DE SQL PARA FORMATEAR REPORTES

Mostrar los empleados de cada departamento que ganan más de $2,000

```
SQL> SELECT DEPTNO,ENAME,SAL
     FROM  DENON
     WHERE SAL >2000;
```

Tabla DENON

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
8001	ANDRUS	CLERK	7902	17-DEC-80	800		20
8002	TEBAS	SALESMAN	7698	20-FEB-81	1,600	300	30
8003	ARGOS	SALESMAN	7698	22-FEB-81	1,250	500	30
8004	MENFIS	MANAGER	7839	02-APR-90	2,875		20
8005	ANDROS	SALESMAN	7698	28-SEP-81	1,250	1,400	30
8006	ORION	MANAGER	7839	01-MAY-93	2,850		30
8007	BABILON	MANAGER	7839	09-JUN-81	2,450		10
8008	TROYA	CLERK	7566	09-NOV-82	3,000		20
8009	BURNEO	CLERK	7902	17-DEC-80	4,000		10
8010	CORSEGA	SALESMAN	7698	08-SEP-81	1,500	0	30
8011	LESBOS	CLERK	7788	23-SEP-81	1,100		20
8012	ACRES	CLERK	7698	03-SEC-81	950		30
8013	ATENAS	ANALYST	7566	03-DEC-80	3,00		20
8014	MALTA	ANALYST	7782	23-JAN-82	3,000		10

DEPTNO	ENAME	SAL
20	TROYA	3,000
30	ORION	2,850
20	MENFIS	2,875
10	BABILON	4,450
10	BURNEO	4,000
20	ATENAS	3,000
10	MALTA	3,000

SQL muestra la información tal y como ésta en la tabla.

En contraste, SQL*plus le permite cambiar los formatos de sus reportes.

```
SQL>  COLUMN DEPTNO  HEADING   DEPARTEMENT

SQL>  COLUMN ENAME   HEADING   NAME

SQL>  COLUMN SAL     HEADING   SALARY

SQL>  COLUMN SAL      FORMAT   $99.999.00

SQL>   TTITLE   'SAMPLE REPORT for | HITECK CORP'

SQL>   BTITLE   'STRCTLY CONFIDENTIAL'

SQL>   BREAK ON DEPTNO

SQL>   COMPUTE SUM OF SAL ON DEPTNO
SQL>    RUN
```

No es necesario poner un punto y coma después de un comando de SQL*plus. Debe correrse un comando de SQL (un query) para ver los efectos de los comandos de formateo de SQL*plus.

ENCABEZADOS Y PIES DE PAGINA

```
SQL> TTITLE 'TITULO DE EJEMPLO' <return>

SQL> BTITLE RIGHT 'ESTRICTAMENTE CONFIDENCIAL' <return>
```

La fecha y el número de pagina aparecen automáticamente hasta arriba de cada página cuando utilizamos **TTITLE**

Ambos comandos de título quedan habilitados hasta que :

- Se defina otro título
- Se termina la sesión
- Se deshabilitan los títulos con los comandos

```
TTITLE OFF
BTITLE OFF
```

NOMBRES DE COLUMNAS

HEADING corresponde al encabezado de la columna

No es necesario poner apóstrofes si el encabezado es de una palabra

```
SQL> COLUMN ENAME HEADING 'EMPLOYEE | NAME'
```

No se necesita punto y coma

Si el encabezado tiene más de una palabra, se pone entre apóstrofes.

```
SQL> COLUMN ENAME HEADING 'EMPLOYEE | NAME'
```

-Puede cancelar la especificación de la columna limpiándola con el siguiente comando

```
SQL> COLUMN ENAME CLEAR
```

FORMATOS PARA COLUMNAS

```
SQL> COLUMN ENAME FORMAT A15

SQL> COLUMN SAL FORMAT $9,999,99

SQL> COLUMN LIKE SAL
```

La cláusula **LIKE** hace que una columna se vea idéntica a otra.

Copia el formato y el encabezado de la columna.

MANEJO DE CORTES

```
SQL> BREAK ON DEPTNO

SQL> SELECT   DEPTNO,ENAME
     FROM  DENON
     ORDER BY DEPTNO;
```

Tabla DENON

DEPTNO	ENAME
20	ANDRUS
30	TEBAS
	ARGOS
	MENFIS
	ANDROS
	ORION
10	BABILON
	TROYA
	BURNEO
	CORSEGA
	LESBOS
	ACRES
	ATENAS
	MALTA

- **BREAK** suprime repetición de valores
- **ORDER BY** es necesario para controlar los cortes.

Solamente un comando **BREAK** puede estar en efecto, pero puede hacer corte por más de una columna

SEPARANDO GRUPOS DE RENGLONES

```
SQL> BREAK ON SKIN 2


SQL> SELECT  DEPTNO,ENAME
     FROM  DENON
     ORDER BY DEPTNO;
```

Tabla DENON

DEPTNO	
10	BABILON
	BURNEO
	MALTA
20	ANDRUS

	MENFIS
	TROYA
	LESBOS
	ATENAS
30	TEBAS
	ARGOS
	ANDROS
	ORION
	CORSEGA
	ACRES

- Doblé espacio entre cada departamento.
- **CLERK BREAK** borra y deshabilita el comando **BREAK**.
- Puede hacer cortes por página con **BREAK ON PAGE**.
- Puede hacer cortes por reporte con **BREAK ON REPORT**.
- Hacer reportes con varias indicaciones como: **BREAK ON PAGE ON REPORT**

DESPLEGAR CALCULOS POR GRUPO

- Suma de salarios por cada departamento

```
SQL> BREAK ON DEPTNO SKIP 2

SQL> COMPUTE SUM OF SAL ON DEPTNO

SQL> SELECT DEPTNO,ENAME,SAL
     FROM DENO
     ORDER BY DEPTNO;
```

La cláusula SKIP permite distinguir mejor los renglones de cada departamento con su respectivos totales.

COMANDO COMPUTE

- COMPUTE AVG OF sal ON deptno
  COUNT número de valores no nulos
  MAX valor mayor
  MIN valor mínimo
  STD desviación estándar
  VAR varianza
  NUMBER número de línea.

- CLEAR COMPUTE
  El COMPUTE queda habilitado hasta que:
  -Sé deshabilita
  -Sé re especifica
  -Termina la sesión

Para deshabilitarlo se debe dar:

CLEAR COMPUTE

COMANDOS DE AMBIENTE DE SQL*plus

Los párametros de la sesión corriente de SQL*plus se pueden desplegar tecleando.

```
SQL> SHOW AUTOCOMMIT
```

SHOW ALL despliega todos los párametros.

- Para especificar valores de los párametros se usa el comando SET.

```
SQL> SET AUTOCOMMIT ON
```

Los comandos **SET** incluyen:

SET AUTOCOMMIT (OFF/ON/INMEDIATE)
ON o INM - habilita COMMIT automático de los cambios

SET ECHO (OFF/ON)
ON - Los comandos ejecutados desde un archivo de comandos serán desplegados en la terminal
OFF - Suprime el desplegado de los comandos.

SET FEEDBACK (OFF/ON)
ON - Los resultados del query serán seguidos por un mensaje indicando el número de registros
retornados por el query.
OFF - Sé suprimen los mensajes.

SET HEADING (OFF/ON)
ON - Sé despliegan los encabezados de las columnas en los reportes.
OFF - Sé suprimen los encabezados.

SET LINESIZE (n)
Fija el número de caracteres que SQL*plus despliega por línea, el default es 80.

SET PAGESIZE (n)
Fija el número de líneas por pagina.

SET PAUSE (OFF/ON/texto)
ON -Hace que el usuario deba presionar <return> antes de desplegar la siguiente pantalla de salida.
OFF -Suprime la espera.
texto -Especifica el mensaje que debe aparecer para dar <return>

SET BUFFER buffer
El buffer se vuelve activo . SQL es el buffer de default. El buffer de SQL puede contener únicamente un comando de SQL. Utilice otro buffer si quiere dar comandos de SQL*plus adicionalmente a los comandos de SQL.

Éstos parámetros de quedan activos para la sesión corriente de SQL*plus

Los parámetros frecuentemente utilizados se pueden colocar en el archivo LOGIN.SQL.

```
SET NULL
```

Poner "NO HAY DATOS " en aquellos empleados del departamento 30 que no ganen comisión.

```
SQL> SET NULL NO HAY DATOS
```

```
SQL> SELECT ENAME,COMM
     FROM DENON
     WHERE DEPTNO = 30;
```

Tabla DENON	
ENAME	COMM
TEBAS	300
ARGOS	500
ANDROS	1,400
CORSEGA	0
ANDRUS	NO HAY DATOS
MENFIS	NO HAY DATOS
ORION	NO HAY DATOS
BABILON	NO HAY DATOS
TROYA	NO HAY DATOS
BURNEO	NO HAY DATOS
LESBOS	NO HAY DATOS
ACRES	NO HAY DATOS
ATENAS	NO HAY DATOS
MALTA	NO HAY DATOS

El comando SET NULL de SQL se utiliza para marcar valores faltantes, el sustituto puede ser cualquier cadena.

SALVAR COMANDOS DE UN ARCHIVO

Para salvar a disco un comando de SQL se utiliza el comando SAVE

```
SAVE nambre_del_archivo
```

El sufijo de SQL es agregado por default al nombre del archivo que tenga un punto.

```
SQL> INPUT
     1 SELECT EMPTNO,ENAME,JOB
     2 FROM DENON
     3 WHERE JOB ='ANALYST';

SQL> SAVE REASERCH
```

Un archivo llamado REASERCH.SQL está ahora en su directorio.

COMO UTILIZAR EL PROCESADOR DE TEXTOS DEL SISTEMA OPERATIVO

EDIT Edita el contenido del buffer

Cualquier cambio hecho durante la edición se salva desde SQL*plus con el comando EDIT.

SQL> EDIT

EDIT edita el contenido dek fuffer

Para regresar a SQL*plus salga del editor del sistema operativo

Para editar el contenido de un archivo, se da en comando **EDIT** seguido por el nombre del archivo.

SQL> EDIT REASERCH

El sufijo .SQL se agrega al nombre del archivo (excepto en los casos en donde el nombre del archivo contiene un punto)

RECUPERACION DE COMANDOS ALMACENADOS

El comando GET trae el contenido de un archivo al buffer de SQL y lo despliega en la pantalla.

Para recuperar el archivo REASERCH dar el comando:

SQL> GET REASERCH

Él sufijo .SQL no se debe de especificar

Ahora que el comando está en el buffer de SQL, puede ser editado o ejecutado con el comando RUN.

FORMA DE CORRER COMANDOS ALMACENADOS

El comando **START** recupera el contenido de un archivo de comandos y los corre

- Para recuperar y correr el archivo REASERCH dar:

SQL> START REASERCH

Un archivo de comandos ejecutado con el comando **START** puede contener comandos de SQL*plus y uno o más comandos de SQL.

El subfijo de .SQL no deben especificarse

MANDAR LA SALIDA A UN ARCHIVO

Para guardar los resultados de un query en un archivo y desplegarlos, utilice le comando **SPOOL**.

```
SQL>  SPOOL  ramiery
```

Toda la información que se despliega en la pantalla se va a almacenar en el archivo ramiery.

A menos que el nombre del archivo tenga un punto, sele agrega un subfijo default al nombre para identificarlo como un archivo de salida. Este sufijo depende del sistema operativo donde se trabaje.

-Para detener el **SPOOL** a un archivo, sele da el comando:

```
SQL> SPOOL  OFF
```

Para imprimir los resultados de un quer y, se debe mandar a un archivo como se vio anteriormente con el comando **SPOOL** nombre_archivo. Después, en lugar de **SPOOL OFF**, se da:

```
SQL> SPOOL  OUT
```

SPOOL OUT cierra el archivo de salida e imprime el archivo en la impresora default del sistema.

PROPOSITO DE LAS FUNCIONES

Las funciones le permiten:

Modificar valores.

Cambiar valores, para crear nuevos valores.

Cambiar formatos de valores.

Sé pueden utilizar en cualquier tipo de query, incluyendo los mas complejos.

FUNCIONES

Cómo una expresión, una función puede ser utilizada en las siguientes cláusulas.

SELECT
WHERE
ORDER BY

Si no se utiliza como expresión, una función tiene el siguiente formato.

```
FUNCION(argumento)
```

También puede tener múltiples argumentos.

```
FUNACION( arg1 , arg2 , arg3 , argN)
```

El primer argumento de una función en SQL es siempre el valor al cual sele aplica la función.

```
SQL> SELECT  ENAME
     FROM  DENON
     WHERE  LENGTH(ENAME)=6;
```

Existen varios tipos de funciones clasificadas según su tipo de valor.

ARITMETICAS
CHAR
DATE

Clasificadas según los renglones que afectan

Funciones individuales: Cada renglón es evaluado por separado. La función se aplica a cada renglón de la tabla.

Funciones en grupo : Evalúan colectivamente conjuntos de renglones.

FUENCIONES DE TIPO CHAR MAS COMUNES

En total existen 15 funciones tipo **CHAR** son la siguientes.

INITCAP (ename)
pone en mayúsculas la primera letra de cada palabra

jack smith se vuelve Jack Smith
smith jones se vuelve Smith Jones

LENGTH(ename)
calcula en número de caracteres en la cadena.

Length de TEBAS = 5
Length de BABILON = 7

SUBSTR(job,1,4)
Lista cuatro caracteres comenzando con el primer carácter de la cadena

CLERK se vuelve CLER
SALESMAN se vuelve SALE

LOWER
Convierte todos los caracteres de la cadena a minúsculas.

UPPER
Convierte todos los caracteres de la cadena en mayúsculas.

LEAST
Regresa el valor de una serie de argumentos que esté primero en orden alfabético.

GREATEST
Regresa el valor, de una serie de argumentos que esté al final en orden alfabético.

FUNCIONES TIPO DATE

Las funciones tipo **DATE** se especifican en la misma forma que las funciones tipo **CHAR**, con el nombre de la función seguido por sus argumentos entre paréntesis.

ADD_MONTHS (hiredate,5)
Suma 5 meses a hiredate

MONTHS BETWEEN (sysdate , hiredate)
Calcula el número de meses entre hiredate y sysdate.

NEXT_DAY (hiredate , friday)
Encuentra la fecha del siguiente viernes a partir de la feche de contratación(hiredate).

Ejemplo

Mostrar qué salarios serán pagados en el departamento 20 el próximo viernes.

```
SQL> SELECT ENAME,SAL,NEXT_DAY,(SYSDATE,'FRIDAY)PROX_V
     FROM  DENON
     WHERE  DEPTNO =20;
```

Función DATE — Primer argumento — segundo argumento

Tabla DENON		
ENAME	SAL	PROX_V
ANDRUS	800	21-NOV-86
MENFIS	2,795	21-NOV-86
BABILON	3,000	21-NOV-86
CORSEGA	1,100	21-NOV-86
ACRES	1,100	21-NOV-8

Fecha del próximo viernes

FUNCION TO_CHAR

Las fechas son desplegadas en el formato default de ORACLE, a menos que se utilice la función **TO_CHAR**.

TO_CHAR (date, date picture)

La fecha va a ser representada como una cadena de caracteres de acuerdo al formato dado en al da en

```
SQL> SELECT ENAME,TO_CHAR(HIREDATE,'Dy Mon  dd, yyy') HIRED
          FROM  DENON
          WHERE  DEPTNO=10;
```

Tabla DENON	
ENAME	HIRED
ORION	Tue-jun-09-1981
TROYA	Tue-non-17-1981
ATENAS	Sat-jan-23- 1982

FUNCION TO_DATE

La representación interna de una fecha va desde el siglo hasta el segundo.

La función **TO_DATE** convierte una cadena de caracteres de una gran variedad de formatos a un dato tipo **DATE** de ORACLE.

TO_DATE(cadena de caracteres , date picture)

La cadena de caracteres va a ser convertida a una fecha ORACLE de acuerdo al formato especificado en el date picture.

```
SQL> INSERT INTO DENON(EMPNO,ENAME,HIREDATE)
       VALUES (7999,'SAMS',
        TO_DATE ('070387083000','MMDDYYHHMISS'));
```

DATE PICTURE

Cuándo utilice **DATE PICTURE** debe ponerlo entre apóstrofes.

Cada **DATE PICTURE** esta compuesto de ciertos componentes básicos.(mayúsculas, minúsculas, abreviaturas).

Día		
dd	número	12
dy	abreviado	fri
day	deletreado	friday
ddsdth	deletreado , ordinal	twelfth
Mes		
mm	número	03
mon	abreviado	mar
manth	deletreado	march
Año		
yy	año	95
yyyy	año y siglo	1995

FUNCIONES NUMERICAS

Las funciones numéricas se especifican de la misma forma que las de tipo **CHAR** y **DATE**, con el nombre de la función seguido por sus argumentos entre paréntesis. Utilizando funciones numéricas usted puede efectuar operaciones sobre un conjunto de valores de su tabla o vista.

- Cúales empleados del depto 20 tienen un número de empleado menor que el de su jefe.

```
SQL> SELECT ENAME , EMPNO , MGR
       LEAST (EMPNO,MGR) LOW NUM
       FROM  DENON
      WHERE DEPTNO = 20;
```

Tabla DENON

ENAME	EMPNO	MGR	LOWNUM
ANDRUS	7369	7902	7369
MENFIS	7566	7839	7566
BABILON	7788	7566	7566
CORSEGA	7876	7788	7788
ACRES	7802	7576	7566

Otras funciones numéricas:

ABS(COMM -SAL)
Retorna el valor absoluto de la diferencia entre **COMM** y **SAL**

GREATTEST(SAL,COMM)
Retorna el valor mayor entre SAL y COMM

ROUND (SAL , 0)
Redondea SAL al dólar más cercano(sin decimales)

SIGN (COMM-SAL)
Puede regresar 3 valores
Sing = -1 si COMM -SAL < 0
Sing = 0 si COMM-SAL = 0
Sing = +1 si COMM-SAL > 0

TRUNCATE (SAL,0)
Trunca SAL al dólar inmediato inferior.

FUNCION NVL

Permite un manejo apropiado de valores nulos (SI, NVL, cualquier número más un dato tipo **NULL** me da un resultado **NULL**.

NVL(arg1 , arg2)

En donde arg1 es el nombre de la columna, si arg1 no es **NULL**, **NVL** trae su valor, pero si arg1es **NULL**, **NVL** regresa el valor de arg2.

- Cuanto recibe, en total(salario más comisión) cada empleado del departamento 30?

```
SQL> SELECT ENAME,COMM+SAL
           NVL (COMM,0) + SAL
     FROM  DENON;
```

Tabla DENON		
ENAME	COMM+SAL	NVL(COMM,0)+SAL
TEBAS	1,900	1,900
ARGOS	1,750	1,750
MARTIN	2,650	2,650
ANDROS		2,850
BURNEO	1,500	1,500
LESBOS		950

FUNCIONES EN GRUPO

- Encontrar el total de comisiones pagadas a los empleados.

```
SQL> SELECT  SUM(COMM)
       FROM  DENON;
```

SUM(COMM)
2,200

En general las funciones de grupo, regresan un valor único para un conjunto de renglones pueden también aplicarse a cualquier valor numérico y algunos valores tipo **CHAR** y **DATE**.

CONSISTENCIAS DE SELECCIÓN

No se pueden seleccionar resultados individuales y de grupo en el mismo query.

```
SQL>  SELECT  ENAME , SUM(COMM)
        FROM DENON;

....ERROR ..NOT A SIGLE
    GROUP SET FUCTION
```

FUNCIONES DE GRUPO PARA CHAR, DATE o NUMERO

MIN, MAX, COUNT pueden ser utilizados con cualquier tipo de dato.

NIM CHAR

```
SQL> SELECT  MIN(ENAME)
       FROM   DENON:
```

MIN(ENAME)
ACRES

MIN DATE

```
SQL> SELECT  MIN(HIREDATE)
       FROM   DENON
```

MIN(HIREDATE)
03-DEC-80

```
SQL> SELECT  MIN(SAL)
     FROM    DENON
```

MIN NUMBER

MIN(SAL)
800

MAX y COUNT funcionan de la misma forma

COUNT DISTINCT

- Cuantos empleados tienen puestos ?

```
SQL> SELECT  COUNT(JOB)
     FROM   DENON
```

COUNT(JOB)
14

Cuantos puestos diferentes existen ?

```
SQL> SELECT COUNT(DISTINCT JOB)
     FROM  DENON
```

COUNT(DISTINCT JOB)
4

AVG ——→ Encontrar el salario promedio de los empleados.

```
SQL> SELECT  AVG(SAL)
     FROM   DENON
```

AVG(SAL)

VGA(SAL)
114

STDDEV ——→ Encontrar la desviación estándar de los salarios de los empleados.

```
SQL> SELECT  STDDEV(SAL)
     FROM  DENON;
```

STDDEV(SAL)
1,182,50

SUM ——————Encontrar el total de los salarios pagados.

```
SQL> SELECT SUM(SAL)
     FROM   DENON
```

SUM(SAL)
29,025,00

CLAUSULA GROUP BY

Se utiliza esta cláusula para definir múltiples grupos de renglones, cada miembro del grupo tiene por lomenos un valor en común, a demás se deben especificar las columnas que contienen estos valores comunes en la cláusula GROUP BY.

Cual es al salario total y el salario promedio para cada departamento.?

```
SQL> SELECT DEPTNO ,SUM(SAL) , AVG(SAL)
     FROM   DENON
     GROUP BY   DEPTNO;
```

DEPTNO	SUM(SAL)	AVG(SAL)
10	8,750,00	2,916,67
20	10,875,00	2,175,00
30	9,400.00	1,566,67

Note que un query con una cláusula GROUP BY regresa un renglón por grupo.

CRITERIOS MULTIPLES PARA AGRUPAR RENGLONES

Dependiendo de su pregunta, usted puede necesitar agrupar por más de una columna.

Cuantos empleados hay en cada departamento. ?

```
SQL> SELECT DEPTNO , COUNT(*)
     FROM   DENON
     GROUP BY   DEPTNO;
```

DEPTNO	COUNT(*)
10	3
20	3
30	6

En el ejemplo anterior se agrupa sólo por columna, en el siguiente se ve la agrupación por más de una columna.

Cuantos empleados hay, de cada puesto, en cada departamento. ?

```
SQL>  SELECT  DEPTNO , JOB , COUNT(*)
      FROM    DENON
      GROUP BY  DEPTNO , JOB;
```

DEPTNO	JOB	COUNT(*)
10	ANALYST	1
10	MANAGER	1
10	CLERK	1
20	CLERK	3
20	MANAGER	1
20	CLERK	3
20	CLERK	3
20	ANALYST	1
30	SALESMAN	4
30	SALESMAN	4
30	SALESMAN	4
30	MANAGER	1
30	SALESMAN	4
30	CLERK	1

SELECCIONAR GRUPOS ESPECIFICOS

La cláusula **HAVING** le permite seleccionar grupos que cumplan con ciertas condiciones.
Es análoga a la cláusula **WHERE**, pero sirve para propósitos diferentes, la cláusula **WHERE** ponecondiciones sobre la cláusula **SELECT**. La cláusula **HAVING** pone condiciones cobre la cláusula **GROUP BY**.

Que departamento tiene una nómina de más de 9,000 ? (no incluir comisiones).

```
SQL>  SELECT  DEPTNO , SUM(SAL)
      FROM   DENON
      GRUOP BY  DEPTNO
      HAVING    SUM(SAL)  > 9000;
```

DEPTNO	SUM(SAL)
20	10,875
30	9,400

A CONTINUACION SE PRESENTE UN QUERY UTILIZANDO TODAS ESTAS CLAUSULAS

Sin tomar en cuenta los CLERKS que departamento tiene una nómina de más de 8,000 (no incluir comisiones y listar los departamentos por orden de cantidad pagada).

```
SQL>  SELECT   DEPTNO , SUM(SAL)
      FROM     DENON
      WHERE  JOB!='CKERK'
      GROUP  BY   DEPTNO
      HAVING       SUM(SAL) >  8000
      ORDER  BY  SUM(SAL);
```

DEPTNO	SUM(SAL)
30	8,450
20	8,975

JOINS

Los **JOINS** se utilizan para combinar, en un mismo query, columnas de diferentes tablas.

Tabla DENON

EMPNO	ENAME	JOB	DEPTNO
8001	ANDRUS	CLERK	20
8002	TEBAS	SALESMAN	30
8003	ARGOS	SALESMAN	30
8004	MENFIS	MANAGER	20
8005	ANDROS	SALESMAN	30
8006	ORION	MANAGER	30
8007	BABILON	MANAGER	10
8008	TROYA	CLERK	20
8009	BURNEO	CLERK	10
8010	CORSEGA	SALESMAN	30
8011	LESBOS	CLERK	20
8012	ACRES	CLERK	30
8013	ATENAS	ANALYST	20
8014	MALTA	ANALYST	10

Tabla RHAM

DEPTNO	DNAME	LOC
40	ENGINEER	BOSTON
30	DOCTOR	CHICAGO
20	REASERCH	DALLAS
10	CHEMICAL	NEW YORK

Tabla DENON y Tabla RHAM

EMPNO	ENAME	JOB	DEPTNO	DNAME
8001	ANDRUS	CLERK	20	CHEMICAL
8002	TEBAS	SALESMAN	30	DOCTOR
8003	ARGOS	SALESMAN	30	DOCTOR
8004	MENFIS	MANAGER	20	CHEMICAL
8005	ANDROS	SALESMAN	30	DOCTOR
8006	ORION	MANAGER	30	DOCTOR
8007	BABILON	MANAGER	10	RESEARCH
8008	TROYA	CLERK	20	CHEMICAL
8009	BURNEO	CLERK	10	RESEARCH
8010	CORSEGA	SALESMAN	30	DOCTOR
8011	LESBOS	CLERK	20	CHEMICAL

8012	ACRES	CLERK	30	DOCTOR
8013	ATENAS	ANALYST	20	CHEMICAL
8014	MALTA	ANALYST	10	RESEARCH

COMO HACER UN JOIN SIMPLE

```
SQL> SELECT   EMPNO , ENAME , JOB
             DENON.DEPTNO , DNAME
     FROM    DENON , RHAM
     WHERE   DENON.DEPTNO = RHAM.DEPTNO;
```

Tabla DENON Y RHAM

TABLA

EMPNO	ENAME	JOB	DEPTNO	DNAME
8001	ANDRUS	CLERK	20	CHEMICAL
8002	TEBAS	SALESMAN	30	DOCTOR
8003	ARGOS	SALESMAN	30	DOCTOR
8004	MENFIS	MANAGER	20	CHEMICAL
8005	ANDROS	SALESMAN	30	DOCTOR
8006	ORION	MANAGER	30	DOCTOR
8007	BABILON	MANAGER	10	RESEARCH
8008	TROYA	CLERK	20	CHEMICAL
8009	BURNEO	CLERK	10	RESEARCH
8010	CORSEGA	SALESMAN	30	DOCTOR
8011	LESBOS	CLERK	20	CHEMICAL
8012	ACRES	CLERK	30	DOCTOR
8013	ATENAS	ANALYST	20	CHEMICAL
8014	MALTA	ANALYST	10	RESEARCH

JOINS EXTERNOS

```
SQL > SELECT ENAME , RHAM.DEPTNO , LOC
        FROM       DEONO , RHAM
      WHERE DENON.DEPTNO (+) = RHAM.DEPTNO;
```

ENAME	DEPTNO	LOC
BABILON	10	NEW YORK
BURNEO	10	NEW YORK
MALTA	10	NEW YORK
ANDRUS	20	DALLAS
MENFIS	20	DALLAS
TROYA	20	DALLAS
LESBOS	20	DALLAS
ATENAS	20	DALLAS
TEBAS	30	CHICAGO
ARGOS	30	CHICAGO
ANDROS	30	CHICAGO
ORION	30	CHICAGO
CORSEGA	30	CHICAGO
ACRES	40	BOSTON

Si existen valores en **RHAM.DEPTNO** que no tengan valores correspondientes en **DENON.DEPTNO** (como el depto 40), el simbolo de **JOIN** externo (+) causa que aparezca un valor NULO en la tabla relacionada.

JOINS DE TABLAS CONSIGO MISMAS

Cuando se quiere relacionar un renglón de la tabla con otro renglón de la misma tabla, se lleva a cabo un **SELFJOIN**

Determinar el nombre del jefe de cada empleado.

```
SQL> SELECT  WORKER.ENAME , MANAGER.ENAME , MANAGER
       FROM   DENON WORKER , DENON MANAGER
       WHERE   WORKER.MGR = MANAGER.EMPNO;
```

Se relaciona una tabla consigo misma en el WHERE, como si fueran dos tablas separadas. Es necesario usar un alias por lo menos para una ocurrencia de la tabla en la cláusula FROM para poder diferencia los nombres de las columnas según la tabla a la que pertenece.

JOINS DE NO EQUIVALENCIA

Si la condición de **JOIN** en la cláusula **WHERE** especifica una equivalencia (=) es un equijoin.

Una condición de **JOIN** con cualquier otro operador es un Non-Equijoin, para mostrar su función crearemos una tabla llamada RODERICK

Tabla RODERICK

GRADO	LOSAL	HISAL
1	700	1,200
2	1,201	1,400
3	1,401	2,200
4	2,001	3,000
5	3,001	9,999

Que salarios caen en el grado 3 ?

```
SQL> SELECT  ENAME , SAL
     FROM   DENON , RODERICK
     WHERE  GRADO = 3
     AND       SAL  BETWEEN LOSAL AND HISAL;
```

ENAME	SAL
TEBAS	1,600
CORSEGA	1,500

OPERADORES DE CONJUNTOS

Los operadores de conjunto cambian 2 ó más querys en un resultado

UNION unión de conjuntos
El resultado trae a los renglones del primer query eliminando renglones duplicados.

INTERSECT intersección de conjuntos
El resultado trae únicamente a los renglones que los dos querys tengan en común.

MINUS diferencia de conjuntos
El resultado trae a los renglones que son únicos para el primer query.

NUEVAS VISTAS

A continuación se presentan tres vistas de la tabla DENON que se utilizaran en los próximos ejemplos.

Vista CHEMICAL

ENAME	SAL	JOB
BABILON	2,450	MANAGER
BURNEO	4,000	CLERK
MALTA	1,300	CLERK

Vista RESEARCH

ENAME	SAL	JOB
TEBAS	1,600	SALESMAN
ARGOS	1,250	SALESMAN
ANDROS	1,250	SALESMAN
ORION	2,850	MANAGER
CORSEGA	1,500	SALESMAN
JAMES	950	CLERK

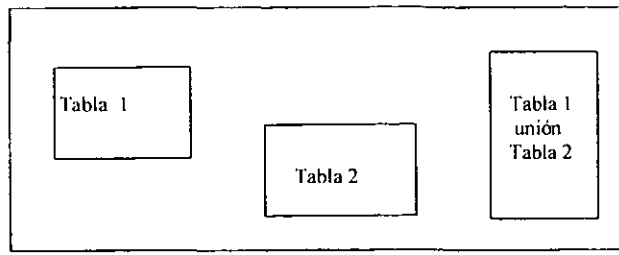
Vista DOCTOR

ENAME	SAL	JOB
ANDRUS	800	CLERK
MENFIS	2,975	MANAGER
TROYA	3,000	ANALYST
LESBOS	1,100	CLERK
ATENAS	3,000	ANALYST

OPERADOR UNION

UNION regresa todos los distintos valores seleccionados por cualquiera de los query a los que se aplica.

Cambia renglones de múltiples tablas y vistas



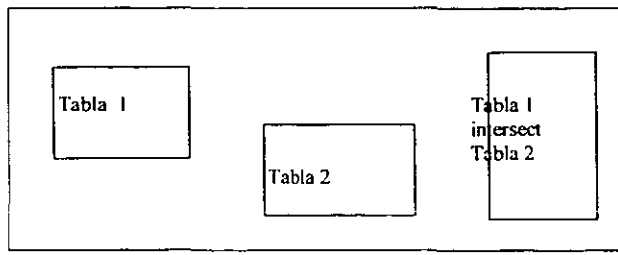
Quien gana más de 2,000 en todos los departamentos. ?

```
SQL>  SELECT   ENAME , SAL
      FROM          CHEMICAL
      WHERE    SAL> 2000

       UNION

      SELECT  ENAME , SAL
      FROM     RESEARCH
      WHERE    SAL> 2000

       UNION

      SELECT   ENAME , SAL
      FROM   DOCTOR
      WHERE   SAL > 2000;
```

ENAME	SAL
ORION	2,850
BABILON	2,450
ATENAS	3,000
MENFIS	2,975
BURNEO	4,000
TROYA	3.000

OPERADOR INTERSECT

INTERSECT Encuentra renglones comunes en múltiples tablas, las columnas deben ser del mismo tipo.



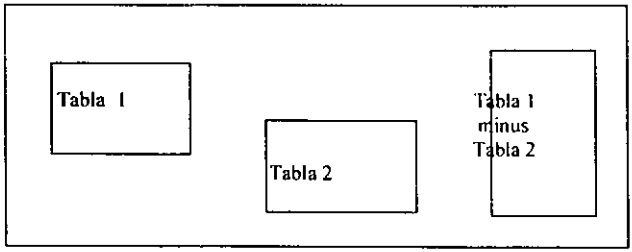
Que puestos tienen en común todos los departamentos. ?

```
SQL> SELECT JOB
     FROM  CHEMICAL

      INTERSECT

     SELECT  JOB
     FROM        RESEARCH

     INTERSECT

     SELECT  JOB
     FROM   DOCTOR;
```

JOB
CLERK
MANAGER

OPERADOR MINUS

MINUS Regresa todos los renglones que sean únicos para el primer query



Existen puestos en el departamento de CHEMICAL que no se encuentren el en departamento de RESEARCH.

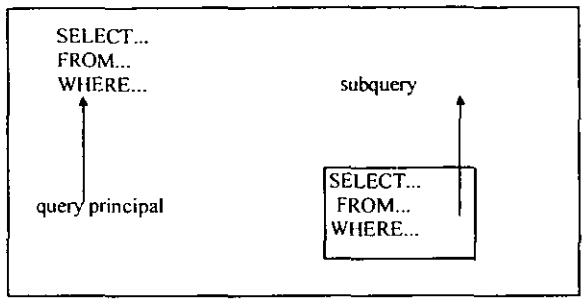
```
SQL> SELECT  JOB
     FROM  CHEMICAL

     MINUS

     SELECT  JOB
     FROM   RESEARCH;
```

CLERK
BURNEO

INTRODUCCION A SUBQUERYS

Un subquery (query anidado) es un query que se encuentra dentro de la cláusula **WHERE**

Los resultados del subquery se utilizan para resolver el query principal.



Ejemplo:

Que empleados trabajan en el mismo departamento que ANDRUS. ?

Estrategia:

1- Subquery
Encontrar el departamento en el que trabaja ANDRUS.

2- Query principal
Seleccionar todos los empleados que trabajan en ese departamento.

```
SQL> SELECT  ENAME , DEPTNO
     FROM    DENON
     WHERE  DEPTNO = (SELECT  DEPTNO
                      FROM    DENON
                      WHERE  ENAME='ANDRUS');
```

ENAME	DEPTNO
MENFIS	20
TROYA	20
LESBOS	20
ATENAS	20

Los subquerys pueden tener:

* El anidamiento de los subquerys pueden continuar identicamente.

* El subquery puede insertar tablas que no son utilizadas por el query principal.

* Un subquery no puede tener una cláusula **ORDER BY**

* Un subquery puede constituir una de las partes de cualquier operador relacional.

Que empleados tienen el mismo puesto que BABILON ó un salario mayor al de él.?

```
SQL> SELECT ENAME , JOB , SAL
     FROM  DENON
     WHERE JOB = ( SELECT JOB
                   FROM   DENON
                   WHERE  EANAME ='BABILON')

         OR  SAL>

                   (SELECT SAL
                    FROM  DENON
                    WHERE ENAME ='BABILON');
```

ENAME	JOB	SAL
MANFIS	MANAGER	1,975
ORION	MANAGER	2,850
BABILON	MANAGER	2,450
TROYA	ANALYST	3,000
BURNEO	CLERK	4,000
ATENAS	ANALYST	3,000

MULTIPLES TABLAS Y SUBQUERYS

Se puede extraer información de más de una tabla en un query que contenga subquerys.

Que empleados en NEW YORK tienen un salario mayor al de TROYA. ?

```
SQL> SELECT ENAME , JOB , SAL
     FROM  DENON , RHAM
     WHERE  LOC = 'NEW YORK'
     AND    DENON.RHAM = RHAM.DEPTNO
     AND    SAL >
                (SELECT SAL
                 FROM    DENON
                 WHERE ENAME ='TROYA');
```

ENAME	JOB	SAL
BURNEO	CLERK	4,000

FUNCIONES DE GRUPO EN SUBQUERYS

Si selecciona una columna regular y una función de grupo en el mismo **SELECT**, recibe un mensaje de error .

Cual fue el primer empleado contratado. ?

```
SQL> SELECT ENAME , MIN(HIREDATE)
     FROM  DENON;
```

```
..ERROR...NOT A SINGLE
GROUP SET FUNTION...
```

Como una alternativa , se puede colocar la función de grupo dentro del subquery y la columna regular en el query principal.

```
SQL> SELECT ENAME , HIREDATE
     FROM  DENON
     WHERE  HIREDATE = (SELECT MIN(HIREDATE)
                        FROM  DENON);
```

SEGURIDAD DE LOS DATOS

* Privilegios.

* Privilegios sobre tablas.

* Privilegios sobre vistas.

REVOKE

* Elimina cualquier tipo de privilegio.

PRIVILEGIOS DE SISTEMA

Existen tres niveles de privilegios del sistema.

DBA
Todos los privilegios.

RESOURCE
Privilegios de conexión y creación de tablas.

CONNECT
Privilegios de conexión, únicamente consultas.

Únicamente un **DBA** tiene autoridad para dar de alta un nuevo usuario.

```
SQL> GRANT CONNECT TO RODERICK
     IDENTIFIED BY LEOPARD;
```

Un usuario puede cambiar su propio password.

```
SQL> GRANT CONNECT TO RODERICK
     IDENTIFIED BY TIGER;
```

PRIVILEGIOS SOBRE TABLAS

Existen dos formas de dar permiso sobre tablas.

* Crear una tabla

* Obtener permisos sobre tablas creadas por otros usuarios. Este lo debe otorgar el dueño de la tabla, de la manera siguiente.

```
SQL> GRANT SELECT , INSERT , DELETE , UPDATE
     ON DENON
     TO RODERICK;
```

* Para dar a un usuario todos los privilegios sobre una tabla.

```
SQL> GRANT ALL
     ON DENON
     TO RODERICK;
```

SINONIMOS

Si otro usuario le ha dado privilegios de **SELECT** sobre alguna de sus tablas, usted la tiene que acceder de la siguiente forma.

```
SQL> SELECT * FROM RODERICK.DENON;
```

Nombre de usuario dueño de la tabla

Como una alternativa, se pueden crear sinónimos utilizando un nombre más significativo y simple. A continuación se muestra la forma de crear un sinónimo llamado TESLA sobre la tabla DENON del usuario TEBAS.

```
SQL> CREATE SYNONIM TESLA
     FOR TEBAS.DENON;
```

Ahora puede consultar la tabla TESLA como cualquier otra

Cuando se accede a la tabla de otro usuario , es posible que se reciba el mensaje
"Table or View does not exist"

Esto significa que no se tiene acceso a la tabla ó vista ó también que dicha tabla ó vista no existe.

También podemos dar a otros usuarios el derecho de otorgar privilegios sobre nuestras tablas.

```
SQL> GRANT ALL
     ON DENON
     TO RODERICK
     WICH GRANT OPTION;
```

COMO ELIMINAR PRIVILEGIOS DEL SISTEMA

Solo pueden ser borrados por el **DBA**

Privilegios sobre tablas.

Pueden eliminarse en cualquier momento

```
SQL> REVOKE INSERT
     ON DENON
     FROM RODERICK;
```

INDICES

INDEXAR TABLAS

El propósito de los INDICES es ayudar a que un **RDBMS**(Research Database Manager system)consulte tablas grandes de una manera más rápida. En lugar de leer toda la tabla, la recorre de manera eficiente.

Para crear un índice

```
SQL> CREATE INDEX DENON_ENAME
     ON DENON(ENAME);
```

Para borrar un índice.

```
SQL> DROP INDEX DENON_ENAME;
```

El uso de índices en únicamente para grandes tablas (por lo menos de 50 renglones).

Sé deben insertar datos antes de índexar.

Una tabla puede tener varios índices.

Generalmente se índexa la columna que identifica a los renglones en forma única (llave primaria).

El uso de índices no tiene ningún impacto sobre la sintaxis de SQL.

Los índices son actualizados automáticamente.

A demás de manejar el performance, se pueden utilizar índices para asegurar que cada valor de una columna sea único.

Ejemplo:

Asegurarse de que cada número de empleado aparezca solamente una vez en la tabla DENON.

```
SQL> CREATE UNIQUE INDEX DENON_EMPNO
     ON DENON(EMPNO);
```

Una vez creado el índice único, se obtendrá un mensaje de error si se trata de insertar ó actualizar un renglón con el mismo EMPNO por un renglón que ya existe.

OBSERVACIONES

El uso de **SQL** permite hacer consultas y modificaciones a la base de datos, por ello es importante conocer a fondo cada uno de sus comandos y tomar una buena decisión a la hora de resolver un problema, no obstante es necesario también conocer como es que se encuentra organizada nuestra información en una base de datos, pues de una buena organización dependerá el buen funcionamiento de una base de datos, es decir si no se tiene una buena *regla de validación* entre nuestros datos, esto puede ocasionar serios problemas en el funcionamiento interno de nuestra fuente de datos. Estas reglas de validación son conocidas por los diseñadores de bases de datos como **NORMALIZACIÓN**. En esta parte es donde se definen las reglas del negocio o necesidades del sistema. De ello hablaremos en el siguiente tema.

Capítulo III

DISEÑO DEL MODELO RELACIONAL DE LA BASE DE DATOS

INTRODUCCIÓN AL ANÁLISIS Y DISEÑO DE BASES DE DATOS

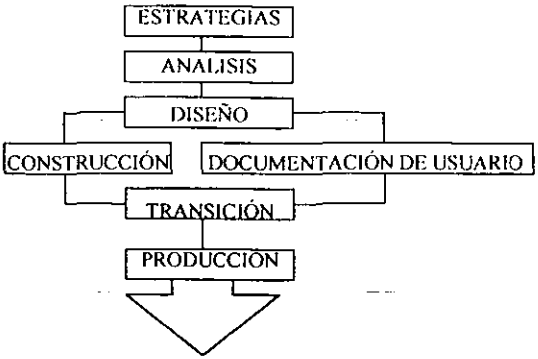
El análisis se encarga de conocer y entender las necesidades comerciales del sistema. En la fase de análisis del proyecto es prematuro pensar en el entorno de software y hardware que se ha usado para la solución. Existen diferentes metodologias para desarrollar el ciclo de vida de un sistema, y entre ellas pueden variar un poco dependiendo del programador. El método que se guiará aquí funciona bien incluso si no hay herramientas automáticas de análisis disponibles. Sobre todo debemos pensar que cualquier metodologia ha de ser flexible. Si se puede hay que considerar el empleo de una herramienta de modelar durante el análisis y el diseño, ya que ayudará a ser mas eficiente y sensible a los cambios. Estas herramientas incluso ayudan produciendo documentación de análisis y diseño.

Los documentos deben expresar lo que el sistema ha de hacer. Durante la fase de análisis hay que completar tanto el modelo de datos conceptual como el lógico. Antes de comenzar el diseño hay que documentar el protocolo con la comprensión de los requerimientos del usuario. En la fase de diseño se define cómo se consiguen esos requerimientos, se selecciona la tecnología y se revisa el modelo de datos lógico bajo consideraciones de rendimiento para el buen desarrollo de un modelo de datos físico. Antes de cualquier desarrollo ha de haberse completado el diseño y se ha de disponer de una base de datos física.

CICLO DEL MODELO DE DESARROLLO DE UNA BASE DE DATOS

REQUERIMIENTOS DEL NEGOCIO

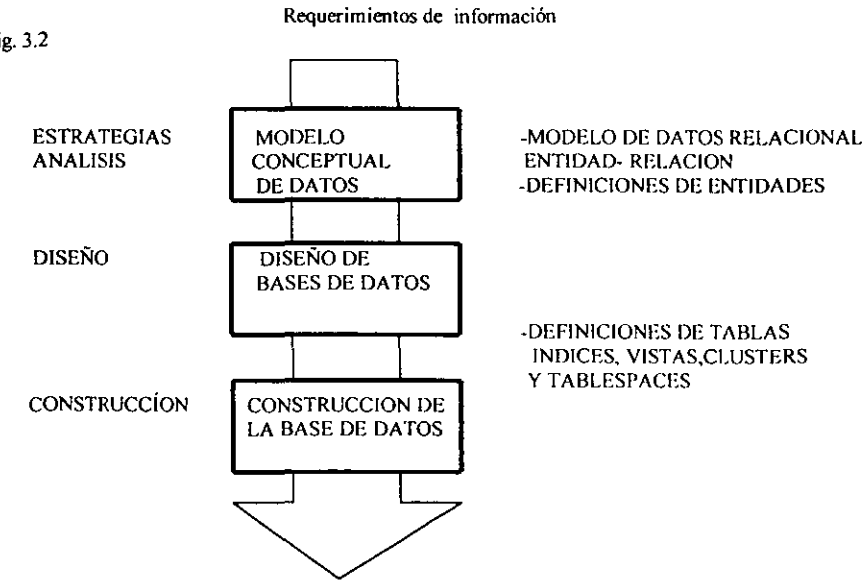
Fig 3.1



PROCESO DE DESARROLLO DE BASES DE DATOS

El desarrollo de base de datos es un enfoque top-down, que transforma los requerimientos de información en una base de datos operacional.

Fig. 3.2



El proceso de desarrollo de bases datos es un corte vertical del ciclo del método de desarrollo case
Requerimientos de información del negocio

Ejemplo:
Tenemos un conjunto de requerimientos de información proporcionados por un usuario

Yo manejo el departamento de recursos humanos de una compañía grande. Necesitamos tener la información de cada uno de los empleados de nuestra compañía. Necesitamos tener seguimiento de los nombres, apellidos, trabajo o empleo, fecha de contratación y salario de cada empleado. Para cualquier empleado por comisión, también necesitamos tener seguimiento de su posible comisión. A cada empleado sele asigna un número único.
Nuestra compañía esta dividida en departamentos. Cada empleado esta asignado a un departamento, por ejemplo, contabilidad, ventas o desarrollo. Necesitamos conocer el departamento responsable de cada empleado y la localización del departamento. cada departamento tiene un número único. Por ejemplo, contabilidad es el 10, y ventas tiene en número 30.
Algunos de los empleados son gerentes. Necesitamos saber quien es el gerente de cada empleado y que empleados tiene cada gerente.

Notas

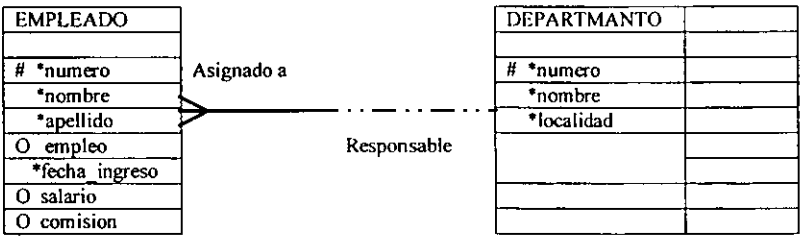
- El alcance de un conjunto de requerimientos pueden variar, pueden ser diferentes las necesidades de departamento a las necesidades de dada compañía.
- Los requerimientos de información están estrechamente ligados con los requerimientos de las funciones del negocio. Por ejemplo, los requerimiento en el departamento de recursos humanos, incluye la administración de la información del empleado

El modeló de datos conceptual define y modela los aspectos importantes acerca de la información que el negocio necesita saber o tener y las relaciones entre dicha información.

Ejemplo

El siguiente modelo entidad relación(E.R), representa los requerimientos del departamento de recursos humanos.

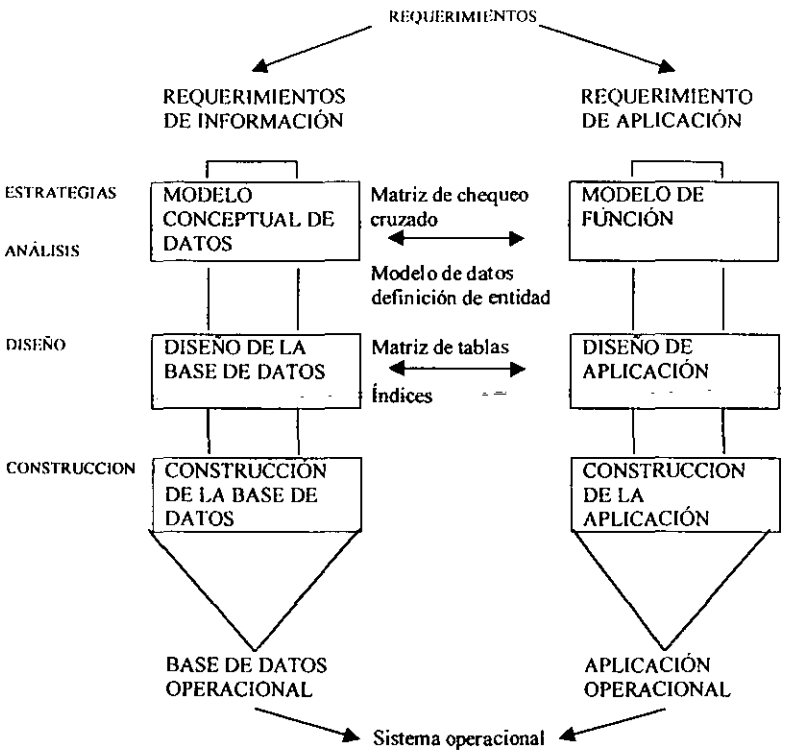
Fig. 3.3



Un modelo de datos entidad relación debería modelar adecuadamente las necesidades de información de la organización y soportar las funciones del negocio.

El proceso de desarrollo de una base de datos esta muy relacionado con el proceso de desarrollo de aplicaciones.

Fig. 3.4



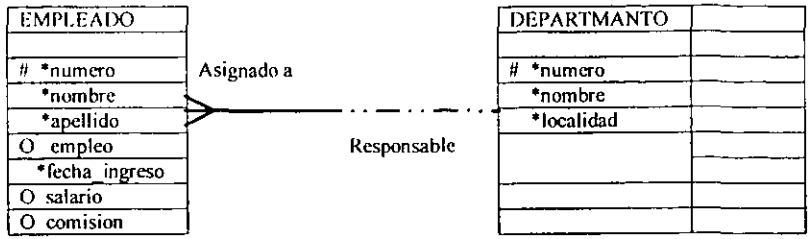
MODELO DE DATOS CONCEPTUAL BÁSICO

El modelo conceptual de datos es el primer paso del modelo TOP-DOWN para el desarrollo de la base de datos, se ejecuta durante las fases de análisis y estrategia en el ciclo de desarrollo de sistema como se muestra en la figura 1.1
El objetivo del modelo conceptual de datos es desarrollar el modelo entidad relación que represente los requerimientos de información de los negocios.

Ejemplo:

El siguiente modelo Entidad-Relación representa los requerimientos de información del departamento de recursos humanos.

Fig. 3.5



COMPONENTES DEL MODELO DE ENTIDAD-RELACIÓN

*Entidades: Son los aspectos importantes acerca de los cuales se necesita información
*Relaciones: Son las distintas formas en como se relacionan las entidades.
*Atributos: Información específica la cual necesita ser almacenada.

Un modelo de Entidad-relación es una forma efectiva para integrar y documentar los requerimientos de información en una organización.

Sintaxis

- Un modelo de E-R documenta los requerimientos de información de la organización en un formato preciso y claro

Comunicación del usuario

- Los usuarios pueden entender fácilmente la forma gráfica de un modelo de E-R

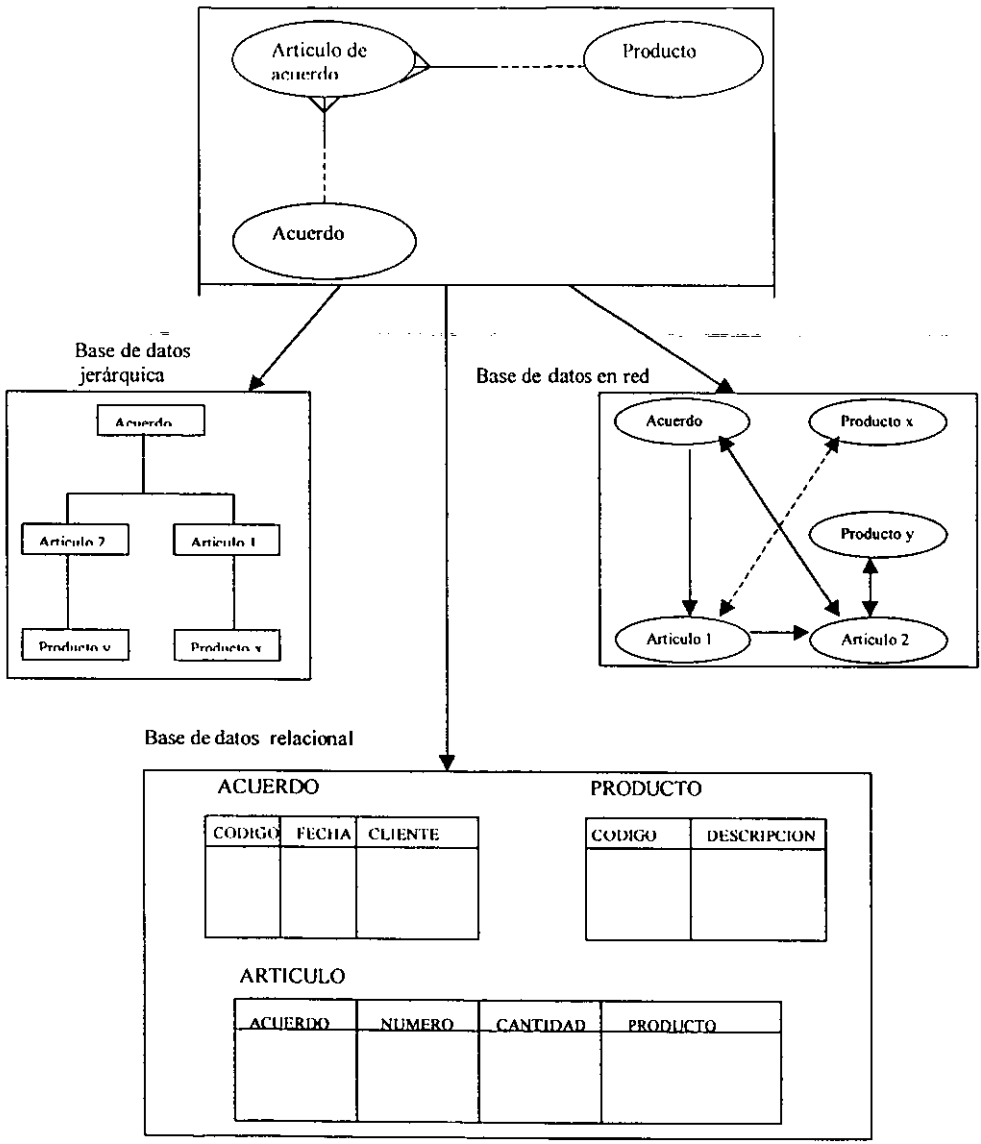
Fácil de desarrollar

- Un modelo E-R puede ser fácilmente desarrollado y definido por alto alcance
- Un modelo E-R provee una clara imagen del alcance de los requerimientos de información a la organizaciones. Integración de múltiples aplicaciones
- Un modelo E-R nos provee una estructura adecuada para la integración de múltiples aplicaciones, desarrollar proyectos, y/o paquetes de aplicaciones adquiridos

El modelo de datos conceptual es independiente del hardware o del software usados para la implementación Un modelo E-R puede ser utilizado para una base de datos de red, jerárquica o relacional.

Fig. 3.6

Modelo Entidad-Ralación



Una entidad es un aspecto importante acerca del cual se necesita tener o conocer información
Ejemplo:

EMPLEADO
DEPARTAMENTO

PROYECTO

Los atributos describen entidades y son las piezas especificas de información las cuales necesitan ser conocidas.
Ejemplo:

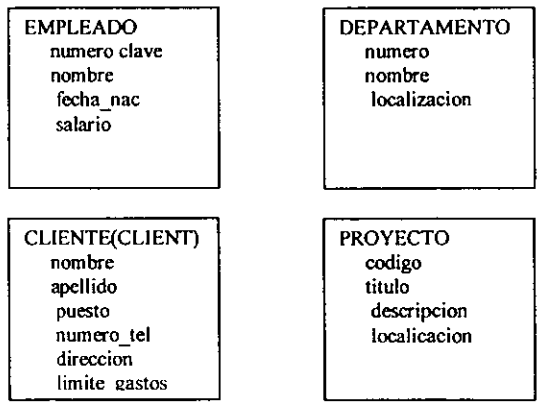
Número de clave, nombre, fecha de nacimiento, y salario, en la tabla de empleados
Ó nombre, número y localización, en la tabla de departamento

ESTÁNDARES PARA LA DIAGRAMACIÓN DE ENTIDADES

- Cajas de cualquier dimensión con las esquinas redondas
- Un nombre único para cada utilidad
- Nombre de la entidad en mayúsculas y en singular
- Nombre de sinónimos, entre paréntesis(opcional)
- Nombre de los atributos en minúsculas

Ejemplo:

Fig. 3.7



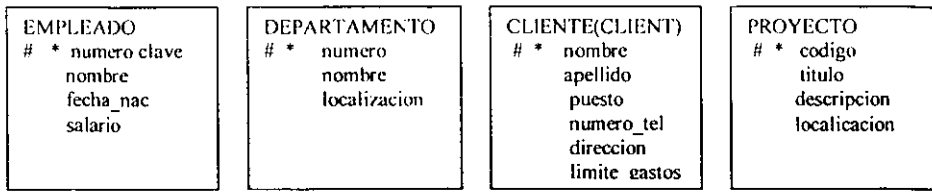
Nota

- Un sinónimo es un nombre alternativo para una entidad
- Los sinónimos son útiles cuando dos grupos de usuarios tienen diferentes nombres para el mismo aspecto diferente.

Cada EMPLEADO tiene una clave de EMPLEADO única. El número de clave del EMPLEADO es un candidato para se el UID de la entidad EMPLEADO.

Ejemplo:

Fig. 3.8



- Si una entidad no puede tener un identificador único (UID), esta no puede ser una entidad
- Los atributos que identifican de manera única una entidad y pertenecen a los UID de las entidades son precedidos por # *

RELACIONES

Una relación es bidireccional y representa la asociación entre dos entidades o entre una entidad consigo misma.

Sintaxis de una relación

Cada entidad {Debe ser o puede ser} nombre de la dirección {Una o mas ó una y solamente una} entidad

Por *ejemplo*:

La relación entre un instructor y un curso es:

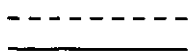
Cada curso puede ser enseñado por uno y solamente un instructor
Cada instructor puede ser asignado a uno o mas cursos

Nota .

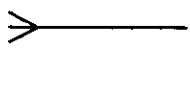
Cardinalidad es un sinónimo para el grado
Un grado de 0 es etiquetado como ser.

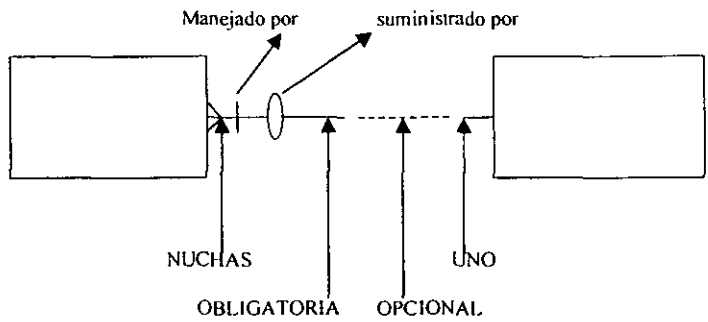
Estándares de diagramación

- Una línea entre dos entidades
- Nombres de relaciones en minúsculas
- Opcional



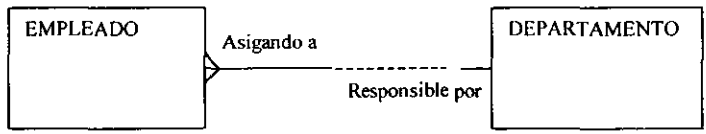
- Grado





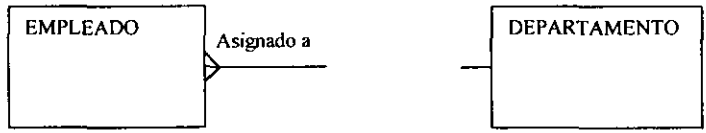
Primero leer la relación en una dirección, y después leer la relación en la otra dirección
Ejemplo:

Leer la relación entre EMPLEADO y DEPARTAMENTO



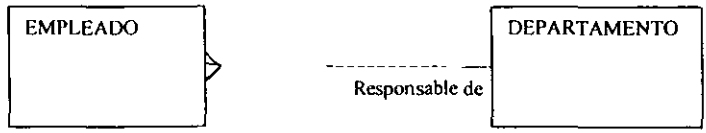
Leer esta relación primero de izquierda a derecha y después de derecha a izquierda

Relación de izquierda a derecha (Diagrama parcial)



Cada EMPLEADO debe ser asignado a uno y solamente un departamento

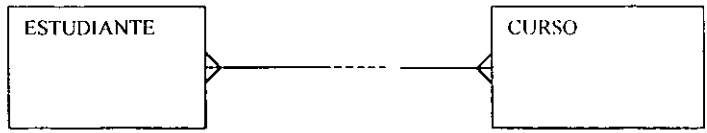
Relación de derecha a izquierda (diagrama parcial)



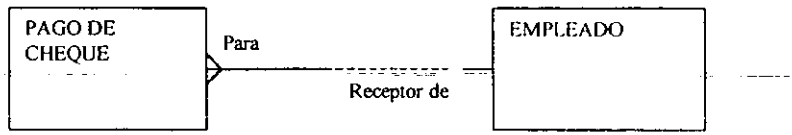
Cada DEPARTAMENTO puede responsable de uno o mas empleados

Otro *ejemplo*

Leer las relaciones entre ESTUDIANTE y CURSO



Un *ejemplo* mas:



Cada PAGO DE CHEQUE debe ser para uno y solo un EMPLEADO

Cada EMPLEADO puede ser el receptor de uno o mas PAGOS DE CHEQUE

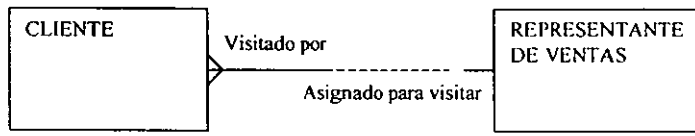
GRADOS DE RELACIÓN

Existen tres grados de relación.

- Relaciones muchos a uno
- Relaciones mucho a muchos
- Relaciones uno a uno

Todas las relaciones deben de representar los requerimientos de información y reglas de negocio. Por ejemplo, una relación (M a 1 o M:1(muchos a uno)) tiene el grado de uno o mas en una dirección y el grado de uno y solamente uno en la otra dirección.

En la siguiente relación de M:1 entre CLENTE y REPRESENTANTE DE VENTAS



Cada CLIENTE debe ser visitado por uno y solamente un REPRESENTANTE DE VENTAS

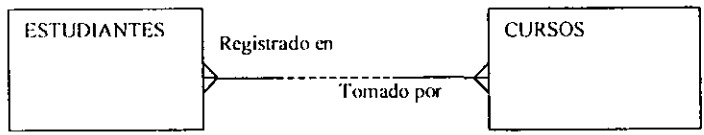
Cada REPRESENTANTE DE VENTAS puede estar asignado para visitar a uno o mas CLIENTE

Nota

- Las relaciones M:1 son las mas comunes.
- Las relaciones M:1 que son obligatorias en ambas direcciones son las mas raras

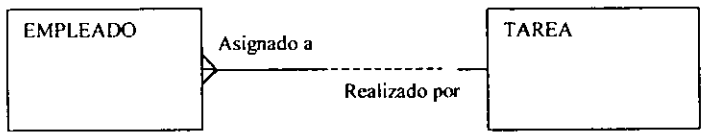
Una relación mucho a muchos (M:M) tiene el grado de uno o mas en ambas direcciones
Ejemplo:

Aquí hay una relación M:M entre ESTUDIANTES Y CURSOS



Cada ESTUDIANTE puede estar registrado en uno o mas CURSOS
Cada CURSO puede ser tomado por uno o mas ESTUDIANTES

En el siguiente diagrama se muestra uno relación M:M entre EMPLEADO y TAREA



Cada EMPLEADO puede estar asignado a una o mas TAREAS

Cada TAREA puede ser realizada por uno o mas EMPLEADOS

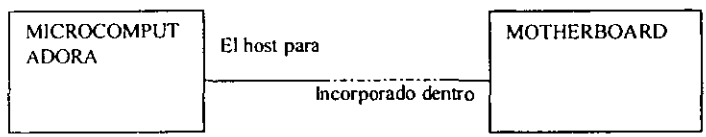
Nota

- Las relaciones muchos a muchos son muy comunes
- Las relaciones muchos a muchos usualmente son opcionales en ambas direcciones, también una relación muchos a muchos puede ser opcional en una sola dirección.

Una relación uno a uno (1:1) tiene un grado de uno y solamente uno en ambas direcciones.

Ejemplo:

Esta es una relación 1:1 entre MICROCOMPUTADORA y MOTHERBOARD



- Cada MICROCOMPUTADORA debe de ser el host para uno y solamente un MOTHERBOARD
- Cada MOTHERBOARD puede ser incorporado dentro de una y solamente un MICROCOMPUTADORA

COMO USAR UNA MATRIZ DE RELACIONES

Se una matriz de relaciones como una ayuda para la colección inicial de información sobre las relaciones entre una serie de entidades.

Estándares de matriz de relaciones

- Una matriz de relaciones muestra si están relacionadas y en qué forma cada entidad (renglón) con cada entidad (columna) mostrada en la matriz.
- Todas las entidades están listadas en el lado izquierdo y en la parte superior de la matriz
- Si una entidad está relacionada con otra entidad entonces el nombre de esta relación se muestra en la caja de intersección.
- Si una entidad no esta relacionada con otra entidad, entonces se dibuja una línea en la caja de intersección.
- Cada relación por encima de la diagonal es el inverso o imagen espejo de la relación por debajo de la línea diagonal
- Las relaciones recursivas (una entidad consigo misma) son representadas por las cajas de la diagonal

Ejemplo:
La siguiente matriz de relaciones muestra una serie de relaciones entre cuatro entidades

	CLIENTE	ARTICULO	ORDEN	BODEGA
CLIENTE			GENERADOR DE	
ARTICULO			COMPRADO A TRAVÉZ DE	GUARDADO EN
ORDEN	GENERADA POR	GENERADA PARA		
BODEGA		CONTENEDOR DE		

El CLIENTE esta relacionado a una ORDEN y el nombre de la relación es **generador de**

La ORDEN está relacionada el CLIENTE y el nombre de al relación es **generada por**

NORMALIZACIÓN DE UN MODELO DE DATOS

Normalizar es un concepto de base de datos relacional, esto quiere decir que un modelo de datos Entidad-Relación normalizado se traslada automáticamente dentro de un diseño de base de datos. En la siguiente figura se valida cada atributo usando las reglas de validación.

Regla de forma normal	**Descripción**
Primera forma normal (1FN)	Todos los atributos deben tener un solo valor para cada instancia.
Segunda forma normal (2FN)	Un atributo debe ser dependiente del identificador único completo
Tercera forma normal (3FN)	Ningún atributo no-IUD puede ser dependiente de otro atributo no-IUD

Nota

- La tercera forma normal en un objetivo generalmente aceptado para eliminar redundancias en un diseño de base de datos.
- Formas normales arriba de la tercera forma normal no son comúnmente utilizadas

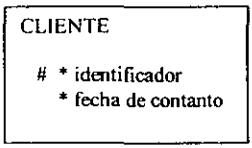
REGLA PRINCIPAL DE LA FORMA NORMAL

Todos los atributos deben tener un solo valor para cada instancia

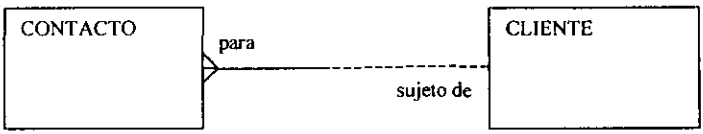
Validación

- Validar que cada atributo tenga un valor único para cada ocurrencia de la entidad ningún atributo deberá tener valores repetidos

Ejemplo
La entidad CLIENTE cumple con 1FN ? Si no cumple, cómo se podría convertir a 1FN?



El atributo fecha de contacto tiene múltiples valores, por lo tanto la entidad CLIENTE no es considera como 1FN



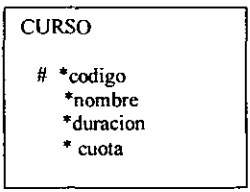
Si un atributo tiene múltiples valores, se crea una entidad adicional y lo relaciona con la entidad original mediante una relación M:1

REGLAS DE LA SEGUNDA FORMA NORMAL

Validación

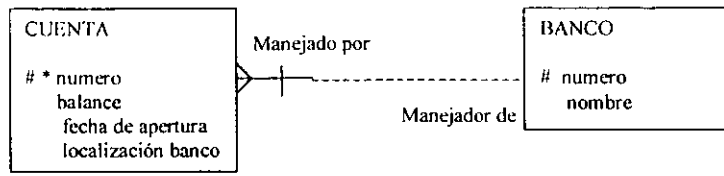
- Validar que cada atributo dependa completamente del UID. Cada instancia especifica del UID debe determinar una sola instancia de cada atributo
- Validar que un atributo asignado no dependa de una sola parte del IUD de la entidad

Ejemplo
Validar los atributos asignados a la entidad CURSO.



Cada instancia del código de curso determina un valor especifica para el nombre , duración y cuota. Los atributos están ubicados correctamente

Ejemplo
Validar que lo atributos de las entidades de CUENTA y BANCO estén correctamente ubicados



Cada instancia de un BANCO y número de cuenta determinan valores especificos del balance y de la fecha de apertura para cada cuenta. El atributo de localización del banco está mal ubicado. Este depende del BANCO , pero no del número de cuenta. Este no deberá ser un atributo de CUENTA.

Nota

* Si un atributo no es dependiente del IUD completo , está fuera del lugar y deberá ser movido.

REGLAS PARA LA TERCERA FORMA NORMAL

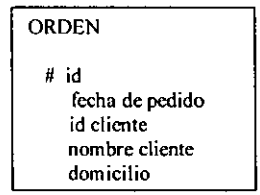
Ningún atributo no-IUD puede ser dependiente de otro atributo no-UID

Validación

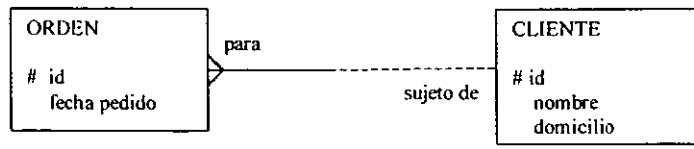
- Validar que cada atributo no-UID no dependa de otro atributo no-UID
- Mover cualquier atributo no-UID que dependa de otro atributo no-UID

Ejemplo

Algún atributo no-UID de esta entidad depende de otro atributo no-UID



Los atributos nombre del cliente y domicilio dependen de la clave de cliente(Id Cliente). Crear otra entidad llamada CLIENTE con el campo de Id Cliente como su UID y ubicar los atributos correctamente



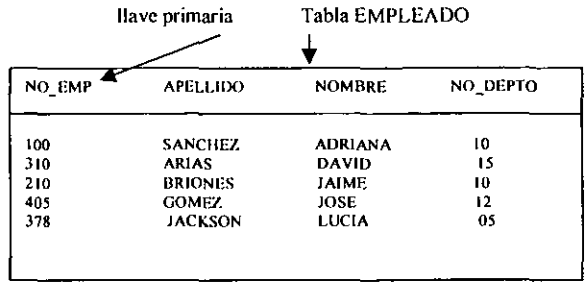
Nota

- Si un atributo de otro atributo no-IUD, es necesario mover ambos, al atributo dependiente y el atributo del que depende, a una entidad relacionada con la entidad actual

LLAVES PRIMARIAS

Una llave primaria (PK) es una columna o grupo de columnas que identifican de manera única a cada renglón en una tabla. Cada tabla debe tener una llave primaria y una llave primaria debe ser única.

Ejemplo:
La llave para la tabla EMPLEADO consiste en la columna NO_EMP. Cada renglón en la tabla es identificado de manera única por su valor en NO_EMP.



NO_EMP	APELLIDO	NOMBRE	NO_DEPTO
100	SANCHEZ	ADRIANA	10
310	ARIAS	DAVID	15
210	BRIONES	JAIME	10
405	GOMEZ	JOSE	12
378	JACKSON	LUCIA	05

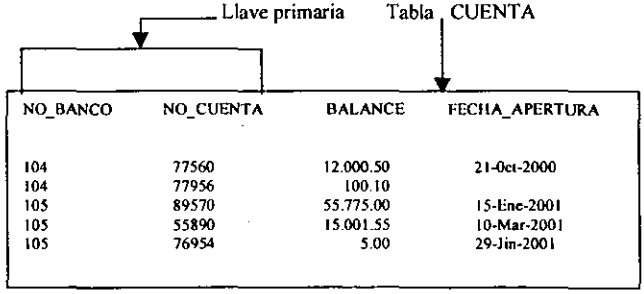
Nota

- No se aceptan duplicado en la llave primaria. La llave primaria debe de ser única
- El valor de las llaves primarias normalmente no se pueden cambiar
- El UID de una entidad ira de acuerdo con la llave primaria en su tabla correspondiente

Una llave primaria que consta de múltiples columnas se llama llave primaria compuesta

Ejemplo

La llave primaria compuesta para la tabla CUENTA consta de la combinación de las columnas NO_BANCO y NO_CUENTA. Cada renglón en la tabla está identificado de manera única por los valores de NO_BANCO y NO_CUENTA.



NO_BANCO	NO_CUENTA	BALANCE	FECHA_APERTURA
104	77560	12.000.50	21-Oct-2000
104	77956	100.10	
105	89570	55.775.00	15-Ene-2001
105	55890	15.001.55	10-Mar-2001
105	76954	5.00	29-Jun-2001

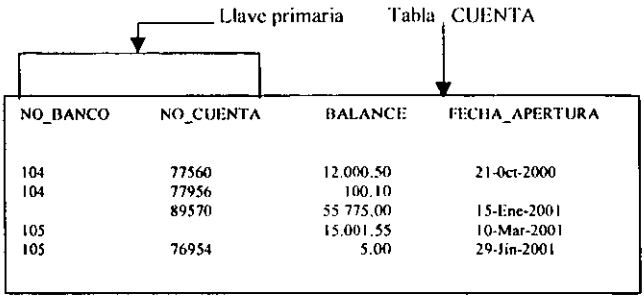
Nota

- Las columnas de una llave primaria compuesta deben de ser únicas en combinación. Las columnas pueden tener duplicados en forma individual, pero en combinación, no se permiten duplicados

Ninguna parte de la llave primaria puede ser NULA.

Ejemplo

NO_EMP es la llave primaria de la tabla EMPLEADO . Por lo tanto NO_EMP debe de ser definida cono NO NULA

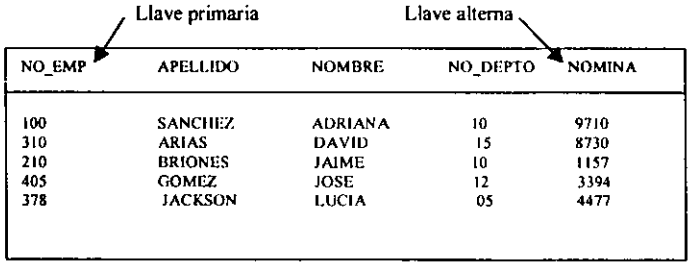


NO_BANCO	NO_CUENTA	BALANCE	FECHA_APERTURA
104	77560	12.000.50	21-Oct-2000
104	77956	100.10	
	89570	55 775.00	15-Ene-2001
105		15.001.55	10-Mar-2001
105	76954	5.00	29-Jin-2001

Note que dos de los renglones tienen valores NULOS en partes de la PK (primary key) . Compuesta NO_BANCO y NO_CUENTA deben de ser definidas como NO NULOS.

Una tabla puede tener mas de una columna o combinación de columnas que pueden servir como la llave primaria de la tabla. Cada una de estas es llamada llave candidata o alterna.

Ejemplo



NO_EMP	APELLIDO	NOMBRE	NO_DEPTO	NOMINA
100	SANCHEZ	ADRIANA	10	9710
310	ARIAS	DAVID	15	8730
210	BRIONES	JAIME	10	1157
405	GOMEZ	JOSE	12	3394
378	JACKSON	LUCIA	05	4477

Aquí podemos observar que NO_EMP y NOMINA son las llaves primarias

- Todas las llaves alternas deben de ser únicas y NO NULAS
- Los UID secundarios concuerdan con las llaves alternas
- Los nombres de personas normalmente no son llaves alternas por que no se puede garantizar que sean únicas. Por ejemplo, en la tabla EMPLEADO , la combinación NOMBRE /APELLIDO posiblemente no podrían ser una llave alterna

INTRODUCCION AL DISEÑO INICIAL DE LA BASE DE DATOS

Documentar cada tabla relacional en un mapa de instancias es una forma correcta de mostrar los detalles de la información, para conocer mejor las ventajas de una buena documentación se muestra el siguiente

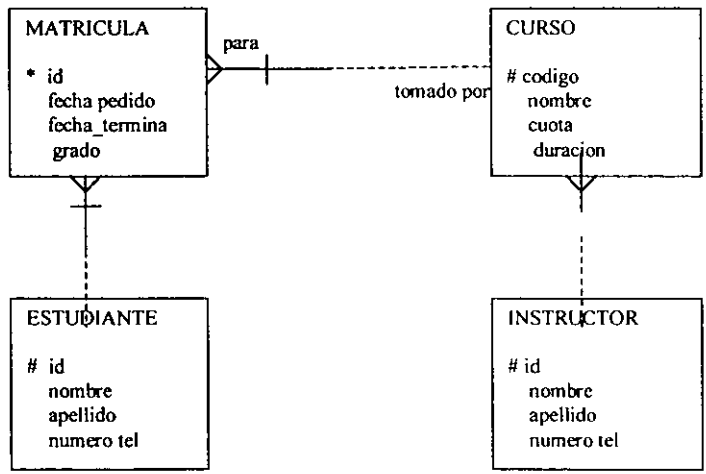
ejemplo.

Columna	EMPNO	NOMBRE	APELLIDO	PUESTO	FECHA_PAGO	SAL	COM	MGR	NODEPT
Tipo clave	PK							FK1	FK2
Nulos/únicos	NN,U	NN	NN		NN				NN
Ejemplo	7369	Martha	Sanchez	Analyst	17-Dic-2000	800		7902	20
	7902	Miguel	Canto	Vendor	03-Dic-2000	1000		7566	50
	7521	Fidel	Cortez	Sell	22-Feb-2001	1250	6000	7998	30
	7698	Peter	Infante	Gerente	1-May-2001	2850	1000	7839	30
	7839	Rodrigo	Bernal	Presid	7-Mar-1998	5000	5000		10

Nota

- Los tipos validos de llaves son PK(Primary Key) para una columna llave primaria, y FK(Foreing Key) para una columna llave foránea.
- Usar sufijos para distinguir entre múltiples columnas FK en una tabla , por ejemplo, FK1, FK2. Etiquetar múltiples columnas son el mismo sufijo.
- Usar NN para una columna que debe ser definida como NO NULA
- Usar U para la columna que debe ser única.
- Si múltiples columnas deben ser únicas en combinación, etiquetarlas con un sufijo, por ejemplo U1.
- Etiquetar una columna sencilla PK como NN, U.
- Etiquetar múltiples columnas PK compuestas como NN, U1 o como NN, U1,U.

En el siguiente modelo ilustraremos las actividades del diseño inicial de la base de dato.



PASOS PARA EL DISEÑO INICIAL DE LA BASE DE DATOS

Paso 1.- Mapear la tabla para cada entidad. Crear un mapa de instancia para la nueva tabla. Registrar únicamente el nombre de la tabla.

Ejemplo:

Crear un mapa de instancia para la entidad INSTRUCTOR. El nombre de la tabla será INSTRUCTOR

NOMNRE DE LA TABLA: INSTRUCTOR

	COLUMNA			
	TIPO DE LLAVE			
INSTRUCTOR	NULOS UNICOS			
	EJEMPLO			

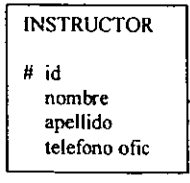
Nota

- El nombre de la tabla debe ser fácil de identificar con el nombre de la entidad. El nombre en plural de una entidad se usa algunas veces por que la tabla debe contener un grupo de renglones.
- Una entidad simple no es un subtipo o supertipo. En el paso 6 el diseñador debe decidir se hace una construcción supertipo y un subtipo.

Paso 2.- Mapear cada atributo de la entidad a una columna en su tabla correspondiente. Establecer los atributos obligatorios para columnas NO NULAS (NN).

Ejemplo:

Asignar los atributos de la entidad INSTRUCTOR a la columna de la tabla INSTRUCTOR. Marcar como atributos obligatorios (NO NULOS) las columnas insert_id , nombre y apellido.



Columna	INST_ID	NOMBRE	APELLIDO	NOTELEF
Tipo de llave				
Nulos / únicos	NN	NN	NN	
Ejemplo				

Nota

Para cada atributo seleccionar un nombre corto pero significativo

- el nombre de las columnas debe ser fácil de identificar en un modelo de E-R
- Prevenir al usuario de no usar las palabras reservadas de SQL para nombre de columnas. Pro ejemplo NUMBER

- Usar abreviaciones consientes que no causen confusión al usuario y al programador. Por ejemplo, Podria ser abreviado NUMERO como NO o NUM y DEPTNO o DEPTNUM?
- Los nombres de columnas cortos o pequeños reducirán el tiempo requerido para el comando de SQL

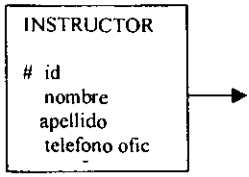
Documentar renglones de ejemplo para las columnas de las tabla INSTRUCTOR

Columna	INST_ID	NOMBRE	APELLIDO	NOTELEF
Tipo de llave	PK			
Nulos / únicos	NN	NN	NN	
ejemplo	10	NANCY	HALL	798-2251
	18	MARIA	GONZALEZ	756-4891
	73	PEET	CASSIDY	301-2291
	95	KATHY	ANDRUS	483-9221
	301	ERIC	CAPLIN	535-3166

Paso 3.- Asignar cualquier atributo (s) que sea parte del UID de la entidad a columnas PK. Etiquetar las columnas PK

Ejemplo
El atributo ID es el UID de la entidad INSTRUCTOR, entonces hacer que la columna correspondiente INST_ ID sea el PK de la tabla INSTRUCTOR.

Nombre de la tabla: INSTRUCTOR



Columna	INST_ID	NOMBRE	APELLIDO	NOTELEF
Tipo de llave	PK			
Nulos / únicos	NN	NN	NN	
Ejemplo	10	NANCY	HALL	798-2251
	18	MARIA	GONZALEZ	756-4891
	73	PEET	CASSIDY	301-2291
	95	KATHY	ANDRUS	483-9221
	301	ERIC	CAPLIN	535-3166

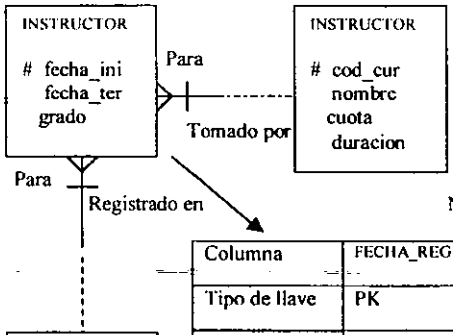
Un tipo de llave PK indica una columna de llave primaria

Nota
- Todas las columnas etiquetadas con PK deben de etiquetarse también con NN y U
- Asignar un IUD que incluya múltiples atributos a una PK compuesta. Etiquetar esas columnas NN y UI

Si una entidad incluye una relación, agregar columnas de llaves foráneas para la tabla y señalarlas como parte de la llave primaria.

Ejemplo

El IUD de la entidad MATRICULA está compuesto de una relación para CURSO y de una relación para ESTUDIANTE. Agregar dos columnas FK a la tabla MATRICULA, para el PK de la tabla CURSO y para el PK de la tabla ESTUDIANTE.



Nombre de la tabla: MATRICULA

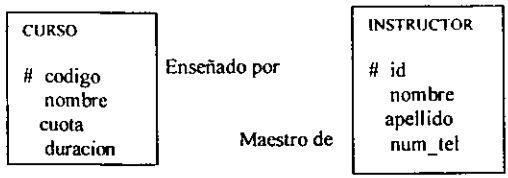
Columna	FECHA_REG	FECHA_TER	GRADO	COD_CUR	ESTUD_ID
Tipo de llave	PK			PK, FK1	PK, FK2
Nulos / únicos	NN			NN, U1	NN, U1
ejemplo	20-Jun-2000	29-Jun-2001		344	47593
	05-Sep-2000			401	15402
	14-Jun-2001	28_Ago-2001	A	717	94572
	8-May-2000	28-Feb-2000	B	117	94572
	05-Feb-2000	21-Mar-2000	A	401	51394

Nota

- Escoger un nombre único para cada columna FK, y etiquetar la(s) columnas(s) PK, NN y FK
- Si existen múltiples columnas PK en una tabla, usar subfijos para distinguirlos, por ejemplo, FK1 y FK2. Etiquetar múltiples columnas llave con el mismo sufijo
- Las PK compuestas deben de ser únicas en combinación y deben de ser etiquetadas como U1
- Agregar ejemplos de datos para las columnas FK

Paso 4.- Mapear relaciones para llaves foráneas

Ejemplo
Poner el PK INST_ID de la tabla 1 y ponerlo en la tabla CURSO (Tabla M)



Nombre de la tabla : CURSO

COLUMNA	CUR_DOD	NOMBRE	CUOTA	DURACION	INST_ID
Tipo de llave	PK				FK
Nulos / únicos	NN, U	NN		NN	
Ejemplo	334	SQL/FORMS	1000	5	81
	974	SL *RV	400	2	73
	401	DB DESING	1500	6	95
	717	DBA	2000	10	73
	659	SQL/PLUS	1000	5	301

Reglas:

M:1 >---------- (__ , __) PK

M:1 >———————— (NN , __) FK

1:1 ---------- (__ , U) FK

1:1 ———————— (NN , U) FK

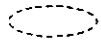
1:1 ———————— (NN , U) FK

Heredada >|———— (NN , U) PK , FK

Tipo de relación:

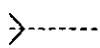
 (__ , __) FK

Recursiva



(___ , U) FK

Recursiva



(___ , U) FK

OBSERVACIONES

El ciclo de vida del desarrollo de un sistema es un conjunto de pasos que un programador recorre para su creación y que, tradicionalmente, se hace uso de todas las etapas ya mencionadas. Existen proyectos con mas o menos fases de su vida y con algunas fases combinadas, como análisis y diseño, o bien con ellas fraccionadas, es decir, pruebas de la unidad, pruebas del sistema y garantías de calidad en fases distintas. En los requerimientos deben estar identificadas todas las reglas comerciales importantes, entradas y salidas (suministros del sistema) y frecuentemente incluir interfaces del usuario y prototipos de desplazamiento interno. Los suministros del sistema pueden consistir en instrucciones sobre lo que el sistema debe hacer, por ejemplo: <<Quiero que el sistema sea capaz de seguir un componente desde el pedido has la entrega>> esto sería un requerimiento del usuario. Para que esto sea mas claro veamos el siguiente tema que es la construcción de una base de datos, en esta parte se mostraran los detalles de como se elabora una fuente de datos y como es que esta puede organizarse.

Capitulo

IMPLANTACIÓN DEL SISTEMA

DISEÑO DE UNA BASE DE DATOS ORACLE

INTRODUCCIÓN AL CICLO VITAL DE DASARROLLO DE UN SISTEMA

EL ciclo de vida del desarrollo de un sistema es un conjunto de pasos que un programador recorre para su creación y que, tradicionalmente, incluyen las siguientes etapas.

1.- **Análisis de los requerimientos** En la fase de análisis se trabaja con los usuarios para conocer y especificar los requerimientos mismos del sistema. Durante esta etapa, se pueden desarrollar prototipos de la interfaz del usuario así como completar modelos lógicos de usuario.

2.- **Arquitectura y diseño** En esta fase se desarrolla un proyecto técnicamente válido para aplicar los requerimientos del sistema.

3.- **Desarrollo** En la fase de desarrollo es donde se codifica realmente la aplicación utilizando el diseño y los requerimientos ya definidos previamente.

4.- **Garantía de calidad** En esta etapa se asegura que la aplicación cumpla con todos los requerimientos y no contenga errores.

5.- **Instalación** Aquí se instala la aplicación en los puestos de trabajo del usuario.

6.- **Seguimiento** Durante esta fase de seguimiento se verifica como funcione el sistema con los operadores. Es una fase importante de cara a definir posibles nuevos requerimientos para ediciones posteriores de la aplicación.

CONOCIMIENTOS DEL REQUERIMIENTO DE LOS USUARIOS.

El análisis de los requerimientos es una fase mucho muy importante, pues antes de empezar el diseño hay que comprender bien los requerimientos, en esta etapa deben estar bien identificadas todas las reglas comerciales mas importantes(entradas y salidas).

INTRODUCCIÓN AL ANÁLISIS.

El análisis se encarga principalmente de conocer y entender las necesidades comerciales del sistema, existen diferentes metodologias para desarrollo de sistemas, estas pueden variar dependiendo del programador. Lo que si es recomendable a todos los programadores, es que el sistema debe ser flexible, si se puede, hay que considerar el empleo de una herramienta de modelar durante el análisis y diseño, ya que ayudara a ser mas eficiente y sensible a los cambios. Esta herramientas ayudan incluso a producir documentos de análisis y diseño.

MODELO CONCEPTUAL DE DATOS.

Un modelo de datos conceptual es una descripción, un diagrama, un dibujo o algo mas sofisticado que explique, en el ámbito del problema, cuales son los conceptos del negocio q resolver. No es una representación técnica y sólo nuestra los elementos de la trama y su relación entre ellos, estos elementos pueden ser personas, lugares, cosas, sucesos o puramente conceptos sobre los que el negocio quiere,. Por ejemplo, Clientes, Facturas, Almacenes, Proveedores, Articulos etc.

Una relación es una conexión lógica entre los elementos del negocio que comunican alguna información sobre el entramado interno. Por ejemplo, un pedido es para uno y solo uno de los clientes, y una factura es para uno y solo uno de los pedidos.

El modelo de datos conceptual que se desarrollará será un diagrama componente/relación(C/R) establecido durante la fase de análisis. Los diagramas C/R son representaciones gráficas de alementos de negocio y sus relaciones. El objetivo de estos diagramas es representar con exactitud en un nivel conceptual los datos que un negocio necesita conocer y recordar.

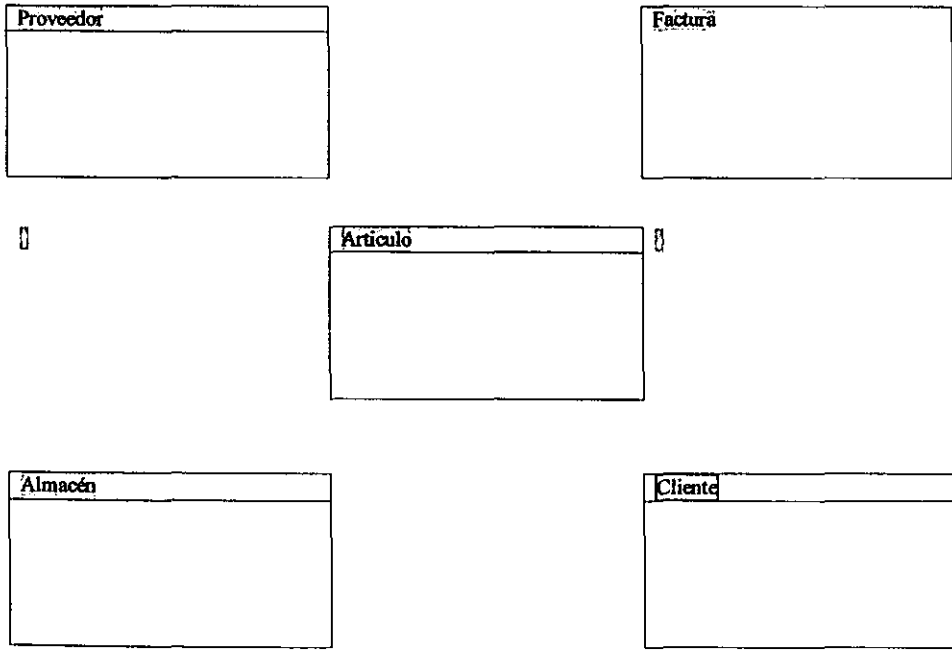
Un diagrama C/R se define en cuatro fases:

1.- Identificar, cualificar y relacionar todos los elementos posibles.
2.- Esquematizar los elementos y sus interrelaciones.
3.- Documentar las reglas del negocio.
4.- Revisar el diagrama C/R y actualizarlo, si procede.

Las siguientes directrices ayudaran a identificar los posibles elementos. Un componente ha de ser:

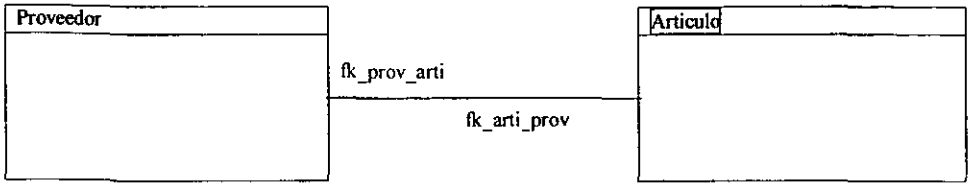
. Persona, lugar, cosa, concepto o soceso.
. Algo para lo cual el negocio ha de tener poder y medios de conseguir información.
. Identificado con un sustantivo singular y único que representa una ocurrencia o réplica del componente
. Poseedor de un atributo singular que lo identifique y que el negocio conozca y pueda crear.
. Diferente y mutuamente exclusivo el relación con los otros elementos del modelo de datos.
. De importancia relativamente igual a la de los otros elementos del modelo de datos.

Con estas características podemos identificar los requerimientos a nuestro sistema, y son los siguientes.



Estos son los elementos del negocio identificados hasta el momento.

Una vez identificados hay que trazar su relación entre ellos, una razón para mostrar las relaciones, es la de asegurarse que se han identificado todos los elementos. Un componente que no tenga relación con otro, puede no ser un componente separado.



Conectar los elementos con una línea y dar un nombre a esta relación.

Dos elementos pueden tener mas de una relación entre ellos, por lo que se puede trazar mas de una linea que los una, para ello no hay que sentirse forzado a elegir entre dos relaciones validas.
Las relaciones se describen con verbos. Si una relación no se identifica con un verbo, es que probablemente no sea una relación, la razón para mostrar las relaciones es la de asegurarse que se han identificado todos los elementos y la mayoría de las mas importantes reglas del negocio en el sistema.

- Un componente que no tenga relación con otro puede no ser un componente separado.
- Si se aprecia una serie de elementos no relacionados con otra serie de elementos, se puede pensar en di señar dos o mas sistemas diferentes.

Como se mostró ya en el segundo capitulo, existen diferentes relaciones.

Relación uno a uno existe donde un acontecimiento del primer elemento tiene una relación con uno y solo un acontecimiento de un segundo elemento. Por ejemplo, un cliente puede tener una y solo una dirección para factura.
Relación una a muchos, existe donde un acontecimiento del primer elemento tiene una relación con muchos acontecimientos de un segundo elemento. Por ejemplo, un cliente puede tener muchos pedidos, pero un pedido es de uno y solo un cliente.
Relación muchos a muchos, existe donde un acontecimiento del primer elemento puede tener una relación con muchos acontecimientos del segundo elemento y un acontecimiento del segundo elemento puede tener una relación con muchos acontecimientos del primer elemento. Por ejemplo, un cliente puede pedir muchos productos y un producto puede ser pedido pos muchos clientes.

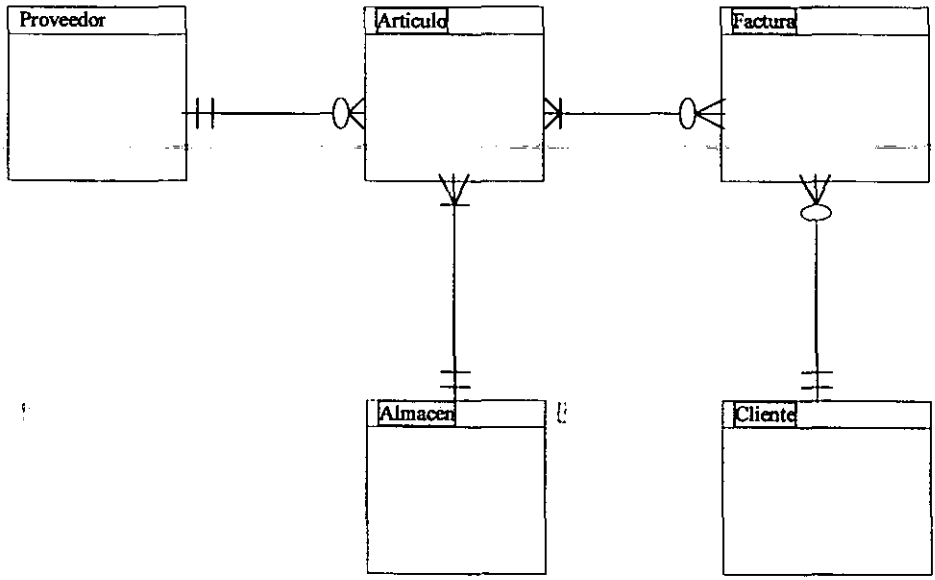
En un modelo de datos conceptual, los elementos pueden tener relaciones muchos a muchos con otros elementos.
El modelo de datos lógico añade atributos al modelo de datos conceptual e identifica las claves principales y foráneas(primary key – foreing key) respectivamente.

ASIGNACIÓN DE ATRIBUTOS A LOS ELEMENTOS

Un atributo o propiedad, es una información sobre el elemento o relación. Los atributos pueden no ser lo mismo que los aclaratorios, porque éstos aceptan un cierto grado de detalles, no necesariamente presentes en los atributo. Por ejemplo, calle-dirección, puede ser un atributo, mientras que calle-numero, calle-nombre o calle-subfijo, pueden ser un dato aclaratorio de calle-dirección.

El dominio de un atributo es el rango de valores aceptable para un atributo
El primer paso para construir el modelo de datos lógico es asignar atributos a los elementos a partir de losrequerimientos:

- Se necesita conocer quien va a recibir las facturas, así que hay que identificar diversos atributos del cliente: Nombre, dirección, teléfono, NIF y persona de contacto.
- Se necesita conocer que es los que se va a facturar, así que hay que identificar la información del producto: Numero de articulo, descripción, precio etc.
- Para seguir los artículos como se piden en los requerimientos, se necesitan atributos para el almacén: Nombre del almacén, número de identificación, dirección, etc.
- Para seguir a los suministradores se necesita: Nombre, dirección, teléfono, NIF, etc.
- Del formato de factura hay que identificar cada campo como un atributo.



Ahora puede verse la relación entre los elementos, y a medida que se continua con el proceso de refinado del modelo lógico, la normalización de algunos elementos puede producir que se identifiquen nuevos elementos y que algunos atributos se cambien de un elemento a otro.
Debido a que el modelo relacional necesita que todos los elementos tengan una clave principal, el número de factura será la clave principal del elemento factura. Es posible que también se quiera añadir los tipos de dato(entero, decimal, etc.) a los atributos, y entonces lo haremos de la manera siguiente:

Factura	
Numero_de_factura(PK)	integer
Fecha_factura	date
Numero_de_cliente	integer
Nombre_de_cliente	char(30)
Direcc1_de_cliente	char(30
Cirecc2_de_cliente	char(30)
Ciudad_de_cliente	char(16)
Edo_cliente	char(2)
Telefono_cliente	char(15)

Numero_de_articulo	integer
Cantidad_de_articulos	integer
Descrip_de_articulo	char(30)
Precio_comer_articulo	decimal(7,2)
Precio_neto_articulo	decimal(7,2)
Numero_de_almacen	integer
Nombre_de_almacen	char(30)
Dir_de_almacen	char(30)
Ciudad_almacen	char(16)
Telefono_almacen	char(10)
Numro_de_proveedor	integer
Nombre_de_proveedor	char(30)
Dir_de_proveedor	char(30)
Ciudad_proveedor	char(16)
Telefono_proveedor	char(15)
Tax Amount	decimal(7,2)
Miscellaneos	decimal(7,2)

Para ilustrar el proceso de normalización de datos, se añade inicialmente todos los atributos al elemento factura.

TERMINOLOGIA RELACIONAL

Antes de construir un modelo de datos lógico es importante conocer los términos mas importantes de un modelo relacional

Modelo relacional	Es un modelo de bases de datos en los que los elementos se representan como tablas, los registros se guardan como filas de la tabla y los datos aclaratorios o campos, como columnas de cada fila.
Tabla	Es una colección de registros en los que cada uno es una fila y cada campo es una columna.
Relación	Una relación esta representada por elementos de una tabla contenidos en una segunda tabla
Atributo	Es la propiedad de un elemento. En una base de datos relacional, los elementos se guardan como tablas y los atributos como columnas.
Valores atómicos	Son valores que no tienen estructura interna, estos valores no se pueden descomponer en valores o tipos de datos mas sencillos. Esto no significa que el valor para un nombre no sea atómico, generalmente no se descompone el valor de un nombre en las letras que lo forman. Se puede descomponer el valor de una fecha en valores de días y meses.
Fila	Son registros guardados en una tabla. Una tabla no puede tener filas duplicadas. Dentro de una tabla, las filas y los atributos están desordenados(izquierda a derecha). Todos los valores de atributos deben ser atómicos. Las filas son acontecimientos de un elemento. Las filas de una relación se llaman a veces tuplas.
Tupla	Se denomina así a las filas de una relación.
Relación base	Una tabla identificada o una relación guardada en una base de datos como oposición a una que contiene el resultado de una consulta, sele llama, relación base.
Base de datos	Es una colección ordenada de datos distribuida en tablas.
Cardinalidad	Es una forma de ayuda cuando se define la relación entre dos elementos. Por ejemplo una factura será para uno y solo un cliente, mientras que un cliente puede tener cero o mas facturas.

Dominio	Es un conjunto de valores legales, son valores que pueden estar contenidos en los atributos.
Clave candidato	Es un conjunto de uno o mas atributos en una tabla o relación, cuyos valores se pueden utilizar para seleccionar filas individuales en una tabla, esta clave debe ser única, debe identificar solo una fila de la tabla, también dese ser mínima: si una clave candidato es compuesta(compuesta de mas de un atributo), ningún subconjuntos de los atributos en la clave compuesta pueden ser también claves candidato. Los atributos de una clave candidato no pueden se NULL. Cada tabla deberá tener al menos una clave candidato.
Llave primaria	Una clave candidato se selecciona como clave principal para la tabla. Una tabla o relación puede tener múltiples claves candidato, pero solo una se elegirá como clave principal. Las reglas para esta clave son las mismas que para las claves candidato(únicas, mínimas y no NULL).
Llave foránea	Es un conjunto de una o mas columnas de una tabla que no son la clave principal de esta tabla pero lo son de otra tabla, una clave foránea es una referencia a una sola fila en otra tabla que contenga el valor que concuerda con la clave principal. La integridad referencial asegura que no haya valores no validos de clave foránea en la tabla.
Llave alternativa	Es cualquier clave candidato en una tabla no seleccionada como clave principal de la mima.
Llave atributo	Es un atributo que es parte de una clave candidato en una tabla.
FD Determinante	El atributo en el lado izquierdo(LS) de un diagrama de Dependencia Funcional(FD) es el determinante, un valor del determinante que identifica uno, y solo un valor del atributo en el lado derecho del diagrama. Cuando el determinante es la clave principal de una tabla, todos los atributos de una tabla son, por definición, funcionalmente dependientes del determinante.
Non-key atributos	Son atributos que no son parte de cualquier clave candidato de la tabla.
Funcional	Un atributo en una tabla(por ejemplo, Dependencia fecha de nacimiento en la tabla de empleados) es funcionalmente dependiente de un segundo atributo en la mis, a tabla(por ejemplo, el DNI del empleado) cuando cada valor del segundo atributo (el DNI del empleado) tiene un valor y sólo uno, para el primer atributo (fecha de nacimiento) en la tabla en cualquier punto en el tiempo, el diagrama de dependencia funcional se ve de la siguiente forma.

EmpleadoID ⟶ fecha de cumpleaños

NORMALIZACIÓN

Como era posible, hay serios problemas en el modelo de datos que se esta haciendo. Hay que refinar ahora el modelo utilizando proceso llamado normalización de datos. El objetivo de la normalización y del buen diseño de la base de datos es reducir, no necesariamente eliminar, la redundancia de datos. Puede haber ocasiones en las cuales una limitada redundancia de datos es necesaria por razones de rendimiento.
La normalización es una serie de pasos para refinar el modelo de datos hasta que satisface ciertas condiciones. Cada paso en el proceso conduce los elementos relaciones a una forma normal mas alta, empezando con la primera forma normal, la segunda forma, la tercera forma, y así sucesivamente. Por ejemplo una tabla esta en la primera forma normal(1NF) solo si los dominios de sus atributos son valores atómicos, a medida que se traten cada una de las formas normales, se vera la utilidad de las mismas.

Primera forma normal(1NF) Una tabla esta en la primera forma normal(1NF) solo si tiene una clave principal y sus atributos son todos tipos de datos simples sin atributos repetidos. Para cumplir con las restricciones de 1NF, los dominios de los atributos de una tabla deben ser valores atómicos, y solo pueden ser grupos repetidos de atributos. Todos estos grupos repetidos de atributos se deberán mover a una tabla nueva. Cuando una tabla se ajusta a 1NF, se le considera como tabla normalizada, pareciendo a un archivo

plano de dos dimensiones. Los elementos que contienen grupos repetidos se consideran como no normalizados.
Mirando la tabla de factura, se debe notar un grupo repetido, en la factura se permiten hasta seis líneas por factura, este diseño se limita únicamente a seis artículos por factura, ver en el siguiente modelo lógico los elementos factura y factura en linea la información repetida de una tabla a otra.

Factura	
Numero_de_factura(PK)	integer
Fecha_factura	date
Numero_de_cliente	integer
Nombre_de_cliente	char(30)
Direcc1_de_cliente	char(30
Cirecc2_de_cliente	char(30)
Ciudad_de_cliente	char(16)
Edo_cliente	char(2)
Telefono_cliente	char(15)
Tax Amount	decimal(7,2)
Miscelaneos	decimal(7,2)

Factura_en_linea	
Numero_de_factura(PK)(FK)	integer
Numero_de_articulo(PK)	integer
Cantidad_de_articulos	integer
Descrip_de_articulo	char(30)
Precio_comer_articulo	decimal(7,2)
Precio_neto_articulo	decimal(7,2)
Numero_de_almacen	integer
Nombre_de_almacen	char(30)
Dir_de_almacen	char(30)
Ciudad_almacen	char(16)
Telefono_almacen	char(10)
Numro_de_proveedor	integer
Nombre_de_proveedor	char(30)
Dir_de_proveedor	char(30)
Ciudad_proveedor	char(16)
Telefono_proveedor	char(15)

Los datos repetidos se han trasladado de la tabla factura a la tabla factura en línea, estos dos elementos están en la primera forma normal(1NF).

La clave principal de factura_en_línea, es una clave compuesta por el número de factura y por el número de artículo. Cada tabla debe tener una sola clave principal. Si no se puede encontrar un identificador único, se creara uno. La clave principal se identifica al final del nombre de la columna.

LLAVES FORANEAS

Las claves foráneas juegan un papel vital en la identificación de la integridad referencial de una base de datos. Una base puede restringir a alguien la posibilidad de cancelar una fila de una tabla si el valor de la clave principal de la fila esta referido al valor de una clave foránea de otro tabla, alternativamente una base de datos podrían producir una cancelación en cascada, es decir, todas las filas referidas a la fila principal serian canceladas también. Sin la integridad referencial podrían aparecer fácilmente valores no válidos de la clave foránea. Cuando se tienen valores no válidos en la clave foránea, se tiene inconsistencia y errores dentro de la base de datos.
Las claves foráneas se identifican con (FK), he aquí algunas consideraciones importantes:

- Se debe escoger una clave principal que solamente identifique una fila de la tabla.
- No se deberán incluir mas columnas en la clave principal que las necesarias para identifica solamente una fila en la tabla.

Ahora que se han normalizado elementos(conforme a 1NF), se puede proceder con el siguiente poso de normalización

Segunda forma normal (2NF) Una tabla esta en segunda forma normal (2NF), solo si esta en primera forma non-key atributos es completa y funcionalmente dependiente de la clave principal (todos los non-key atributos, deben ser dependientes de la clave principal).
Para cumplir las restricciones de la segunda forma normal, se crea una nueva tabla: artículo, y se trasladan

todos los atributos que no son completa y funcionalmente dependientes de la clave principal de la facrura_en_línea desde la factura_en_línea a la tabla articulo.

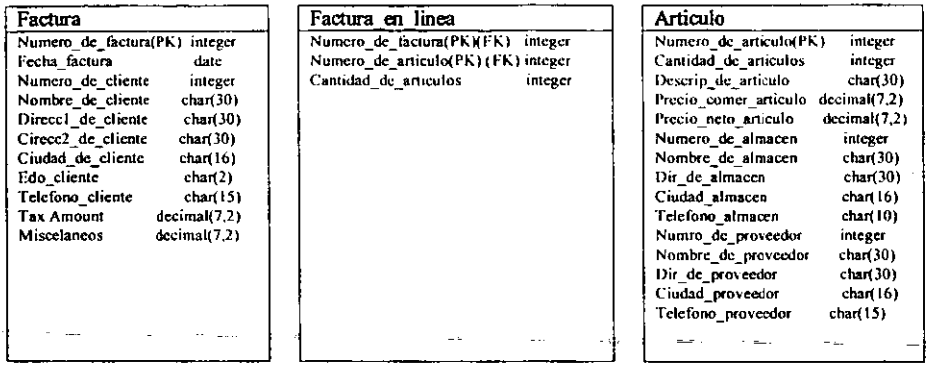
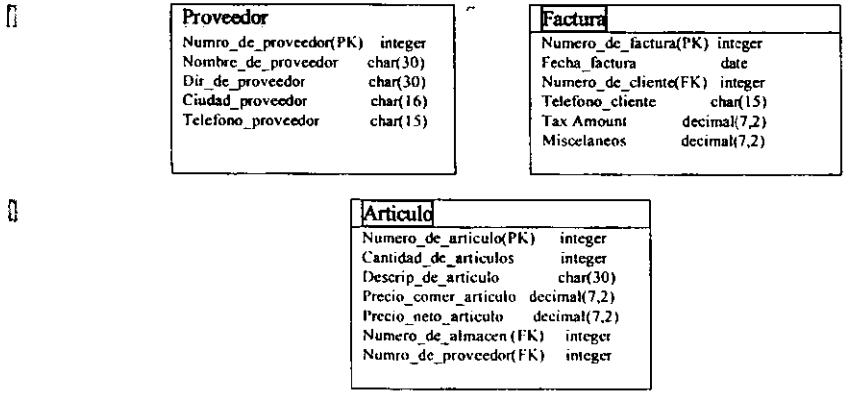
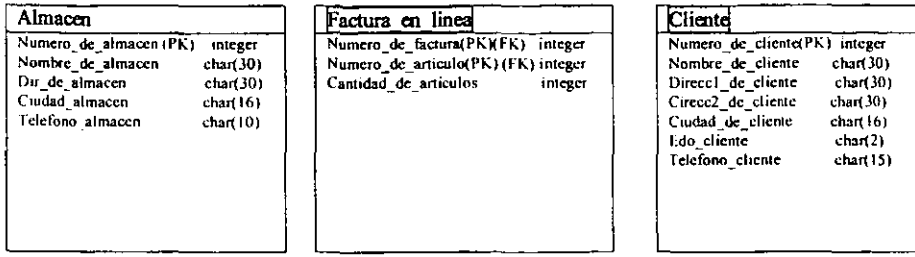


Fig. 3.4 Se han trabsladado los atributos desde la tabla factura_en_línea a la tabla artículo, consiguiendo que ambos elementos estén conforme a la segundo forma normal (2NF).

Tercera forma normal(3NF) Una relación esta en tercera forma normal(3NF) si esta en segunda(2NF) y no contiene ninguna dependencia transitiva. Las dependencias transitivas son dependencias funcionales entre non-key atributos. Cualquier atributo sin clave que sea funcionalmente dependiente de otro atributo sin clave de la misma tabla genera una dependencia transitiva y debe ser transladado a una nueva tabla.
En este ejemplo, los atributos del cliente en la tabla factura, son funcionalmente dependientes del número de cliente, que no forma parte de la clave principal. El número de cliente, es funcionalmente dependiente del numero de factura, haciendo a los atributos del clientetales como el nombre y la dirección, transitivamente dependientes del número de factura.
Se deberán trasladar todos los atributos del cliente en la tabla factura s una nueva tabla cliente(con el nombre de cliente como clave principal). El número de cliente continuara en la tabla factura como la clave foránea, también deberían trasladares los atributos de las tablas almacén y proveedor en la tabla de artículo a un nuevo elemento como se muestra en la siguiente gráfica.

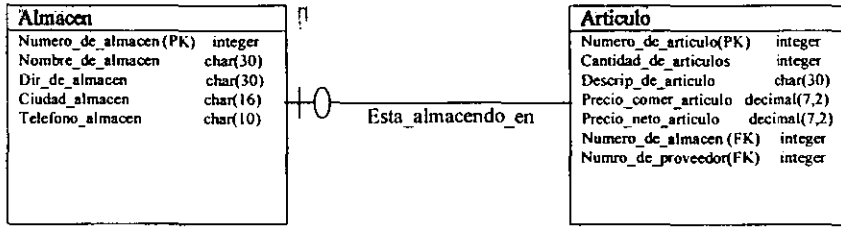




Aquí se han añadido estos tres nuevos elementos(cliente, proveedor, almacén) para transladar las dependencias transitivas desde los elementos factura y articulo.

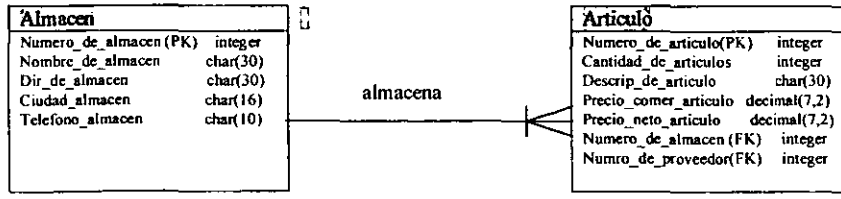
AFINACIÓN DE LA CARDINALIDAD

Después de haber normalizado el modelo de datos, hay que revisar y afinar las reglas del negocio y la cardinalidad. Esta propiedad ayuda a documentar las reglas mas importantes del negocio, la siguiente gráfica es una representación de cardinalidad.



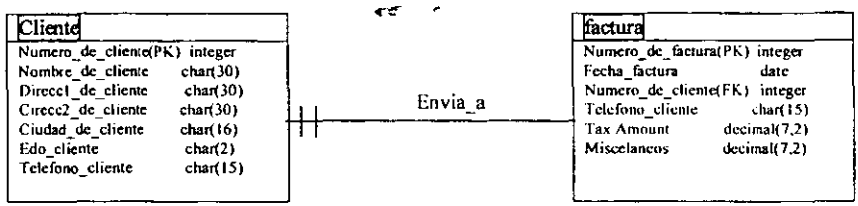
Articulo tiene una relación uno o ninguno con almacén.

Esta representación indica que cada articulo tiene ninguno o un solo almacén asocido con el, lo que se denomina relación uno o ninguno. Con frecuencia, los elementos tienen grandes campos de texto opcionales, tales como comentarios, separandolos en una tabla asociada con una relación uno-o-ninguno entre la primera tabla y sus comentarios.



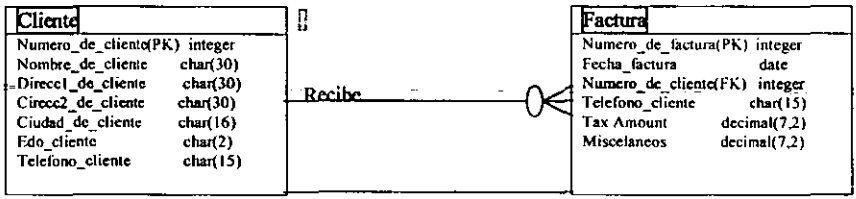
Almacén tiene una-o-muchas relaciones con articulo.

Esta gráfica indica que cada almacén contendrá uno o mas artículos. Como otro ejemplo, una factura relaciona una-o-muchas facturas_en_línea. Por lo tanto factura tiene una-o-muchas relaciones con factura_en_línea.



Factura tiene una-o-muchas relaciones con articulo.

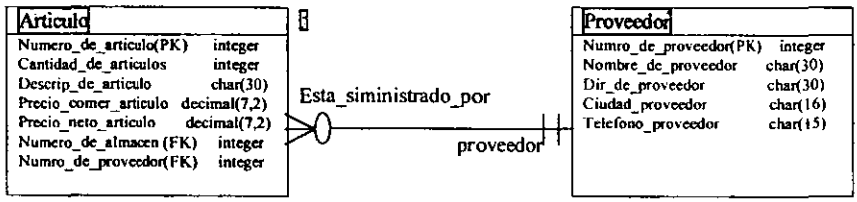
Esto indica que una factura se envía a uno y solo un cliente. Por lo tanto articulo tiene una relación uno-y-solo-uno con proveedor.



Cliente tiene uno-o-muchas relaciones con factura.

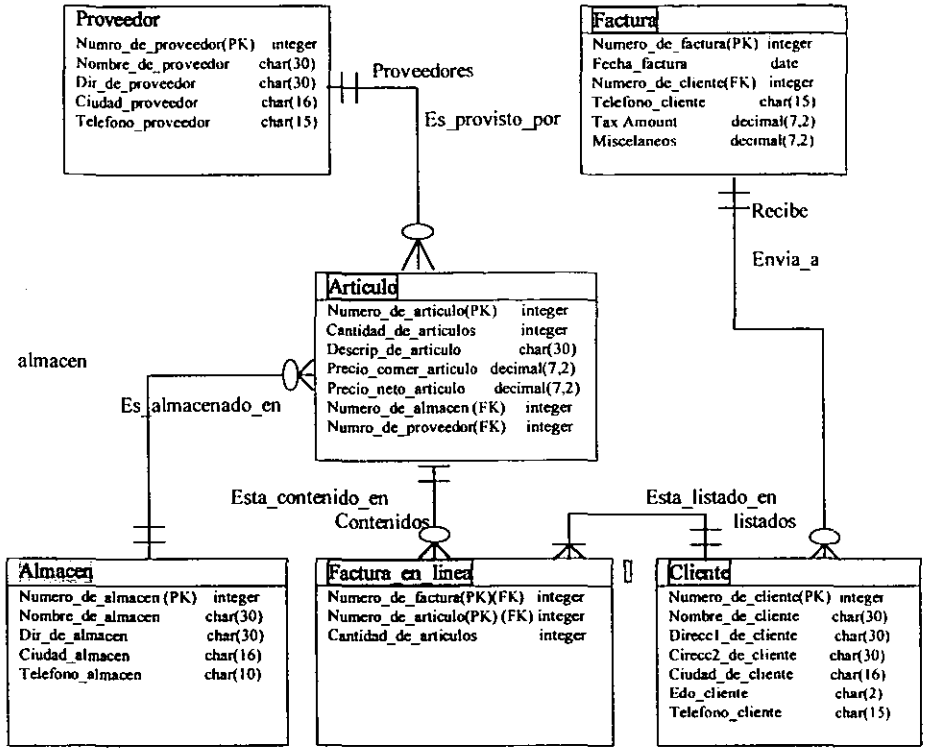
Esto indica que cada réplica de cliente puede recibir ninguna o muchas facturas. Por ejemplo, un suministrador podría vender ninguno-o-muchos artículos.

Aplicación de cardinalidad a modelos de datos Se puede definir cardinalidad en ambos lados de cadarelación entre elementos, como en la siguiente fig.



Se ha aplicado cardinalidad tanto en articulo como en proveedor

En la siguiente gráfica se añade cardinalidad a todo el modelo de datos, se podrá notar que proveedor y artículo tienen una relación muchos a uno (cada suministrador puede suministrar muchos artículos, pero cada articulo debe ser suministrado por un solo suministrador). En cuanto que factura y artículo tienen una relación muchos-a-muchos (cada factura puede contener muchos artículos y cada articulo puede estar en muchas facturas). No obstante, almacén y artículo tienen una relación uno-a-muchos (cada almacén puede contener muchos artículos, pero cada artículo solo puede estar en un almacén de acuerdo con los requerimientos).



Ahora se incluyen todas las cardinalidades

Aunque hay ahora más elementos que cuando se empezó, cada tablas es más sencilla que la tabla factura lo era al principio. El resultado es que hay menos redundancia que antes. La tabla factura_en_linea es una tabla asociativa (o de unión). Si no existiera en el modelo de datos, habría una relación muchos-o-muchos entre factura y articulo, con la creación de una tabla asociativa se divide una relación muchos-o-muchos. Casi siempre los elementos asociados tienen claves principales compuesta, en la que cada atributo de la clave principal es también una clave foránea para cada uno de los elementos en la relación original muchos-o-mucho.

Es posible normalizar en exceso una base de datos, y, si lo es, qué se debe hacer? . Una base de datos normalizada en exceso puede sufrir algunos problemas de rendimiento. Cuando ocurre esto, los diseñadores de la base de datos suelen desnormalizar un modelos de base de datos, cambiando dos o mas elementos y reducir el número de uniones de tabla en ciertos tipos de consultas, esto significa que es importante además de comprender la estructura de los datos, es importante también conocer las relaciones entre los elementos, así como la forma en que se accede y se actualizan los datos.
La desnormalización puede causar mas problemas de los que resuelva si no se realiza con cuidado. Los siguientes son algunos consejos para la desnormalización:

. No se debe nunca desnormalizar sin realizar antes un modelo y saber cómo la forma normalizada del modelo de datos puede afectar el rendimiento. Hay que asegurarse de que desnormalizar elementos no es una simple excusa para evitar el pasar por el proceso de normalización.

. Todos los elementos deben tener una clave principal única. Hay que asignar siempre un único para la tabla u asegurarse de que no se tienen atributos adicionales en la clave que no sean absolutamente necesarios para identificar unitariamente las filas de una tabla.
. La desnormalización de grupos repetidos se hace algunas veces cuando esta limitada la repetición de atributos. Por ejemplo, con frecuencia se guarda varios números de teléfono de un mismo cliente; se puede considerar que estos números son un atributo repetido, o se podría definir un conjunto limitado de atributos identificados separadamente, tales como el teléfono de la oficina, el teléfono de fax, el del domicilio, etc. Así definiendo atributos identificados separadamente se puede limitar el diseño, pero separar los teléfonos en una tabla aparte para resolver el problema de la repetición de atributos, esto significa que siempre habrá que unir la tabla cliente a la tabla teléfono para obtener todos los teléfonos de un cliente.
. Las desnormalizaciones de dependencias transitivas son utilizadas, algunas veces, para ingresar dos elementos que tienen una relación uno-o-uno, o en elementos donde todos los atributos transitivamente dependientes serian parte de una clave principal si se trasladaran a una tabla separada.

CREACIÓN DE UNA BASE DE DATOS ORACLE

Los procedimientos que se presentan en este capítulo pueden utilizarse para crear una base de datos que se ajuste a las normas de las disposiciones lógicas y físicas discretas, la creacion de una base de datos oracle se ayuda de algunos archivos de disco depositados en un directorio de unix, los archivos son:

- **cr_CC.sql,** archivo de SQL que se llama desde SQLDBA y que crea de hecho la base de datos.
- **initcc10.ora,** Archivo unit.ora inicial para la creación de las base de datos.
- **initcc1.ora,** archivo unit.ora de producción para la base de datos.
- **configcc1.ora,** archivo config.ora tanto para producción como para la creación inicial de la base de datos(en caso de que no haya allacenamiento, deberá crearse una versión aparte para producción)

Nota
Estos archivos podrian cambiar de nombre de acuerdo a la versión de oracle que se este empleando.

Antes de ejecutar el archivo cr_CC.sql, el usuario debe haber especificado los componentes lógicos de la base de datos(variable de entorno). En unix, esto supone definir la variable de netorno ORACLE_SID , y la variable ORACLE_HOME en el directorio raíz de oracle, el archivo cr_CC.sql debe ejecutarse desde el directorio $ORACLE_HOME/dbs.
A continuación se verán los pasos necesarios para la crear una base de datos que se ajuste a las reglas de diseño de bases de datos proporcionadas como en el capítulo anterior.

CREACIÓN INICIAL DE LA BASE DE DATOS ORACLE

1.- Crear la base de datos RX , especificando el arvhivo de datos del tablespace SYSTEM y los redologs en línea.
2.- Crear los catalogos de oracle.
3.- Crear un segmento de rollback (r0) en el tablespace SYSTEM.
4.- Modificar el segmento de rollback (r0) para hacerlo disponible. Podrian crearse entonces otros tablespaces.

Creción de nuevos tablespaces

5.- Crear un tablespace RBS para los segmentos de rollback adicionales.
6.- Crear un tablespace TEMP para los segmentos temporales.
7.- Crear un tablespace TOOLS para las tablas de los productos de las bases de datos.
8.- Crear un tablespace TESTS para la comprovación de las aplicaciónes y supervisión de RX.
9.- Crear un tablespace INDEX para la aplicación de supervisión de RX y deberá llamarse CCINX.

Creación de segmentos de rollback

10.- Crear los segmentos de rollback (r1, r2) en el tablespace RBS. Para otras aplicaciones será necesario crear otros segmentos de rollback.
11.- Desactivar el segmento de rollback (r0) en el tablespace de SYSTEM.

Modificación de los usuario SYS y SYSTEM

12.- Modificar la desigmación del tablespace temporal de los usuario SYS y SYSTEM para que lo utilice TEMP.
13.- Modificar el tablespaces por omisión del usuario SYSTEM para que lo utilice TOOLS.
14.- Crear los sinonimos del DBA para el usuario de SYSTEM.

Guión de creación de la base de datos RX para orcle cr_CC.sql ubicado en $ORACLE_HOME/dbs. Personalizar las ubicaciones de los archivos según las necesidades $ORACLE_HOME = /orasw/v7. Activar la salida de las terminal y la variable echo de las ordenes $ORACLE_SID = SID, sid es la variable de instancia en la parte de unix.

Paso 1. **Creación inicial de la base de datos**

Crear la base de datos, los archivos control se crean de forma automática cuando se especifica en el archivo initcc10.ora.

```
$ echo ORACLE_SID= SID
$ export ORACLE_SID
```

```
$connect internal
$ startup pfile= /orasw/v7/dbs/initcc10.ora nomount
```

```
SQLDBA> create database RX
                Maxinstances 1
                Maxlogfiles 16
                  Datafile
                   '/db01/oracle/RX /sys01.dbf' size 2M reuse,
                   logfile
                    '/db05/oracle/RX/redo01RX.dbf' size 2M reuse,
                    '/db05/oracle/RX/redo02RX.dbf' size 2M reuse,
                    '/db05/oracle/RX/redo03RX.dbf' size 2M reuse;
```

Analisis

Creación de la base de datos RX , usa directrices de configuración del tablspace SYSTEM de proposito general, tanbién directrices de archivos redulogs, los cuales deben ser mas de 3 para reducir las esperas por registro, ademas utilizan 2M por cada redulog. En este momento la base de datos ya debe de haber arrancado con: pfile= /orasw/v7/dbs/initcc10.ora

Paso 2. **Crear el catálogo(diccionario de datos) oracle**

Se refiere a instalar las vista del diccionario de datos.
@/orasw/v7/rdbms/admin/catalog.sql

Paso3. **Crear un segmento de rollback (r0) en SYSTEM**

Creación de segmentos de rollback adicionales en SYSTEM

```
$connect internal
SQLDBA> create rollback segment r0 tablespace system
            Storage (initial 16K next minextents 2 maxextents 20);
```

Paso 4. **Modificar el nuevo segmento de rollback para hacerlo disponible. A partir de aquí, podrian crearse mas tablespaces**

Utilizar ALTER ROLLNACK SEGMENT ONLINE para poner r0 en linea sin tener que apagar y volver a arrancar la base de datos.

```
SQLDBA> alter rollback segment online;
```

Paso 5. **Creación de nuevos tablespaces**

Crear un tablespace RBS para los segmentos de rollback, crear también directrices de configuración de los segmentos de rollback, un segmento de rollback por cada 4 transacciones concurrentes, es lo mas comvenente, todos los segmentos de rollback del mismo tamaño.

```
Create  tablespace rbs
        datafile
             /'db02/oracle/RX/rbs01.dbf' size 10M reuse,
             /'db02/oracle/RX/rbs02.dbf' size 10M reuse,
             /'db02/oracle/RX/rbs03.dbf' size 10M reuse
        default storage (
             initial   1M
             next      1M
             pctincrease  0
             miniextents  9
             maxextents  50);
```

Paso 6 **Crear un tablespace TEMP, para los segmentos temporales**

Crear un tablespace para los segmentos temporales con sus respectivas directrices de configuración.

```
SQLDBA> create tablespace temp
                datafile
                    /'db05/oracle/RX/temp1.dbf' size 15M reuse
                    default storage (
                     initial     500K
                     next        500K
                     pctincrease 0);
```

Paso 7. **Crear un tablespace TOOLS para las herramientas del producto**

Crear un tablespace para las herramientas de la base de datos.

```
SQLDBA> create tablespace tools
                datafile
                    /'db05/oracle/RX/tools01.dbf' size 20M reuse;
```

Paso 8. **Para la aplicación de supervision**

Crear un tablespce TESTS para utilizarlo para comprovar las modificaciones

```
SQLDBA> create tablespace tests
                    datafile
                        /'db05/oracle/RX/tests01.dbf' size 30M reuse;
```

Paso 9. **Crear un tablespace DATA**

Crear un tablespace DATA para la aplicación de supervisión de RX y se llamara RX

```
SQLDBA> create tablespace data
                    datafile
                    /'db03/oracle/RX/rx.dbf' size 30M reuse;
```

Paso 10. **Crear un tablespace INDEX**

Crear un tablespace para la aplicación de supervisión de RX, se llamara ccindex

```
SQLDBA> create tablespace index
                    datafile
                    /'db04/oracle/RX/ccindex.dbf' size 20M reuse;
```

Paso 11. **Creación de segmentos de rollback adicionales**

Creación de segmentos de rollback en el tablespace RBS, para otras aplicaciones

```
SQLDBA> create rollback segment r1 tablespace rbs
                        Storage (optimal 9M);
SQLDBA> create rollback segment r2 tablespace rbs
                        Storage (optimal 9M);
```

Paso 12. **Desactivar el segmento de rollback r0 en el tablespace de SYSTEM**

Utilizar ALTER ROLLBACK SEGMENT ONLINE para poner en línea los segmentos de rollback sin apagar y volver a arrancar la base de datos

```
SQLDBA> alter rolback segment r1 online;
SQLDBA> alter rollback segment r2 online;
```

Como se han creado y puesto en línea 2 segmentos de rollback (r1, r2), ya no se necesita r0 del tablespace de SYSTEM . Debera ponerse fuera de línea, pero dejandolo en el tablespace de SYSTEM, de esa forma podra ponerse en línea nuevamente en caso de que RBS tenga que ponerse fuera de línea.

```
SQLDBA> alter rollback segment r0 offline;
```

Paso 13. Modificar las designaciones del tablesapce temporal(TEMPORARY TABLESPACE) de los usuarios SYS y SYSTEM para que utilicen el tablepace temporal(TEMPORARY)

Paso 14. Modificar el tablespace el DEFAULT TABLESPACE del usuario SYSTEM para que utilice el tablespace TOOLS(herramientas).

```
SQLDBA>alter user sys temporary tablespace temp;
SQLDBA> alter user system quota 30M on tools
            Default tablespace tools temporary tablesapce temp;
```

Paso 15. Crear los sinonimos del DBA para el usuario SYSTEM.

```
SQLDBA> connect system/manager

@/orasw/v7/rdbms/admin/catdbsyn.sql

spool off
```

Para metros temporales del archivo init.ora

Este archivo se llama en el paso 1 del guion cr_cc1.sql. Se utiliza para montar la base de datos. Solo se utilixa durante la craación de la base de datos. Debe almacenarse en el directorio $ORACLE_HOME/dbs

```
#Archivo initcc10.ora de oracle7, utilizado durante la creacion de la base de datos
#Archivo: initcc10.ora
#Ubicación: $ORACLE_HOME/dbs
#
#Incluir los parametros de configuracion de la base de datos
    pfile                          = /orasw/v7/dbs/configcc1.ora
#
    rollback_segments              = ()
    db_files                       = 60
    db_file_multiblock_read_count  = 8
    db_block_buffers               = 500
    shared_pool_size               = 350000
    log_checkpoint_interval        = 1000
    processes                      = 50
    dml_locks                      = 100
     log_buffers                   = 8192
    sequences_cache_entries        = 10
    sequences_cache_hash_buckets = 10
#audit_trail_                      = true
#timed_statistics                  = true
     max_dump_file_size            =  10240
#log_archive_start                 = true
mst_dispatchers                    = " IPC, 1"
mts_max_dispatchers                = 10
mts_servers                        = 1
mts_max_servers                    = 10
mts_service                        = dev7
mts_listener_address               = "(ADDRESS=(PROTOCOL=ips) (KEY = dev7))"
```

Para metros del producto (Archivo unitcc1.ora)

Este archivo no se llama de manera directa durante el proceso de creación de la base de datos. Es el archivo INIT.ORA que se utiliza durante el arranque de la aplicación de supervisión de la base de datos. En su nombre se incluye el nombre de la instancia que se utiliza para CC1. Para utilizarlo en una instancia, debera cambiarsesu nombre para que utilize el nombre de esa instancia en su nombre de archivo.
Este archivio debe almacenarse en el directiorio $ORACLE_HOME/dbs. Debe tenerse en cuenta que contieene entradas para los dos segmentos de rollback (r1 y r2) creados durante el proceso de creación de la base de datos. Si se utiliza un conjunto de segmentos de rollback distintos, habra que modificar la linéa <<rollback segs>> para que refleje sus nombres.

```
#Archivo initcc1.ora de oracle7, Archivo INIT.ORA de producción para  CC1
#Archivo: initcc10.ora
#Ubicación: $ORACLE_HOME/dbs
#
#Incluir los parametros de configuracion de la base de datos
     pfile                              = /orasw/v7/dbs/configcc1.ora
#
#    rollback_segments                  = (r0)
     rollback_segments                  = (r1,r2)
     db_files                           = 60
     db_file_multiblock_read_count      = 8
     db_block_buffers                   = 500
     shared_pool_size                   = 350000
     log_checkpoint_interval            = 1000
     processes                          = 50
     dml_locks                          = 100
      log_buffers                       = 8192
     sequences_cache_entries            = 10
     sequences_cache_hash_buckets       = 10
#audit_trail_                           = true
#timed_statistics                       = true
      max_dump_file_size                =  10240
#log_archive_start                      = true
mst_dispatchers                         = " IPC, 1"
mts_max_dispatchers                     = 10
mts_servers                             = 1
mts_max_servers                         = 10
mts_service                             = dev7
mts_listener_address                    = "(ADDRESS=(PROTOCOL=ips) (KEY = dev7))"
```

Configuración de la base de datos(Archivo configcc1.ora)

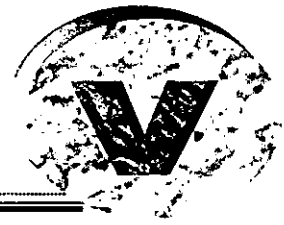
En este erchivo se especifican los valores de los parametros de la nase de datos que no suele modificarse. Lo llama archivo initcc10.ora durante el paso 1 de la creación de la base de datos. También lo llaman archivo initcc1.ora que se utiliza en producción. Este archivo debe encontrarse en el directorio $ORACLE_HOME/dbs, el cual debera tener en cuenta que en su nombre se incluye la instancia. Para admitir una instancia habra que modificar el parametro << db_name>> al nuevo nombre de la base de datos.

```
#Archivo configcc1.ora de configuración de oracle7
#
#Archivo :                          = configcc1.ora
#ubicación:                         = $ORCLE_HOME/dbs
#
control_files                       = (db01/oracle/CC1/ctrl1cc1.ctl,
                                       db02/oracle/CC1/ctrl2cc1.ctl,
                                       db03/oracle/CC1/ctrl3cc1.ctl)
background_dump_dest                =  /orasw/v7/rdbms/log
core_dump_dest                      = /orasw/v7/dbs
user_dump_dest                      = /orasw/v7/rdbms/log
log_archive_dest                    = /orasw/v7/arch/arch
db_block_size                       = 2048
db_name                             = CC1
```

OBSERVACIONES

Los procedimientos que se han presentado en esta capitulo pueden utilizarse para crear una base de datos que se ahuste a las normas de disposición lógica y fisica de sus usuarios. Este guión proporciona el compromiso de obtener un mejor rendimientos de su aplicación. Los codigos que se presentan permiten crear la base de datos como centro de información y estan basados en SQL. Los nombres de los archivos de disco y de los directorios son especificos de UNIX
El guion principal de creación de bases de datos se denomina cr_CC1.sql, este prmite crear una base de datos cuyo nombre de instancia esta dado por el DBA. Para especificar un nombre deinstancia distinto, habrá que cambiar el nombre del archivo de creación de la base de datos para que utilice el nombre del nuevo archivo(por ejemplo rc_nuevo_nombre.sql). Se deberan cambiar después todas las referencias que se hagan a CC1 en los siguientes guiones para que utilicen el nuevo nombre de la instancia. También deberá renombrarse el nombre de la instancia en el parametro db_name del archivo INIT.ORA y así sucesivamente con cada uno de los parametros que que se encuentrane en este archivo y demas archivos de conectividad(CONFIG.ORA, TNSNAMES Y LISTENER

Capitulo



TIPS DE ADMINISTRACIÓN ORACLE

INTRODUCCION

Tanto los DBA(Administradores de bases de datos) experimentados como los DBA recientes o los desarrolladores de aplicaciones, necesitan conocer la forma en que trabajan e interactuan las estructuras internas de la base de datos oracle. Una adecuada gestión de las estructuras de la base de datos permitirá que esta cumpla los propósitos deseados: Que funcione y que funcione bien.
En este tema se encontrara la información necesaria para alcanzar todos los objetivos planeados. Para que esto sea posible debemos enfatizar la gestión de las capacidades de la base de datos de forma eficaz y eficiente para obtener un producto de calidad. El resultado final debe ser una base de datos seria, robusta, segura y extendible, su diseño debe cumplir los objetivos de las aplicaciones que admite. Varios componentes son esenciales para este objetivo, y todos ellos se han tratado con profundidad. Una arquitectura lógica y física de una base de datos bien diseñada mejora el rendimiento y facilitara la administración mediante la adecuada distribución de los objetos de la base de datos. La determinación del número y tamaño correcto de los segmentos de rollback(anulación) permitirá que la base de datos admita todas las transacciones, también contemplaremos los respectivos tablespace que irán asociados con los segmentos de rollback. Veremos como hacer uso del diccionario de datos y de toda una serie de conceptos.

PONER EN MODO DE LINEA EL SERVER MANAGER

$orapwd file = $ORACLE_HOME/dbs/orapwu15

password = admin entries = 5

svrmgrl command = @credb.sql

ó

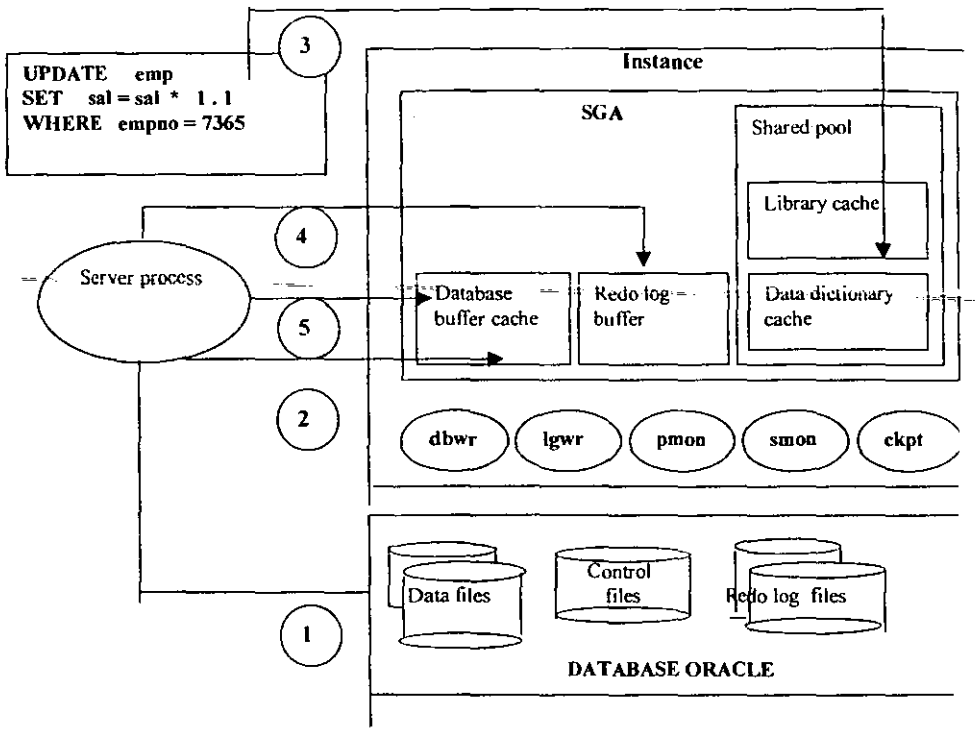
svrmgrl > CONNECT system/manager

svrmgrl > CONNECT / INTERNAL

vrmgrl > STARTUP pfile = /DISCK1/initU15.ora

TIPS PARA ADMINISTRAR BASES DE DATOS ORACLE

PROCESO DE FUNCIONAMIENTO



Esta es la manera en que ORACLE manipula los datos en nuestras tablas dentro de una base de datos

TABLESPACE

Es una división lógica de la base de datos, cada base de datos tiene al menos un tablespace de tablas llamado System, este tablespace agrupa todas las tablas, pueden utilizarse otros tablespace para agrupar aplicaciones y facilitar el mantenimiento y mejorar así el rendimiento, como ejemplo de estos tablespaces consideramos un tablespace USERS de uso general, y RBS(rollback segment) para los segmentos de rollback; Un tablespace de tablas solo puede pertenecer a una base de datos.

Sintaxis:

```
CREATE TABLESPACE tablespace
        DATAFILE filespec [ autoextent_clause ]
        [ ,      filespec [ autoextent_clause ] ] ...
        [ MINIMUM EXTENT integer [ K | M ] ]
        [ DEFAULT storage_clause ]
        [ PERMANENT | TEMPORARY ]
        [ ONLINE | OFFLINE ]
STORAGE ( [ INITIAL integer [ K | M ] ]
          [ NEXT    integer [ K | M ]
          [ MINEXTENTS     integer ]
          [ MAXEXTENTS     { integer | UNLIMITED } ]
          [ PCTINCREASE    integer ] )
```

Donde:

Tablespace	Es el nombre del tablespace a ser creado
DATAFILE	Contiene los datos que necesita el tablespace
DEFAULT STORAGE	Contiene los parámetros para todos los objetos creados en el tablespce
MINIMUM EXTENT	Se asegura de que todos los extents usados por el tablespace, sean enteros
ONLINE	Hace que el tablespace este disponible una vez creado
OFFLIN E	Deshabilita el tablespace una vez creado
PERMANENT	Especifica que tablespace puede ser usado de manera permanente por todos los Objetos
TEMPORARY	Especifica que el tablespace va a ser usado de manera temporal por los objetos

Ejemplo:

```
CREATE TABLESPACE app_data
   DATAFILE '/DISCK4/app01.dbf' SIZE 100M,
            '/DISCK5/app02.dbf' SIZE 100M
   MINIMUM EXTENTS 500K
   DEFAULT STORAGE ( INITIAL 500K NEXT 500K
   MAXEXTENTS 500 PCTINCREASE 0 );
```

Podemos modificar un tablespace

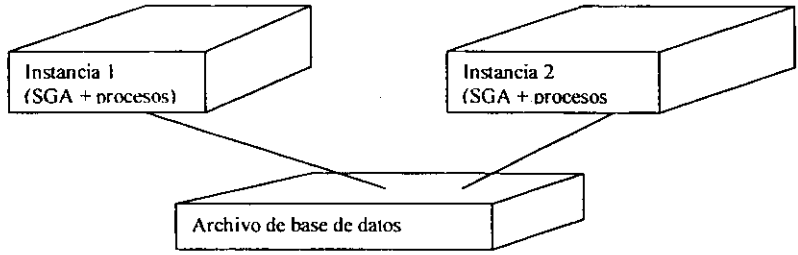
```
ALTER TABLESPACE app_data
ADD DATAFILE '/DISCK3/app03' SIZE 200M
AUTOEXTEND ON NEXT 10M
MAXSIZE 500M;
```

Borrar un tablespace

```
DROP TABLESPACE app_data
INCLUDING CONTENTS;
```

INSTANCIA

Para acceder a la base de datos, oracle utiliza un conjunto de procesos de fondo compartido por todos los demas usuarios. A demás existen ciertas estructuras de memoria que se utilizan para almacenar los datos mas consultados, estas áreas de memoria ayudan a mejorar el rendimiento de la base de datos al disminuir la cantidad de operaciones de acceso a los archivos de datos.



ELEMENTOS INTERNOS DE LA BASE DE DATOS

Tablas, columnas, tipos de datos y restricciones
Usuarios y esquemas
Indices, grupos y grupos hash
Vistas
Secuencias
Procedimientos, funciones, paquetes y triggers
Sinónimos
Privilegios y funciones(roles)
Enlaces de base de datos
Segmentos, Extensiones y blockes
Segmentos de rollback

TABLAS Y COLUMNAS

Las tablas son mecanismos de almacenamiento de datos dentro de la base de datos oracle, constan de un conjunto fijo de columnas que describen los atributos de la entidad que se maneja mediante la tabla cada columna tiene un nombre y características especiales.
Las tablas propiedades del usuario SYS se llaman tablas del diccionario de datos y proporcionan un catalogo que la base de datos utiliza para gestionarse a si misma; Las tablas se relacionan entre ellas mediante las columnas que tienen en común. Puede utilizar la base de datos para imponer esta relación por medio de la integridad referencial(references integrity), la integridad referencial se impone al nivel de la base de datos a través de las restricciones de las tablas.

Sintaxis:

```
CREATE  TABLE [ schema. ] table
   ( column-definition [ , column-definition ] ...
   [ , out-of-line-constrint [ , out-of-line-constraint ] ....] )
ORGANIZATION INDEX
[ PCTFREE integer ]
[ INITRNS integer ]
[ MAXTRANS integer ]
[ storage-clause ]
[ TABLESPACE tablespace ]
[ PCTTHRESHOLD integer ]
[ INCLUDING column ] ]
[ OVERFLOW segment_attributes_clause ]
```

Donde:

Schema	Es el propietario del cluster
Table	Es el nombre de la tabla
ORGANIZATION INDEX	Especifica que es una tabla con indice organizado
PCTTHRESHOLD	Representa el porcentaje de espacio reservado en los bloques del índice
INCLUDING	Define una columna con la cual se divide la tabla
OVERFLOW	Indica cual es el índice organizado de la tabla

Ejemplo:

```
CREATE TABLE  scott.sales
       ( office_cd  NUMBER(3),
         qtr_end    DATE,
         Revenue    NUMBER(10, 2),
         Review     VARCHAR(1000),
           CONSTRAINT  sales_pk
             PRIMARY  KEY( office_code, qtr_end ))
ORGANIZATION  INDEX TABLESPACE  data01
PCTTHSHOLD 20
OVERFLOW TABLESPACE data02;
```

Cambiar la estructura de una tabla

```
ALTER  TABLE scott.employees
ENABLE VALIDATE  CONSTRAINT  emp_dept_fk;
```

Borrar una tabla

```
DROP  TABLE   scott.employees;
```

Tipos de datos

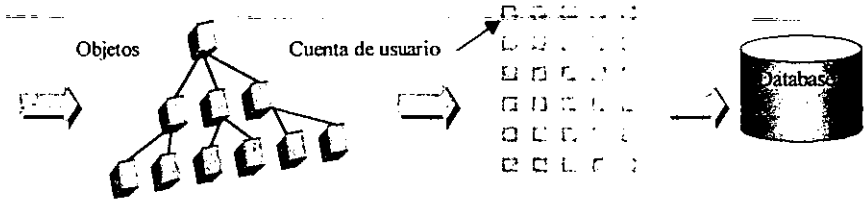
- Char — Campo de longitud fija, con una longitud máxima de 255 caracteres.

- Varchar — En la actualidad idéntico a Varchar2, aunque su funcionalidad puede cambiar en versiones futuras, por tanto debe utilizarse Varchar2 para almacenar cadenas de caracteres de longitud variable.

- Varchar2 Campo de longitud variable, con una longitud máxima de 2000 caracteres.
- Date Campo de una longitud de siete bytes que se utiliza para almacenar todas las fechas . La hora se almacena como parte de la fecha . En la consultas, la fecha aparece en el formato Dia-Mes-Año, como por ejemplo, 13-Mar-2000.
- Number Columna numérica de longitud variable. Los valores permitidos son cero y números positivos y negativos desde 1.0E-130 hasta 9.99.. E125.
- Long row Campo de longitud variable que se utiliza para datos binarios con una longitud máxima de 2G .
- Mlslabel Solo para trusted Oracle. Este tipo de datos utiliza entre dos y cinco bytes por fila.

ESQUEMAS

Es un conjunto de objetos que posee la cuenta de un usuario, es posible crear usuarios que no tengan la capacidad de acceder a la base de datos. Dichas cuentas de usuario proporcionan un esquema que puede utulizarse para mantener un conjunto de objetos de bases de datos separados de otros esquemas de los usuarios.



INDICES

En una base de datos relacional, la situación física de una fila es irrelevante(a menos que la base de datos tenga que encontrarla). Para que sea posible encontrar las datos, cada fila de cada tabla se etiqueta con un ROWID (IDENTIFICADOR DE FILA). Este ROWID le indica a la base de datos el lugar exacto en que encuentra la fila(mediante el archivo, bloque dentro de ese archivo y fila dentro de ese bloque).
Un índice es una estructura de la base de datos que le permite al servidor localizar rápidamente una fila en una tabla. Existen dos tipos de índice, índices de grupo e índices de tabla.
Los índices contienen una lista de entradas cada una de las cuales consta de un valor clave y de un ROWID
El valor clave es el valor de una columna o la combinación de valores de varias columnas en una fila.

Sintaxis:

```
CREATE [ UNIQUE ] INDEX [schema. ] index
ON [ schema. ] table
( column [ ASC | DESC ] [ , column [ ASC | DESC ] ]...)
[ TABLESAPCE tablesapce ]
  [ PCTFREE integer ]
  [ INITRANS integer ]
  [ MAXTRANS integer ]
  [ storage – clause ]
  [ LOGGING | NOLOGGING ]
  [ NOSORT ] ————————→ Significa que la tabla ya esta ordenada
```

Ejemplo:

```
CREATE INDEX scott.emp_lname_idx
ON scott.employees(last_name)
PCTFREE 30
```

```
STORA<GE( INITIAL 200K NEXT 200K
PCTINCREASE 0 MAXEXTENTS 50)
TABLESPACE indx01
```

Modificar un indice

```
ALTER INDEX scott.emp_lname_idx
STORAGE( NEXT 400K
MAXEXTENTS 100);
```

La vista del diccionario de datos que contiene todas las caracteristicas de un indice, se llama DBA_INDEXES, para utilizarla solo debemos emplear el siguiente query:

```
SVRMGRL> SELECT index_name, tablespace_name, index_type,
                Uniqueness, status
                FROM dba_indexes WHERE owner = ' SCOTT ';
```

Borrar un indice

```
DROP INDEX scott.emp_lname_idx;
```

CLUSTER

Un cluster se usa para almacenar datos de diferentes tablas, es decir, se usa cuando se agrupan dos a mas tablas en un mismo espacio.

Sintaxis:

```
CREATE CLUSTER [ schema. ] cluster ( column dattype [ , column datatype ] ... )
    [ PCTFREE integer ]
    [ PCTUSED integer ]
    [ INITRANS integer ]
    [ MAXTRANS integer ]
    [ SIZE integer [ K | M ] ]
    [ storage-clause ]
    [ TABLESPACE tablespace ]
    [ INDEX ]
```

Donde:

schema	Es el propietario del cluster
cluster	Es el nombre del cluster
column	Es el nombre de una llave
data type	Es el tipo de dato y el tamaño de la llave
SIZE	Especifica el espacio requerido por todos los registros correspondientes al valor de la llave en bytes
INDEX	Especifica cual es el indice del cluster

Ejemplo:

```
CREATE CLUSTER scott.ord_clu
        ( ord_no NUMBER(3))
        SIZE 200 TABLESPACE DATA01
        STORAGE ( INITIAL 5M NEXT 5M PCTINCREASE 0 );
```

Cambiar un cluster

```
ALTER CLUSTER scott.ord_clu
   SIZE 300 STORAGE ( NEXT 2M );
```

Borrar un cluster

```
DROP CLUSTER scottord_clu
 INCLUDING TABLES;
```

PASOS PARA CREAR UN USUARIO

Sintaxis:

```
CREATE USER user
IDENTIFIED { BY password | EXTERNALLY }
[ DEFAULT TABLESPACE tablespace ]
[ TEMPORARY TABLESAPCE tablespace ]
[ QUOTA { integer [ K | M ] UNLIMITED } ON tablespace
   [ QUOTA [ integer [ K | M ] | UNLIMITED } ON tablespace ] .... ]
[ PASSWORD EXPIRED ]
[ ACOUNT { LOCK | UNLOCK }]
[ PROFILE { profile | DEFAULT }]
```

Donde:

User Es el nombre del usuario
BY password Controla la autenticidad del usuario
EXTERNALLY Especifica que el usuario este autorizado por system
DEFAULT/TEMPORARY/TABLESAPCE Especifica el tablespace temporal al cual se apuntara el usuario
QUOTA Define el espacio máximo permitido por el dueño del objeto
PASSWORD EXPIRED Tiempo que durara el password
ACOUNT LOCK/UNLOCK Lock se usa para bloquear, mientras que unlock es por default
PROFILE Se usa para controlar los recursos asignados a el usuario

Ejemplo:

```
CREATE USER rodrigo
IDENTIFIED BY bernal
DEFAULT TABLESPACE data01
TEMPORARY TABLESPACE temp
QUOTA 15M ON data01
PASSWORD EXPIRED;
```

Cambio de la estructura

```
ALTER USER rodrigo
IDENTIFIED BY jimenez
QUOTA 0 ON data01
PASSWORD EXPIRED;
```

Borrar un usuario

```
DROP USER rodrigo CASCADE;
```

La opción de CASCADE borra todos los objetos contenidos por el usuario.

GRUPOS

Las tablas a las que se suele acceder conjuntamente pueden almacenarse físicamente juntas. Para almacenarlas juntas se crea un grupo(cluster) que contenga las tablas. Las datos de las tablas se almacenan entonces juntos esto es para minimizar el numero de operaciones de entrada y salida y mejorar así el rendimiento.

Ejemplo:

ORD_NO	PROD	QTY
101	A4102	20
102	A2091	11
102	G7830	20
102	N9587	26
101	A5675	19
101	W0824	10

ORD_NO	ORD_DT	CUST_CD
101	05-Jun-2000	R01
102	07-Jun-2001	N45

```
Cluster Key
 ( ORD_NO )
101   ORD_DT      CUST_CD
        05-Jan-2000     R01
          PROD        QTY
          A4102       20
          A5675       19
          W082        10
102   ORD_DT      CUST_CD
        07-Jun-2001     N45
          PROD        QTY
          A2091       11
          G7830       20
```

GRUPOS HASH

Estos grupos(Hash cluster) utilizan funciones sobre la clave del grupo de la fila para determinar la posición física sobre la que se debe almacenar la fila. Así se obtiene los mayores beneficios en el rendimiento en consulta.

VISTAS

Una vista(view) tiene el mismo aspecto que una tabla con columnas y se consulta de la misma forma que esta, puede pensarse de una vista como una mascar que cubre una o mas tabla, las vistas no pueden indexarse ya que no hay datos físicas directamente asociados a ellas.

Ejemplo:

```
CREATE VIEW contacts_ca AS
  SELECT * FROM customers
  WHERE state = 'CA';
```

SECUENCIAS

Las definiciones de las secuencias(sequences), también se almacenan en el diccionario de datos. Las secuencias proporcionan una lista consecutiva de números exclusivos que sirven para simplificar las tareas de programación. La primera vez que se llama a una secuencia en una consulta, devuelve un valor predeterminado. En cada consulta subsiguiente a la secuencia se obtendrá un valor aumentado en el incremento especificado.

PAQUETES

Los paquetes(package) sirven para organizar los procedimientos y funciones en agrupamientos lógicos, sus definiciones se almacenan en el diccionario de datos, los paquetes son muy útiles en las tareas administrativas necesarias para gestionar los procedimientos y funciones. Los elementos de los paquetes pueden definirse como públicos y privados. Los elementos públicos están accesibles para los usuarios del paquete, mientras que los elementos privados están ocultos. Entre los elementos privados suelen encontrarse procedimientos a los que llaman otros procedimientos del paquete.

SINÓNIMOS

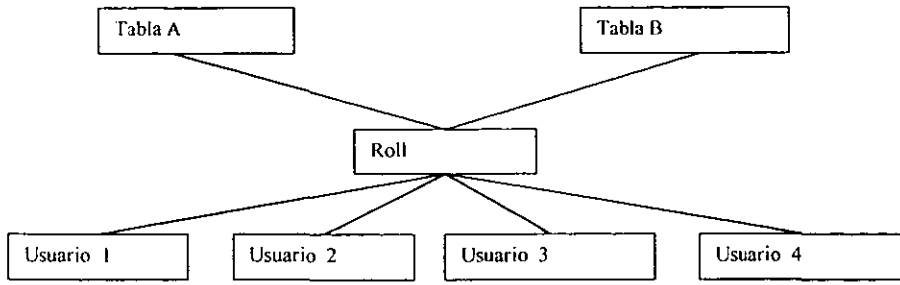
Para utilizar una identificación completa de un objeto de base de datos(como una tabla o una vista) en una base de datos distribuida, es necesario especificar el nombre de la maquina anfitriona, el nombre del servidor y el propietario del objeto a demás de el nombre, serán necesarios de entre uno y cuatro de estos parámetros en función del emplazamientos del objeto. Los desarrolladores pueden crear sinónimos que apunten al objeto adecuado para ocultar este proceso a los usuarios que solo tienen que conocer el nombre del sinónimo. Los sinónimos públicos los comparten todos los usuarios de una base de datos, mientras que los sinónimos privados pertenecen a propietarios de cuentas de base de datos individuales.
Por ejemplo, la tabla EMPLEADO descrita anteriormente, debe ser propiedad de una cuenta (digamos que el propietario es BJ). Desde otra cuenta de usuario de la misma base de datos, dicha tabla podría referenciarse como BJ.EMPLEADO. No obstante, esto implica que la segunda cuenta debe conocer que la tabla EMPLEADO es propiedad de la cuenta BJ. Para evitar esto, puede crearse un sinónimo público llamado EMPLEADO que apunte a BJ.EMPLEADO. Siempre que se haga referencia, este sinónimo apuntará a la tabla adecuada. La siguiente sentencia SQL permitirá crear dicho sinónimo.

CREATE PUBLIC SYNONYM empleado FOR bj.empleado;

PRIVILEGIOS Y ROLES

Para que una cuenta pueda acceder a un objeto propiedad de otra cuenta tiene que habérsele concedido el privilegio de acceso. Lo habitual es que a los no propietarios seles conceda el privilegio de **INSERT , SELECT, UPDATE, DELETE, EXECUTE** , no se conceden privilegios sobre los índices o los triggers ya que la base de datos los utiliza durante la actividad de la tabla. Los privilegios pueden concederse a los usuarios individuales o públicos.

En la siguiente figura se muestra la relación entre privilegios y funciones



ASIGNAR PRIVILEGIOS A UN ROLE

Ejemplo:

```
GRANT SELECT, UPDATE, DELETE, INSERT, CREATE TABLE, ALTER TABLE
        ON empleado TO rodrigo;
```

Podemos asignar todos los privilegios a un usuario

```
GRANT ALL ON empleado TO rodrigo;
```

Borrar privilegios a un usuario

```
REVOKE SELECT, UPDATE, DELETE, INSERT, CREATE TABLE, ALTER TABLE
    FROM rodrigo;

REVOKE ALL ON empleado FROM rodrigo;
```

Forma de crear un ROLE

```
CREATE ROLE rham IDENTIFIED BY password;
```

Asignar privilegios a un ROLE

```
GRANT CREATE TABLE, DROP TABLE, ALTER TABLE TO mi_role;
```

Asignar un ROLE a un usuario

```
GRANT mi_role TO rodrigo;
```

Asignar un ROLE a otro ROLE

```
GRANT mi_role TO mi_otro_role;
```

Mi_role es el role origen, mientras que mi_otro_role a donde voy asignar

ENLACE DE BASES DE DATOS

Las bases de datos oracle utilizan SQL*NET, tienen la capacidad de hacer referencia a datos almacenados fuera de la base de datos local, en estos casos debe especificarse el nombre completo del objeto remoto. Para especificar una vía de acceso a un objeto situado en una base de datos remota es necesario crear un enlace de datos(database linck), los enlaces de bases de datos pueden ser tanto públicos(disponibles para todas las cuentas de esa base de datos), privados(creados por el usuario para el uso exclusivo de esa cuenta).

Ejemplo:

```
CREATE database linck  mi_enlace
  CONNECT  TO usuario IDENTIFIED BY password
 USING 'T:143.4.1.8:A';
```

Donde:

mi_enlace	Es el nombre del dblinck o del enlace
T	Es el protocolo de comunicación (TCP/IP)
143.4.1.8	Es el host del equipo remoto
A	Es el nombre de la instancia de la base de datos remota

Forma en que podemos utilizar un enlace o dblink

```
CREATE  TABLE  customer
 AS
SELECT  *  FROM  cliente@mi_enlace
```

Este query crea una tabla llamada customer con todos los datos de una tabla remota llamada cliente. Con el siguiente ejemplo vamos a copiar una tabla de un equipo remoto a un equipo local.

```
COPY  FROM  scott / tiger@T:lrmxn03:B
CREATE  tiempo ( clave,  cve,  temp )
USING
SELECT  llave,  contraseña,  temperatura  FROM  temperatura
```

Donde:

scott/tiger	Usuario y password en la base de datos local
@	Es una regla de conectividad
T	Es el protocolo de comunicación
lrmxn03	Es el nombre del host local
B	Es el nombre de la instancia local
tiempo	Es el nombre de la tabla a generar
clave, cve, temp	Son los campos con los que se crea la tabla tiempo
llave, contraseña, temperatura	Son los campos que se desean de la tabla temperatura
temperatura	Es el nombre de la tabla remota

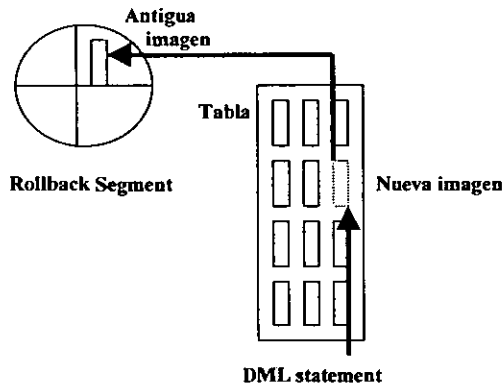
SEGMENTOS EXTENCIONES Y BLOQUES

Los segmentos(segments) son la contra partida física de los objetos lógicos de la base de datos. En los segmentos de índice por ejemplo: se almacenan los datos asociados con los índices . Para una gestión eficaz de los segmentos es necesario que el DBA conozca los objetos que una aplicación va a utilizar, la forma en que se van a introducir los datos en esos objetos y las maneras en que se van a recuperar.

Como un segmento es una entidad física, debe estar asignado a un tablespace de la base de datos. U n segmento consta de secciones llamadas Extensiones((extents) conjunto contiguo de bloques de oracle), cuando las extensiones existen en un segmento ya no pueden contener datos nuevos, el segmento consigue otra extensión, este proceso de extensión continua hasta agotar el espacio disponible en los archivos de datos del segmento, si un segmento consta de varias extensiones no se garantiza que dichas extensiones sean contiguas.

SEGMENTOS DE ROLLBACK

Para mantener la consistencia de lectura entre varios usuarios de la base de datos y poder anular las transacciones. Oracle debe disponer de un mecanismo de reconstrucción de una imagen anterior de los datos para transacciones no confirmadas. Oracle utiliza segmentos de rollback dentro de la base de datos para llevar acabo esta tarea.
Los segmentos de rollback crecen tan grandes como las transacciones lo admiten.



Sintaxis:

```
CREATE [ PUBLIC ] ROLLBACK SEGMENT rollback_segment
       [ TABLESPACE tablespace ]
       [ STORAGE ( [ INITIAL integer [ K | M ] ]
         [ NEXT    integer [ K | M ] ]
         [ MINEXTENTS    integer ]
         [ MAXEXTENTS { integer | UNLIMITED }]
         [ OPTIMAL  { integer [ K | M ] | NULL }] ) ]
```

Ejemplo:

```
CREATE ROLLBACK SEGMENT rbs01
TABLESPCE rbs
STORAGE (
INITIAL 100K NEXT 100K OPTIMAL 4K
MINEXTENTS 20 MAXEXTENTS 100);
```

Modificar un segmento de rollback

```
ALTER ROLLBACK SEGMENT rbs01 ONLINE;
 STORAGE (MAXEXTENTS 200 );
```

Borrar un segmento de rollback

```
DROP ROLLBACK SEGMENT rbs01;
```

ESTRUCTURA INTERNA DE LA BASE DE DATOS ORACLE

La base de datos oracle utiliza dos tipos distintos de estructuras de memoria, áreas globales y áreas de proceso, las áreas de proceso se describirán en la sig. sección, por lo que las áreas globales esta compuestas por los siguientes elementos:

* Area global del sistema(System Global Area(SGA))
* Buffers de bloques de datos.
* Cache del diccionario.
* Buffers de redo logs
* Fondo común SQL compartido
* Areas de contexto
* Area global del programa(Program Global Area(PGA))

CONSTRAINTS

Sobre una tabla pueden imponerse restricciones (constraints), en este caso, cada fila de la tabla debe satisfacer las condiciones especificadas en las cláusulas de la restricción.

Ejemplo:

```
CREATE TABLE empleado
(numemp  NUMBER(10)  PRIMARY KEY,
 nombre      CHAR(40)  NOT NULL,
 numdept     NUMBER(2)  DEFAULT 10,
 sueldo      NUMBER(7,2)  CHECK  sueldo < 1000000,
 fecha_nac   DATE,
 num_seg_soc     CHAR(9)   UNIQUE,
 FOREING KEY (numdept) REFERENCES  dept.numdept )
```

Se aprecia en primer lugar que a la tabla se le asigna un nombre EMPLEADO, todas sus columnas tienen un nombre NUMEMP, NOMBRE etc. Cada columna tiene un tipo de dato y longitud especificada. La columna NUMEMP viene especificada como un tipo NUMBER , sin escala(equivalente a cero). La columna NOMBRE viene especificada como CHAR(40). La clave primaria(PRIMARY KEY) de la tabla es la columna o conjunto de columnas que hacen que cada fila de dicha tabla sea exclusiva. Una columna se identifica como clave primaria por estar definida dentro de la base de datos como NOT NULL (NO NULA) lo que significa que todas las filas almacenadas en esta tabla, como en la columna NOMBRE del ejemplo anterior. Una columna de la base de datos puede tener una restricción DEFAULT (por omisión), que indica que se generara un valor para dicha columna cuando se introduzca una fila en la tabla pero no se especifique ningún valor para ella. La restricción CHECK (comprobación) sirve para asegurar que los valores de una columna en concreto cumplan cierto criterio (en este caso que el valor de la columna SUELDO sea menor al de 1.000.000).
Existe otra restricción UNIQUE (exclusivo), que se utiliza para especificar la exclusividad de columnas que deben ser exclusivas pero que no forman parte de la clave primaria. En este ejemplo, la columna NUM_SEG_SOC tiene una restricción UNIQUE , lo que implica que todos los registros de esta tabla deben tener un valor exclusivo en esta columna. Una restricción de clave externa (FOREING KEY) sirve

para especificar la naturaleza de la restricción entre tablas. Una clave externa de una tabla hace referencia a una clave primaria definida con anterioridad en cualquier otro lugar de la base de datos. Por ejemplo, si otra tabla llamada DEPT, tuviera una clave primaria denominada NUMDEPT, entonces los registros de esa tabla contendrían todos los valores validos de NUMDEPT. La columna NUMDEPT de la tabla EMPLEADO, hace referencia a dicha columna DEPT.NUMDEPT al especificar EMPLEADO.NUMDEPT como clave externa a DEPT.NUMDEPT, con esto se garantiza que no se puede introducir valores de NUMDEPT en la tabla EMPLEADO a menos que dichos valores ya existan en la tabla DEPT.

Un Ejemplo mas de constraints

```
create table anticipo
 (SOLICITUD NUMBER(8),
 CONSTRAINT FK_CAP_ANT FOREIGN KEY(SOLICITUD)
 REFERENCES DESCRIPCION(SOLICITUD),
 EMPLEADO NUMBER(6),
 CONSTRAINT FK_ANT_CEMP FOREIGN KEY(EMPLEADO)
 REFERENCES CATALOG_EMP(EMPLEADO),
 FECHA_INIC DATE,FECHA_FIN DATE,IMPORTE NUMBER(10,2),
 SALDO_IND NUMBER(10,2))
```

TRIGGERS

Los triggers son procedimientos que se ejecutan cuando se produce un evento de bases de datos. Los eventos que pueden ser afectados por un trigger son: INSERT, UPDATE y DELETE

Los triggers se utilizan para aumentar la integridad referencial y conseguir una seguridad adicional. Existen dos tipos de triggers:

-Triggers de sentencias, estos se activan una vez por cada sentencia de disparo.
-Triggers de fila, se activan una vez por cada fila de una tabla afectada por las sentencias.

Por ejemplo: Usando un procedimiento almacenado que envía dos parámetros mi_num_emp, y aumento, para después insertar en la misma tabla cuatro nuevos campos, seria el siguiente query.

```
CREATE PROCEDURE mi_aumento (mi_num_emp IN NUMBER, aumento IN NUMBER)
    AS BEGIN
          UPDATE emp
          SET sueldo= suledo + aumento
          WHERE numemp = mi_num_emp;
    CREATE TRIGGER copia_dato
      AFTER INSERT ON emp
      FOR EACH ROW
      BEGIN
            INSERT INTO emp
            VALUES
            (:nuevo.numemp, : nuevo.nombre, :nevo.numdept, :nuevo.sueldo);
       END

    END;
```

El script anterior es un ejemplo de un trigger de fila, esta forma es útil si la acción del disparo cuenta con los datos afectados por la transacción.

Siguiente ejemplo:

```
SELECT empno FROM  emp
       WHERE SUELDO > 20000
  CREATE  TRIGGER  copia_dato
       BEFORE  INSERT ON emp
       FOR  EACH  ROW
       BEGIN
            INSERT  INTO emp
            VALUES
            (:nuevo.numemp, : nuevo.nombre,  :nevo.numdept,  :nuevo.sueldo);
        END;
```

Este es un ejemplo de un trigger de sentencia, este tipo de trigger es útil si el código de la acción de disparo no cuenta con los datos afectados.

Para cada uno de estos puede crearse un disparador BEFORE y otro AFTER, para cada tipo de evento de disparo, entre estos eventos de disparo se encuentran las operaciones INSERT, UPDATE y DELETE, EXECUTE, INDEX, REFERENCES, SELECT.

Los triggers de sentencia son útiles en el código de acción. Por ejemplo se puede crear un trigger de sentencia BEFORE INSERT en una tabla para impedir operaciones de INSERT en ella, excepto durante periodos de tiempos especificados.
Los triggers de fila, son útiles si la acción de disparo cuente con los datos afectados por l<transacción. Por ejemplo puede crearse un trigger de fila AFTER INSERT que introduzca filas nuevas en una tabla.

Ejemplos:

Antes de crear el trigger, es necesario crear un enlace de base de datos, para que pueda ser usado por el trigger, en este caso el enlace se crea en la base de datos propietaria de los datos y accesible para los datos de la tabla que se esta replicando.

```
CREATE  DATABASE LINK  enlace_trigger
 USING 'REMOTO1';
```

. enlace_trigger, es el nombre del enlace.
. REMOTO1, es el nombre del servicio de datos remoto

Forma en que genera el enlace

1.- **Primero creamos el db_link de manera siguiente**:

```
CREATE  DATABASE  LINK db_link    /* db_link  es el nombre del database link de connección*/
    CONNECT  TO SYSTEM  IDENTIFIED BY  password      /* será conectado al usuario system*/
     USING 'T: 143.4.1.8: A'                  /* 'T' indica que usará protocolo TCP/IP,  una dirección ip,
                                                 Mientras que 'A'  representa la instancia del equipo*/
```

2.- **Si se desea usar para generar una tabla, entonces la sintaxis quedaría de la manera siguiente.**

```
CREATE TABLE  nueva_tabla /*  nueva_tabla es la tabla a crear*/
As
SELECT *  FROM  tabla_remota@db_link   /* tabla_remota es la tabla de donde se traerán los
                                        datos para llenar nueva_tabla, db_link es la
                                        connección*/
```

El siguiente trigger utiliza este enlace, se activa al introducir renglones ó registros en la tabla EMPLEADO. Como el trigger se ejecuta después de la introducción de la fila, los datos de esta ya se han validado, después se introduce la misma fila en una tabla remota con la misma estructura utilizando en enlace de bases de datos ENLACE_TRIGGER.

Nota

Los datos se validan desde que se hace el SELECT

```
CREATE  TRIGGER  copia_dato
ARTER  INSERT  ON empleado
 FOR  EACH  ROW
 BEGIN
       INSERT  INTO empleado@ENLACE_TRIGGER
       VALUES
       (:nuevo.numemp, : nuevo.nombre,  :nevo.numdept,  :nuevo.sueldo);
 END;
```

Para obtener información acerca de los triggers, se utiliza la vista DBA_TRIGGERS del diccionario de datos.

Ejemplo.

El siguiente trigger permite hacer una consulto, obtener información de la tabla sobre la que se realiza la llamada.

```
SELECT trigger_type, triggering_event, table_name
    FROM dba_triggers
    WHERE trigger_name='COPY_DATA'
```

Análisis : El script anterior obtiene el tipo de trigger, el evento del trigger, y el nombre de la tabla de la vista del diccionario de datos dba_triggers, donde el nombre del trigger es igual a COPY_DATA. La clase de datos que se desee seleccionar dependen de los requerimientos del usuario.

Un ejemplo mas sin usar db_link

```
CREATE  TRIGGER auditoria
       AFTER  UPDATE  OF  sal  ON  empleados
       FOR  EACH  ROW
BEGIN
      INSERT  INTO  auditoria_empleado  VALLUES ( ..... )
END;
```

PROCEDIMIENTOS ALMACENADOS

Un procedimiento almacenado son bloques de sentencias PL/SQL que se guardan en el diccionario de datos, de donde es llamado por las aplicaciones. Estos procedimientos permiten almacenar dentro de la base de datos relaciones de las transacciones que se utilizan con frecuencia. Cuando se ejecuta un procedimiento, sus sentencias se ejecutan como una unidad, los procedimientos no devuelven ningún valor al programa que los llama.

Los procedimientos ayuda a reforzar la seguridad de los datos, con ello puede conseguirse en no acceso directo a los usuarios sobre determinadas tablas en una aplicación, pues el procedimiento contiene solo los privilegios que han sido asignado por el DBA al usuario.

ESTRUCTURA GENERAL DEL UN PL/SQL

```
[DECLARE
        -------- declaraciones]
BEGIN
        --------sentencias
[EXCETION
        ---------controles]
END;
```

Variables y constantes

Pl/sql permite declarar variables y constantes para ser usadas en cualquier modulo del procedimiento

```
Part_no   number(4);
Int_stock   boolean;
```

Puede también asignarse registros de tablas una variable.

```
Tax := price * tax_rate;
Bobus := current_salary * 0.10;
Amount := TO_NUMBER(SUBSTR('750 dollars', 1, 3));
Valid := FALSE;
```

Otra de las formas de utilizar variables, es con el uso de un fetch

```
SELECT sal * 0.10 INTO bonus  FROM  emp  WHERE  empno = emp_id;
```

Declaración de un constante.

```
Credit_limit CONSTANT  REAL := 5000.00;
```

Ejemplo:

```
CREATE  PROCEDURE  OR REPALCE   mi_aumento (mi_num_emp IN  NUMBER,  aumento
 IN  NUMBER)
 AS  BEGIN
        UPDATE  empleado
        SET  sueldo= suledo + aumento
        WHERE  numemp = mi_num_emp;
 END;
```

Siguiente ejemplo:

Este programa registra el marcador en un juego de tennis a través de una variable numérica que permite almacenar la cantidad de aciertos y faltas, para después guardar los cambios en una tabla dentro de la base de datos

```
DECLARE
      Cursor_juego  NUMBER(5);
BEGIN
```

```
SELECT cantidad INTO cursor_juego FROM juegos
   WHERE producto = 'tennis'
     FOR UPDATE OF cantidad
IF cursor_juego > 0 THEN
   UPDATE juegos SET cantidad = cantidad - 1
        WHERE producto = 'tennis'
    INSERT INTO marcador
         VALUES('comienza juego ',SYSDATE);
ELSE
     INSERT INTO marcador
          VALUES('Fuera de tiempo',SYSDATE);
 END IF
 COMMIT;
END;
```

CURSORES

Oracle utiliza espacios de trabajo para ejecutar sentencias sql y procedimientos almacenados, los cursores permiten el control de información según sea la tarea de la sentencia, existen dos tipos de cursores, cursores explícitos y cursores implícitos. Para los cursores de tipo implícito el principal uso es en aquellas sentencias que devuelvan un solo registro, para querys que devuelvan mas de un registro entonces se emplea la forma explicita Por ejemplo.

```
DECLARE
    CURSOR c1 IS
      SELECT empno , ename, job FROM emp WHERE deptno = 20;
```

cursor >

7369	SMITH	CLERK
7566	JONES	MANAGER
...	...	...
7876	ADAMS	CLERK
7902	FORD	ANALYST

Los cursores hacen uso delas sentencias **OPEN, FETCH** y **CLOSE,** son sentencias de control de un cursor

OPEN Esta sentencia ejecuta el query asociado con el cursor identificando la información activa y posición del cursor y verificar así si este se encuentra en la primer línea,

FETCH Esta sentencia recupera el registro actual y avanza el cursor al siguiente registro una vez que haya procesado el anterior .

CLOSE Deshabilita el cursor y libera el espacio que se encuentra ocupando en ese momento

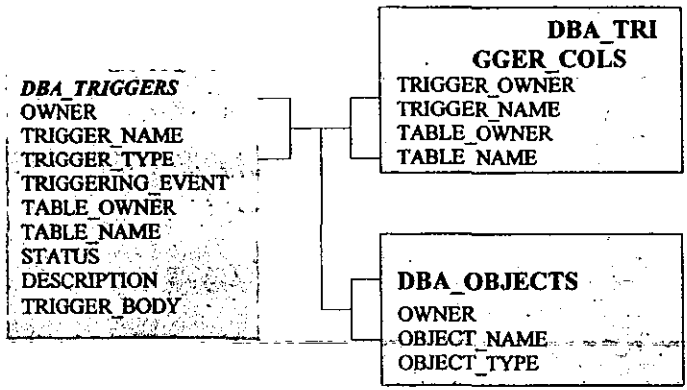
USO DEL DICCIONARIO DE DATOS

Para hacer uso del diccionario de datos es necesario conocer su contenido y como debemos usarlo, para ello daremos un ejemplo mostrando una sintaxis general para no tener ningún problema.

```
SELECT OWNER, TRIGGER_NAME, TRIGGER_TIPO FROM DBA_TRIGGERS
```

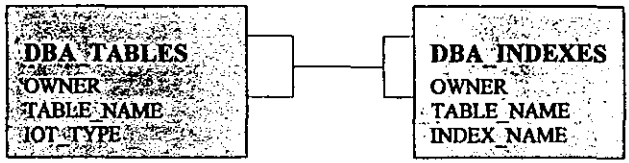
Una manera fácil para describir este ejemplo es: La selección se hace como si seleccionaramos campos de alguna tabla, en este caso estamos seleccionando el propietario, el nombre del trigger y el tipo de trigger de

la visrta DBA_TRIGGERS, Esta funciona como si fuera el nombre de alguna tabla. Estas vistas se encuentran organizados de la siguiente forma:



Es esta gráfica se muestran 3 vistas **DBA_TRIGGERS**, **DBA_TRIGGER_COL**, **DBA_OBJECT**, cada uno contiene descriptores de la base de datos, a través de ellos podemos obtener información acerca del estado actual de nuestra base datos, en este caso **DBA_TRIGGERS**, **DBA_TRIGGER_COL**, **DBA_OBJECTS** proporcionan información de un trigger. Existen otras vista que nos ayudan a conocer detalles de un segmento de rollback, de un tablespace, de una tabla, de un role etc.

Otro ejemplo:



Esta vista nos proporcionan detalles de una tabla y de un índice respectivamente, la manera en que podemos conocer su estructura es dando un:

```
svrmgrl> DESC DBA_TABLES
```

Para mirar todo el contenido del diccionario de datos basta con hacer lo siguiente:

```
Svrmgrl> DESC DICT
```

CONCLUSIONES

Al utilizar cualquiera de estas configuraciones se añade un nivel de complejidad a la administración del sistema, así como también un número de posibilidades interesantes a la arquitectura. Uno de los casos seria que ningún usuario podrá acceder directamente al servidor de la base de datos sin autorización, esto implica que el usuario se deba ajustar a la memoria y recursos que le sean otorgados. Otro de los puntos importantes es que los usuarios podrán separase de los desarrolladores, así como también distintas clases de usuarios podrán acceder a diversos ejecutables. Al desarrollar aplicaciones en este tipo de entorno, se descubre la eficiencia absoluta del trabajo en red y la gestión de bases de datos distribuidas así como todo el funcionamiento de oracle y utilidades de terceros fabricantes. También se han presentado ejemplos del mundo real para cada una de las configuraciones principales, de manera que si consideramos las técnicas que se presentan en este trabajo ya no habrá que preocuparse por los desastres que amenazan a las bases de datos pues hemos presentado una manera optima para diseñarlas, con estas técnicas, los esfuerzos de ajuste serán mínimos y la administración de la base de datos se ara mas sencilla y los usuarios obtendrán un mejor producto, a la vez que la base de datos funcionara adecuadamente. Los temas que se han mostrado en el presente trabajo reúnen las características necesarias para un material de consulta, no obstante, se recomienda consultar bibliografías de apoyo para comprender y extender aun mas el interés y el conocimiento necesario para resolver cualquier problema en una fuente de datos.

BIBLIOGRAFIA

Configuration and capacity planning for solaris server
Brian L Wong
Prentice Hall

TCP/IP Network Administration
Craig Hunt
O'Reily

Building Storage Networks
Osborne
McWrawHill

Database administration
Oracle university
Oracle

Building database Object-Oriented Applications
William B. Heys
Prentice Hall

www.oracle.com/oramag/