HERRAMIENTA DIDÁCTICA PARA EL APRENDIZAJE DE ORACLE

FARIDES MARTÍNEZ FONSECA

MARÍA MARGARITA PAYARES GARCÍA

TECNOLÓGICA DE BOLÍVAR INSTITUCIÓN UNIVERSITARIA

FACULTAD DE INGENIERÍA DE SISTEMAS

ÁREA DE ADMINISTRACIÓN DE BASES DE DATOS

CARTAGENA D .T Y C

2003

HERRAMIENTA DIDÁCTICA PARA EL APRENDIZAJE DE ORACLE

FARIDES MARTINESZ FONSECA
MARIA MARGARITA PAYARES GARCIAS

Tesis de grado presentada como requisito parcial para optar el título de Ingeniero de Sistemas

Director
GIOVANNY VASQUEZ
Ingeniero de Sistemas

TECNOLÓGICA DE BOLÍVAR INSTITUCIÓN UNIVERSITARIA
FACULTAD DE INGENIERÍA DE SISTEMAS
ÁREA DE ADMINISTRACIÓN DE BASES DE DATOS
CARTAGENA D .T Y C
2003

Cartagena, marzo 30 de 2003

Señores
TECNOLÓGICA DE BOLIVAR INSTITUCIÓN UNIVERSITARIA
FACULTAD DE INGENIERIA DE SISTEMA
ATN. COMITÉ DE EVALUACIÓN
Ciudad

Reciban un cordial saludo,

La presente es para informar a ustedes que la tesis que lleva como título "**Herramienta Didáctica para el Aprendizaje de Oracle"**, que ha sido presentada por las estudiantes **Maria Margarita Payares** y **Farides Martinez Fonseca**, cuenta con todo mi aval ya que es un proyecto que le trae beneficios a nuestra Institución.

Cordialmente,

Ing. GIOVANNY VASQUEZ

Cartagena, marzo 30 de 2003

Señores
TECNOLÓGICA DE BOLIVAR INSTITUCIÓN UNIVERSITARIA
FACULTAD DE INGENIERIA DE SISTEMA
ATN. COMITÉ DE EVALUACIÓN
Ciudad

Reciban un cordial saludo,

Por medio de la presente nos permitimos hacer entrega formal del trabajo de grado "**Herramienta Didáctica para el Aprendizaje de Oracle"**, como requisito parcial para optar el titulo de Ingeniero de Sistema

Atentamente,

FARIDES MARTINEZ FONSECA **MARIA MARGARITA PAYARES**

Cartagena, marzo 30 de 2003

Señores
TECNOLÓGICA DE BOLIVAR INSTITUCIÓN UNIVERSITARIA
FACULTAD DE INGENIERIA DE SISTEMA
ATN. ING. GONZALO GARZON
DECANO DE FACULTAD
Ciudad

Reciban un cordial saludo,

Comedidamente nos dirigimos a usted con el fin de presentar a consideración para su estudio y aprobación el trabajo de grado **"Herramienta Didáctica para el Aprendizaje de Oracle"**, con el objeto de optar el titulo de Ingeniero de Sistema.

Atentamente,

FARIDES MARTINEZ FONSECA **MARIA MARGARITA PAYARES**

REGLAMENTO ACADEMICO
(ARTICULO 105)

La Corporación Universitaria Tecnológica de Bolívar se reserva el derecho de propiedad intelectual de todos los trabajos de grado aprobados y no pueden ser explotados sin su autorización.

NOTA DE ACEPTACIÓN

_____

_____

_____

_____

JURADO

JURADO

Cartagena, marzo 30 de 2003

DEDICATORIA

Dedicamos este trabajo a nuestras maravillosas familias, quienes nos vieron comenzar esta carrera cuando apenas éramos unas adolescentes, y ahora, que necesitamos muchas manos para llegar a la meta aún están allí para asegurarse de que triunfemos.

De todo corazón, ¡Muchas gracias!

Farides y María Margarita

AGRADECIMIENTOS

Gracias a Ti, Nuestro Buen Pastor, por ser nuestra fuerza en medio de la debilidad, porque no hay nadie como tú para abrir caminos en medio de la nada, donde el ojo humano no ve la solución.

Gracias a todas las maravillosas personas que con su tiempo, esfuerzo, palabras de ánimo, dinero, conocimiento y amor, hicieron esta tesis posible. Especialmente nuestro director de facultad Gonzalo Garzón, nuestro director de tesis Giovanny Vásquez, nuestra asesora María Claudia Bonfante, y todo el cuerpo docente de la Tecnológica de Bolívar quienes nos transmitieron los valiosos conocimientos que fueron una base sólida sobre la cual edificamos esta tesis.

A todos, incluyéndote a ti estimado lector, quien le das razón de ser a esta tesis, a todos...

¡Un millón de Gracias!.

Farides y María Magarita

TABLA DE CONTENIDO

Pág.

LISTA DE ANEXOS

OBJETIVO DE LA INVESTIGACIÓN

OBJETIVO GENERAL

Implementar y desarrollar una serie de prácticas de laboratorio de Administración de Bases de Datos en el área de Oracle, como complemento de la asignatura de Administración de Bases de Datos para la facultad de Ingeniería de Sistemas de la Tecnológica de Bolívar.

OBJETIVOS ESPECÍFICOS

- Montar un servidor que sirva como base para la implementación de las prácticas.

- Diseñar e implementar 20 prácticas de pruebas para el laboratorio de Oracle.

- Presentar como base de procedimiento de estas prácticas una estructura que ayude al estudiante en el desarrollo de la misma de la siguiente forma:

    - Sustentación teórica
    - Objetivos para alcanzar
    - Contenido de la práctica en sí.
    - Cuestionario de retroalimentación final.

SÍNTESIS DE LA METODOLOGÍA

Para llevar a cabo el desarrollo de la tesis se realizó un estudio del Administrador de Bases de Datos Oracle, que arrojó como resultado la selección de los temas necesarios que le permitirían al alumno adquirir el conocimiento básico fundamental para el manejo de Oracle.

SÍNTESIS DE LA CONCLUSIÓN

Este trabajo que se presenta a la comunidad académica hoy, siempre estuvo inspirado en lo que su título claramente señala: ser un instrumento didáctico para la enseñanza de Oracle. La elaboración del mismo exigió un gran esfuerzo inicial en el conocimiento profundo de la herramienta y, posteriormente, en el diseño de las guías de tal manera que cumplieran los objetivos de enseñar y ser didáctico. Después de todo este tiempo de trabajo y sacrificio de las estudiantes siento que todo ha valido la pena, puesto que el producto que se entrega, y del que este documento forma parte, satisface totalmente las metas propuestas en el documento del proyecto de grado. Las prácticas diseñadas se montaron y probaron rigurosamente para que se pudieran alcanzar cada uno de sus objetivos, el de las prácticas, y el cumplimiento de los objetivos de cada laboratorio, se convirtió en una meta más dentro de lo que definimos como el objetivo general del proyecto. ¿Faltaron algunos laboratorios? ¿Incluyeron algunos que no tenían por qué? Esas respuestas dependen de los gustos y los objetivos de aprendizaje de cada individuo. Este trabajo cumple totalmente las competencias cognitivas de un curso de base de datos tomando como

herramienta de laboratorio el motor de base de datos Oracle, y su diseño tuvo siempre como principio guía el de enseñar de manera sencilla, ya que lo complejo lo establecerá los límites de aprendizaje, y la aplicación de estos saberes, que se impongan los mismos estudiantes.

Director: GIOVANNY VASQUEZ M.

1. ESTADO DEL ARTE

En la actualidad Oracle tiene un fuerte liderazgo en el mercado moderno de bases de datos relacionales.

Un estudio reciente sobre empresas que figuran en Fortune 100 realizado por FactPoint descubrió que Oracle tiene la mayor penetración de bases de datos en dichas empresas por sobre cualquier otro competidor, y esto se debe a que Oracle se encuentra a la vanguardia de la tecnología del manejo de Bases de Datos.

Teniendo en cuenta toda la proyección que Oracle presenta en el mercado laboral, el Gobierno Nacional lo introdujo como parte del Programa de Formación de Base de Datos el cual hace parte de uno de los programas desarrollado y establecidos en La Agenda de Conectividad la cual es un conjunto de estrategias desarrolladas a través de programas y proyectos específicos, articulados entre si, con el propósito de lograr que el país aproveche el uso de las tecnologías de la información para su desarrollo económico, social y político.

Oracle cuenta con las últimas tendencias tecnológicas en el área de Bases de Datos (BD's) y su aprendizaje a través de la "Herramienta Didáctica de Aprendizaje de Oracle" proporciona muchos beneficios como son:

Le permitirá a alumno prepararse como profesionales con excelentes conocimientos técnicos y prácticos en el campo de las bases de datos.

Analizar, evaluar y proponer soluciones a problemas tecnológicos en el área de las bases de datos pues el estudiante contrastara los conceptos con la praxis.

Administrar sistemas de bases de datos y asesorar la evaluación e implementación de soluciones con productos comerciales de Hardware y Software.

Posibilidad por parte del docente de la actualización tecnológica, hecho necesario y de suma importancia, debido a la velocidad con que aparecen nuevas conceptos en cuanto a Oracle y sus herramientas.

Para lograr esto se implementarán prácticas de los siguientes temas:

- Sentencias SQL Básicas.
- Restringiendo y Ordenando Datos.
- Funciones SQL.
- Obteniendo Datos de Múltiples Tablas.
- Subconsultas.

- Produciendo Salidas Legibles con SQL*Plus.
- Manipulando Datos.
- Creación y Manejo de Tablas.
- Agregando Ligaduras.
- Otros Objetos de Base de Datos.
- Controlando el Acceso a la Base de Datos.
- Declaración de Variables en PL/SQL
- Uso de SQL en PL/SQL
- Estructuras de Control
- Declaración y Uso de Cursores
- Excepciones en PL/SQL
- Procedimientos en PL/SQL
- Funciones en PL/SQL
- Paquetes en PL/SQL
- Triggers en PL/SQL

Estas experiencias contarán de un manual para el estudiante que contiene el procedimiento para desarrollar la práctica. Cada práctica llevará una documentación en la que se hace referencia a la teoría y estrategias utilizada para

desarrollar la experiencia, para que le ayude a la mejor comprensión y aprendizaje de esta.

El profesor contará con un manual que le permitirá entender la directriz de la teoría y la practica elaborada, adicionalmente el manual contendrá la respuesta a las preguntas de evaluación que se dan al alumno en su manual del estudiante, para que resuelva, y también contendrá sugerencias para la realización eficiente de la práctica.

Las prácticas se probaran con anticipación por las autoras con la ayuda del asesor de la tesis, para poder retroalimentar y mejorar el trabajo.

2. DISEÑO DE LOS MANUALES PARA APRENDIZAJE DE ORACLE

Los manuales orientan y ayudan en forma inmediata el trabajo individual. Ellos no pasan de ser un instrumento de trabajo, un medio para un logro posterior.

El manual del estudiante se confeccionó, respetando el ritmo de trabajo de cada estudiante, y al mismo tiempo permite la planificación continua mediante constante reorientación en el proceso instructivo; hasta tal punto han influido y tal ha sido su difusión en el medio educativo, que es difícil actualmente concebir un sistema individualizado en educación, sin el empleo de múltiples guías didácticas.

Los tipos de manuales o guías son muy variados dependiendo de los autores, de los objetivos que quieran obtenerse y las funciones especiales que deban desempeñar. Los más conocidos son:

- *Manual Directo,* le indican al estudiante lo que debe hacer, lo remiten al material o a otros libros de consulta que más le puedan ayudar. Son los manuales base, e indispensables en todo trabajo individualizado.

- *Manual de Información,* contienen una exposición o información; texto para estudiar, datos y documentos. Suelen ser más amplios en el contenido.

- *Manual de Recuperación,* para aquellos estudiantes con dificultades en el aprendizaje.

- *Manual de Control,* o más apropiadamente de autocontrol con el que el estudiante juzga su propio trabajo, y viene a ocupar el puesto que tendrían los exámenes, natural revisado por el profesor, el contraste entre el juicio del estudiante y el del profesor le ayudará a formarse en la objetividad de sus apreciaciones.

El manual como tal es un medio que el profesor se vale para enseñar, como podría ser el Internet, el televisor o un libro; este es el valor que se le puede atribuir. Por medio de èl, el profesor puede adoptar la enseñanza en mejor forma a la sicología y especiales circunstancias en que se encuentran los estudiantes. Podría decirse que el manual es una lección preparada, pensada y orientada al estudiante.

Para el estudiante, encontrarse con una tarea definida y clara preparada por el maestro debe ser un estímulo; por esto el ideal es que cada profesor, o un grupo de maestros con materias afines, sean quien las confeccionen, esto permite que de por si los manuales no sean inmutables, antes bien puedan ser renovados y adaptados según las capacidades y necesidades presentes en los estudiantes. Las confrontaciones de experiencias conjunta de algunas guías, sirven en gran manera para amortizar enseñanzas e integrar conceptos aun en materias diferentes.

Los manuales, si se requiere prescindir de enseñanza catedrática, exige para cada lección orientaciones especiales y en general una introducción que ayude a su mismo desarrollo. El permite un control personal y como es natural, solo se puede pasar al siguiente laboratorio cuando el control ha sido cumplido. El hecho que al completar los planes de trabajo se haya cumplido diversas etapas por medio del manual, ayuda notablemente a la amplitud de conocimientos y despierta el interés, frente el éxito escolar.

El valor de las programaciones distribuidas en pequeñas unidades reside en que éstas permiten formar un trabajo proporcionado sin abarcar mucho a la vez, al tiempo que concede al alumno la satisfacción de haber superado una dificultad después de otra; más esta misma puede frenar el impulso creador y hacer pensar

al alumno que solo logrará encontrar lo que busca, respondiendo a mil preguntas preparadas y escogidas de antemano.

Conviene prevenir contra el peligro de que el manual sea utilizado como medio fácil de ocupar a los estudiantes, también es peligroso el dejar a los alumnos constantemente frente a si mismo, sin intervención del maestro. La acción del profesor en cuanto a la formación es insustituible, y la acción mutua de la clase sobre el alumno es de suma importancia.

Gran importancia debe concederse a la elaboración del manual del estudiante y lograr que en el se ponga un especial cuidado e interés, con el fin de que no se reduzca a un cuestionario, sino que el alumno encuentre un verdadero programa al tiempo que orientaciones para avanzar con seguridad al plan de trabajo y hacer por si mismo descubrimiento de los elementos correspondientes. Conviene anotar el peligro de confundir una programación con la confección lineal de ejercicios en el manual, con la sola intención de individualizar la enseñanza dentro de una clase numerosa no debe perderse de vista que la importancia de los métodos activos, más que una ayuda el aprendizaje y a la conclusión de muchos

ejercicio, radica en la invitación que ellos dan al trabajo personal y la conciencia de su propia realización que deja en el alumno.

Los manuales o guías nunca pueden ser suficiente por si solos para la educación. Ellos serán directivas de trabajo, y ayudaran a sacar a flote conocimientos adquiridos, como también orientan posteriormente a la búsqueda, pero su presentación al profesor no puede ser sustituida, vendrán luego las puntuaciones cualitativas estímulo que impulsa a continuar el trabajo o quizás un regreso para rehacer lo que desde un principio no había quedado bien hecho. Puede también prepararse otros laboratorios dentro del manual con preguntas de contenido más difícil para aquellos que, a juicio del profesor, se encuentran en un proceso avanzado y rápido. De una buena confección del manual, trabajo previo del profesor, depende en gran parte que el alumno aproveche más y realice sus tareas personales en forma eficiente.

Como condiciones para que el manual sea verdaderamente cauce orientador para el alumno en su propio trabajo debe tenerse en cuenta lo siguiente:

- El manual debe estar adoptado para los alumnos a quien va dirigido, y preparado cada laboratorio por su propio maestro, por tanto el que un

profesor copie los laboratorios de un libro o de un colega no deja de ser negativo.

- El manual debe redactarse de forma directa con órdenes e indicaciones precisas dirigidas al alumno y no de forma impersonal o anónima.

- El manual debe llevar las indicaciones necesarias para su desarrollo en forma clara y precisa: no debe contener poca información para que el alumno no entienda, ni mucha que llegue aburrir o distraer al estudiante. Un manual que sea incomprensible y que exija a la mayor parte de los alumnos a recurrir al profesor para solicitar explicaciones, es un manual mal elaborado.

- Los temas contenidos en el manual debe estar expresados en forma clara a manera de titulo.

- El manual debe, por su redacción, promover la iniciativa y en tal forma estar organizado que lleve en su ejecución a realizar el objeto que se pretende como una expresión clara de lo aprendido. Por lo tanto el objetivo debe estar anunciado ante de todo el texto, el cual debe ser logrado y evaluado al terminar la elaboración del manual.

- Ha de preocuparse para que el manual lleve un ejercicio de los valores personales y si fuera el caso fomentar también los valores grupales o comunitarios, y en cuanto se pueda valores sociales.

- Siempre ha de preocuparse el profesor que el manual vaya acompañado de la documentación (hojas, libros, elementos que permitan al alumno realizar su trabajo).

- El manual nunca debe dar la impresión de ser un cuestionario, por eso su elaboración debe tener cuidado de no todas las frases tengan el signo de interrogación.

2.1. MODELO GENERAL DE LA ESTRUCTURA DE LOS MANUALES PARA EL APRENDIZAJE DE ORACLE

Por cada unidad estudiada los manuales contendrán un laboratorio tanto para el practicante como para el docente, los cuales como es de esperar estar relacionados por unidades.

Así como el manual del estudiante contribuye a la orientación de este en el desarrollo de un laboratorio dado, el manual del profesor debe brindarle apoyo al profesor en el manejo de cada laboratorio (aclaraciones de dudas a los practicantes, direcciones de las experiencias de laboratorio, evaluación de los resultados, etc).

Las estructura de dichos manuales esta regido por el siguiente modelo:

GUIA DEL ESTUDIANTE

NOMRE DE LA PRACTICA
- Objetivos
- Introducción
- Recursos (Equipo, Software)
- Marco Teórico
- Supuestos
- Procedimientos
- Cuestionario
- Bibliografía

- **Objetivos,** Denotaran los logros que se persiguen con la experiencia e irán acorde con la práctica.

- **Introducción,** Se coloca con el fin de dar una breve descripción del tema especial de cada experiencia.

- **Recursos,** Herramientas tipo hardware y software necesarios para llevar a cabo la practica.

- **Marco Teórico,** Bases y conceptos aclaratorios acerca de la experiencia a realizar.

- **Procedimiento,** Paso a seguir en el desarrollo de la practica.

- **Supuestos,** Postulados hechos con el fin de que los practicantes demuestren o refuten con base a los resultados arrojados en la experiencia.

- **Cuestionario,** Serie de preguntas realizadas con el fin de medir el nivel de asimilación de las experiencias por parte de los practicantes.

- **Bibliografía,** Conjunto de fuentes de información las cuales se pueden remitir los interesados en aclarar dudas o realizar estudios más profundos acerca de la experiencia.

GUÍA DEL PROFESOR

NOMBRE DE LA PRÁCTICA
- Objetivos
- Marco Teórico
- Sugerencia
- Respuesta al laboratorio

- **Objetivos,** Denotaran los logro que persigue el profesor con la lección e irán acorde con la practica.

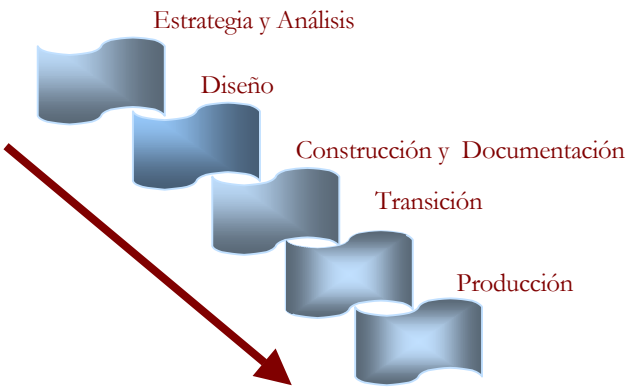
- **Marco Teórico,** Bases y conceptos aclaratorios acerca de la lección y experiencia a realizar con mayor profundidad.

- **Sugerencia,** Pautas para que el profesor de mejor la lección y pueda realizar la experiencia de una forma más clara o pueda hacer futuras modificaciones a las mismas.

- **Respuesta al laboratorio,** Solución a las preguntas que se plantearon al estudiante.

3. MARCO TEORICO

3.1. CICLO DE VIDA DEL DESARROLLO DEL SISTEMA



El Ciclo de Vida del Desarrollo del Sistema es la parte superior e inferior del enfoque sistemático para el desarrollo de la base de datos que transforma la información de negocio y los requerimientos para una base de datos operacional, este Ciclo de Vida se desarrolla en varias etapa que corresponden a:

Estrategia y Análisis

- Estudia y analiza los requerimientos del negocio. encuestas a los usuarios y dirigentes para identificar los requerimientos de la información. Incorpore la empresa y la misión de la empresa a la declaración como también para las futuras especificaciones del sistema.

- Construya el modelo del sistema. transfiera la narración de los asuntos del negocio a una representación grafica de las necesidades y reglas. confirme y refine el modelo con el analista y experto.

Diseño

- Diseño de base de dato basado en el modelo desarrollado en las estrategias y en la fase de análisis.

Construcción y Documentación

- Construya el prototipo del sistema, escriba y ejecute los comandos de creación de tablas y los objetos que componen la base de datos.

- Desarrolle la documentación, texto de ayuda, y manuales operacionales para apoyar el uso de operaciones del sistema.

Transición

- Refine el prototipo. Mueva una aplicación en producción de prueba de aceptación del usuario, conversión de datos existentes y operaciones paralelas. Haga cualquiera modificación si se requiere.

Producción

- Desarrolle el sistema para el usuario, opere el sistema de producción. Monitoree su actuación y refine el sistema.

Las varias fases de ciclo de vida y desarrollo se pueden realizar iterativamente. Esta carrera enfoca en la fase del ciclo de v ida para el sistema de desarrollo.

3.1.1. ALMACENAMIENTOS DE DATOS POR DIFERENTES MEDIOS

3.1.1.1 Almacenar la Información

Cada organización tiene una necesidad de información. Una librería mantiene una lista de los miembros, libros, fecha de vida y multas. Una compañía necesita almacenar información acerca de los empleados, departamentos, y salarios. Esta información se llama datos.

Las organizaciones pueden almacenar datos en varios medios y diferentes formatos, por ejemplo, un documento en borrador en un gabinete de datos de archivos o en una hoja electrónica de base de datos.

Una base de datos es una colección organizada de información.

Para manejar la base de datos, usted necesita el sistema de manejo de base de datos (DBMS). El DBMS es un programa que almacena, recupera y modifica datos en la base de datos que se necesite.

3.1.2. CONCEPTO DE BASE DE DATOS

3.1.2.1. Modelo Relacional

Los principios del modelo relacional fueron diseñado por el Dr. E.F. Codd en junio 1970 en un documento llamado "un modelo de datos relacional para compartir datos de bancos" en este documento el Dr. Codd propuso un modelo relacional para el sistema de base de datos.

El más popular de los modelos usado era el jerárquico o el trabajo neto, o un simple archivo con manejo de estructuras de datos. El manejo del sistema de datos relacional (RDBMS) pronto se hizo muy popular especialmente por su facilidad de usar y flexibilidad en estructura. Además un número de vendedores innovadores, tales como Oracle, suplementaron al RDBMS con una serie de aplicaciones poderosas en el desarrollo y producto de los usuarios, previendo una solución total.

3.1.2.2. Definición de una base de datos relacional

Una Base de Datos Relacional es una colección de relaciones de tablas con dos dimensiones para almacenar información. Por ejemplo: usted puede querer almacenar la información acerca de todos los empleados de una compañía, en una Base de Datos Relacional, usted crea varias tablas para almacenar diferentes piezas de información acerca de sus empleados, tales como una tabla de empleo, una tabla de departamentos y una tabla de salarios.

Modelos de datos

Los modelos son como un pilar para diseñar. Los ingenieros construyen un modelo de carro para evaluar cualquier detalle antes de ponerlo en producción, de la misma manera los diseñadores de sistema desarrollan un modelo para explorar ideas y mejorar la comprensión del diseño de base de datos.

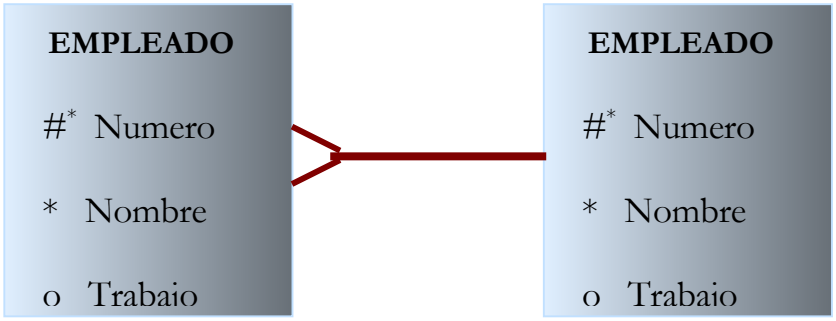
Propósitos de modelos

Los modelos pueden ayudar a comunicar los conceptos en la mente de la gente. Ellos pueden usarse para hacer lo siguiente:

- Comunicar
- Categorizar
- Describir
- Especificar
- Investigar
- Evolucionar
- Analizar
- Imitar

El objeto es producir un modelo que se ajuste a la multitud de estos usos, y que puedan ser entendidos para el usuario final, y que contenga suficiente detalles para desarrollar la construcción de un sistema de base de datos.

3.1.3. MODELO ENTIDAD RELACION (ER)



En un sistema efectivo, los datos se dividen en una categoría discretas o entidades. Un Modelo Entidad Relación (ER) es una ilustración de varias entidades en un negocio y las relaciones entre ellas. Un ER, es decir una relación entre entidades y el modelo sería de las especificaciones del negocio o informaciones construidas en la fase de análisis de Ciclo de Vida del Desarrollo del Sistema del ER, separado de las informaciones requeridas para un negocio de las actividades realizadas con un negocio, aunque los negocio puedan cambiar sus actividades el cielo de información tiende a permanecer constante. Por lo tanto las estructuras de datos tienden a permanecer constantes.

Beneficios de un modelo ER

- Información para la documentación son clara y en un formato preciso.
- Suministra un cuadro claro del enfoque de la información requerida.
- Suministra una fácil comprensión a través de un mapa pictórico para el diseño de base de datos.
- Ofrece un marco efectivo para integrar las aplicaciones múltiples.

Componentes claves

- **Entidad:** Es un objeto acerca del cual tiene que conocerse o mantenerse información. Ejemplo, departamento, empleados, facturas.

Definiciones de Entidad

- Un objeto de interés para el negocio.
- Una clase o categoría de una cosa.
- Es un objeto con nombre.

Una entidad requiere tener atributos que necesitan conocerse desde el punto de vista del negocio, para que pueda ser considerada una entidad en el ámbito del negocio.

- **Atributos:** Algo que describe una cualidad o entidad.

Los atributos son las piezas de información que necesitan saberse acerca de las entidades. ejemplo, para el empleado de la entidad emp los atributos serán: el número del empleado, nombre, título del empleado, fecha en que inició a trabajar, número del departamento, etc. cada uno de los atributos ya sea requerido o es opcional, este estado se llama opcionalidad.

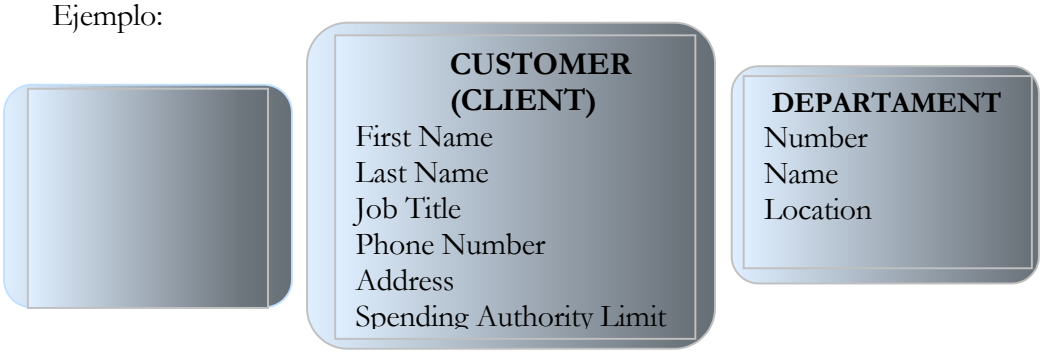
- **Relaciones:** Una asociación nombrada de entidades muestra opcionalidad y grado. Ejemplo, un empleado, departamento, factura y artículo.

Convención del modelo entidad relación

- Rectángulo de esquinas curvas de cualquier dimensión.
- Nombres de entidades en singular, en mayúsculas y únicos.
- Nombre del sinónimo opcional (en paréntesis).

- Nombres de atributos en minúscula.

Ejemplo:



Notas Rápidas:

- Un sinónimo es un nombre alterno para esa entidad.
- Los sinónimos son útiles cuando dos grupos usan diferentes nombres sobre el mismo objeto de interés para la organización.

Cada Entidad puede tener múltiples ocurrencias o instancias.

Ejemplos**:**

La entidad EMPLOYEE tiene una ocurrencia para cada empleado en el negocio:

Jim Brown, Mary Jones, Juan Gómez, and Bill Ludge son todas ocurrencias de la entidad EMPLOYEE.

La entidad DEPARTMENT tiene una ocurrencia para cada departamento de la compañía:

- El departamento Financiero, el departamento de Ventas, y el departamento de desarrollo son todas instancias de la entidad DEPARTMENT.

Cada instancia de entidad tiene valores específicos para los atributos de la entidad.

Ejemplos**:**

La entidad EMPLOYEE tiene los siguientes atributos:

- name, badge number, date of birth, y salary.

La instancia Jim Brown tiene los siguientes valores:

- name = *Jim Brown,* badge number = 1322, date of birth = *15-MAR-70,* and salary = *$55,000.*

Cada instancia debe poder ser identificable de manera única con relación a las otras instancias de la misma entidad. Un atributo o conjunto de atributos que identifican de manera única una entidad es llamado Identificador Único (Unique Identifier UID).

Ejemplo:

Cada empleado tiene un único *badge number*. Entonces Badge number es un candidato para ser el UID de la entidad EMPLOYEE.

Notas Rápidas:

- Las instancias son, algunas veces, mal interpretadas como entidades.
- Una entidad es una clase o categoría de una cosa Ej: EMPLOYEE.
- Una instancia es un elemento en particular de esa entidad Ej: el empleado Jim Brown.

3.1.4. Relaciones

Una relación es una asociación bidireccional significante entre dos entidades o entre una entidad consigo misma.

Sintaxis de la Relación:

Entidad1 {(Debe / puede tener)Nombre de la relación (una o más/ uno y solamente uno)}Entidad2

Ejemplo*:*

La relación entre MAESTRO y CURSO es:

Cada curso puede ser enseñado por uno y solo un maestro. Cada maestro puede estar asignado a uno o más cursos.

Cada dirección de la relación tiene:

- **Un nombre:** Ej. Enseñado por o Asignado a.
- **Una opcionalidad:** Debe ser o Puede Ser.
- **Un grado:** Uno a uno, Uno a muchos, Muchos a uno o Muchos a muchos.

Los atributos que sean UID o parte de la UID son marcados con #*.

Cardinalidad es sinónimo de grado.

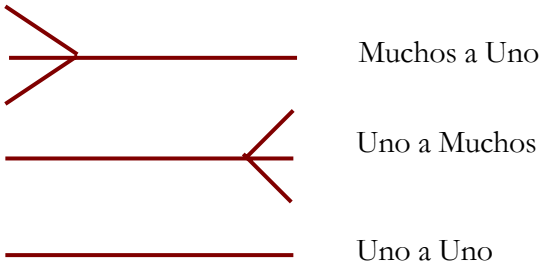
Un grado de cero en un sentido para una instancia implica una relación opcional.

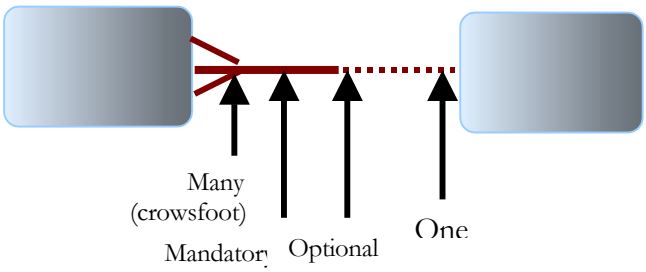
Convenciones en diagramas

- Una línea entre dos entidades
- Nombres en minúsculas para las relaciones

Opcionalidad:

Grado:

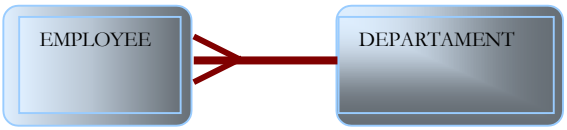




Primero lea la relación en un sentido y luego del lado.

Ejemplo:

Lea la relación entre EMPLOYEE Y DEPARTAMENT



La Relación de Izquierda a Derecha del diagrama Parcial:

Cada empleado debe estar asignado a uno y solo un Departamento.

La Relación de Izquierda a Derecha del diagrama Parcial:

Cada Departamento puede ser responsable de uno o más Empleados.

3.1.4.1. Tipos de relaciones

Hay tres tipos de Relaciones

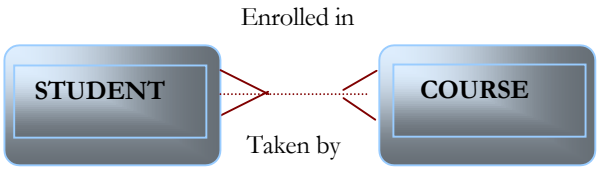
- Uno a muchos.

- Mucho a muchos.
- Uno a uno.

➢ **Uno a Muchos:** Un elemento está relacionado con 1 o más elementos del otro conjunto (1:N).

➢ **Mucho a Muchos: Tienen** grado de 1 a N en ambas direcciones.

Ejemplos:



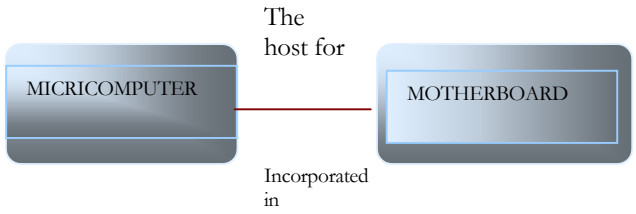
Notas Rápidas:

- Todas las relaciones deberían representar los requerimientos de información y las reglas del negocio.
- Las relaciones 1 a muchos son comunes.
- Relaciones 1:N que sean obligatorias en ambos sentidos son raras.
- Las relaciones mucho a mucho son muy comunes.

- **Uno a Uno:** Tienen grado de uno a uno en ambas direcciones.

Ejemplo:

Hay una relación 1:1 entre MICROCOMPUTER y MOTHERBOARD.



Notas Rápidas:

- Todas las relaciones deberían representar los requerimientos de información y las reglas del negocio.
- Las relaciones 1 a muchos son comunes.
- Relaciones 1:N que sean obligatorias en ambos sentidos son raras.
- Las relaciones mucho a mucho son muy comunes.
- Las relaciones mucho a mucho generalmente son opcionales en ambas direcciones, aunque pueden ser opcionales en un solo sentido.
- Las relaciones uno a uno son raras.
- Las relaciones uno a uno obligatorias son muy raras.
- Pueden ser realmente una sola entidad.

3.1.5. Terminología de la base de datos relacional

En un sistema de base de datos relacional, la información se organiza en forma de tablas. Una tabla es la estructura básica de almacenamiento de un RDBMS. Una tabla contiene todos los datos necesarios acerca de algo en el mundo real, por ejemplo, empleados, facturas o clientes.

Tabla EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEP
7839	KINK	President		17/11/81	5000		10
7698	BLAKE	Manager	7839	01/05/81	2850		30
7782	CLARK	Manager	7839	09/06/81	2450		10
7566	JONES	Manager	7839	02/04/81	2975		20
7654	MARTI	Salesman	7698	28/09/81	1250	1400	30
7499	ALLEN	Salesman	7698	20/02/81	1600	300	30
7844	TURNE	Salesman	7698	08/09/81	1500	0	30

Notas Rápidas:

- Las categorías de información se listan en la parte de arriba de cada tabla.
- Los casos individuales se listan al lado izquierdo.
- En esta forma, usted puede visualizar, entender y usar esta información inmediatamente.

Cada columna contiene un tipo de información. Cada fila esta compuesta de columnas que contienen un valor único.

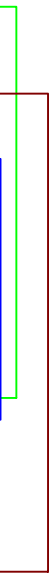
3.1.6. Tabla de relaciones múltiples

NOMBRE DE LA TABLA: EMP

EMPNO	ENAME	JOB	DEPTNO
7839	**KING**	**PRESIDENT**	**10**
7698	**BLACK**		
7782	**CLARK**		

NOMBRE DE LA TABLA: DEP

DNAME		
ACCOUNTING		
	DALLAS	**20**
SALES	**CHICAGO**	**30**



Los datos de las diferentes entidades están almacenados en diferentes tablas, usted puede necesitar combinar dos o más tablas para responder a un interrogante particular.

Por ejemplo, usted puede querer saber la localización de un departamento donde un empleado trabaja. En este escenario, usted necesita información de la tabla EMP (contiene los datos del empleado) y la tabla DEPT (contiene información acerca de los departamentos). Un RDBMS lo capacita para relacionar los datos de una tabla con los datos de otra usando la clave foránea. Una llave foránea es una columna o una serie de columnas que se refieren a una clave primaria en la misma tabla o en otra tabla.

La habilidad para relacionar los datos de una tabla con los datos de otra tabla lo capacita a usted para organizar la información en separado, y en unidades manejables, emplee datos que se puedan lógicamente distinguir del departamento de datos para almacenarlo en una tabla separada.

Notas Rápidas:

- Cada fila de datos en una tabla es únicamente identificada por una llave o clave primaria.

- Usted puede lógicamente relacionar datos de múltiples tablas usando llaves foráneas.

GUÍA PARA LLAVES PRIMARIA Y LLAVES FORÁNEAS

- No duplique los valores que se usan en la llave primaria.
- Las llaves primarias generalmente no se pueden cambiar.
- Las llaves foráneas se basan en los valores de los datos y son puramente lógicas, no física, ni apuntadores.

Nota Rápida:

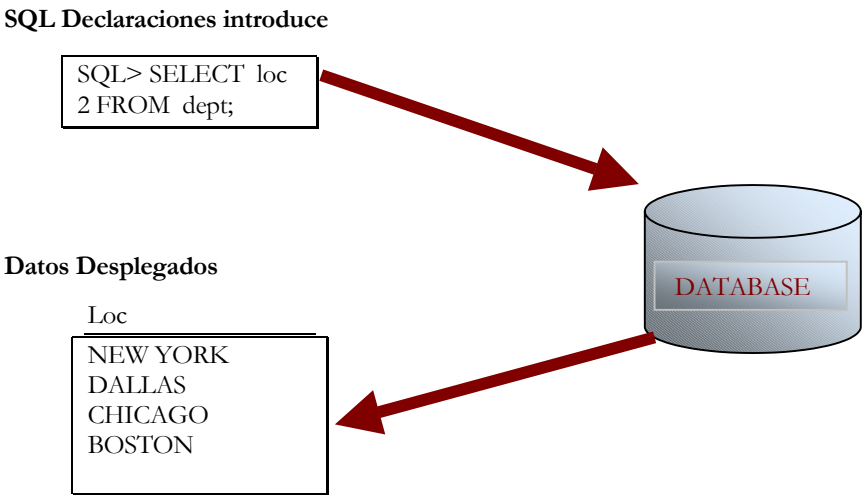
- Usted no puede definir una llave foránea sin la existencia de una llave primaria (única).

3.1.7. Propiedades de una base de datos relacional

En una Base de Datos Relacional, usted no puede especificar el acceso a las filas de una tabla, y usted puede no saber como los datos están ordenados físicamente.

Para acceder a la Base de Datos, usted ejecuta una instrucción de búsqueda del lenguaje estructurado (SQL), el cual es el Estándar del Instituto Americano Nacional (ANSI), el cual tiene un lenguaje estándar para operar en las Base de Datos Relacional. El lenguaje contiene una larga serie de operadores para particionar y combinar relaciones. La base de datos puede ser modificada usando los estamentos SQL.

3.1.8. Comunicándonos con el RDBMS con SQL



Lenguaje de Búsqueda Estructurado

El lenguaje de consultas estructurado SQL permite que usted se comunique con el servidor y tiene las siguientes ventajas:

Eficiencia.

Facilidad para aprender y usar.

Funcionalidad completa (SQL le permite a usted definir, recuperar, y manipular datos en las tablas).

3.1.9. Manejo del sistema de base de datos relacional

Oracle suministra una flexibilidad RDBMS llamado Oracle7. Usando sus expresiones, usted puede almacenar y manejar datos con todas las ventajas de la estructura relacional PL/SQL, una "MAQUINA" que le suministra a usted con la habilidad de almacenar y ejecutar unidades de programas.

El servidor ofrece la opción de recuperar los datos basados en técnicas de optimización. Incluye rangos de seguridad que controla el acceso a la base de datos y el uso. Otra característica es que incluye seguridad de inconsistencia y protección de los dato a través de un mecanismo centinela.

Las aplicaciones Oracle pueden correr en el mismo computador como el de servidor Oracle. Alternativamente, usted puede ejecutar aplicaciones en un sistema local para usuario y ejecutar el servidor Oracle en otro sistema (Arquitectura Cliente/ Servidor).

El ambiente Cliente/Servidor, tiene amplios recursos que se pueden usar. Por ejemplo, una aplicación para reservaciones en una aerolínea se puede ejecutar en un computador personal (cliente), mientras que el acceso a los datos de vuelos son convenientemente manejados por el servidor de Oracle en un computador central.

Nota rápida:

- Para mayor información, vea el Manual de Conceptos del Servidor Oracle.

3.1.10. ORACLE8: SISTEMA DE MANEJO DE LOS OBJETOS EN UNA BDR

Oracle8, es la primera base de datos objeto capaz desarrollada por Oracle. Se extiende al modelo de dato con la capacidad Oracle7 para soportar un nuevo objeto del Modelo Base de Datos Relacional. Oracle8 provee una nueva herramienta que brinda la programación de objetos-orientados, los tipos de datos complejos, los objetos complejos del negocio y completa compatibilidad con el mundo relacional.

Nota Rápida:

- Tipos de datos y objetos definidos por el usuario.
- Completamente compatible con Base de Datos Relacional.
- Soporte de multimedia y objeto largos.
- Base de Datos con alta cualidad de característica del servidor.

3.1.11. ORACLE8I: Plataforma Internet de base de datos

Oracle8i, La Base de Datos para Computación por Internet, provee herramientas avanzadas para manejar todo tipo de datos en los sitios web.

Es mucho más que un simple almacenamiento de datos relacional.

El sistema de archivo de Internet (IFS) combina el poder de Oracle8i, para el fácil uso de un sistema de archivo. Permite a los usuarios mover todos sus datos en la Base de Datos Oracle8i, donde se puede almacenar y manejar más eficientemente.

El usuario final puede fácilmente tener acceso a los archivos y directorios en Oracle iFS, posee una gran variedad de protocolos, como el HTML, FTP, y IMAP4, dándole un acceso universal a sus datos.

Oracle8i Inter-Media permite a los usuarios el manejo de los datos de multimedia, incluyendo imágenes, texto, audio, y datos de video. El oracle8i incluye un robusto, integrado, y escalable Maquina Virtual Java con el Servidor (Jserver), que soporta Java con todos sus enlaces y aplicaciones. Esto elimina la necesidad de compilar o modificar los códigos Javas cuando no se despliega con otro enlace diferente.

Por el nuevamente introducido recurso de manejo, el DBA puede escoger el mejor método para ajustar a un perfil de aplicación y volumen de trabajo. Oracle8i provee completa, integración nativa con el servidor de transacción de Microsoft (MTS) en el entorno Windows NT.

Las aplicaciones en el ambiente se simplifican en el Oracle por la aplicación Wizard (AppWizard) para Visual Estudio, el cual provee desarrolladores con herramientas GUI para crear un Visual C++, Visual Interdev, o aplicaciones de Visual Basic para acceder a los datos en la Base de Datos Oracle.

Nota Rápida:

- Para mayor información, vea el Manual de Conceptos del Servidor Oracle. Publicación 8i

3.1.12. TABLA USADAS EN EL CURSO

EMP

EMPNO	ENAME	JOB	MGR	H.DATE	SAL	COMM	DEP
7839	KINK	President		17/11/81	5000		10
7698	BLAKE	Manager	7839	01/05/81	2850		30
7782	CLARK	Manager	7839	09/06/81	2450		10
7566	JONES	Manager	7839	02/04/81	2975		20
7654	MARTI	Salesman	7698	28/09/81	1250	1400	30
7499	ALLEN	Salesman	7698	20/02/81	1600	300	30
7844	TURNE	Salesman	7698	08/09/81	1500	0	30
7900	JAMES	Clerk	7698	03/12/81	950		30

TABLA DEPT

EMPNO	ENAME	JOB	DEPTNO
7839	KINK	President	10
7698	BLAKE	Manager	20
7782	CLARK	Manager	30

TABLA SALGRADE

GRADE	LOSAL	HI SAL
1	700	1200
2	1201	1400
2	1401	2000
4	2001	3000
5	3001	9999

Las anteriores tres tablas se usaran en este curso

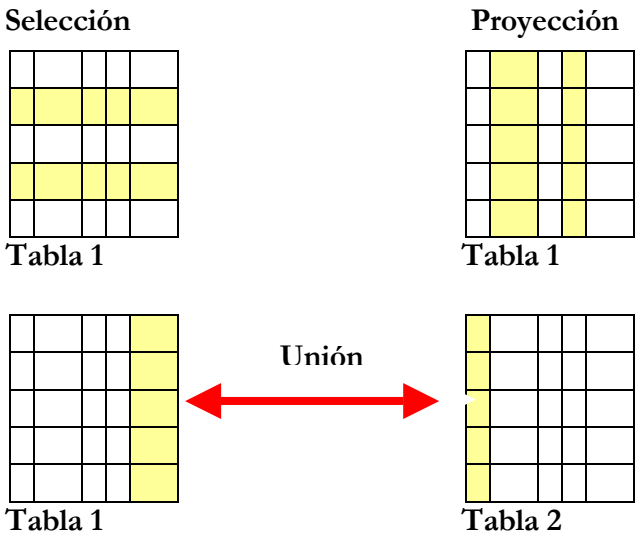
- La tabla EMP, la cual contiene todos los detalles del empleado.
- La tabla DEPT, la cual contiene todos los detalles del departamento.
- La tabla SALGRADE, la cual contiene todos los detalles para varios datos.

Nota Rápida:

- Estas tablas son las que viene por defectos en Oracle.

3.2. SENTENCIAS SQL BÁSICAS

3.2.1 Capacidades de la sentencia SELECT de SQL



Una sentencia SELECT recupera información de la base de datos. Usando una sentencia SELECT usted puede hacer lo siguiente:

- SELECCIÓN: Usted puede usar la capacidad de seleccionar de SQL escogiendo las filas de una tabla que quiere que recupere la consulta.

- PROYECCIÓN: Usted puede usar la capacidad de proyección de SQL escogiendo las columnas de una tabla que quiere que recupere la consulta.

- UNION: Usted puede usar la capacidad de unión de SQL tomando juntos los datos almacenados en diferentes tablas para crear una unión entre ellos.

3.2.2. La sentencia SELECT

En su forma más simple, una sentencia SELECT debe incluir lo siguiente:

Una cláusula SELECT: Especifica las columnas que se van a mostrar.

Una cláusula FROM: Especifica las tablas que contienen las columnas listadas en la cláusula SELECT.

Sintaxis:

```
SELECT    [ DISTINCT] {*, columna, [ alias]...}
FROM      tabla
```

Donde:

SELECT	Es una lista de una o más columnas.
DISTINC	Elimina los duplicados.
*	Selecciona todas las columnas.
Columna	Selecciona el nombre de la columna.
Alias	Da a una columna seleccionada un encabezamiento diferente.
FROM tabla	Especifica la tabla que contiene las columnas.

Notas Rápidas:

- Una cláusula es una parte de una sentencia SQL.

  Por ejemplo: SELECT ename es una cláusula.

- Una sentencia es una combinación de dos o más cláusulas.

  Por ejemplo: SELECT ename FROM emp es una sentencia.

3.2.3. Sentencias SQL

Reglas para escribir sentencias SQL

- Las sentencias SQL no son de caso sensitivo, a menos que se indique de esta manera.
- Las sentencias SQL se pueden ingresar en una o más líneas.
- Las palabras claves no pueden dividirse con líneas o abreviarse.
- Las cláusulas son usualmente colocadas en líneas separadas para mejorara la lectura y facilitar la edición.
- Las palabras claves típicamente se ingresan en mayúsculas; todas las otras palabras, tales como el nombre de las tablas y columnas, se ingresan en minúsculas.

- Dentro del SQL*Plus, una sentencia SQL es ingresada en el prompt de SQL, y las siguientes líneas se enumeran. Esto es llamado el buffer de SQL. Solo una sentencia SQL puede correr en un momento dado en el buffer.

Reglas para ejecutar sentencias SQL

- Coloque un punto y coma (;) al final de la última cláusula.
- Coloque un slash (/) en la línea final en el buffer.
- Coloque un slash (/) en el prompt de SQL.
- Edite un comando SQL*Plus RUN en el prompt de SQL.

3.2.4. Seleccionado todas las columnas

Usted puede desplegar todas las columnas de datos en una tabla colocando un asterisco (*) o la lista de todas las columnas después de la palabra clave SELECT.

Ejemplo: Mostrar todas las columnas de la tabla DEPT.

```
SQL> SELECT  *
  2    FROM    dept;
```

```
DEPTNO   DNAME          LOC
         ---------  ----------  -------------
      10  ACCOUNTING    NEW YORK
      20  RESEARCH      DALLAS
      30  SALES         CHICAGO
      40  OPERATIONS    BOSTON
```

Para este ejemplo también podemos ejecutar la siguiente sentencia SQL obteniendo el mismo resultado.

```
SQL> SELECT  deptno, dname, loc
  2     FROM    dept;
```

3.2.5. Seleccionando columnas específicas

Usted puede usar la sentencia SELECT para desplegar columnas específicas de la tabla especificando los nombres de las columnas, separados por comas.

Ejemplo: Mostrar el número (deptno) y nombre (dname) de cada departamento en la tabla DEPT.

```
SQL> SELECT deptno, dname
   2     FROM    dept;
```

```
    DEPTNO   DNAME
---------- --------------
        10   ACCOUNTING
        20   RESEARCH
        30   SALES
        40   OPERATIONS
```

En la cláusula SELECT, se especifican las columnas que usted quiere ver, en el orden en el que usted quiere que aparezcan en la salida. Por ejemplo, para desplegar nombre antes de número de departamento, use la siguiente sentencia:

```
SQL> SELECT dname, deptno
  2  FROM   dept;
```

```
DNAME          DEPTNO
-------------  ----------
ACCOUNTING         10
RESEARCH           20
SALES              30
OPERATIONS             40
```

3.2.6. Eliminando filas duplicadas

A menos que se indique de otra manera, SQL*Plus despliega los resultados de la consulta sin eliminar las filas duplicadas.

Ejemplo: Mostrar todos los números de departamento (deptno) en la tabla EMP.

```
SQL> SELECT deptno
     2  FROM   emp;
```

```
DEPTNO
-----------
        20
        30
        30
        20
                                    . . .
14 filas seleccionadas.
```

Para eliminar las filas duplicadas en el resultado, se incluye la palabra clave DISTINCT en la cláusula SELECT inmediatamente después de la palabra clave SELECT. Como se muestra en la siguiente sentencia:

```
SQL> SELECT DISTINCT  deptno
     2  FROM   emp;
```

```
 DEPTNO
-------------
          10
          20
          30
```

Usted puede especificar múltiples columnas después del calificador DISTINCT. El calificador DISTINCT afecta todas las columnas seleccionadas y el resultado representa una combinación distinta de las columnas.

Notas Rápidas:

- Por defecto SQL*Plus despliega los resultados de la consulta sin eliminar las filas duplicadas.
- Para eliminar las filas duplicadas use la palabra clave DISTINCT en la cláusula SELECT.

3.2.7 Encabezamiento de columnas por defecto

Las columnas con tipos de datos caracteres y fecha son justificadas a la izquierda dentro del ancho de la columna. Las columnas con tipo de datos numérico se justifican a la derecha.

```
SQL> SELECT empno, ename, hiredate
     2  FROM   emp;
```

```
 EMPNO    ENAME       HIREDATE
-----------    ------------     ---------------
      7369     SMITH       17-DIC-80
      7499     ALLEN       20-FEB-81
      7521     WARD        22-FEB-81
                    7566    JONES        02-ABR-81
      7654     MARTIN     28-SEP-81
. . .
14 filas seleccionadas.
```

El encabezamiento de la columna aparece en mayúsculas por defecto. Usted puede sobrescribir el encabezamiento de la columna desplegando con un alias.

Notas Rápidas:

Justificación por defecto:

- Izquierda: Tipos de datos fecha y carácter
- Derecha: Tipos de datos numéricos
- Despliegue por defecto: Mayúsculas

3.2.8 Expresiones aritméticas

Para realizar cálculos, o modificar la manera en la cual se despliegan los datos se pueden usar las expresiones aritméticas. Una expresión aritmética puede contener nombres de columnas, valores de constantes numéricas, y operadores aritméticos.

Operadores aritméticos

Usted puede usar operadores aritméticos en cualquier cláusula de sentencias SQL excepto en la cláusula FROM.

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División

Ejemplo: Use el operador suma para calcular un aumento de salario de $50000 para todos los empleados y despliegue una nueva columna SAL + 500.

```
SQL> SELECT ename, sal, sal+500
  2  FROM   emp;
```

```
ENAME       SAL    SAL+500
----------  ---------  ------------
SMITH         800          1300
ALLEN        1600          2100
WARD         1250          1750
JONES         2975          3475
MARTIN       1250          1750
BLAKE         2850          3350
. . .
14 filas seleccionadas.
```

3.2.9 Precedencia de operadores

Si una expresión aritmética contiene más de un operador, la multiplicación y la división son evaluadas primero. si los operadores dentro de una expresión tiene la misma prioridad, la evaluación se hace de izquierda a derecha.

Usted puede usar paréntesis para forzar las expresiones dentro de paréntesis a ser evaluadas primero.

```
* / + -
```

Ejemplo: Despliegue una nueva columna que calcule el salario anual más una bonificación de $ 200 para cada empleado en la tabla EMP.

```
SQL> SELECT ename, sal, 12*sal+200
  2    FROM    emp;
```

```
ENAME        SAL 12*SAL+200
---------- --------- ---------------
SMITH         800          9800
ALLEN        1600         19400
WARD         1250         15200
JONES        2975         35900
MARTIN       1250         15200
BLAKE        2850         34400
CLARK        2450         29600
SCOTT        3000         36200
. . .
14 filas seleccionadas.
```

3.2.10 Definiendo valores nulos

Un valor nulo (NULL) es un valor no disponible, no asignado, desconocido, o no aplicable. Un valor nulo (NULL) nos es lo mismo que un cero o un espacio en blanco. Cero es un número y un espacio en blanco es un carácter.

Las columnas de cualquier tipo de dato pueden contener valores nulos, a menos que la columna se defina como NOT NULL, o como PRIMARY KEY cuando fue creada.

Ejemplo: Mostrar el trabajo, salario y la comisión que gana cada empleado en la tabla EMP.

```
SQL> SELECT ename, job, sal, comm
  2  FROM   emp;
```

```
ENAME   JOB              SAL   COMM
---------  -----------------  ---------  ---------
SMITH   CLERK              800
ALLEN   SALESMAN          1600        300
. . .
TURNER  SALESMAN          1500          0
ADAMS   CLERK             1100
. . .
14 filas seleccionadas.
```

En el ejemplo que se presenta, note que sólo los vendedores (SALESMAN), ganan comisión. Para representar el hecho de que los otros empleados no ganan comisión se usa un valor nulo en la columna COMM para esos empleados. Es diferente el caso de TURNER quien siendo un vendedor su comisión es de valor 0, no nulo.

3.2.11 Valores nulos en expresiones aritméticas

Si en cualquier columna el valor en una expresión aritmética es un nulo, el resultado es nulo. Si usted intenta realizar una división por cero, usted obtendrá un error. Sin embargo, si usted divide un número por un valor nulo, el resultado es nulo o desconocido.

Ejemplo: Mostrar en una columna el salario anual más la comisión de cada empleado.

```
SQL> SELECT ename, job, sal, 12*sal+comm
  2  FROM   emp;
```

```
ENAME      JOB              SAL    12*SAL+COMM
---------- ---------- --------- -----------
. . .
KING       PRESIDENT    5000
TURNER     SALESMAN     1500          18000
. . .
14 filas seleccionadas.
```

En el ejemplo que se presenta, el empleado KING no es un vendedor (SALESMAN) y no obtiene ninguna comisión. Debido a que la columna COMM en la expresión aritmética es nulo, el resultado es nulo.

3.2.12 Definiendo un alias de columna

Cuando se despliega el resultado de una consulta, SQLl*PLUS normalmente usa el nombre de la columna seleccionada como encabezado de columna. En muchos casos, este encabezamiento puede no ser descriptivo lo cual puede dificultar la comprensión. Usted puede cambiar una cabecera de columna usando un alias de columna.

Especifique el alias después de la columna en la lista del SELECT usando un espacio como separador, o después de la palabra clave AS. Por defecto, el encabezamiento alias aparece en mayúsculas. Si el alias contiene espacios, caracteres especiales (tales como # o $), o es caso sensitivo, encierre el alias entre comillas dobles (“ ”).

Ejemplo: Definir el alias Nombre para la columna ename en la tabla EMP.

```
SQL> SELECT ename AS nombre
  2     FROM    emp;
```

```
NOMBRE
----------
…
```

En el ejemplo que se presenta, el resultado de la consulta sería el mismo si se usa o no la palabra clave AS. También note que la sentencia SQL tiene el alias de columna en minúsculas, mientras que el resultado de la consulta despliega el encabezado de columna en mayúscula.

Para cambiar el despliegue por defecto (mayúsculas) del encabezado de columna en la salida encierre el alias como quiere que aparezca en a salida entre comillas dobles (“ ”). Por ejemplo: “Nombre” o “nombre”.

Notas Rápidas:

Un alias de columna:

- Renombra una cabecera de columna.
- Es útil al realizar cálculos.

- Se define colocando la palabra clave AS o un espacio entre el nombre de la columna y el alias.
- Requiere doble comillas si contiene espacios, caracteres especiales o es de caso sensitivo.

3.2.13. Operador concatenación

Usted puede unir columnas a otras columnas, expresiones aritméticas, o valores constantes para crear expresiones de caracteres usando el operador concatenación (||). Columnas que se mostraban separadas con el operador son combinadas para hacer una sola columna de salida.

La palabra clave AS antes del nombre del alias hace que la cláusula SELECT sea fácil de leer.

Ejemplo: Concatenar las columnas dname y loc de la tabla dept, y usar como alias para la nueva columna el nombre Departamentos.

```
SQL>  SELECT    dname|| loc AS "Departamentos"
  2    FROM       dept;
```

```
Departamentos
----------------------------
ACCOUNTINGNEW YORK
RESEARCHDALLAS
SALESCHICAGO
OPERATIONSBOSTON
```

3.2.14 Cadenas de caracteres literales

Un literal es un carácter, un número, o un dato incluido en la lista de SELECT que no es un nombre de columna o un alias de columna. Este se imprime para cada fila que retorna. Las cadenas literales de texto de libre formato se pueden incluir en el resultado del consulta y se tratan de igual forma que una columna en la lista de SELECT.

Los literales deben encerrarse en comillas simples (‘ ’).

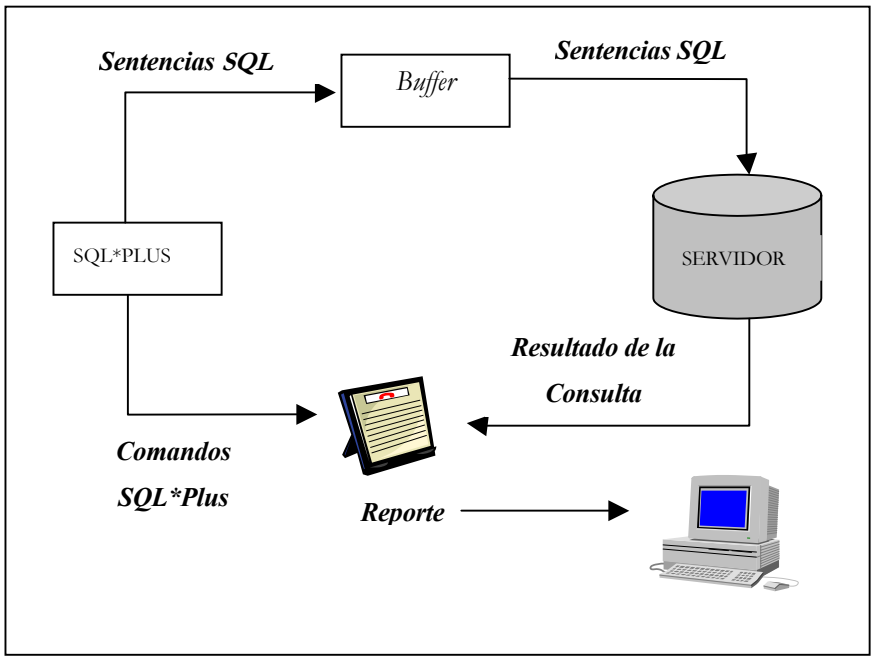
Ejemplo:

```
SQL>  SELECT    dname||' está en '||loc AS
  2                 "Departamentos"
  3     FROM      dept;
```

```
Departamentos
----------------------------
ACCOUNTING está en NEW YORK
RESEARCH está en DALLAS
SALES está en CHICAGO
OPERATIONS está en BOSTON
```

El ejemplo que se presenta despliega los nombres y localizaciones de todos los departamentos. La columna tiene el encabezado Departamentos. Note los espacios entre las comillas sencillas en la sentencia SELECT. Estos espacios mejoran la lectura de la salida.

3.2.15 Interacción SQL - SQL*PLUS



SQL: Es un lenguaje que consiste en comandos para guardar, recuperar, mantener y regular el acceso a una base de datos.

SQL*Plus: Es una aplicación que reconoce y ejecuta comandos SQL y comandos especializados de SQL*Plus que pueden personalizar reportes, proveer facilidades de ayuda y edición, así como mantener variables del sistema.

3.2.16 Comparación SQL - SQL*PLUS

SQL	SQL*PLUS
Es un lenguaje de comunicación con el servidor de Oracle para el acceso de datos	Reconoce sentencias SQL y las envía al servidor
Está basado en el American National Standars Institute (ANSI) estándar en el SQL	Es propietario de la Interfase para ejecutar sentencias SQL
Manipula los datos en la tabla de definiciones en la base de datos	No permite la manipulación de los valores en la base de datos
Se entra en el SQL buffer en una o más líneas	Se entra en una línea a la vez; no almacena en el SQL buffer
No tiene un carácter de continuación	Tiene un guión como carácter de continuación si el comando es más largo que una línea
No se puede abreviar	Se puede abreviar
Usa un carácter de terminación para ejecutar un comando inmediatamente	No requiere caracteres de terminación; los comandos se ejecutan inmediatamente
Usa funciones para realizar formatos	Usa comandos para el formato de los datos

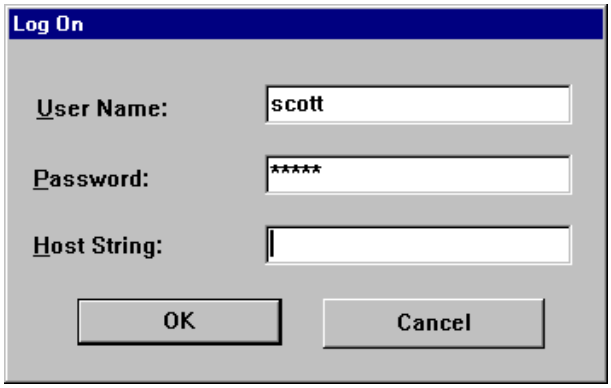
3.2.17 Visión general de SQL*Plus

Los comandos de SQL*PLUS pueden dividirse en las siguientes categorías principales:

CATEGORÍA	PROPÓSITO
Ambiente	Afecta en general el comportamiento de las sentencias SQL para la sesión
Formato	Da formato a los resultados de las consultas
Manipulación de archivos	Guarda, carga y corre archivos de script
Ejecución	Envía sentencias SQL de el SQL buffer al Oracle8 Server
Edición	Modifica sentencias SQL en el buffer
Interacción	Permite crear y pasar variables a sentencias SQL, imprime valores de variables, e imprime mensajes para la pantalla
Miscelánea	Tiene varios comandos de conexión a bases de datos, manipula el SQL*Plus Environment, y muestra definiciones de columnas

3.2.18 Registrándose en SQL*Plus

a) Interfaz Grafica



1. Clic inicio → programas → Oracle para Windows NT → SQL*Plus8.0

2. Llenar el nombre de usuario, contraseña, y cadena conexión.

b) Desde la línea de Comando

SQL> connect [username[/password[@database]]]

Username = nombre de usuario

Password = clave de acceso

@database = alias de la base de datos

***c)* Cierre de sesión**

SQL> exit <ENTER>

3.2.19 Mostrando la estructura de la tabla

En SQL*Plus, usted puede mostrar la estructura de la tabla usando el comando describe. El resultado del comando es ver los nombres de las columnas y los tipos de datos.

Sintaxis:

DESCRIBE nombre_tabla

nombre_tabla: Es el nombre cualquier tabla existente, vista o sinónimo accesible para el usuario.

Ejemplo: mostrar la estructura de la tabla dept.

```
SQL> DESCRIBE dept
```

```
Nombre                    Nulo?    Tipo
------------------------- -------- ----
DEPTNO                    NOT NULL NUMBER(2)
DNAME                              VARCHAR2(14)
LOC                                VARCHAR2(13)
```

En el resultado:

nulo? Indica si una columna debe contener datos;

NOT NULL Indica que una columna debe contener datos

Tipo Muestra el tipo de datos de la columna

Los tipos de datos se describen en la siguiente tabla:

Tipo de Dato	**DESCRIPCIÓN**
NUMBER(p,s)	Valor numérico, donde p es el máximo número de dígitos, y s es el número de dígito a la derecha de un punto decimal
VARCHAR2(s)	Carácter de longitud variable con valor máximo s
DATE	Fecha y tiempo con valor entre enero 1 de 4712 a.c y diciembre 31 de 9999 dc
CHAR(s)	Carácter de longitud fija con tamaño s

3.2.20 Comandos de edición SQL*Plus

COMANDO	DESCRIPCION
A[PPEND] text	Añade texto al final de la línea corriente
C[HANGE]/old/new C[HANGE]/text/ CL[EAR]BUFF[ER]	Cambia texto viejo por nuevo en la línea corriente Borra texto de la línea corriente Borra todas las líneas del SQL buffer
DEL I[INPUT] I[INPUT]TEXT L[IST]	Borra la línea corriente Inserta un indefinido número de líneas Inserta una líneas consistente de texto Lista todas las líneas en el SQL buffer
L[IST]n L[IST]m n R[UN]	Lista un a línea (especificada por n) Lista un rango delineas (de m a n) Despliega y corre la sentencia SQL actual en el buffer
n n text o text	Especifica la línea que se va a correr Remplaza la línea n con texto Inserta una línea antes de la línea 1

Notas Rápidas:

- Usted puede entrar sólo en el SQL*Plus comandos por el prompt de SQL.

- Los comandos SQL*Plus nos se almacenan en el buffer.

- Para continuar una comando SQL*Plus en la siguiente línea, termine la línea actual con un guión (-).

3.2.21 Comandos de archivo SQL*Plus

COMANDO	DESCRIPCION
SAV[E] filename [.ext]	Guarda el contenido actual del SQL buffer en un archivo. Usa APPEND para añadir al archivo existente; usa REPLACE para sobrescribir el archivo existente. La extensión por defecto es .sql
GET filename[.ext]	Escribe el contenido previamente guardado en el archivo del SQL buffer. La extensión por defecto es .sql
STA[RT]filename [.ext]	Corre el comando de archivo previamente guardado
@filename	Corre el comando de archivo previamente guardado (al igual que START)
ED[IT]	Invoca el editor y guarda el contenido del buffer en una archivo llamado afiedt.buf
ED[IT] [filename [.ext]]	Invoca el editor y edita el contenido de un archivo guardado
SPO[OL]filename[.ext] OFF[OUT]	Guarda el resultado de las consulta en un archivo. OFF cierra la pila del archivo. OUT cierra la pila del archivo y envía el resultado del archivo a la impresora del sistema
EXIT	Sale del SQL*Plus

3.3 RESTRINGIENDO Y ORDENANDO DATOS

3.3.1 Limitando filas usando la cláusula WHERE

Usted puede limitar las filas recuperadas en una consulta usando la cláusula WHERE.

Sintaxis:

```
SELECT      [DISTINCT] {*, columna, [ alias]...}
FROM        Tabla
[WHERE      Condición (s)];
```

Donde:

WHERE Restringe la consulta a las filas que cumplen la condición

Condición Está compuesta de nombres de columnas, expresiones, constantes y un operador de comparación.

La cláusula WHERE puede comparar los valores en columnas, valores literales, expresiones aritméticas, o funciones.

Ejemplo: mostrar el nombre de todos los empleados que trabajen como vendedores (SALESMAN).

```
SQL> SELECT ename, job
  2  FROM   emp
  3  WHERE  job = 'SALESMAN';
```

```
ENAME      JOB
---------- ---------
ALLEN      SALESMAN
WARD       SALESMAN
MARTIN     SALESMAN
TURNER     SALESMAN
```

El nombre del trabajo (SALESMAN) debe ir en mayúsculas porque las cadenas de caracteres son de caso sensitivo.

3.3.2 Cadenas de caracteres y fechas

Los valores de tipo de datos cadenas de caracteres y fechas en la cláusula WHERE deben encerrarse en comillas simples (‘ ’); mientras que las constantes numéricas no. Todos los caracteres buscados son de caso sensitivo.

Oracle almacena las fechas en un formato número interno, representando el siglo, el año, el mes, el día, las horas, los minutos y los segundos. Por defecto la fecha se despliega DD-MM-AA.

Ejemplo: mostrar el resultado de la consulta de la lección anterior sin colocar en mayúsculas SALESMAN.

```
SQL> SELECT ENAME, JOB
  2    FROM   EMP
  3    WHERE  JOB = 'SALESMAN';
```

```
ninguna fila seleccionada
```

En este caso ninguna fila es seleccionada porque la columna job almacena los valores en mayúsculas y los caracteres buscados son de caso sensitivo.

3.3.3 Operadores de comparación

Los operadores de comparación son usados cuando se compara una expresión con otra. Estos son usados en la cláusula WHERE con el siguiente formato:

Sintaxis:

WHERE Expresión Operador Valor

OPERADOR	DESCRIPCIÓN
=	IGUAL A
>	MAYOR QUE
>=	MAYOR O IGUAL QUE
<	MENOR QUE
<=	MENOR O IGUAL QUE
<>	DIFERENTE A

Ejemplo: Mostrar el nombre de todos los empleados que devenguen un salario mayor de $2500.

```
SQL> SELECT ename, sal
  2  FROM   emp
  3  WHERE  sal > 2500;
```

```
ENAME          SAL
----------     ---------
JONES           2975
BLAKE           2850
SCOTT           3000
KING            5000
FORD            3000
```

3.3.4 Otros operadores de comparación

Operador	**Descripción**
BETWEEN ....AND...	Entre dos valores
IN (Lista)	Igual que algún miembro de la lista
LIKE	Concuerda con un patrón de caracteres
IS NULL	Es un valor nulo

- **Operador BETWEEN**

Usted puede desplegar filas basadas en un rango de valores usando el operador BETWEEN. El rango que usted especifica contiene un valor máximo y uno mínimo.

Ejemplo: mostrar el nombre de todos los empleados que devenguen un salario entre $ 2000 y $ 4000.

```
SQL> SELECT ename, sal
  2  FROM       emp
  3  WHERE      sal  BETWEEN 2000 AND 4000;
```

```
ENAME          SAL
----------     ---------
JONES           2975
BLAKE           2850
CLARK           2450
SCOTT           3000
FORD            3000
```

- **Operador IN**

Para probar los valores en una lista específica use el operador IN. Este operador puede usarse con cualquier tipo de datos. Los caracteres y fechas se encierran en comillas simples.

Ejemplo: mostrar el nombre de todos los empleados que trabajen en los departamentos número 10 o 30.

```
SQL> SELECT ENAME, DEPTNO
  2  FROM   EMP
  3  WHERE  DEPTNO IN (10, 30);
```

```
ENAME    DEPTNO
---------  --------------
BLAKE              30
CLARK              10
. . .
9 filas seleccionadas.
```

- **Operador LIKE**

Para el encaje de patrones sobre cadenas de caracteres use el operador LIKE. Para la descripción se patrones se usan los dos caracteres especiales siguientes:

% Representa una secuencia de cero o más caracteres.

_ Representa un solo carácter.

Ejemplo: Mostrar el nombre de todos los empleados, cuyo nombre empiece con la letra M.

```
SQL> SELECT ename
  2  FROM   emp
  3  WHERE  ename LIKE 'M%';
```

```
ENAME
----------
MARTIN
MILLER
```

Para que los patrones puedan contener los caracteres especiales (% y _), SQL permite la especificación de un caracter de escape que se utiliza inmediatamente antes de un caracter especial, para indicar que este va a ser tratado como un caracter normal. El caracter de escape se define utilizando la palabra clave ESCAPE.

Ejemplo: Mostrar la información del departamento cuyo nombre contenga el caracter subrayado (_) .

```
SQL> SELECT *
  2  FROM  dept
  3  WHERE  dname LIKE '% \ _ %' ESCAPE ' \ ';
```

En el ejemplo que se presenta el carácter de escape es el backslash (\).

Operador IS NULL

Para probar los valores que son nulos use el operador IS NULL. Un valor nulo es un valor no disponible, no asignado, desconocido, o no aplicable. Usted no puede probar los valores nulos con el signo igual (=) porque un valor nulo no es igual a nada.

Ejemplo: Mostrar los nombres de todos los empleados que no están habilitados para ganar comisión.

```
SQL> SELECT ename, job, comm
  2  FROM   emp
  3  WHERE  comm IS NULL;
```

En el ejemplo que se presenta se obtendrá como resultado los datos de todos los trabajadores que no sean SALESMAN, ya que sólo ellos están habilitados para ganar comisiones.

3.3.5 Operadores lógicos

Los operadores lógicos combinan el resultado de dos condiciones compuestas produciendo un solo resultado basado en estas o invirtiendo el resultado de una condición simple.

Operador	***Descripción***
AND	Retorna TRUE (Verdadero) si las condiciones son TRUE.
OR	Retorna TRUE (Verdadero) si cualquiera de las condiciones es TRUE.
NOT	Retorna TRUE (Verdadero) si la siguiente condición es FALSE (Falsa).

Operador AND

Requiere que las dos condiciones sean TRUE. Todos los caracteres son de caso sensitivo. Las cadenas de caracteres deben encerrase entre comillas simples (‘ ’).

Ejemplo: Mostrar el nombre de todos los empleados cuyos nombres empiecen con la letra M y devenguen más de $1000.

```
SQL> SELECT ename, sal
  2  FROM   emp
  3  WHERE  ename LIKE 'M%' AND sal > 1000;
```

```
ENAME       SAL
----------  ---------
MARTIN      1250
MILLER      1300
```

Operador OR

Requiere que cualquiera de las condiciones sea TRUE.

Ejemplo: Mostrar el nombre de todos los empleados cuyos nombres empiecen con la letra M o devenguen más de $1000.

```
SQL> SELECT ename, sal
  2  FROM   emp
  3  WHERE  ename LIKE 'M%' OR sal > 1000;
```

```
ENAME          SAL
---------- ---------
ALLEN          1600
WARD           1250
JONES          2975
. . .
12 filas seleccionadas.
```

En el ejemplo que se muestra el resultado es diferente al de la consulta anterior, porque se muestran las filas que cumplan cualquiera de las dos condiciones.

Operador NOT

Requiere que la siguiente condición sea FALSE.

Ejemplo: Mostrar la información de todos los departamentos que no se localicen en DALLAS ni en CHICAGO.

```
SQL> SELECT deptno, dname, loc
  2  FROM   dept
  3  WHERE  loc NOT IN ('DALLAS', 'CHICAGO');
```

```
DEPTNO  DNAME          LOC
------  -------------- -------------
    10  ACCOUNTING     NEW YORK
    40  OPERATIONS     BOSTON
```

3.3.6 Precedencia de operadores

Orden De Evaluación	Operador
1	Todos los operadores de comparación
2	NOT
3	AND
4	OR

Nota Rápida:

- Se puede forzar la prioridad usando paréntesis.

Ejemplo: Mostrar el nombre, trabajo y salario de todos los vendedores y gerentes que ganen más de $ 1500

```
SQL> SELECT ename, job, sal
    2  FROM   EMP
    3  WHERE  sal > 1500 AND ( job = 'MANAGER' OR
                  job = 'SALESMAN');
```

ENAME	JOB	SAL
ALLEN	SALESMAN	1600
JONES	MANAGER	2975
BLAKE	MANAGER	2850
CLARK	MANAGER	2450

3.3.7 Cláusula ORDER BY

Para definir el orden en que retornarán las filas resultantes de una consulta use la cláusula ORDER BY. Para especificar el tipo de ordenación use la cláusula DESC para orden descendente y ASC para orden ascendente. El orden por defecto es ascendente. Se puede orden con respecto a más de una columna.

Sintaxis:

```
SELECT          expresión
FROM            tabla
[WHERE          condición (s)]
[ORDER BY       {columna, expression}[ASC/DESC]];
```

Donde:

ORDER BY especifica el orden en que retornarán las filas

ASC/DESC orden ascendente y descendente

Ejemplo: Mostrar el nombre, trabajo y salario de los empleados del departamento 20 en orden ascendente por salario.

```
SQL> SELECT ename, job, sal, deptno
  2  FROM   EMP
  3    WHERE  deptno = 20
  4   ORDER BY sal ASC;
```

ENAME	JOB	SAL	DEPTNO
SMITH	CLERK	800	20
ADAMS	CLERK	1100	20
JONES	MANAGER	2975	20
SCOTT	ANALYST	3000	20
FORD	ANALYST	3000	20

Para ordenar en forma descendente coloque la palabra clave DESC, en lugar de ASC.

Ordenando por un alias de columna

Usted puede usar un alias de columna en la cláusula ORDER BY.

Ejemplo:

```
SQL> SELECT ename, sal, sal+500 aumento
  2  FROM   emp
  3  ORDER BY aumento;
```

```
ENAME        SAL   AUMENTO
----------   ---------   ---------------
SMITH          800            1300
JAMES          950            1450
ADAMS         1100            1600
. . .
14 filas seleccionadas.
```

Ordenando por múltiples columnas

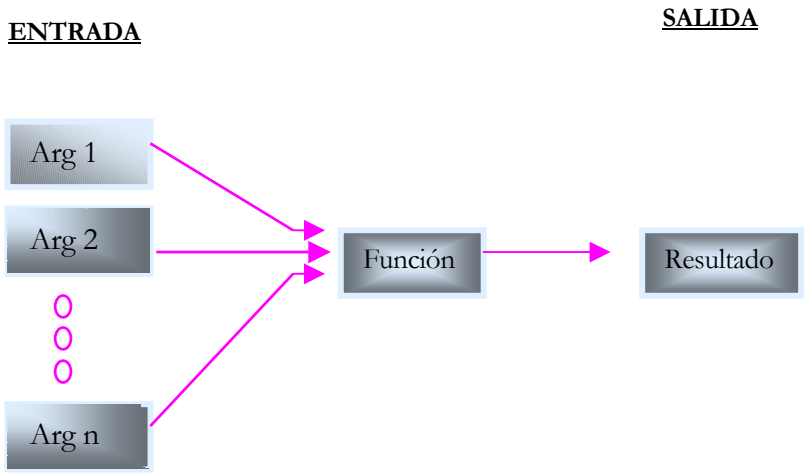
Usted puede ordenar el resultado de una consulta por más de una columna. Tantas como tenga la tabla. Especifique el nombre de las columnas en la cláusula ORDER BY separados por comas.

Ejemplo: Mostrar el nombre, trabajo y salario de todos los empleados ordenados por trabajo por nombre si tiene trabajos iguales.

```
SQL> SELECT ename, job
  2  FROM emp
  3  ORDER BY job, ename;
```

3.4 FUNCIONES SQL

3.4.1 Funciones SQL



Las funciones son una característica poderosa de SQL y pueden usarse para hacer lo siguiente:

- Realizar cálculos sobre datos.
- Modificar datos individuales.
- Manipular la salida de grupos de filas.
- Dar formato a fechas y números para ser mostrados.
- Convertir tipos de datos de columna.

Las funciones SQL pueden aceptar argumentos y siempre retornan un valor.

Existen dos tipos de funciones en SQL:

- Funciones de una fila.
- Funciones de grupo.

En el desarrollo de esta unidad se cubrirá el estudio de los dos tipos de funciones.

3.4.2 Funciones de una fila

Estas funciones operan sobre una fila solamente y retornan un resultado por fila. Hay varios tipos de funciones de una fila, en esta unidad se cubrirán los tipos caracter, numérica y fecha.

Sintaxis:

nombre_función (columna / expresión, [arg1, arg2, ...])

Donde:

nombre_función	Es el nombre de la función
columna	Es el nombre de cualquier columna de la base de datos
expresión	Es cualquier cadena de caracteres o expresión calculada

arg1, arg2 Es cualquier argumento a usar por la función.

Funciones de tipo de dato caracter

Aceptan valores de tipo caracter de entrada y pueden retornan tanto valores de tipo carácter como numérico.

Función	Descripción
LOVER (colu/exp)	Convierte los caracteres a minúscula
UPPER (colu/exp)	Convierte los caracteres a mayúscula
INITCAP (colu/exp)	Convierte la primera letra a mayúscula y los demás en minúsculas
CONCAT(colu1/exp1. (colu2/exp2)	Concatena dos cadenas
SUBSTR(colu/exp,m,n)	Devuelve n caracteres desde m
LENGTH (colu/exp)	Devuelve el número de caracteres
INSTR (colu/exp, m)	Devuelve la posición de un caracter
TRIM (caracter FROM fuente)	Borra un caracter de una cadena

Ejemplo: Mostrar el nombre de cada empleado y la longitud de cada nombre.

```
SQL> select ename, LENGTH(ename)
  2  FROM emp;
```

```
ENAME      LENGTH(ENAME)
---------- -------------
SMITH                  5
ALLEN                  5
WARD                   4
JONES                  5
. . .
14 filas seleccionadas.
```

Funciones de tipo de dato numérico

Aceptan valores de tipo numérico de entrada y retornan valores de tipo numérico.

Función	***Descripción***
ROUND (colu/exp,n)	Retorna colu/exp redondeado a n dígitos después del punto decimal
TRUNC (colu/exp,n)	Retorna colu/exp truncado a n dígitos después del punto decimal
MOD (m,n)	Retorna el residuo de dividir m entre n

Ejemplos:

ROUND (2.438,2)		2.44
TRUNC (2.431,2)	⟶	2.43
MOD (10, 3)		1

Funciones de tipo de dato fecha

Operan sobre valores de tipo fecha y todas retornan valores de tipo fecha, excepto MONTHS_BETWEEN que retorna un valor numérico.

Oracle almacena las fechas con el formato interno por defecto: DD-MES-AA.

La aritmética con tipos de datos fechas puede resumirse de la siguiente manera:

- Fecha + número: Suma un número de días a una fecha, produciendo otra fecha.
- Fecha – número: Resta un número de días a una fecha, produciendo otra fecha.
- Fecha – Fecha: Resta una fecha de otra, produciendo el número de días entre las dos.

Función	*Descripción*
ADD_MONTHS	Suma meses a la fecha
LAST_DAY	Ültimo día del mes
NEXT_DAY	Próximo día de la fecha especificada
MONTHS_BETWEEN	Número de meses entre dos fechas
ROUND	Redondea fechas
TRUNC	Trunca fechas

Ejemplos:

ADD_MONTHS ('11-JAN-97',6) ⟶ '11-JUL-97
NEXT_DAY('01-SEP-97','FRIDAY') ⟶ '08-SEP-97'
ROUND('27-JUL-97',MONTH) ⟶ '01-AUG-97'
TRUNC('27-JUL-97',MONTH) ⟶ '01-JUL-97'

3.4.3 Referencia a los valores de fecha

Elementos del formato de los valores de fecha

Elemento	*Descripción*
YYYY	Año con cuatro dígitos
MM	Mes con dos dígitos
MONTH	Nombre completo del mes
DY	Día de la semana con tres letras
DAY	Nombre completo del día

Para formatear la visualización de los valores de fecha se usa la función de conversión TO CHAR

Ejemplo:

SELECT TO_CHAR (hiredate, 'fmDD " of " MONTH YYYY') "Date of Hire"

Función SYSDATE

La función SYSDATE devuelve la fecha y hora actual.

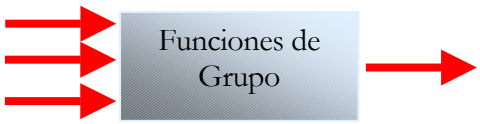
Ejemplo:

Mostrar el nombre y las semanas de trabajo de los empleados del departamento 10.

```
SQL> SELECT ename, ROUND((SYSDATE - hiredate)/7,0) SEMANAS
 2  FROM  emp
 3  WHERE deptno=10;
```

```
ENAME      SEMANAS
----------  ----------
CLARK            1112
KING             1089
MILLER           1079
```

3.4.4 Funciones de grupo



Estas funciones operan sobre grupos de filas y retornan un resultado por grupo.

Los tipos de funciones de se muestran en la siguiente tabla:

Función	Descripción
AVG([DISTINCT/ALL]n)	Promedio aritmético de n
COUNT({*/[DISTINCT/ ALL]expr})	Número de valores no nulos de expr. Número de filas no nulas em un grupo.
MAX([DISTINCT/ALL] expr)	Valor máximo de expr
MIN([DISTINCT/ALL] expr)	Valor mínimo de expr
SUM([DISTINCT/ALL]n)	Suma de los valores de n

Ejemplo 1: Muestre el promedio, valor máximo y la suma de los salarios anuales de todos los gerentes.

```
SQL> SELECT AVG(sal), MAX(sal), SUM(sal)
     2  FROM   EMP
     3  WHERE  job ='MANAGER';
```

```
AVG(SAL)  MAX(SAL)  SUM(SAL)
--------- --------- ---------
 2758.3333      2975      8275
```

Ejemplo 2: Muestre el número de empleados.

```
SQL> SELECT COUNT(ename)
  2  FROM emp;
```

```
COUNT(ENAME)
----------
        14
```

3.4.5 Cláusula GROUP BY

Usted puede usar la cláusula GROUP BY para dividir las filas de una tabla en grupos.

Sintaxis:

```
SELECT          columna, función_grupo(columna)
FROM            tabla
[WHERE          condición]
[GROUP BY       expresión_group_by]
[ORDER BY       columna];
```

Donde:

expresión_group_by especifica las columnas cuyos valores determinan las bases para agrupar las filas

La columna del GROUP BY no tiene que estar en la lista de SELECT.

Ejemplo: Muestre cada departamento y su número de empleados agrupados por departamentos.

```
SQL> SELECT deptno, COUNT(*)
  2  FROM   emp
  3  GROUP BY deptno
```

```
DEPTNO  COUNT(*)
--------- ---------
      10         3
      20         5
      30         6
```

Agrupando por múltiples columnas

Ejemplo: Muestre la cantidad de empleados para cada categoría de cargos en cada departamento.

```
SQL> SELECT deptno, job, COUNT(*)
     2  FROM   emp
     3  GROUP BY deptno, job;
```

```
DEPTNO JOB              COUNT(*)
------------ ---------------- -------------
          10 CLERK                       1
          10 MANAGER
          10 PRESIDENT                   1
          20 ANALYST                     2
          20 CLERK                       2
. . .
9 filas seleccionadas.
```

3.4.6 Cláusula HAVING

Usted puede especificar cuáles grupos van a ser mostrados usando la cláusula HAVING.

Sintaxis:

```
SELECT      columna, función_grupo(columna)
FROM        tabla
[WHERE      condición]
[GROUP BY   expresión_group_by]
[HAVING     condición _group]
[ORDER BY   columna];
```

Donde:

condición _group restringe los grupos de filas retornadas a aquellos grupos para los cuales la condición es verdadera.

Ejemplo: Mostrar el salario anual promedio para todos los tipos de cargos con más de dos empelados.

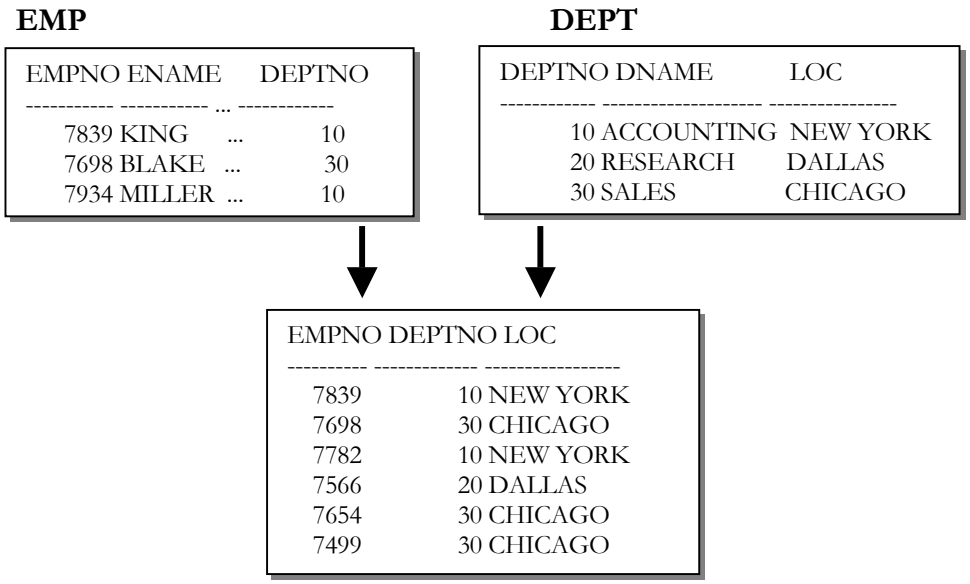
```
SELECT job, 12*AVG(sal)
FROM   emp
GROUP BY job
HAVING COUNT(*) >2
```

```
JOB           12*AVG(SAL)
--------------- ------------------
CLERK                 12450
MANAGER               33100
SALESMAN              16800
```

3.5 OBTENIENDO DATOS DE MULTIPLES TABLAS

3.5.1 Datos de múltiples tablas

Algunas veces usted necesitará usar datos de más de una tabla, esto se consigue uniendo las tablas por medio de los campos que las relacionan. En el ejemplo que se muestra a continuación se presenta un reporte con datos de dos tablas.



3.5.2 ¿Qué es un JOIN?

Un JOIN es una consulta de datos de más de una tabla.

Sintaxis:

```
SELECT    tabla1.columna, tabla2.columna
FROM      tabla1, tabla2
WHERE     tabla1.columna1 = tabla2.columna2
```

Donde:

tabla1.columna Denota la tabla y la columna de cada dato que retorna

tabla1.columna1= tabla2.columna2 Es la condición para relacionar las tablas

Características del JOIN

- Las tablas a ser unidad se especifican en la cláusula FROM.
- En la cláusula WHERE se especifica como unir las tablas.
- Las columnas que tienen nombres iguales en las tablas especificadas en la cláusula FROM deben ser identificadas utilizando NOMBRE_TABLA.NOMBRE_COLUMNA.
- Si las columnas no tienen homónimos en las tablas, no es necesario especificar el nombre de la tabla en las cláusulas WHERE y SELECT.

- El criterio de coincidencia entre las tablas es denominado el predicado del JOIN o criterio del JOIN.
- Se pueden unir cuantas tablas se requieran.
- Cuando n tablas son unidas, es necesario tener al menos n-1 condiciones de JOIN entre parejas de tablas para evitar el producto cartesiano.

3.5.3 Tipos de JOINS

- **Equijoin: Cuando** en la condición del Join los valores en las columnas de ambas tablas deben ser iguales.

Ejemplo: Mostrar el nombre del empleado, nombre del departamento y su localización.

```
SQL> SELECT ename, emp.deptno, loc
  2  FROM emp, dept
  3  WHERE emp.deptno = dept.deptno;
```

```
ENAME      DEPTNO  LOC
----------  ------------  -------------
SMITH              20  DALLAS
ALLEN              30  CHICAGO
. . .
14 filas seleccionadas.
```

- **Non-Equijoins:** Se basan en los operadores: !=, <, <=, >, >=, between.

Ejemplo: Muestre el nombre, salario y grado de salario de cada empleado.

```
SQL> SELECT ename, sal, grade
  2  FROM emp, salgrade
  3  WHERE sal BETWEEN LOSAL AND HISAL
```

```
ENAME      SAL  GRADE
---------- --------- ----------
SMITH        800          1
ADAMS       1100          1
JAMES        950          1
. . .
14 filas seleccionadas.
```

- **Outer Joins:** Es el signo más entre paréntesis (+), este obliga a que un registro que contenga valores nulos en uno de los criterios del JOIN coincida con cada valor de la segunda tabla sobre la cual ordinariamente no estaría esa coincidencia.

Sintaxis:

```
SELECT    tabla1.columna, tabla2.columna
FROM      tabla1, tabla2
WHERE     tabla1.columna1(+) = tabla2.columna2
```

Ejemplo: Muestre el número y la localización de cada departamento y los empleados que trabajan en ellos. Incluya los departamentos en los cuales no trabaja ningún empleado.

```
SQL> SELECT ename, dept.deptno, loc
  2  FROM emp, dept
  3  WHERE emp.deptno(+) = dept.deptno
```

```
ENAME   DEPTNO  LOC
----------  ------------  -----------------
CLARK             10  NEW YORK
KING              10  NEW YORK
MILLER            10  NEW YORK
SMITH             20  DALLAS
ADAMS             20  DALLAS
FORD              20  DALLAS
SCOTT             20  DALLAS
. . .
                  40 BOSTON
15 filas seleccionadas.
```

- **Self Joins:** Es utilizado para buscar coincidencia entre registros sobre columnas diferentes de la misma tabla.

Sintaxis:

```
SELECT      alias1.columna, alias2.columna
FROM        tabla1 alias1, tabla2 alias 2
WHERE       alias1.columna1 = alias2.columna2
```

Ejemplo: Muestre el número y nombre de cada empleado, así como el número y nombre del jefe correspondiente.

```
SQL> SELECT a.empno, a.ename, b.empno, b.ename
  2  FROM  emp a, emp b
  3  WHERE a.mgr=b.empno
```

```
EMPNO ENAME   EMPNO  ENAME
----------  ----------   ----------   ---------
    7369  SMITH          7902  FORD
    7499  ALLEN          7698  BLAKE
    7521  WARD           7698  BLAKE
    7566  JONES          7839  KING
    7654  MARTIN         7698  BLAKE
    7698  BLAKE          7839  KING
    7782 CLARK           7839  KING
    7788  SCOTT          7566  JONES
    7844  TURNER         7698  BLAKE
    7876  ADAMS          7788  SCOTT
    7900  JAMES          7698  BLAKE
    . . .
13 filas seleccionadas.
```

3.5.4 Operadores de conjunto

Los operadores de conjunto combinan dos o más tablas para obtener un resultado.

- **OPERADOR UNION:** Retorna todos los registros distintos seleccionados través de múltiples consultas.

- **OPERADOR INTERSECT:** Retorna valores comunes de múltiples tablas.

- **OPERADOR MINUS:** Retorna todos los registros seleccionados por la primera consulta que no fueron también seleccionados por la segunda consulta.

Sintaxis general:

```
SELECT      columna
FROM        tabla
[WHERE      condiciones]
OPERADOR
SELECT      columna
FROM        tabla
[WHERE      condiciones]
```

Ejemplo:

```
SQL> SELECT ename nombres
  2  FROM emp
  3  UNION
  4  SELECT dname
  5  FROM dept;
```

3.6 SUBCONSULTAS

3.6.1 Subconsultas

Una subconsulta es una sentencia SQL que está embebida en una cláusula de otra sentencia SQL.

La subconsulta (consulta interna) se ejecuta inmediatamente antes de la consulta principal (consulta externa). El resultado de esta subconsulta es usado por la consulta principal.

La subconsulta se encierra entre paréntesis.

Sintaxis:

```
SELECT      lista_select
FROM        tabla
WHERE       expr        operador
                    (SELECT     lista_select
                     FROM       tabla);
```

Donde:

Operador incluye los operadores de comparación como: >, =, o IN.

Ejemplo: Mostrar el nombre de los empleados que trabajan en el mismo departamento que SMITH.

```
SQL> SELECT ename
  2  FROM emp
  3  WHERE deptno=
  4
  5
  6
```

```
ENAME
----------
SMITH
JONES
SCOTT
ADAMS
FORD
```

3.6.2 Subconsultas de una fila

Una subconsulta de una fila es aquella que retorna una sola fila de la sentencia SELECT interna. Este tipo de subconsultas usan operadores de una sola fila como =, >, >=, <, <=, <>.

Ejemplo: Mostrar el nombre y número del jefe del empleado 7499.

```
SQL> SELECT ename, empno
  2  FROM emp
  3  WHERE empno=
  4                    (SELECT mgr
  5
  6
```

```
ENAME   EMPNO
----------  ----------
BLAKE        7698
```

Uso de las Funciones de Grupo en una Subconsulta

Ejemplo: Mostrar el nombre y salario del empleado que devenga el salario máximo.

```
SQL> SELECT ename, sal
  2  FROM emp
  3  WHERE sal=
  4
  5
```

```
ENAME          SAL
---------- ---------
KING          5000
```

3.6.3 Subconsultas de múltiples filas

Una subconsulta de múltiples filas es aquella que retorna más de una fila y en la cual se usan operadores de múltiples filas como IN, ANY, ALL.

Ejemplo 1: Mostrar el nombre de todos los empleados cuyos salarios sean los máximos de cada departamento.

```
SQL> SELECT deptno, ename, sal
  2   FROM emp
  3   WHERE sal IN
  4                    (SELECT MAX(sal)
  5
  6
  7  ORDER BY deptno;
```

```
DEPTNO ENAME      SAL
---------- ---------- ---------
        10 KING         5000
        20 SCOTT       3000
        20 FORD         3000
                         30 BLAKE       2850
```

Ejemplo 2: Mostrar el nombre, salario y departamento de todos los empleados con salario superior a al menos uno de los empleados del departamento 30.

```
SQL> SELECT ename, sal, deptno
  2  FROM emp
  3  WHERE sal > ANY
  4
  5
  6
  7  AND deptno<>30;
```

```
ENAME      SAL DEPTNO
---------- --------- ------------
JONES       2975            20
CLARK      2450            10
SCOTT      3000            20
KING        5000            10
ADAMS     1100            20
FORD       3000            20
MILLER     1300            10
```

3.6.4 Subconsultas de múltiples columnas

Una subconsulta de múltiples columnas es aquella que retorna más de una columna. Las subconsultas de múltiples columnas hacen posible combinar condiciones WHERE duplicadas con condiciones WHERE simples.

Sintaxis:

```
SELECT      columna, columna...
FROM        tabla
WHERE       (columna, columna,...) IN
                                    (SELECT     columna, columna...
            FROM        tabla
                                    WHERE     condición);
```

Ejemplo: Mostrar el número, nombre, número de departamento y salario de todos los empleados que tengan el mismo salario y trabajen en el mismo departamento que el empleado número 7788.

```
SQL> SELECT empno, ename, deptno, sal
  2  FROM emp
  3  WHERE (deptno, sal) IN
  4
  5
  6
  7  AND empno <> 7788;
```

```
EMPNO  ENAME   DEPTNO       SAL
----------  ----------  ----------  ----------
    7902  FORD                20       3000
```

3.7. PRODUCIENDO SALIDAS LEGIBLES CON SQL *PLUS

3.7.1. Variables de sustitución

En SQL*Plus usted puede usar las variables de sustitución para almacenar valores temporalmente.

AMPERSAND (&)

SQL*Plus hace uso de a variables para limitar los datos retornados dinámicamente. Use un ampersand (&) para identificar cada variable en su sentencia SQL. No necesita definir el valor de cada variable.

Ejemplo: Mostrar el número, nombre y salario de un empleado cuyo número digitará el usuario.

```
SQL> SELECT empno, ename, sal
  2  FROM emp
  3  WHERE empno = &numero_empleado;
```

```
Ingresar un valor para numero_empleado: 7369
antiguo   3: where empno=&numero_empleado
nuevo   3: where empno=7369

EMPNO ENAME      SAL
----------- ---------- ---------
     7369 SMITH        800
```

DOBLE AMPERSAND (&&)

Use un doble ampersand (&&) si quiere volver a usar el valor de la variable fuera del prompt del usuario.

Ejemplo: Mostrar el número, nombre del empleado y una columna más que el usuario digite. Ordenar los datos por la columna digitada.

```
SQL> SELECT empno, ename, &&columna
  2  FROM emp
  3  ORDER BY &columna
```

```
Ingresar un valor para columna: deptno

EMPNO ENAME DEPTNO
---------- ---------- ----------
    7782 CLARK          10
                  7839 KING              10
    7934 MILLER         10
    7369 SMITH          20
    7876 ADAMS          20
    . . .
14 filas seleccionadas.
```

3.7.2 Comandos de formato SQL*Plus

Usted puede controlar las características del reporte usando los siguientes comandos que se describen en la siguiente tabla.

Comando	**Descripción**
COL[UMN] [columna opción]	Controla el formato de columna
TTI[TLE] [texto/OFF/ON]	Especifica el encabezado
BTI[TLE] [texto/OFF/ON]	Especifica el pie de página
BRE[AK] [ON/elemento_reporte]	Suprime valores duplicados, salta líneas, permite el control de puntos de ruptura

COMANDO COLUMN

COL[UMN] [{columna/alias} [opción]]

OPCIONES DEL COMANDO COLUMN

- CLE[AR]: Limpia cualquier formato de columna.
- FOR[MAT]format: Cambia el despliegue de una columna.

- HEAD[ING]texto: Establece un encabezado de columna.
- JUS[TIFY]{alineación}: Alinea el encabezado de columna.

Ejemplo:

```
COLUMN  ename HEADING 'Nombre'
COLUMN  dept  JUSTIFY LEFT
```

COMANDOS TTITLE Y BTITLE

TTI[TLE] [texto/OFF/ON]
BTI[TLE] [texto/OFF/ON]

Ejemplo:

```
TTITLE
BTITLE                                    'Preparado para:
```

COMANDO BREAK

BRE[AK] [ON/elemento_reporte]

Ejemplo:

```
BREAK ON ename ON job
BREAK ON deptno SKIP 1 ON REPORT
```

3.7.3. Creando un archivo script para ejecutar un reporte

Para crear un archivo script usted puede seguir los siguientes pasos:

1. Cree la sentencia SELECT SQL.
2. Guarde la sentencia SELECT en un archivo script.
3. Cargue el archivo script en el editor.
4. Agregue los comandos de formato antes de la sentencia SELECT.
5. Verifique que el caracter de terminación siga la sentencia SELECT.
6. Borre los comandos de formato después de la sentencia SELECT.
7. Guarde el archivo script.
8. Ingrese "START nombre_archivo" para correr el script.

Ejemplo: Crear un archivo script para crear un reporte que muestre el nombre, departamento y salario de todos los empleados que devenguen más de $ 1500 ordenados por departamento. Agregar un encabezado centrado que se lea Reporte de Empleados, y un pie de página centrado que se lea Preparado para el departamento de nómina. Renombrar la columna ename como Empleado.

```
TTITLE 'Reporte de Empleados'
BTITLE 'Preparado para el departamento de Nómina'
COLUMN ename HEADING 'Empleados'
SELECT ename, sal, deptno
FROM emp
WHERE sal > 1500
ORDER BY deptno;
```

```
Vie Oct 11
                        Reporte de Empleados

Empleados       SAL    DEPTNO
------------   ---------   ------------
CLARK          2450             10
KING           5000              10
JONES          2975              20
SCOTT          3000              20
FORD           3000              20
ALLEN          1600             30
BLAKE          2850             30


                    Preparado para el departamento de Nómina

7 filas seleccionadas.
```

3.8 MANIPULANDO DATOS

3.8.1 Lenguaje de manipulación de datos

Un lenguaje de manipulación de datos (DML) es un lenguaje que permite a los usuarios acceder o manipular los datos de la base de datos.

Por manipulación de datos se quiere decir:

- La recuperación de la información almacenada en la base de datos.
- La inserción de información nueva en la base de datos.
- El borrado de información de la base de datos.
- La modificación de información almacenada en la base de datos.

Una colección de sentencias DML que forman una unidad lógica de trabajo es llamada transacción.

3.8.2 Sentencia INSERT

Usted puede añadir filas a una tabla haciendo uso de la sentencia INSERT.

Sintaxis:

```
INSERT INTO   tabla [ (columna1, columna2, …) ]
VALUES        (valor1, valor2, ...)
```

Donde:

Valor es el valor correspondiente para cada columna de la tabla

Sólo una fila es insertada a la vez con esta sintaxis.

Ejemplo 1: Inserte el departamento mantenimiento con código 50 con sede en Boston en la tabla Dept.

```
SQL> INSERT INTO dept
  2   VALUES (50, 'MAINTENANCE', 'BOSTON');

1 fila creada.
```

3.8.3. Insertando filas con valores nulos

Usted puede insertar filas con valores nulos por dos métodos:

- Método implícito: Omita las columnas en lista de columnas.
- Método explícito: Especifique con la palabra clave NULL.

Ejemplo: Inserte el departamento mantenimiento con código 60 en la tabla Dept.

No ingresar el valor de la localización. Realizarlo por los dos métodos.

Método implícito:

```
SQL> INSERT INTO dept (deptno, dname )
    2    VALUES (60, 'MAINTENANCE')

1 fila creada.
```

Método explícito:

```
SQL> INSERT INTO dept
    2    VALUES (60, 'MAINTENANCE', NULL)

1 fila creada.
```

3.8.4. Insertando valores nulos con sustitución de variables

Usted puede escribir una sentencia SQL donde el usuario introduzca los valores de forma interactiva haciendo uso de la sustitución de variables de SQL*Plus.

Ejemplo: Inserte un nuevo departamento, cuyos valores serán ingresados por el usuario.

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

```
Ingresar un valor para deptno: 70
Ingresar un valor para dname: FINANCE
Ingresar un valor para loc: BOSTON

1 fila creada.
```

3.8.5. Sentencia UPDATE

Usted puede modificar filas existentes en una tabla haciendo uso de la sentencia UPDATE.

Sintaxis:

```
UPDATE    tabla
SET       columna1 = valor, columna2 = valor
[ WHERE      condición ]
```

Donde:

Valor es el correspondiente valor o subcosulta de columna

Condición identifica las filas que serán modificadas

Todas las filas en la tabla serán modificadas si se omite la cláusula Where.

Ejemplo 1: Cambiar al empleado 7566 al departamento 10.

```
SQL> UPDATE   emp
  2    SET
  3    WHERE     empno= 7566;
1 fila actualizada.
```

Ejemplo 2: Cambiar el jefe del empleado 7521 por el jefe del empleado 7902.

```
SQL> UPDATE   emp
  2    SET   mgr = (SELECT mgr
  3                  FROM emp
  4                  WHERE empno= 7902)
  5    WHERE empno= 7521;
 1 fila actualizada.
```

3.8.6 Sentencia DELETE

Usted puede remover filas existentes de una tabla usando la sentencia DELETE.

Sintaxis:

```
DELETE [FROM ]          tabla
[WHERE                  condición ]
```

Donde:

Condición identifica las filas que serán borradas

Todas las filas en la tabla serán borradas si se omite la cláusula Where.

Ejemplo 1: Borrar el empleado número 7499.

```
SQL> DELETE FROM employee
  2   WHERE empno = 7499;
1 fila suprimida.
```

Ejemplo 2: Borrar a todos los empleados que trabajan en el departamento de ventas.

```
SQL> DELETE FROM employee
  2   WHERE          deptno =
  3
  4
  5                              WHERE  dname =
  6                              'SALES');

5 filas suprimidas.
```

3.8.7 Controlando transacciones

Usted puede controlar las transacciones usando las sentencias COMMIT, SAVEPOINT Y ROLLBACK

COMMIT

Use la sentencia Commit para hacer permanentes todos los cambios realizados en la transacción actual.

Ejemplo: Hacer Commit a una sentencia SQL.

```
SQL> DELETE FROM emp
  2    WHERE ename = 'JONES'

1 fila suprimida.
```

```
SQL> COMMIT;
```

ROLLBACK

Use la sentencia Rollback para que los cambios realizados sean desechos.

Ejemplo: Deshacer la sentencia del ejemplo anterior.

```
SQL> DELETE FROM emp
  3    WHERE ename = 'JONES'

1 fila suprimida.
```

```
SQL> ROLLBACK;
Descarte finalizado..
```

SAVEPOINT

Use la sentencia Savepoint para marcar un punto en una transacción, hasta el cual podría hacer un Rollback sin necesidad de deshacer toda la transación.

Ejemplo: Crear un punto de salva en una sentencia SQL.

```
SQL> INSERT INTO dept
  2  VALUES (70, 'PERSONNEL', 'CHICAGO');
1 fila creada.
```

```
SQL> SAVEPOINT A;
```

3.9 CREACIÓN Y MANEJO DE TABLAS

3.9.1 Objetos de la base de datos

Los objetos más importantes en una base de datos se describen en la siguiente tabla.

Objeto	**Descripción**
Tabla	Unidad básica de almacenamiento, compuesta por filas y columnas
Vista	Representación lógica sustentanda en datos de una o más tablas.
Secuencia	Generadores de valores de llaves primarias.
Índice	Mejora el desempeño de algunas consultas.
Sinónimo	Da nombres alternativos a los objetos.

El nombre de una tabla debe seguir las reglas estándar para los nombres de cualquier objeto de una base de datos ORACLE.

- Debe tener entre 1 y 30 caracteres y el primer carácter debe ser alfabético.
- Debe contener solo los caracteres: A-Z, a-z, 0-9, _ (guión bajo)
- No debe ser ninguna palabra reservada de ORACLE.
- No debe tener el mismo nombre que otro objeto dentro del mismo nombre de usuario de ORACLE.

3.9.2 Sentencia CREATE TABLE

Usted puede crear tablas para almacenar datos ejecutando una sentencia CREATE TABLE. Esta es una sentencia del Lenguaje de Definición de Datos (DDL).

Para que un usuario pueda crear una tabla debe tener un privilegio CREATE TABLE y un área de almacenamiento en la cual crear el objeto.

Sintaxis:

```
CREATE TABLE [user.]  tabla
```

(columna tipo [, . . .]);

Donde:

User es el dueño de la tabla

Tabla es el nombre de la tabla

Columna es el nombre de la columna

Tipo es el tipo de datos y tamaño de la columna

TIPOS DE DATOS

Tipo de Dato	***Descripción***
VARCHAR2(Tamaño)	Dato caracter longitud variable
CHAR(tamaño)	Dato caracter longitud fija
NUMBER(p,s)	Dato numérico longitud variable
DATE	Valores de fecha y tiempo
LONG	Dato caracter longitud variable de hasta 2 Gigabytes
CLOB	Dato caracter de único byte de hasta 4 Gigabytes
RAW Y LONG RAW	Dato binario puro
BLOB	Dato binario de hasta 4 Gigabytes
BFILE	Dato binario almacenado en un archivo externo de hasta 4 Gigabytes

Ejemplo: Crear la tabla dept2 y confirmar su creación.

```
SQL> CREATE TABLE dept2



Tabla creada.
```

```
SQL> DESCRIBE dept2;
 Nombre                  Nulo?  Tipo
 ----------------------  -------  -------------
 DEPTNO                           NUMBER(2)
 DNAME                            VARCHAR2(14)
 LOC                              VARCHAR2(13)
```

3.9.3 Creación de una tabla usando una subconsulta

Usted puede crear tablas usando la cláusula AS en la sentencia CREATE TABLE para utilizar los atributos de columnas y los datos de una tabla ya existente.

Sintaxis:

```
CREATE TABLE [user.]          tabla
[(  columna  , columna. . . )];      AS subconsulta;
```

Donde:

Tabla	Es el nombre de la tabla.
Columna	Es el nombre de la columna.
Subconsulta	Es la sentencia SELECT que define el grupo de filas que se insertarán en la nueva tabla.

Ejemplo: Crear la tabla dept10. Esta tabla contendrá los números, nombres y salario de todos los empleados que trabajen en el departamento 10. Confirmar su creación.

```
SQL> CREATE TABLE dept10
  2  AS SELECT empno, ename, sal
  3  FROM emp
  4  WHERE deptno = 10;
```

```
SQL> DESCRIBE dept10;
 Nombre          Nulo?   Tipo
 --------------- -------- ---------
 EMPNO                    NUMBER(4)
 ENAME                    VARCHAR2(10)
 SAL                      NUMBER(7,2)
```

3.9.4 Sentencia ALTER TABLE

Para modificar la estructura de una tabla use la sentencia ALTER TABLE. Se pueden hacer dos operaciones:

- Adicionar columnas o restricciones.
- Modificar la definición de las columnas (tipos de datos y tamaño, restricciones y demás).

Sintaxis – Adicionar:

```
ALTER TABLE          tabla
ADD   ( columna      tipo
      [, columna     tipo]. . . );
```

Ejemplo: Adicionar a la tabla dept10 el trabajo que cada empleado desempeña.

```
SQL> ALTER TABLE dept10
  2  ADD  (job  VARCHAR2(9));
Tabla modificada.
```

Sintaxis – Modificar

```
ALTER TABLE          tabla
MODIFY               ( columna     tipo
                     [, columna    tipo]. . . );
```

Ejemplo: Modificar el tamaño de la columna ename en la tabla dept10 a VARCHAR2(14).

```
SQL> ALTER TABLE dept10
```

3.9.5 Cláusula DROP COLUMN

Usted puede borrar una columna de una tabla usando la sentencia ALTER TABLE con la cláusula DROP COLUM.

Sintaxis:

```
ALTER TABLE     tabla
DROP COLUMN     columna;
```

Guías:

- La columna puede o no contener datos.
- Sólo una columna puede ser borrada a la vez.
- La tabla debe permanecer por lo menos con una columna luego de ser alterada.
- Una vez una columna es borrada, ésta no se puede recuperar.

Ejemplo: Borrar la columna job de la tabla dept10.

```
SQL> ALTER TABLE dept10
  2  DROP COLUMN job;
Tabla modificada.
```

3.9.6. Sentencia DROP TABLE

La sentencia DROP TABLE elimina la definición de una tabla en Oracle. Cuando usted borra una tabla, la base de datos pierde todos los datos en la tabla y todos los índices asociados con ésta.

Sintaxis:

```
DROP TABLE  tabla;
```

Donde:

Tabla es el nombre de la tabla

Guías:

- Todo dato es borrado de la tabla.
- Las vistas y sinónimos permanecerán, pero son inválidas.
- Sólo el creador de la tabla, o un usuario con el privilegio DROP ANY TABLE podrá eliminar una tabla.
- Una vez ejecutada esta sentencia es irreversible.

Ejemplo: Borrar la tabla dept10.

```
SQL> DROP TABLE dept10;
Tabla eliminada.
```

3.9.7. Sentencia RENAME

Usted puede cambiar el nombre de una tabla, vista, secuencia o sinónimo haciendo uso de la sentencia RENAME. Para esto , usted debe ser el dueño del objeto a renombrar.

Sintaxis:

```
RENAME   nombre_viejo   TO  nombre_nuevo;
```

Donde:

nombre_viejo	es el antiguo nombre del objeto
nombre_nuevo	es el nuevo nombre del objeto

Ejemplo: Cambiar el nombre de la tabla dept2 por departamento.

```
SQL> RENAME dept2  TO  departamento;
Tabla renombrada.
```

3.10. AGREGANDO LIGADURAS

3.10.1. Ligaduras de integridad

El servidor de Oracle usa las ligaduras para prevenir que datos inválidos sean introducidos en las tablas.

Usted puede usar las ligaduras para hacer lo siguiente:

- Hacer cumplir las reglas de la tabla siempre que se inserten, actualicen o borren filas de la tabla. Las ligaduras deben satisfacerse para cada operación realizada.
- Prevenir que se borre una tabla si tiene dependencias con otras tablas.

Guías:

- Todas las ligaduras son almacenadas en el diccionario de datos.
- Las ligaduras pueden ser definidas en el momento de crear la tabla, o después de crearla.
- El nombre de una ligadura debe seguir las reglas estándar para los nombres de cualquier objeto de una base de datos ORACLE.

Ligaduras de Integridad

Ligadura	*Descripción*
NOT NULL	Especifica que dicha columna no puede contener valores nulos.
UNIQUE	Especifica una columna o columnas que deben tener valores únicos a lo largo de todas las filas.
PRIMARY KEY	Identifica de manera única una fila de la tabla.
FOREIGN KEY	Establece e impone una relación entre esta columna y una PRIMARY KEY en la tabla referenciada.
CHECK	Especifica una condición que debe ser verdadera.

3.10.2 Definiendo ligaduras

Las ligaduras pueden ser definidas por uno de dos niveles:

1. Columna: Referencia a una sola columna; puede definirse cualquier tipo de ligadura de integridad.

Sintaxis:

columna [CONSTRAINT nombre_ligadura] tipo_ligadura

Sintaxis:

```
columna, . . .
[CONSTRAINT nombre_ligadura] tipo_ligadura
(columna, . . .),
```

En la sintaxis:

nombre_ligadura	es el nombre de la ligadura
tipo_ligadura	es el tipo de la ligadura

Ejemplo:

```
SQL> CREATE TABLE emp2
2 (empno NUMBER(4),
3 CONSTRAINT empno_constr PRIMARY KEY(empno),
. . .
9 comm NUMBER(7,2),
10 deptno NUMBER(2) NOT NULL );
```

Usted puede ver las ligaduras que ha creado consultando la tabla user_constraints.

3.10.3 Ligadura NOT NULL

La ligadura NOT NULL asegura que los valores nulos no serán permitidos en la columna. Las columnas sin la ligadura NOT NULL pueden contener valores nulos por defecto. Esta ligadura se define sólo a nivel de columna.

Ejemplo: Crear la tabla dept3, asegurándose que la columna deptno no permita valores nulos.

```
SQL> CREATE TABLE dept3
  2                (deptno NUMBER(2) NOT NULL,
  3                 dname VARCHAR2(14),
  4                  loc   VARCHAR2(13));
```

3.10.4 Ligadura UNIQUE

La ligadura UNIQUE requiere que cada valor en una columna o grupo de columnas sea único. Esta ligadura se puede definir tanto a nivel de columna como de tabla.

Ejemplo: Crear la tabla dept4 asegurándose que cada nombre de departamento es único en la columna dname. Nombrar la ligadura dept4_dname_uk.

```
SQL> CREATE TABLE dept4


 5          CONSTRAINT dept4_dname_uk
```

3.10.5 Ligadura PRIMARY KEY

La ligadura Primary Key crea una llave primaria para una tabla. Sólo una llave primaria puede ser creada para cada tabla. La llave primaria es una columna o grupo de columnas que identifican de manera única cada fila de la tabla. Esta ligadura se puede definir a nivel de columna o a nivel de tabla.

Ejemplo: Crear la tabla dept5 y asignar la columna deptno como llave primaria. Nombrar la ligadura dept5_deptno_pk

```
SQL> CREATE TABLE dept5
  2        (deptno NUMBER(2),
  3         dname VARCHAR2(14),
  4         loc   VARCHAR2(13),
  5         CONSTRAINT dept5_deptno_pk PRIMARY
  6
```

3.10.6. Ligadura FOREING KEY

La ligadura FOREIGN KEY designa una columna o combinación de columnas como una llave foránea y establece una relación entre esta columna y una llave

primaria en la tabla referenciada. Esta ligadura se puede definir a nivel de columna o a nivel de tabla.

Ejemplo: Crear la tabla emp2, donde la columna deptno es una llave foránea que hace referencia a la columna deptno de la tabla dept. Nombrar la ligadura emp2_deptno_fk.

```
SQL> CREATE TABLE emp2
  2       (empno NUMBER(4),
  3        ename CHAR(10),
  4        job CHAR(9),
  5        mgr NUMBER(4),
  6        hiredate DATE,
  7        sal NUMBER(7,2),
  8        comm NUMBER(7,2),
  9        deptno NUMBER(2) NOT NULL,
 10         CONSTRAINT emp2_deptno_fk FOREIGN
 11         KEY(deptno) REFERENCES ;dept(deptno));
```

3.10.7. Ligadura CHECK

La ligadura CHECK define una condición que cada fila de satisfacer. No hay límite para el número de ligaduras CHECK que puede definir en una columna. Esta ligadura se puede definir a nivel de columna o a nivel de tabla.

Ejemplo: Crear la tabla dept6 verificando que sólo se puedan introducir números de departamento entre 10 y 99 en la columna deptno. Nombrar la ligadura dept6_deptno_ck.

```
SQL> CREATE TABLE dept6
  2        (deptno NUMBER(2),
  3         dname VARCHAR2(14),
  4         loc   VARCHAR2(13),
  5         CONSTRAINT dept6_deptno_ck
  6                  CHECK(deptno BETWEEN 10 AND 99));
```

3.10.8. Manejando ligaduras

Adicionado una Ligadura

Usted puede adicionar una ligadura a una tabla existente usando la sentencia ALTER TABLE y la cláusula ADD.

Sintaxis:

```
ALTER TABLE    tabla
ADD     [CONSTRAINT   ligadura]    tipo(columna);
```

Ejemplo:

```
SQL> ALTER TABLE emp
  2    ADD CONSTRAINT emp_mgr_fk
  3     FOREIGN KEY(mgr) REFERENCES emp(empno);
Tabla modificada.
```

Borrando una Ligadura

Usted puede borrar una ligadura a una tabla existente usando la sentencia ALTER TABLE y la cláusula DROP.

Sintaxis:

```
ALTER TABLE   tabla
DROP   CONSTRAINT   ligadura   [CASCADE];
```

Ejemplo:

```
SQL> ALTER TABLE emp
```

La opción CASCADE de la cláusula DROP ocasiona que cualquier ligadura dependiente también sea borrada.

Deshabilitando Ligaduras

Usted puede deshabilitar una ligadura usando la sentencia ALTER TABLE con la cláusula DISABLE.

Sintaxis:

```
ALTER TABLE    tabla
DISABLE    CONSTRAINT   ligadura   [CASCADE];
```

Ejemplo:

```
SQL> ALTER TABLE emp
    2    DISABLE CONSTRAINT emp_mgr_pk;
Tabla modificada.
```

Habilitando Ligaduras

Usted puede habilitar una ligadura usando la sentencia ALTER TABLE con la cláusula ENABLE.

Sintaxis:

```
ALTER TABLE    tabla
DISABLE    CONSTRAINT   ligadura;
```

Ejemplo:

```
SQL> ALTER TABLE emp
```

3.11. OTROS OBJETOS DE LA BASE DE DATOS

3.11.1. Objetos de la base de datos

Los objetos más importantes en una base de datos se describen en la siguiente tabla.

Objeto	**Descripción**
Tabla	Unidad básica de almacenamiento, compuesta por filas y columnas
Vista	Representación lógica de un subconjunto de datos de una o más tablas.
Secuencia	Generadores de valores de llaves primarias.
Índice	Mejora el desempeño de algunas consultas.
Sinónimo	Da nombres alternativos a los objetos.

Las tablas han sido estudiadas en la unidad 8, en esta unidad se estudiarán los demás objetos de la base de datos.

3.11.2 Creando una vista

Una vista es una tabla lógica basada en una tabla o en otra vista. Las vistas no contienen datos propios, se almacenan como una sentencia SELECT en el diccionario de datos y son usadas por seguridad, conveniencia y presentación.

Usted puede crear una vista por una subconsulta embebida con la sentencia CREATE VIEW. La subconsulta no puede contener una cláusula ORDER BY.

Sintaxis:

```
CREATE [OR REPLACE] VIEW   vista
[ (alias, alias, ... ) ]
 AS subconsulta
[WITH CHECK OPTION [CONSTRAINT ligadura]]
[WITH READ ONLY];
```

Donde:

OR REPLACE	vuelve a crear la vista si esta ya existe
Vista	es el nombre de la vista
Alias	especifica nombres de expresiones seleccionadas por la consulta de la vista
Subconsulta	es un sentencia SELECT
WITH CHECK OPTION	especifica que sólo filas accesibles a la vista pueden ser insertadas o modificadas

Ligadura	Es el nombre asignado a la ligadura CHECK OPTION
WITH READ ONLY	especifica que la vista es de sólo lectura

Ejemplo: Crear una vista empvd30 que contenga el número, nombre y salario de todos los empleados del departamento 30. Confirme la creación de la vista usando el comando DESCRIBE.

```
SQL> CREATE VIEW empvd30
  2   AS SELECT empno, ename, sal
  3   FROM emp
  4   WHERE deptno = 30;
Visualización creada.
```

```
SQL> DESCRIBE empvd30;
```

```
Nombre              Nulo?  Tipo
 ------------------- -------- --------------
 EMPNO                        NUMBER(4)
 ENAME                        VARCHAR2(10)
 SAL                          NUMBER(7,2)
```

Una vez creada la vista usted puede consultar la tabla user_views para mirar el nombre de la vista y su definición.

3.11.3 Manejando vistas

Consultando una Vista

Usted puede recuperar datos de una vista como lo haría de cualquier tabla.

Ejemplo: Mostrar todas las columnas de la vista empvd30.

```
SQL> SELECT *
  2    FROM empvd30;
```

```
EMPNO ENAME  SAL
----------- ----------- ---------
    7521 WARD     1250
    7654 MARTIN   1250
    7698 BLAKE     2850
    7844 TURNER   1500
    7900 JAMES        950
```

- **Modificando una Vista**

Usted puede usar la opción OR REPLACE para modificar una vista.

Ejemplo: Modificar la vista empvd30 usando la cláusula CREATE OR PREPLACE VIEW. Añada un alias por cada nombre de columna.

```
SQL> CREATE OR REPLACE VIEW empvd30
    2   (numero_empleado, nombre_empleado, salario)
    3  AS SELECT empno, ename, sal
    4  FROM emp
    5  WHERE deptno = 30;
```

- **Borrando una Vista**

Usted puede usar la sentencia DROP VIEW para borrar una vista. Al borrar una vista los datos de la tabla en la cual se basa la vista no son afectados. Sólo el creador de la vista o un usuario con el privilegio DROP ANY VIEW pueden borrar una vista.

Ejemplo: Borrar la vista empvd30.

```
SQL> DROP VIEW empvd30;
Visualización eliminada.
```

- **Cláusula WITH CHECK OPTION**

Usted puede asegurase de que las inserciones y actualizaciones realizadas sobre una vista, no afectarán los datos que la vista no sea capaz de acceder, mediante la utilización de la cláusula WITH CHECK OPTION.

Ejemplo: Crear una vista con los datos de los empelados del departamento 20. Usar de la cláusula WITH CHECK OPTION. Modificar el departamento del empleado 7369 de 20 a 10.

```
SQL> CREATE VIEW empvd20
  2    AS SELECT *
  3    FROM emp
  4     WHERE deptno = 20
  5     WITH CHECK OPTION;
Visualización creada.
```

```
SQL> UPDATE empvd20
  2  SET deptno = 10
  3  WHERE empno = 7369;
UPDATE empvd20
       *
ERROR en línea 1:
ORA-01402: violación de la cláusula-WHERE en la visualización
WITH CHECK OPTION
```

3.11.4 Creando una secuencia

Las secuencias se pueden utilizar para generar llaves primarias de forma automática. Use la sentencia CREATE SEQUENCE para crear una secuencia.

Sintaxis abreviada:

```
CREATE SEQUENCE    secuencia
[ INCREMENT BY  n  ]
```

[START WITH n]
[{ MAXVALUE n | NOMAXVALUE }]
[{ MINVALUE n | NOMINVALUE]

Donde:

Secuencia	es el nombre del generador de secuencia
INCREMENT BY	especifica el intervalo entre los números de secuencias donde n es un entero
START WITH n	especifica el primer número de secuencia a generar
MAXVALUE n	especifica el máximo valor de secuencia que puede generarse
NOMAXVALUE	especifica el valor máximo por defecto
MINVALUE n	especifica el mínimo valor de la secuencia
NOMINVALUE	especifica el valor mínimo por defecto

Ejemplo: Crear una secuencia llamada s_dept_deptno empezando desde el valor 91 hasta el 100 con un incremento de 1.

```
SQL> CREATE SEQUENCE s_dept_deptno
  2    INCREMENT BY 1
  3    START WITH 91
  4    MAXVALUE 100;
Secuencia creada.
```

3.11.5 Manejando secuencias

Usted puede verificar la creación de una secuencia consultando la tabla user_sequences.

- **NEXTVAL y CURRVAL**

Use NEXTVAL para retornar el próximo valor disponible de la secuencia y CURRVAL para retornar el valor actual de la secuencia.

Ejemplo: Mostrar el siguiente valor disponible de la secuencia s_dept_deptno.

```
SQL> SELECT s_dept_deptno.nextval
  2  FROM dual;
```

Modificando una Secuencia

Usted puede modificar una secuencia usando la sentencia ALTER SÉQUENCE.

Ejemplo: Modificar el valor máximo de la secuencia s_dept_deptno a 1000.

```
SQL> ALTER SEQUENCE s_dept_deptno
  2    INCREMENT BY 1
  3    MAXVALUE 1000;
```

Borrando una Secuencia

Usted puede borrar una secuencia usando la sentencia DROP SEQUENCE.

Ejemplo: Borrar la secuencia s_dept_deptno.

```
SQL> DROP SEQUENCE s_dept_deptno;
Secuencia eliminada.
```

3.11.6. Creando un índice

Los índices son estructuras opcionales, asociadas con tablas, que se usan para agilizar la ejecución de consultas y/o garantizar unicidad. Usted puede crear un índice usando la sentencia CREATE INDEX.

Sintaxis:

```
CREATE INDEX    indice
ON              tabla (columna [ , columna ] ... );
```

Donde:

Indice	es el nombre del índice
Tabla	es el nombre de la tabla
Columna	es el nombre de la columna en la tabla a indexar.

Ejemplo: Crear un índice sobre la columna ename en la tabla emp. Nombrar al índice i_emp_ename.

```
SQL> CREATE INDEX i_emp_ename
  2    ON  emp(ename)
 Índice creado.
```

En la tabla user_indexes del diccionario de datos se encuentran todos los índices pertenecientes al usuario actual.

Ejemplo: Crear una consulta que muestre el nombre de todos los índices de la tabla emp.

```
SQL> SELECT index_name
  2  FROM user_indexes
  3  WHERE table_name = 'EMP';
```

3.11.7 Manejando índices

Borrando un Índice

Usted puede borrar un índice usando la sentencia DROP INDEX.

Ejemplo: Borrar el índice i_emp_ename.

```
SQL> DROP INDEX i_emp_ename;
Índice eliminado.
```

3.11.8. Creando un sinónimo

Los sinónimos se pueden crear por razones de seguridad y conveniencia. Use la sentencia CREATE SYNONYM para crear un sinónimo.

Sintaxis:

```
CREATE SYNONYM  [PUBLIC]                    sinónimo
 FOR                                        objeto;
```

Donde:

PUBLIC	crea un sinónimo accesible a todos los usuarios
Sinónimo	es el nombre del sinónimo a crear

Objeto	identifica el objeto para el cual el sinónimo es creado

Ejemplo: Crear un nombre corto para el índice i_emp_ename.

3.11.9 Manejando sinónimos

```
SQL> CREATE SYNONYM i_ename
  2  FOR i_emp_ename;
Sinónimo creado.
```

Borrando un Sinónimo

Usted puede borrar un sinónimo usando la sentencia DROP SYNONYM.

Ejemplo: Borrar el sinónimo i_ename.

```
SQL> DROP SYNONYM i_ename;
Sinónimo eliminado.
```

3.12. CONTROLANDO EL ACCESO A LAS BASE DE DATOS

3.12.1. Privilegios

Un privilegio es el derecho a ejecutar un sentencia SQL particular. El usuario requiere privilegios del sistema para ganar acceso a la base de datos, y privilegios de objeto para manipular el contenido de los objetos de la base de datos.

Privilegios típicos del DBA.

Privilegio	**Operación autorizada**
CREATE USER	Crear otro usuario de Oracle
DROP USER	Borrar otro usuario
DROP ANY TABLE	Borrar cualquier tabla
BACKUP ANY TABLE	Hacer copia de seguridad a cualquier tabla

Privilegios típicos de los usuarios.

Privilegio	**Operación autorizada**
CREATE SESSION	Conectar a la base de datos
CREATE TABLE	Crear tablas
CRÉATE SEQUENCE	Crear una secuencia
CREATE VIEW	Crear una vista
CREATE PROCEDURE	Crear un procedimiento almacenado

El administrador de la base de datos es un nivel alto de usuario con la capacidad de conceder a los usuarios acceso a la base de datos y a sus objetos.

3.12.2 Creando usuarios

El DBA crea usuarios usando la sentencia CREATE USER.

Sintaxis:

```
CREATE USER        usuario
IDENTIFIED BY      contraseña;
```

Donde:

Usuario	es el nombre del usuario a crear
Contraseña	especifica que el usuario puede registrarse con esta contraseña

Ejemplo: Crear al usuario scott con la contraseña tigre.

```
SQL> CREATE USER scott
  2    IDENTIFIED BY tiger;
Usuario creado
```

3.12.3 Concediendo privilegios

El DBA concede privilegios a los usuarios usando la sentencia GRANT.

Sintaxis:

```
GRANT      privilegio [, privilegio . . .]
 TO        usuario    [, usuario . . .];
```

Donde:

Privilegio	es el privilegio de sistema que se va a conceder
Usuario	es el nombre del usuario.

Ejemplo: Conceder al usuario scott el privilegio de crear tabla.

```
SQL> GRANT create table
  2   TO scott;
```

3.12.4. Creando roles

Un role es un grupo de privilegios relacionados que pueden ser concedidos al usuario. Primero el DBA debe crear el role usando la sentencia CREATE ROLE. Luego, puede asignar privilegios y usuario al role.

Sintaxis:

CREATE ROLE role;

Donde:

Role es el nombre del role a crear

Ejemplo: Crear el role salesman, asignarle los privilegios de crear tabla y crear vista y asignarle como usuarios a Ward, Martin y Turner.

```
SQL> CREATE ROLE salesman;
```

```
SQL> GRANT create table, create view
  2  TO salesman;
```

3.12.5 Cambiando su contraseña

El DBA crea una cuenta y una contraseña para cada usuario. Usted puede cambiar su contraseña usando la sentencia ALTER USER.

Sintaxis:

```
ALTER USER          usuario
IDENTIFIED BY       contraseña;
```

Donde:

Usuario es el nombre del usuario

Contraseña especifica la nueva contraseña

Ejemplo: Cambiar la contraseña del usuario scott a rabbit.

```
SQL> ALTER USER scott
  2  IDENTIFIED BY rabbit;
Usuario modificado.
```

3.12.6 Concediendo privilegios sobre objetos

El DBA concede privilegios a los usuarios usando la sentencia GRANT.

Sintaxis:

```
GRANT   privilegio  | ALL [columna]
```

```
ON          objeto
TO          {usuario | role  | PUBLIC }
            [WITH GRANT OPTION];
```

Donde:

Privilegio es el privilegio sobre el objeto a ser concedido

ALL especifica todos los privilegios de objeto

columna especifica la columna de la tabla o vista sobre la cual el privilegio es concedido

objeto es el objeto sobre el cual el privilegio es concedido

TO identifica a quién es concedido el privilegio

PUBLIC concede privilegios de objeto a todos los usuarios

WITH GRANT OPTION habilidad de otorgar los privilegios a otros usuarios y roles

Ejemplo

Conceder el privilegio de hacer consultas a la tabla emp al usuario Blake.

```
SQL> GRANT select
  2  ON emp
  3  TO blake;
```

3.12.7 Revocando privilegios

Usted puede revocar los privilegios concedidos a otros usuarios usando la sentencia REVOKE.

Sintaxis:

```
REVOKE      {privilegio [, privilegio . . .] | ALL}
ON          objeto
FROM        {usuario [ ,usuario . . . ] | role  | PUBLIC};
```

Ejemplo: Revocar el privilegio de consultar la tabla emp dado al usuario blake.

```
SQL> REVOKE select
  2  ON emp
  3  FROM blake;
```

3.13 DECLARACION DE VARIABLES EN PL/SQL

3.13.1 Bloques de instrucciones PL/SQL

PL/SQL

PL/SQL (Lenguaje Procedimental/SQL) es una extensión de SQL, que agrega ciertas construcciones propias de lenguajes procedimentales.

Estructura del Bloque PL/SQL

Todos los programas de PL/SQL están conformados por bloques, significando que el programa puede ser dividido dentro de bloques lógicos.

Un bloque PL/SQL consiste en tres secciones: declarativa (opcional), ejecutable (obligatoria), manejo de excepciones (opcional). Sólo las palabras claves BEGIN y END son obligatorias.

Estructura del bloque PL/SQL

- DECLARE - Opcional
  Declaración de variables, cursores, definición de excepciones.
- BEGIN - Obligatorio
  Sentencias SQL
  Sentencias PL/SQL
- EXCEPTION - Opcional
- END; - Obligatorio

3.13.2 Tipos de variables en PL/SQL

Los tipos de variables en PL/SQL son:

ESCALARES

- *BYNARY_ INTEGER*
- DEC
- DECIMAL
- DOUBLE
- PRECISION
- FLOAT
- INT
- INTEGER
- NATURAL
- NATURALN
- NUMBER
- NUMERIC
- PLS_INTEGER
- POSITIVE
- POSITIVEN
- REAL

- *SMALLINT*
- *CHAR*
- *CHARACTER*
- *LONG*
- *LONGRAW*
- *NCHAR*
- *NVARCHAR2*
- *RAW*
- *ROWID*
- *STRING*
- *UROWID*
- *VARCHAR*
- VARCHAR2
- BOLEAN
- DATE

COMPUESTAS

- RECORD
- TABLE
- VARRAY

REFERENCIA

- REF CURSOR
- REF OBJECT_TYPE

LOB (Objetos Grandes)

- BFILE
- BLOB
- CLOB
- NCLOB

3.13.3 Declaración de variables escalares

Sintaxis:

Identificador [CONSTANT] tipo_dato [NOT NULL]
[: = | DEFAULT expresión]

Donde:

Identificadores	el nombre de la variable
CONSTANT	fuerza la variable de modo que su valor no pueda cambiar; las constantes deben inicializarse
tipo_dato	es un tipo de dato escalar, compuesto, referencia o LOB.

NOT NULL	fuerza la variable de modo que esta debe contener un valor; estas variables deben inicializarse.
Expresión	es una expresión PL/SQL que puede ser un literal, otra variable, o una expresión que contenga operadores y funciones

Ejemplos:

```
DECLARE
v_ename
v_deptno
v_hiredate
c_comm.
```

Guías:

- Seguir las convenciones para los nombres.
- Inicializar las variables designadas como NOT NULL y CONSTANT.
- Inicializar los identificadores usando el operador de asignación (: =) o la palabra reservada DEFAULT.
- Declarar un identificador por línea.

3.13.4 Declaración de variables compuestas

Declaración de tipos TABLE

Sintaxis:

```
TYPE        nombre_tipo IS TABLE OF
            (tipo_columna | variable%TYPE | tabla.columna%TYPE}
            [NOT NULL] [INDEX BY BINARY INTEGER]
identificador     nombre_tipo;
```

Donde:

nombre_tipo	es el nombre de tipo TABLE
tipo_columna	es cualquier tipo de dato escalar
identificador	es el nombre del identificador que representa una entidad tabla PL/SQL.

Ejemplo: Declarar una tabla PL/SQL para almacenar nombres.

```
DECLARE

TYPE ename_table_type IS TABLE OF VARCHAR2(10)
        INDEX BY BINARY_INTEGER;
ename_table
```

Declaración de tipos RECORD

Sintaxis:

```
TYPE          nombre_tipo IS RECORD
              (nombre_campo{tipo_campo|variable%TYPE
              tabla.columna%TYPE} [NOT NULL] {: = expr},
              [,(nombre_campo{tipo_campo|variable%TYPE
              tabla.columna%TYPE} [NOT NULL] {:= expr}, . .]
identificador  nombre_tipo;
```

Donde:

nombre_tipo	es el nombre de tipo RECORD
nombre_campo	es el nombre de un campo del registro
tipo_campo	es el tipo de dato del campo
exp	es el tipo_campo o un valor inicial

Ejemplo

Declarar un registro que almacene el nombre, trabajo y salario de un nuevo empleado.

```
DECLARE

TYPE emp_record_type IS RECORD
        (ename
         job
         sal
emp_record
```

3.13.5 Declaración de atributos

Uso de %TYPE

El atributo %TYPE define el tipo de una variable utilizando una definición previa de otra variable o columna de la base de datos.

Ejemplos:

```
DECLARE

v_ename
v_sal
v_sal_min
```

En los ejemplos, la variable v_ename se declara del mismo tipo que la columna ename de la tabla emp, y la variable v_sal_min del mismo tipo que la variable v_sal.

3.13.6 Asignación de variables

Sintaxis:

Identificador : = expresión

Donde:

Identificador	es el nombre de la variable escalar
Expresión	puede ser un literal, una variable, llamada a función, pero no una columna de la base de datos.

Ejemplos:

```
v_ename : = 'SCOTT';
v_deptno : = 20;
```

3. 14 USO DE SQL EN PL/SQL

3.14.1 Sentencias SQL en PL/SQL

Las sentencias DML (Lenguaje de Manipulación de Datos) de SQL soportadas en PL/SQL son SELECT... INTO, INSERT, UPDATE y DELETE.

Las sentencias DDL (Lenguaje de Definición de Datos), como CREATE TABLE, ALTER TABLE, o DROP TABLE, no son soportadas en PL/SQL.

Las sentencias DCL (Lenguaje de Control de Datos), como GRANT y REVOKE, no son soportadas en PL/SQL.

3.14.2 Sentencia SELECT INTO

Use la sentencia SELECT para recuperar datos de la base de datos.

Sintaxis:

```
SELECT      lista_select
INTO        variable [, variable ... ]
FROM        tabla

WHERE       condicion;
```

Donde:

lista_select	es la lista de por lo menos una columna
variable	es la variable cuyo valor va a ser recuperado
tabla	especifica el nombre de la tabla de la base de datos
condición	es la condición que debe cumplirse

Ejemplo: Retornar el nombre y salario del empleado número 7369.

```
DECLARE
    v_ename VARCHAR2(10)
    v_sal NUMBER(7,2);
BEGIN
    SELECT
    INTO
    FROM
    WHERE
END;
```

La sentencia SELECT…INTO debe retornar uno o cero registros. Si retorna múltiples registros produce un error.

3.14.3 Modificando los datos

Sentencia INSERT

Ejemplo: Adicionar la información de un nuevo empleado.

```
BEGIN
    INSERT INTO emp(empno, ename, job, deptno)
          VALUES (empno_sequence.NEXTVAL,
                'PAUL','SALESMAN', 30);
END;
```

Sentencia UPDATE

Ejemplo: Incrementar el salario de los empleados cuyo cargo sea CLERK.

```
DECLARE
    v_sal_inc  emp.sal%TYPE := 1000;
BEGIN
    UPDATE emp
    SET sal = sal + v_sal_inc
    WHERE job = ' CLERK ';
END;
```

Sentencia DELETE

Ejemplo: Borrar los registros de la tabla EMP que se encuentran en el Departamento 20.

```
DECLARE
     v_deptno
BEGIN
    DELETE FROM emp
    WHERE deptno = v_deptno;
END;
```

3.14.4 Uso de COMMIT

La sentencia COMMIT finaliza la transacción actual y efectúa los cambios en la base de datos de forma permanente. Mientras un usuario no efectúa el COMMIT, el resto de usuarios que accedan la misma base en forma concurrente no verán los cambios que este primer usuario ha estado efectuando.

Ejemplo:

```
BEGIN
…
UPDATE cuenta SET bal = mi_bal – debito WHERE num_cta
= 7715 ;
…
UPDATE cuenta SET bal = mi_bal + credito WHERE num_cta
= 7720 ;
COMMIT WORK;
END;
```

La sentencia COMMIT libera todas las filas bloqueadas de la tabla "cuenta". La palabra clave "WORK" no tiene otro efecto que permitir la fácil lectura de la instrucción, pero es perfectamente prescindible dentro del lenguaje.

Note que la palabra END al final del código indica el final del bloque, no el fin de la transacción. Una transacción puede abarcar más de un bloque, como también dentro de un mismo bloque pueden ocurrir muchas transacciones.

3.14.5 Uso de ROLLBACK

La sentencia ROLLBACK finaliza la transacción actual y deshace todos los cambios realizados en la base de datos en la transacción activa.

Considérese el caso del ejemplo siguiente, donde se inserta información en tres tablas diferentes y se toma la precaución de que si se trata de insertar un valor duplicado en una clave primaria, se genera un error (controlado con la sentencia rollback).

Ejemplo:

```
DECLARE
    emp_id integer;
…
BEGIN
  SELECT empno, … INTO emp_id, … FROM new_emp
  WHERE …
  …
  INSERT INTO emp VALUES(emp_id, …);
  INSERT INTO tax VALUES(emp_id, …);
  INSERT INTO pay VALUES(emp_id, …);
  …
  EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
  ROLLBACK;
…
END;
```

3.14.6 Uso de SAVEPOINT

Con la sentencia SAVEPOINT es posible nombrar y marcar un punto determinado donde se podrá retornar el control luego de ejecutarse una sentencia ROLLBACK.

Ejemplo:

```
DECLARE
emp_id emp.empno%TYPE;
BEGIN
UPDATE emp SET … WHERE empno=emp_id;
DELETE FROM emp WHERE …
…
SAVEPOINT do_insert;
INSERT INTO emp VALUES (emp_id, …);
…
EXCEPTION
WHEN DUP_VAL_ON_INDEX THEN
ROLLBACK TO do_insert;
                    END;
```

3.15 ESTRUCTURAS DE CONTROL

3.15.1 Estructuras de control

Las estructuras de control de un lenguaje de programación son métodos de especificar el orden en que las instrucciones de un algoritmo se ejecutarán.

Las tres estructuras de control básico son:

- Secuencia: ejecutan una secuencia de estamentos en el orden que ocurren.

- Selección: verifican cierta condición, después ejecutan cierta secuencia de expresiones dependiendo si la condición resultó ser verdadera o falsa.

- Repetición: ejecutan una secuencia de sentencias repetidamente mientras la condición permanezca verdadera.

3.15.2 Control de selección – sentencia IF

La sentencia IF permite ejecutar una secuencia de acciones condicionalmente. Esto es, si la secuencia es ejecutada o no depende del valor de la condición a evaluar.

Existen tres modos para esta instrucción:

IF – THEN, IF – THEN – ELSE y IF – THEN – ELSIF.

IF – THEN

Este es el modo más simple y consiste en asociar una condición con una secuencia de sentencias encerradas entre las palabras reservadas THEN y END IF. La secuencia de sentencias es ejecutada sólo si la condición es verdadera.

Sintaxis:

```
IF condición THEN
    secuencia_de_sentencias
END IF;
```

Ejemplo:

```
IF v_ename = 'SMITH' THEN
    v_job := 'CLERK';
END IF;
```

IF – THEN – ELSE

Sintaxis:

```
IF condición THEN
    secuencia_de_sentencias_1
ELSE
    secuencia_de_sentencias_2
END IF;
```

La secuencia de sentencias en la cláusula ELSE es ejecutada solamente si la condición es falsa o nula.

Ejecución de alguna de las dos secuencias de estamentos.

Ejemplo:

```
...
IF v_deptno := 20  THEN
    v_sal := v_sal + 1000;
ELSE
    v_sal := v_sal + 500;
END IF;
...
```

IF – THEN – ELSIF

El tercer modo de la sentencia IF utiliza la clave ELSIF para introducir condiciones adicionales. Si la primera condición es falsa o nula, la cláusula ELSIF verifica una nueva condición.

Cada sentencia IF puede poseer un número indeterminado de cláusulas ELSIF.

Sintaxis:

```
IF condición_1 THEN
    secuencia_de_sentencias_1
ELSIF condición_2 THEN
    secuencia_de_sentencias_2
ELSE
    secuencia_de_sentencias_3
END IF;
```

Ejemplo:

```
IF v_sal >3000  THEN
   bono := 700;
ELSIF v_sal >2000  THEN
   bono := 500;
ELSE
   bono := 200;
END IF;
```

3.15.3 Control de repetición – sentencia LOOP

La sentencia LOOP permite ejecutar una secuencia de acciones múltiples veces, todas ellas gobernadas por una condición que regula la ejecución de la iteración.

Existen tres modalidades para esta instrucción:

LOOP, WHILE – LOOP y FOR – LOOP.

LOOP

El modo básico (o infinito) de LOOP encierra una serie de acciones entre las palabras clave LOOP y END LOOP. Con cada iteración del ciclo las sentencias son ejecutadas.

Sintaxis:

```
LOOP
        secuencia_de_instrucciones
END LOOP;
```

Existen dos modalidades para utilizar esta sentencia: EXIT y EXIT – WHEN.

Ejemplo usando EXIT:

```
DECLARE
        v_Counter BINARY_INTEGER := 1;
BEGIN
        LOOP
        INSERT INTO temp_table VALUES (v_Counter,
        'Loop index');
        v_Counter := v_Counter + 1;
        IF v_Counter > 50 THEN
        EXIT;
        END IF;
        END LOOP;
END;
```

Ejemplo usando EXIT – WHEN

```
DECLARE
        v_Counter BINARY_INTEGER := 1;
BEGIN
        LOOP
        INSERT INTO temp_table VALUES
        (v_Counter, 'Loop index');
        v_Counter := v_Counter + 1;
        EXIT WHEN v_Counter > 50;
        END LOOP;
END;
```

WHILE – LOOP

Esta sentencia se asocia a una condición con una secuencia de sentencias encerradas por las palabras clave LOOP y END LOOP.

Sintaxis:

```
WHILE condición LOOP
        secuencia_de_sentencias
END LOOP;
```

Ejemplo:

```
DECLARE
        v_Counter BINARY_INTEGER := 1;
BEGIN
        WHILE v_Counter <= 50 LOOP
        INSERT INTO temp_table VALUES (v_Counter,
        'Loop index');
        v_Counter := v_Counter + 1;
        END LOOP;
END;
```

FOR – LOOP

Con una instrucción del tipo FOR-LOOP, la iteración se efectúa un número finito y conocido de veces.

Sintaxis:

```
FOR contador IN [REVERSE] valor_min..valor_max LOOP
    secuencia_de_sentencias
END LOOP;
```

Ejemplo:

```
DECLARE
        v_LowValue NUMBER := 10;
BEGIN
        FOR v_Counter IN 1..50 LOOP
        INSERT INTO temp_table VALUES (v_Counter, 'Loop
        Index');
END LOOP;
```

3.16 DECLARACION Y USO DE CURSORES

3.16.1 Cursores

Los cursores permiten manejar grupos de datos que se obtienen como resultado de una consulta. PL/SQL utiliza dos tipos de cursores: implícitos y explícitos.

Siempre se declara un cursor implícito para cualquier sentencia de manipulación de datos, incluyendo aquellas que retornan sólo una fila. Sin embargo, para las consultas que retornan más de una fila, usted debe declarar un cursor explícito.

Los Cursores Explícitos son aquellos que devuelven cero, una o más filas, dependiendo de los criterios con que hayan sido construidos. Un cursor puede ser declarado en la primera sección de un programa PL/SQL ("declare").

Existen tres comandos para controlar un cursor: OPEN, FETCH y CLOSE. En un principio, el cursor se inicializa con la instrucción OPEN. Enseguida, se utiliza la instrucción FETCH para recuperar la primera fila o conjunto de datos. Se puede ejecutar FETCH repetidas veces hasta que todas las filas hayan sido

recuperadas. Cuando la última fila ya ha sido procesada, el cursor se puede liberar con la sentencia CLOSE.

Es posible procesar varias consultas en paralelo, declarando y abriendo múltiples cursores.

3.16.2 Declarando cursores

Los cursores deben ser declarados antes de ser utilizados en otras sentencias. Cuando se declara un cursor, a éste se le da un nombre y se asocia con una consulta específica.

Sintaxis:

```
DECLARE
        CURSOR nombre_cursor [ (parámetro1 [, parámetro2]…) ]
        [RETURN tipo_de_retorno] IS sentencia_select ;
```

Donde:

tipo_de_retorno — debe representar a un registro o una fila en una tabla de la base y los parámetros siguen la siguiente sintaxis:

parametro [IN] tipo_de_dato [{ := | DEFAULT} expresión]

Ejemplo:

```
DECLARE
        CURSOR c1
        IS SELECT empno, ename, job, sal FROM emp
        WHERE sal>1000 ;
        CURSOR c2 RETURN dept%ROWTYPE IS
        SELECT * from dept WHERE deptno = 10 ;
```

El nombre del cursor (c1 y c2 en el ejemplo) corresponde a un identificador no declarado, no a un nombre de variable de PL/SQL. No se pueden asignar valores a un nombre de cursor ni utilizarlos en una expresión.

Un cursor puede recibir parámetros, los cuales deben ser declarados con la cláusula IN para formalizar su incorporación.

3.16.3 Apertura de un cursor

Al abrir un cursor se ejecuta inmediatamente la consulta y se identifica el resultado. Para abrir un cursor se utiliza la sentencia OPEN.

Sintaxis:

```
OPEN nombre_cursor;
```

Ejemplo:

```
DECLARE
    CURSOR c1 IS SELECT ename, job FROM emp
                    WHERE deptno = 10;
…
BEGIN
    OPEN c1;
…
END;
```

La sentencia OPEN también se utiliza para pasar los parámetros al cursor, en caso de que éste los requiera.

Ejemplo:

```
DECLARE
    emp_name emp.name%TYPE;
    deptno emp.deptno%TYPE;
    CURSOR c1 (name VARCHAR2, deptno NUMBER)
    IS SELECT…
```

Cualquiera de estas sentencias abre el cursor:

```
OPEN c1(emp_name, 10);
OPEN c1("Blake", 20);
```

3.16.4 Recuperación de filas

Para recuperar las filas se usa la sentencia FETCH, que permite recuperar los conjuntos de datos de a uno a la vez. Después de cada recuperación y carga de un conjunto de datos el cursor avanza a la fila siguiente.

Sintaxis:

```
FETCH nombre_cursor INTO variable1, variable2, …
```

Para cada columna retornada en un cursor y especificada en la declaración del mismo debe existir una variable compatible en tipo en la lista INTO.

Ejemplo:

```
LOOP
FETCH c1 INTO mi_registro;
EXIT WHEN c1%NOTFOUND;
--- se procesa el registro
END LOOP;
```

3.16.5 Atributos de cursores

Algunos atributos de cursores se muestran en la siguiente tabla.

Atributo	**Tipo Retorno**	**Descripción**
%ISOPEN	Boolean	Devuelve TRUE si el cursor está abierto
%FOUND	Boolean	Devuelve TRUE si el cursor retorna registros
%NOTFOUND	Boolean	Devuelve TRUE si el cursor no retorna registros
%ROWCOUNT	number	Devuelve el número de registros retornados por el cursor

Uso de %ISOPEN

Este atributo toma el valor verdadero (TRUE) cuando un cursor se encuentra abierto. De otra manera, retorna FALSO.

Ejemplo:

```
IF NOT emp_cursor%ISOPEN THEN
OPEN emp_cursor;
END IF;
LOOP
FETCH emp_cursor...
```

Uso de % FOUND

Antes de recuperar la primera fila de un cursor abierto, el valor del atributo %FOUND es nulo. Tomará el valor TRUE si el cursor retorna registros, y FALSE cuando ya no existan más filas para mostrar en el conjunto de resultados.

Ejemplo:

```
LOOP
FETCH c1 INTO …
IF c1%FOUND THEN -- fetch exitoso
…
ELSE -- fetch falló; se sale del loop
EXIT;
END IF;
END LOOP;
```

Uso de % NOTFOUND

Es el opuesto lógico de %FOUND. Cada vez que una sentencia FETCH retorne una fila válida, este atributo devolverá FALSO. Sólo alcanzará el valor TRUE cuando no haya más filas en un cursor y se ejecute la sentencia FETCH (sin éxito por lo tanto).

Ejemplo.

```
LOOP
FETCH c1 INTO …
EXIT WHEN c1%NOTFOUND;
…
END LOOP;
```

Uso de % ROWCOUNT

Cuando un cursor es abierto, este atributo recibe el valor de 0 (cero). En adelante, cada vez que se recuperen filas exitosamente con un FETCH, este valor se irá incrementando en uno.

Cuando se utiliza con cursores implícitos, este atributo devuelve el total de filas afectadas por una instrucción del tipo INSERT, UPDATE o DELETE.

3.16.6 Cierre de un cursor

Para cerrar un cursor se utiliza la sentencia CLOSE.

Sintaxis:

```
CLOSE nombre_cursor;
```

Una vez que un cursor ya ha sido cerrado, es posible volverlo a abrir sin tener que declararlo otra vez. Cualquier otra operación que se desee efectuar sobre un cursor no operativo (cerrado) provocará una excepción del tipo "invalid_cursor".

Ejemplo:

```
DECLARE
    CURSOR c1 IS SELECT ename, job FROM emp
                        WHERE deptno = 10;
…
BEGIN
    OPEN c1;
…
    CLOSE c1;
…
END;
```

3.16.7 Cursores en el ciclo FOR

Un cursor en el bucle FOR procesa filas en un cursor explícito. Es un camino corto porque el cursor es abierto, los registros son recuperados cada uno en la iteración del ciclo., y el cursor es cerrado automáticamente cuando todos los registros son procesados. El ciclo es terminado automáticamente cuando el último registro es procesado.

Sintaxis:

```
FOR nombre_registro IN nombre_cursor LOOP
sentencias 1;
. . .
END LOOP;
```

Donde:

nombre_registro	Es el nombre implícito del registro declarado
nombre_cursor	Es un identificador para el previo cursor declarado

Ejemplo:

```
DECLARE
   CURSOR emp_cursor IS SELECT ename, deptno
   FROM emp;
BEGIN
   FOR emp_record IN emp_cursor LOOP
-- implicita acción del open y fetch
   IF emp_record.deptno = 30 THEN
...
END LOOP; --implicito cierre
END;
```

Notas Rápidas:

- No declare el registro que controla el ciclo. Su alcance es sólo en el ciclo.
- Los Atributos del cursor pueden ser usados dentro del ciclo.

3.17 EXCEPCIONES EN PL/SQL

3.17.1 Excepciones

Una excepción es una advertencia o condición de error. Estas pueden ser definidas en forma interna o explícitamente por el usuario.

Cuando ocurre un error se alcanza la excepción, esto quiere decir que se ejecuta la porción del programa donde ésta se encuentra implementada, transfiriéndose el control a ese bloque de sentencias. Las excepciones definidas por el usuario deben ser alcanzadas explícitamente utilizando la sentencia RAISE.

Las excepciones proveen claridad y comodidad al código.

3.17.2 Excepciones predefinidas

Las excepciones predefinidas no necesitan ser declaradas. Simplemente se utilizan cuando estas son accionadas por algún error determinado.

Excepciones predeterminadas en PL/SQL.

Nombre	**Accionada cuando...**	**SQLCODE**
ACCESS_INTO_ NULL	El programa intentó asignar valores a los atributos de un objeto no inicializado	-6530
COLLECTION_ IS_NULL	El programa intentó asignar valores a una tabla anidada aún no inicializada	-6531
CURSOR_ ALREADY_ OPEN	El programa intentó abrir un cursor que ya se encontraba abierto	-6511
DUP_VAL_ ON_INDEX	El programa intentó almacenar valores duplicados en una columna que se mantiene con restricción de integridad de un índice único (unique index)	-1
INVALID_ CURSOR	El programa intentó efectuar una operación no válida sobre un cursor	-1001
INVALID_ NUMBER	El programa intentó convertir una cadena de caracteres a número y resultó un número invalido	-1722

Nombre	Accionada cuando...	SQLCODE
LOGIN_ DENIED	El programa intentó conectarse a Oracle con un nombre de usuario o contraseña inválida	-1017
NO_DATA_ FOUND	Una sentencia SELECT INTO no devolvió valores o el programa referenció un elemento no inicializado en una tabla indexada	+100
NOT_LOGGED _ON	El programa efectuó una llamada a Oracle sin estar conectado	-1012
PROGRAM_ ERROR	PL/SQL tiene un problema interno	-6501
ROWTYPE_ MISMATCH	El valor a asignar y la variable que lo contendrá tienen tipos incompatibles o un parámetro pasado a un subprograma no es del tipo esperado.	-6504
SELF_IS_NULL	El parámetro SELF (el primero que es pasado a un método MEMBER) es nulo	-30625
STORAGE_ ERROR	La memoria se terminó o está Corrupta	-6500
SUBSCRIPT_ BEYOND_ COUNT	El programa está tratando de referenciar un elemento de un arreglo indexado que se encuentra en una posición más grande que el número real de elementos de la colección	-6533

Nombre	Accionada cuando...	SQLCODE
SUBSCRIPT_ OUTSIDE_ LIMIT	El programa está referenciando un elemento de un arreglo utilizando un número fuera del rango permitido (por ejemplo, el elemento "-1")	-6532
SYS_INVALID _ROWID	La conversión de una cadena de caracteres hacia un tipo rowid falló porque la cadena no representa un número	-1410
TIMEOUT_ON _RESOURCE	Se excedió el tiempo máximo de espera por un recurso en Oracle	-51
TOO_MANY_ ROWS	Una sentencia SELECT INTO devuelve más de una fila	-1422
VALUE_ ERROR	Ocurrió un error aritmético, de conversión o truncamiento. Por ejemplo, sucede cuando se intenta calzar un valor muy grande dentro de una variable más pequeña	-6502
ZERO_DIVIDE	El programa intentó efectuar una división por cero	-1476

3.17.3 Excepciones definidas por el usuario

PL/SQL permite al usuario definir sus propias excepciones, las que deberán ser declaradas y accionadas explícitamente utilizando otros comandos del lenguaje.

Declaración

Las excepciones sólo pueden ser declaradas en el segmento DECLARE de un bloque, subprograma o paquete.

Sintaxis:

Nombre _ excepción EXCEPTION;

Ejemplo:

```
DECLARE
error_01 EXCEPTION;
```

Reglas de Alcance

Una excepción no puede ser declarada dos veces en un mismo bloque. Tal como las variables, una excepción declarada en un bloque es local a ese bloque y global a todos los sub-bloques que comprende.

La Sentencia "RAISE"

La sentencia RAISE permite accionar una excepción en forma explícita. Es factible utilizar esta sentencia en cualquier lugar que se encuentre dentro del alcance de la excepción.

Ejemplo:

```
DECLARE
out_of_stock EXCEPTION; -- declaración de la excepción
total NUMBER(4);
BEGIN
…
IF total < 1 THEN
RAISE out_of_stock;          -- llamado a la excepción
END IF;
EXCEPTION
WHEN out_of_stock THEN
-- manejar el error aquí
WHEN OTHERS THEN
…
END;
```

La excepción OTHERS simboliza cualquier condición de excepción que no ha sido declarada. Se utiliza comúnmente al final del bloque de excepciones para absorber cualquier tipo de error que no ha sido previsto por el programador. En ese caso, es común observar la sentencia ROLLBACK en el grupo de sentencias

de la excepción o alguna de las funciones SQLCODE – SQLERRM, que se detallan en a continuación en esta unidad.

3.17.4 Uso de SQLCODE y SQLRRM

Al manejar una excepción es posible apoyarse con las funciones predefinidas SQLCode y SQLErrm para aclarar al usuario la situación de error acontecida.

SQLCODE siempre retornará el número del error de Oracle y un "0" (cero) en caso exitoso al ejecutarse una sentencia SQL.

SQLERRM retornará el correspondiente mensaje de error para la situación ocurrida. También es posible entregarle a la función SQLERRM un número negativo que represente un error de Oracle y ésta devolverá el mensaje asociado.

Estas funciones son muy útiles cuando se utilizan en el bloque de excepciones, para aclarar el significado de la excepción OTHERS, cuando ésta ocurre.

Estas funciones no pueden ser utilizadas directamente en una sentencia SQL, pero sí se puede asignar su valor a alguna variable de programa y luego usar esta última en alguna sentencia.

Ejemplo:

```
DECLARE
err_num NUMBER;
err_msg VARCHAR2(100);
BEGIN
…
EXCEPTION
WHEN OTHERS THEN
err_num := SQLCODE;
err_msg := SUBSTR(SQLERRM, 1, 100);
INSERT INTO errores VALUES(err_num, err_msg);
END;
```

3.18 PROCEDIMIENTOS EN PL/SQL

3.18.1 Procedimientos

Un procedimiento es un subprograma PL/SQL para encapsular tareas que se usan frecuentemente. Un subprograma es un programa PL/SQL con nombre que puede tomar parámetros y puede ser llamado una y otra vez para realizar una tarea concreta.

La sintaxis para construirlos es la siguiente*:*

```
CREATE OR REPLACE PROCEDURE nombre [ (parámetro [,
parámetro, …] ) ] IS
[declaraciones_locales]
BEGIN
sentencias_ejecutables
[EXCEPTION
condiciones_de_excepción]
END [nombre] ;
```

Cada parámetro se escribe con la siguiente notación:

nombre_parámetro [IN | OUT [NOCOPY] | IN OUT [NOCOPY] tipo_de_dato
[{ := | DEFAULT } expresión]

3.18.2 Uso de parámetros

Los parámetros formales pueden tener tres modos: IN, OUT, IN OUT. Si no se especifica su modo será por defecto el modo IN.

En la siguiente tabla se describen los tres modos que pueden tener los parámetros formales.

Modo	**Descripción**
IN	El valor del parámetro real se pasa al procedimiento cuando éste es invocado. El parámetro se considera de sólo lectura y no puede ser cambiado.
OUT	Se ignora cualquier valor que tenga el parámetro real cuando se llama al procedimiento. El parámetro formal se considera de solo lectura.

IN OUT	El valor del parámetro real se pasa al procedimiento cuando éste es invocado, el parámetro formal puede ser tanto de lectura como de escritura.

3.18.3 Creando un procedimiento

Un procedimiento posee dos partes: una especificación y un cuerpo.

La especificación es simple, comienza con la palabra PROCEDURE y termina (en la misma línea) con el nombre del procedimiento o la lista de parámetros (que es opcional).

El cuerpo del procedimiento comienza con la palabra reservada "IS" y termina con "END", seguido opcionalmente por el nombre del procedimiento.

El siguiente ejemplo que muestra cómo crear y almacenar el procedimiento PRINTLINE en una base de datos.

Ejemplo:

```
CREATE OR REPLACE PROCEDURE PRINTLINE
(width in integer, chr in char default '-' )
AUTHID DEFINER
IS
BEGIN
FOR i IN 1..width LOOP
DBMS_OUTPUT.PUT(chr);
END LOOP;
DBMS_OUTPUT.PUT(' ');
END PRINTLINE;
/
```

Si crea el procedimiento con opción AUTHID CURRENT_ USER, Oracle ejecuta el procedimiento (cuando sea llamado) usando el dominio de privilegio del usuario que llama al procedimiento. Para ejecutar el procedimiento con éxito, quien realiza la llamada debe tener los privilegios necesarios para acceder a todos los objetos de bases de datos referenciados en el cuerpo del procedimiento almacenado.

Si crea el procedimiento con la opción predeterminada AUTHID DEFINER, Oracle ejecuta el procedimiento usando el dominio de privilegio del propietario del procedimiento. Para ejecutar el procedimiento con éxito, el propietario del procedimiento debe tener los privilegios necesarios para acceder a todos los objetos de la base de datos referenciados en el cuerpo el procedimiento almacenado.

Para simplificar la administración de los privilegios para los usuarios de aplicaciones, la opción predeterminada AUTHID DEFINER debería ser la elección típica a la hora de crear un procedimiento almacenado, de esta forma, no

tiene que conceder privilegios a todos los usuarios que necesitan llamar al procedimiento.

3.18.4 Usando un procedimiento

Para usar el procedimiento PRINTLINE en cualquier otro programa PL/SQL simplemente se llama dicho procedimiento almacenado en la base de datos.

Por ejemplo, se pueden introducir los siguientes bloques PL/SQL anónimos, que usan el procedimiento almacenado PRINTLINE.

```
BEGIN
PRINTLINE(40,'*');
END;
/
BEGIN
PRINTLINE(10);
END;
/
```

Las salidas del programa son las siguientes:

```
PL/SQL procedure successfully completed
----------
PL/SQL procedure successfully
completed
```

3.19 FUNCIONES EN PL/SQL

3.19.1 Funciones

Una función es un subprograma que calcula un valor.

Para crear una función almacenada en la base de datos de ORACLE, use el comando CREATE FUNCTION.

La sintaxis para construir funciones es la siguiente:

```
CREATE FUNCTION nombre [ (parámetro [, parámetro, …] ) ] RETURN tipo_de_dato IS
BEGIN
      sentencias_ejecutables
[EXCEPTION
 condiciones_de_excepción]
END [nombre] ;
```

Y la sintaxis de los parámetros es idéntica al caso de los procedimientos:

nombre_parámetro [IN | OUT [NOCOPY] | IN OUT [NOCOPY] tipo_de_dato [{ := | DEFAULT } expresión]

3.19.2 Uso de parámetros

Los subprogramas traspasan información utilizando parámetros. Las variables o expresiones referenciadas en la lista de parámetros de una llamada a un subprograma son llamados parámetros actuales. Las variables declaradas en la especificación de un subprograma y utilizadas en el cuerpo son llamados parámetros formales.

En este último caso (para los parámetros formales) se pueden explicitar tres modos diferentes para definir la conducta de los parámetros: IN, OUT e IN OUT.

Sin embargo, evite usar los tipos de parámetros OUT e IN OUT en las funciones, ya que por su naturaleza éstas devuelven un sólo dato y no a través de parámetros. Es una mala práctica hacer que una función devuelva muchos valores; para eso utilice los procedimientos.

Parámetros de modo IN

Estos parámetros son la entrada a las funciones y procedimientos y actúan dentro de ellas como constantes, es decir, sus valores no pueden ser alterados dentro de un subprograma.

Los parámetros actuales (que se convierten en formales dentro de un subprograma) pueden ser constantes, literales, variables inicializadas o expresiones.

Parámetros de modo OUT

Un parámetro de salida (OUT) permite comunicar valores al bloque de programa que invocó al subprograma que utiliza este parámetro.

Parámetros de modo OUT (Continuación)

Esto quiere decir además, que es posible utilizar un parámetro de este tipo como si fuera una variable local al subprograma.

En el programa que invoca la función o procedimiento, el parámetro de modo OUT debe ser una variable, nunca una expresión o una constante.

Parámetros de modo IN OUT

Esta modalidad permite proporcionar valores iniciales a la rutina del subprograma y luego devolverlos actualizados. Al igual que el tipo anterior, un parámetro de este tipo debe corresponder siempre a una variable.

Si la salida de un subprograma es exitosa, entonces PL/SQL asignará los valores que corresponda a los parámetros actuales (los que reciben los valores de los parámetros formales en la rutina que llama al subprograma). Por el contrario, si la salida del subprograma se produce con un error no manejado en alguna excepción, PL/SQL no asignará ningún valor a los parámetros actuales.

3.19.3 Creando una función

La función posee una especificación y un cuerpo.

El segmento de especificación comienza con la palabra FUNCTION y termina con la cláusula RETURN, la cual especifica el tipo de dato retornado por la función.

El cuerpo comienza con la palabra "IS" y termina con la palabra "END", es decir, incluye las secciones de declaraciones, sentencias ejecutables y una parte opcional de manejo de excepciones.

Ejemplo:

```
CREATE OR REPLACE FUNCTION revisa_salario
(salario in REAL, cargo in CHAR(10) := 'SALESMAN')
RETURN BOOLEAN IS
salario_minimo REAL;
salario_maximo REAL;
BEGIN
SELECT lowsal, highsal INTO salario_minimo,

salario_maximo


FROM EMP WHERE job = cargo ;
if salario >= salario_minimo and salario <=  salario_maximo
then
return true;
else
return false;
```

La Sentencia RETURN

Esta sentencia termina inmediatamente la ejecución de un programa, retornando el control al bloque de programa que lo llamó.

No se debe confundir con la cláusula return de las funciones, que especifica el tipo de dato devuelto por ella.

Un subprograma puede contener varias sentencias Return. Si se ejecuta cualquiera de ellas, el subprograma completo se termina.

La sintaxis para los procedimientos es simple, sólo se necesita la palabra RETURN. Sin embargo, en el caso de las funciones, esta sentencia debe contener un valor, que es aquel que se va a devolver al programa que la llamó.

La expresión que sigue a la sentencia puede ser tan compleja como se desee pero siempre debe respetar el tipo de datos que está definido en la cabecera (especificación) de la función.

Una función debe contener como mínimo una sentencia RETURN, de otra manera, al no encontrarla, PL/SQL generará la excepción PROGRAM_ERROR.

3.19.4 Usando una función

Para usar la función revisa_salario debe ser llamada desde una sentencia PL/SQL que reciba un valor booleano, como por ejemplo, en:

```
DECLARE
renta_actual REAL;
codcargo CHAR(10);
BEGIN
…
IF revisa_salario (renta_actual, codcargo) THEN
```

La función revisa_salario actúa como una variable de tipo booleano, cuyo valor depende de los parámetros recibidos.

3.20 PAQUETES EN PL/SQL

3.20.1 Paquetes

Un paquete es un esquema u objeto que agrupa tipos de PL/SQL relacionados, ítems y subprogramas.

Los paquetes se constituyen de dos partes: la especificación y el cuerpo.

La especificación es la interfaz con las aplicaciones. En ella es posible declarar los tipos, variables, constantes, excepciones, cursores y subprogramas disponibles para su uso posterior.

El cuerpo define completamente a cursores y subprogramas e implementa lo que se declaró inicialmente en la especificación.

Es posible depurar y modificar cuantas veces se desee el cuerpo de un paquete sin necesidad de alterar por ello la especificación del mismo.

3.20.2 Creando un paquete

Para crear un paquete se usa la sentencia CREATE PACKAGE.

Ejemplo:

```
CREATE OR REPLACE PACKAGE partmgmt is
PROCEDURE insertpart (partrecord in parts%rowtype);
PROCEDURE updatepart (partrecord in parts%rowtype);
PROCEDURE deletepart (partid IN INTEGER);
PROCEDURE printpartsprocessed;
end partmgmt;
/
```

```
CREATE OR REPLACE PACKAGE body partmgmt as
/*variable global privada*/
                    rowsprocessed integer := 0;
```

```
/*Subprogramas publicos*/
PROCEDURE insertpart (partrecord IN parts%rowtype) is
begin
insert into parts values(partrecord.p_id, partrecord.unitprice,
partrecord.description);
rowsprocessed := rowsprocessed +1;
end insertpart;

PROCEDURE updatepart (partrecord in parts%rowtype) is
begin
update parts set description = partrecord.description,
unitprice = partrecord.unitprice
where p_id = partrecord.p_id;
rowsprocessed := rowsprocessed +1;
end updatepart;
```

```
PROCEDURE deletepart (partid IN INTEGER) IS
BEGIN
DELETE PARTS WHERE ID = PARTID;
rowsprocessed := rowsprocessed +1;
END deletepart;
PROCEDURE printpartsprocessed is
begin
dbms_output.put_line('parts processed this
session:'||rowsprocessed);
end printpartsprocessed;
end partmgmt;
/
```

3.20.3 Usando un paquete

Para hacer uso de un paquete almacenado en la base de datos desde un bloque PL/SQL, se hace referencia al objeto de paquete usando la notación de puntos .

Ejemplo 1: El siguiente bloque PL/SQL anónimo usa el procedimiento insertpart del paquete partmgmt para insertar una nueva pieza en la tabla.

Ejemplo 2: El siguiente bloque PL/SQL anónimo actualiza el nombre de un ítem o producto usando el procedimiento updatepart en el paquete partmgmt.

```
Declare
newpart parts%rowtype;
Begin
newpart.p_id := 6;
newpart.unitprice := 49;
newpart.description := 'MOUSE';
partmgmt.insertpart(newpart);
end;
/
```

```
Declare
apart parts%rowtype;
Begin
Select * into apart from part where p_id = 6;
Apart.description := 'MOUSE 2N'
partmgmt.updatepart(apart);
end;
/
```

Comando EXECUTE

Use el comando EXECUTE para ejecutar un procedimiento de un paquete.

El comando EXECUTE es equivalente a rodear una llamada de procedimiento con las palabras clave Begin y end que delimitan el inicio y el final de un bloque PL/SQL anónimo.

Ejemplo 1: Usar el comando EXECUTE para ejecutar el procedimiento deletepart del paquete partmgmt para eliminar el ítem más reciente.

```
EXECUTE PARTMGMT.DELETEPART(6);
```

Ejemplo 2: Usar el comando EXECUTE para obtener el número de filas de la tabla PARTS procesada por la sesión actual usando el paquete partmgmt;

```
EXECUTE PARTMGMT.PRINTPARTSPROCESSED;
```

3.20.4 Paquetes de utilidades incorporados

En la siguiente tabla se muestran los paquetes de utilidades incorporados en PL/SQL.

Nombre Paquete	**Descripción**
DBMS_ALERT	Procedimientos y funciones que permiten a las aplicaciones nombrar y emitir señales de alerta sin sondeo
DBMS_AQ DBMS_AQADM	Procedimientos y funciones para poner en cola la ejecución de transacciones y administrar los mecanismos de encolamiento
DBMS_DDL DBMS_UTILITY	Procedimiento que proporciona acceso a un número limitado de sentencias DDL dentro de los programas PL/SQL

Nombre Paquete	**Descripción**
DBMS_DESCRIBE	Procedimiento que describen la API de funciones y procedimientos almacenados

DBMS_JOB	Procedimientos y funciones que administran los mecanismos de las tareas de encolamiento de una base de datos
DBMS_LOB	Procedimientos y funciones que manipulan BLOB, CLOB, NCLOB y BFILE
DBMS_LOCK	Procedimientos y funciones que permiten a las aplicaciones coordinar el acceso a los recursos compartidos
DBMS_OUTPUT	Procedimientos y funciones que permiten a un programa PL/SQL generar salida de terminal
DBMS_PIPE	Procedimientos y funciones que permiten a las sesiones de base de datos comunicarse usando conductos (canales de comunicación)
DBMS_ROWID	Procedimientos y funciones que permiten a las aplicaciones interpretar fácilmente un carácter ROWID externo de base 64
DBMS_SESSION	Procedimientos y funciones que controla la sesión de usuario de una aplicación
DBMS_SQL	Procedimientos y funciones que realizan SQL dinámico desde dentro de un programa PL/SQL
Nombre Paquete	Descripción
DBMS_ TRANSACTION	Procedimientos que realizan un control limitado de las transacciones
UTL_FILE	Procedimientos y funciones que permiten a un programa PL/SQL leer y escribir archivos de texto al sistema de archivos del servidor

3.21 TRIGGERS EN PL/SQL

3.21.1 Triggers

Un disparador de base de datos o trigger es un procedimiento almacenado que se asocia con una tabla específica.

Cuando las aplicaciones apuntan a la tabla con una sentencia SQL DML que verifica las condiciones de ejecución del disparador, Oracle ejecuta automáticamente el disparador para que realice su trabajo.

Se pueden usar disparadores para personalizar la reacción de un servidor de Base de Datos de ORACLE ante eventos de aplicaciones.

3.21.2 Sintaxis de definición de un trigger

La sintaxis para construir un trigger es la siguiente:

```
CREATE [OR REPLACE] TRIGGER NOMBRE_TRIGGER
{BEFORE|AFTER}
{DELETE|INSERT|UPDATE [OF col1, col2, . . ., colN]
[OR {DELETE|INSERT|UPDATE [OF col1, col2, . . ., colN]. . .]}
ON table
[REFERENCING OLD AS oldname, NEW as newname]
[FOR EACH ROW [WHEN (condition)]]
.....bloque PL/SQL
                    END NOMBRE_TRIGGER
```

Una definición de un trigger incluye las siguientes partes:

- La definición de un disparador incluye una lista de sentencias de disparador, incluyendo INSERT, UPDATE y/o DELETE, que lanza el disparador. Un disparador está asociado con una, y sólo una tabla.
- Un disparador puede configurarse para que se lance antes o después de la sentencia del disparador para proporcionar lógica especifica de la aplicación.
- La definición de un disparador indica si el disparador es un disparador de sentencias o un disparador de filas. Un disparador de sentencias se dispara solo una vez, no importa a cuantas filas afecte la sentencia del disparador.
- El uso de OR REPLACE permite sobrescribir un trigger existente. Si se omite, y el trigger existe, se producirá, un error.
- El modificador FOR EACH ROW indica que el trigger se disparará cada vez que se desee hacer operaciones sobre una fila de la tabla. Si se acompaña del modificador WHEN, se establece una restricción; el trigger solo actuará, sobre las filas que satisfagan la restricción.

3.21.3 Eliminando un trigger

Si se desea eliminar (borrar) un trigger, se usa la instrucción:

DROP TRIGGER NombreTrigger;

3.21.4 Usando un trigger

Introduzca la sentencia CREATE TRIGGER para crear un disparador que registre automáticamente alguna información básica sobre los cambios DML realizados a la tabla EMP. EL disparador LOGEMPCHANGES es un disparador posterior a las sentencias, que se lanza después de la sentencia de disparo, sin que importe el número de filas a que afecta la sentencia del disparador.

```
CREATE OR REPLACE TRIGGER logempchanges
AFTER INSERT OR UPDATE OR DELETE ON
EMP
DECLARE
statementType CHAR(1);
BEGIN
IF INSERTING THEN
statementType := 'I';
ELSIF UPDATING
statementType := 'U';
ELSE
statementType := 'D';
END IF;
INSERT INTO emplog
VALUES (SYSDATE, statementType,USER);
END logempchanges
/
```

CONCLUSIONES

- El desarrollo de este trabajo implico un gran estudio ya que son muchos los temas y conceptos que se deben tener en cuenta, debido a que el trabajo de grado es un Herramienta para el Aprendizaje de Oracle, se deben abarcar todos los temas necesarios para la asignatura de Administración de Bases de Datos.

- Todos los laboratorios que realizaron para el aprendizaje de Oracle fueron puestos en practicas en la Corporación Universitaria Tecnológica de Bolívar por parte de las autoras de la tesis, para probar que el diseño de estos fueran adecuados y con base en esto se pudieron hacer mejoras de los mismos.

- El estudiante de Oracle requiere que día a día este informado, puesto que Oracle constantemente esta presentando nuevas herramientas que facilitan y mejoran su calidad.

- Con los laboratorios se logra el propósito que el estudiante pudiera comprender mejor y pudiera poner en práctica los conocimientos

teóricos en la materia Administración de Base de Datos.

- Las herramientas utilizadas en los laboratorios fueron elaboradas para que el profesor utilizara estas mismas, como herramientas de evaluación del trabajo del alumno. De igual forma se le pueden hacer cambios para futuros laboratorios.

RECOMENDACIONES

- Para la realización de los laboratorios el profesor debe realizar con anticipación una clase teórica de los temas del laboratorio.

- Para la implementación exitosa de estos laboratorios es de gran importancia que la Universidad disponga del software y hardware necesario para la realización de estos.

- Es importante que el profesor a cargo este al tanto de todos los avances e innovaciones de Oracle con respectos a los temas tratados en la materia para ver la forma en que puedan ser aplicados.

- El profesor deberá cambiar los ejercicios planteados en las guías a medida que pase los semestres para que estos no se vuelvan repetitivos y los alumnos de los semestres venideros encuentren todos los laboratorios resueltos.

- El alumno debe primero leer el manual del usuario incluido dentro de la guía del estudiante para que pueda desarrollar el laboratorio de una forma satisfactoria.

BIBLIOGRAFÍA

Manual de Oracle 8i Versión Server.

Loney, Kevin. Oracle. Manual de Administrador, McGraw – Hill / Interamericana de España. 1999.

Loney, Kevin**.** Oracle8. Manual de Administrador MH 2000.

Loney, Kevin. Oracle8. Manual de Administrador, McGraw – Hill / Interamericana de España. 1999.

Oracle education. Oracle8. Database Adminitration. Oracle corporation, 1998

Venus, Alberto. Teoría del diseño de las galaxias. Guatemala, Prensa Universitaria, 1997.

ANEXOS

ANEXO A.

En el CD se encuentran almacenados los archivos del documento escrito del manuscrito de la tesis, el manual del profesor y el manual del alumno en archivos punto PDF, además se encuentra almacenados los archivos de ejecución de las respuestas de todos laboratorios en dos clases de formatos como es el formato punto TXT y el formato punto SQL.