Universidad Autónoma de Querétaro

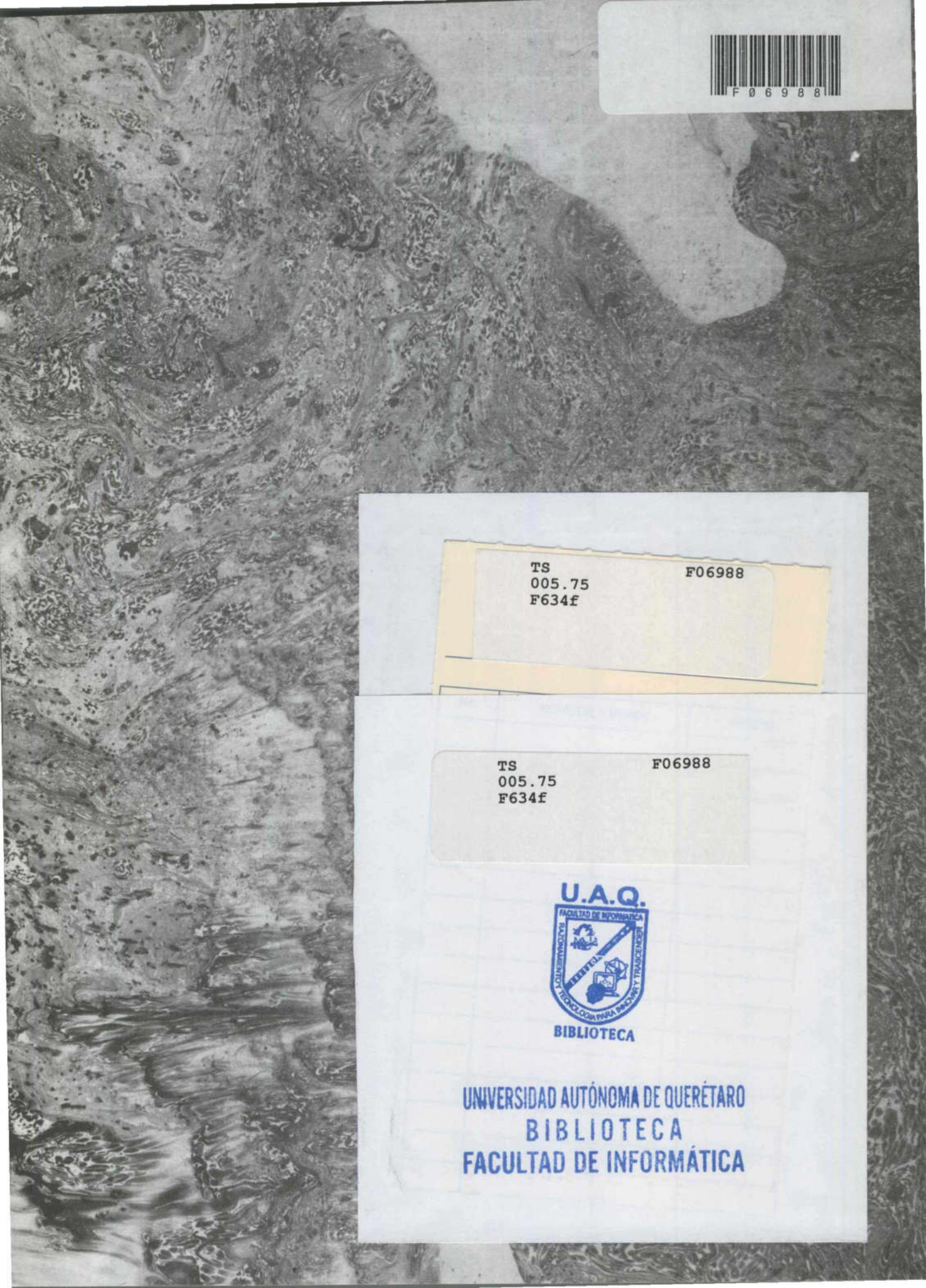
Facultad de Informática

Tesina: Fundamentos PL/SQL

Presenta: Luis Guillermo Flores Ceja

Expediente: 80324

Asesor: I.S.C. Jabel Reséndiz González



CARTA DE ACEPTACIÓN

Por este medio, se otorga constancia de aceptación de tesina para obtener el título de Licenciado en Informática, que presenta el pasante *LUIS GUILLERMO FLORES CEJA*, con el tema denominado *"FUNDAMENTOS PL/SQL"*.

Este trabajo fue desarrollado como una investigación derivada del curso de titulación *"ADMINISTRACIÓN DE BASE DE DATOS ORACLE"*, dando cumplimiento a uno de los requisitos contemplados en el artículo 34 del reglamento de titulación vigente, en lo referente a la opción de titulación por realización y aprobación de cursos de actualización.

Se extiende la presente para los fines legales a que haya lugar y para su inclusión en todos los ejemplares impresos de la tesina, a los seis días del mes de septiembre de dos mil cuatro.

ATENTAMENTE

I.S.C. JABEL RESÉNDIZ GONZÁLEZ
PROF. CURSO DE TITULACIÓN

FUNDAMENTOS DE PL/SQL

AGRADECIMIENTOS

A mi madre, Martha Lucila Ceja Barrera, amigos y familiares por haber creído en mi, apoyándome en las decisiones que tomamos y estar presentes en todos nuestros logros.

Al I.S.C. Jabel Resendiz González, catedrático de esta facultad, por haberme dado la oportunidad de ser mi asesor y apoyado en la realización de este proyecto.

A nuestra escuela, Universidad Autónoma de Querétaro, por la educación que nos ha proporcionado haciendo de nosotros verdaderos agentes de cambio.

RESUMEN

Hoy en día la actualización y protección de la información constituye una prioridad para las empresas que manejan tanto volúmenes pequeños o muy grandes de información. Por lo tanto, la necesidad de integridad, estabilidad y seguridad de la información constituye unos de los rubros donde se invierte una mayor cantidad de recursos.

En el mundo actual de los negocios, la información es dinero. Mas aún el manejo adecuado de ella, como acceder a la información, cómo analizarla, como presentar resultados, darle significado a esta información en la toma de decisiones, etc. Todo ello y mas es posible gracias a la simplificación de procesos que nos permiten los lenguajes estructurados como PL/SQL de Oracle.

La ventaja principal y distintiva de un lenguaje estructurado como PL/SQL es su compartimentación de código y datos, que es la habilidad del lenguaje para seccionar y ocultar del resto del programa toda la información y las instrucciones que son necesarias para realizar rutinas separadas. Esto nos permite a los desarrolladores una mayor facilidad al manejar información, pues los comandos que se utilizan en un lenguaje estructurado son concretos y simples.

A través de PL/SQL podemos emplear las estructuras de SQL para manipular datos en ORACLE, y estructuras de flujo para procesar los datos. Además podemos declarar variables y constantes, definir subprogramas (procedimientos y funciones) y atrapar los errores de ejecución. Esto es, PL/SQL combina el poder de la manipulación de datos de SQL con el poder de procesamiento de datos de un lenguaje procedural.

INDICE

ÍNDICE DE TABLAS Y FIGURAS

TABLAS:

FIGURAS:

INTRODUCCIÓN

Definición

SQL es un lenguaje estructurado de consulta que permite a través de un SMBD por ejemplo el Oracle Server, acceder, manipular y definir los datos de un BD en particular.

SQL * Plus es una herramienta propietaria de Oracle que reconoce y permite el envío de sentencias SQL y PL/SQL al Oracle Server y al motor de ejecución PL para su ejecución y contiene su propio lenguaje de comandos.

Cuadro Comparativo entre las sentencias SQL y los comandos de SQL*PLUS (Tabla.1)

SQL	SQL*PLUS
Es un leguaje	Es un ambiente
Estándar ANSI	Propietario de Oracle
Manipula la data y la definición de las estructuras de los mismos	No manipula los valores de los datos de la BD
Las sentencias se almacenan en el buffer	Los comandos SQL*Plus no se almacenan en el buffer
No pueden ser abreviados	Se pueden abreviar
Utiliza un carácter (;) de terminación para ser ejecutado	No necesita carácter de terminación

Tabla .1 Cuadro Comparativo entre las sentencias SQL y los comandos de SQL*PLUS

¿Qué podemos hacer con SQL*PLUS?

- Conectarse a SQL*PLUS
- Describir la estructura de las tablas
- Editar las sentencias SQL
- Ejecutar sentencias SQL desde el SQL*Plus
- Salvar las sentencias SQL en archivos y editarlos.
- Ejecutar archivos salvados (scripts)
- Carga comandos SQL desde el buffer y editarlos.

Comandos de archivos de SQL*PLUS

- SAVE *filename*
- GET *filename*
- START *filename*
- *@filename*
- *EDIT filename*
- *SPOOL filename*
- *EXIT*

Comandos de edición de SQL*PLUS

- A[PPEND] text
- C[HANGE]/old/new
- Cl[EAR] BUFF[ER]
- DEL
- DEL n
- DEL m n
- I[NPUT]
- I[NPUT] text
- L[IST] n
- L[IST] m n
- R[UN]
- n
- n text

Objetivo

Por medio de este trabajo explico que es PL/SQL y la utilización de éste, mencionando conceptos, sintaxis y ejemplos para una mejor comprensión.

Objetivos particulares

Capítulo 1, doy a conocer los tipos de datos y los comparadores que se utilizan para la manipulación de información.

Capítulo 2, presentaré las funciones de SQL más utilizadas con respecto a fechas, caracteres, numéricas, fecha y conversión de datos.

Capítulo 3, Se muestran las consultas de selección.

Capítulo 4, En éste capítulo se muestran las consultas de selección que se utilizan para indicar al motor de datos que devuelva información de las bases de datos.

Capítulo 5, Se muestra el agrupamiento de los registros con valores idénticos, en la lista de campos especificados, en un único registro.

Capítulo 6, en este capítulo veremos la manipulación de la información y algunas sentencias que nos la facilitan.

Capítulo 7, las tablas son de gran importancia, por lo que veremos que son, como se crean y algunos ejemplos de éstas.

Capítulo 8, en este capítulo se muestra la importancia de los constraints.

Capítulo 9, mencionaré cómo PL/SQL es utilizado, en dónde las estructuras de control, procedimientos, funciones y triggers son utilizados

Justificación

Este trabajo lo realice con la intención de facilitar los conocimientos que he adquirido sobre PL/SQL durante mi preparación profesional, proporcionando un compendio de los puntos más relevantes.

Alcances

Por medio de este trabajo deseo llegar a las personas que tengan la intención de adquirir fundamentos de PL/SQL y Oracle, llevándolos a una conceptualización ejemplificada de los temas mas utilizados en el desarrollo de bases de datos con Oracle.

Limitaciones

Las versiones de Oracle, al irse actualizando van sufriendo ligeras modificaciones, por lo que algunas de estas mejoras pueden no estar incluidas en estos momentos. Pero se muestran los conceptos mas utilizados

1. TIPOS DE DATOS EN ORACLE (SQL)

A continuación se muestran los tipos de datos que se utilizan en SQL. (Tabla 1.1)

TIPO DE DATOS	DESCRIPCIÓN
VARCHAR2(*Tamaño*)	Representa a un dato de tipo carácter de tamaño variable (Un tamaño máximo puede ser especificado, pero por defecto el tamaño mínimo es 1, y el tamaño máximo es 4000).
CHAR(*Tamaño*)	Representa a un dato de tipo carácter de tamaño fijo (Por defecto el tamaño mínimo es 1 y el tamaño máximo es 2000).
NUMBER(p,s)	Representa a un dato de tipo numérico de tamaño variable. Número que posee una precisión p y a una escala s (La precisión es el número total de dígitos decimales, y la escala es el número de dígitos a la derecha del punto decimal. La precisión puede tener un rango del 1 al 38, y la escala puede tener un rango del -84 al 127).
DATE	Permite representar un dato de tipo fecha y hora. La fecha y hora poseen valores entre 1 de Enero de 4712 AC y el 31 de Diciembre del 9999 DC)
LONG	Permite representar datos de tipo carácter pero de tamaño variable hasta los 2 gigabytes.
CLOB	Permite representar un dato de tipo carácter solo en bytes hasta 4 gigabytes.
RAW (*Tamaño*)	Específica a datos binarios primos de un tamaño largo (El tamaño máximo puede ser especificado. El tamaño máximo es 2000).
LONG RAW	Representa a datos binarios primos de longitud variable hasta los 2 gigabytes)
BLOB	Especifica a datos binarios hasta los 4 gigabytes.
BFILE	Almacena datos de tipo binario en un archivo externo, hasta los 4 gigabytes.

Tabla 1.1 Tipos de datos

1.1 OPERADORES DE COMPARACIÓN

1.1.1 DEFINICIÓN

Los operadores de comparación son usados en condiciones que compara una expresión con otra. (Tabla 1.2)

OPERADOR	SIGNIFICADO
=	Igual a
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
<>	No igual que

Tabla 1.2 Operadores

1.1.2 SINTAXIS

```
SELECT [ ] FROM [ ]  WHERE   expr operador valor
```

1.1.3 EJEMPLO

```
SQL> SELECT ename,sal,comm
  2  FROM emp
  3  WHERE sal<=comm;
```

1.1.4 OTROS OPERADORES DE COMPARACIÓN

OPERADORES	SIGNIFICADO
BETWEEN ...AND..	Se usa para mostrar filas asadas en un rango de valores, es decir, entre 2 valores (BETWEEN Limite inferior AND Límite superior).
IN (list)	Se usa para evaluar los valores especificados en una lista.
LIKE	Se usa para ejecutar la búsqueda de letras o números que estén contenidos en un String o literal. - % denota cero o más caracteres. - _ denota un carácter.
IS NULL	Es un valor nulo

Tabla 1.3 Operadores de comparación

1.1.5 EJEMPLOS

```
1) SQL> SELECT ename,sal
       2  FROM emp
       3  WHERE sal  BETWEEN 1000 AND 1500;

2) SQL> SELECT ename,sal, mgr
       2  FROM emp
       3  WHERE mgr  IN (78,79,90,91);

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

```
4) SQL> SELECT ename
       2  FROM emp
       3  WHERE ename LIKE '_A%';

5) SQL> SELECT ename,mgr
       2  FROM emp
       3  WHERE mgr IS NULL';
```

1.2 OPERADORES LÓGICOS

OPERADOR	SIGNIFICADO
AND	Retorna verdad sí ambos componentes de la condición es verdad.
OR	Retorna verdad sí algunos de los componentes de la condición sea verdad
NOT	Retorna verdad si la siguiente condición es falso.

Tabla 1.4 Operadores lógicos

1.2.1 ORDEN DE PRECEDENCIA

El orden de precedencia se anula por el uso de los paréntesis (Tabla 1.5).

ORDEN DE EVALUACION	OPERADOR
1	Todos los operadores de comparación.
2	NOT
3	AND
4	OR

Tabla 1.5 Orden de precedencia

1.2.2 EJEMPLOS

```
1) SQL> SELECT ename,jo,sal
       2  FROM emp
       3  WHERE job='SALESMAN'
       4  OR job='PRESIDENT'
       5  AND sal>1500;
```

Se ejecutan en el siguiente orden:

- La primera condición es que job sea igual a 'PRESIDENT' y el salario sea mayor que 1500.
- La segunda condición es que job sea igual a 'SALESMAN'.

```
2) SQL> SELECT ename,jo,sal
       2  FROM emp
       3  WHERE (job='SALESMAN'
       4  OR job='PRESIDENT' )
       5  AND sal>1500;
```

Se ejecuta en el siguiente orden:

- La primera condición es que job sea igual a 'PRESIDENT' ó 'SALESMAN'.
- La segunda condición es que el salario sea mayor que 1500.

2. FUNCIONES DE SQL

2.1 DEFINICIÓN

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

2.2 USO DE LAS FUNCIONES

Las funciones pueden ser usadas para realizar lo siguiente:

- Realizar cálculos de unos datos.
- Modificar un el dato de un items individual.
- Manipular la salida para un grupo de filas.
- Cambiar el tipo de datos de una columna.
- Mostrar el formato de las fechas y números.

2.3 FUNCIONES DE CARACTERES

2.3.1 DEFINICIÓN

Acepta caracteres como entrada y puede retornar ambos valores: carácter y numérico. (Tabla 2.1)

FUNCION	PROPOSITO
LOWER (Columna | expresión)	Convierte los valores de cada carácter a

Tabla 2.1 Funciones de Caracter

	minúscula.
UPPER (Columna | expresión)	Convierte los valores de cada carácter a mayúscula.
CONCAT (Columna1 | expresión1, Columna2 | expresión2)	Concatena el valor del primer carácter con el valor del segundo carácter, es equivalente al operador de concatenación (||).
SUBSTR (columna | expresión, m[n])	Retorna los caracteres específicos del comienzo de cada carácter, como la posición m, y n caracteres de largo (Sí es negativo, comienza a contar desde el final del valor del carácter. Sí n es omitido, todos los caracteres para el fin del String son retornadas.
INITCAP (Columna | expresión)	Convierte el valor de cada carácter a mayúscula para la primera letra de cada palabra, y las demás en minúsculas.
LENGTH (Columna | expresión)	Retorna el valor del número de caracteres.
INSTR (Columna | expresión,m)	Retorna la posición numérica del carácter especificado.
LPAD (Columna | expresión, n, 'String')	Conjunto de valores de caracteres trasladados a la derecha para un ancho total n posiciones de caracteres.

Tabla 2.1 Funciones de Carácter (continuación)

2.3.2 EJEMPLOS

- LOWER ('SQL Course') → sql course
- UPPER ('SQL Course') → SQL COURSE
- INITCAP ('SQL Course') → Sql Course
- CONCAT ('Buen', 'String') → BuenString
- SUBSTR ('String',1,3) → Str

- INSTR ('String','r') → 3
- LENGTH ('String') → 6
- LPAD (sal,10,'*') → ******5000

2.4 FUNCIONES NUMÉRICAS

2.4.1 DEFINICIÓN

Acepta valores numéricos como entrada y retorna valores numéricos. Entre ellas tenemos (Tabla 2.2)

FUNCION	PROPOSITO
ROUND (Columna | expresión,n)	Redondea el valor especificado en el número decimal n posiciones.
TRUNC (Columna | expresión,n)	Trunca los valores que especificado en el número decimal n posiciones..
MOD (m,n)	Retorna el resto de la división.

Tabla 2.2 Funciones numéricas

2.4.2 EJEMPLOS

- ROUND(45.948,2) → 45,95
- TRUNC(45.948,2) → 45.94
- MOD(1600,300) → 100

2.5 FUNCIONES DE FECHA

2.5.1 DEFINICIÓN

Opera en valores del tipo de datos (Todas los datos de las funciones retornan un valor del mismo tipo de dato que el dato exceptuando a la función MONTHS_BETWEEN, la cual retorna un número. (Tabla 2.3)

FUNCION	DESCRIPCION
MONTHS_BETWEEN (fecha1, fecha2)	Retorna el número de meses entre dos fechas.
ADD_MONTHS (fecha,n)	Añade n números de meses del calendario a la fecha.
NEXT_DAY(fecha,'char')	Retorna el próximo día de la semana de la fecha especificada.
LAST_DAY(fecha)	Retorna el último día del mes que contiene la fecha.
ROUND (date[,'fmt'])	Retorna la fecha redondeada para la unidad especificada por el formato fmt.
TRUNC(date[,'fmt'])	Retorna la fecha con la porción de tiempo del día truncada a la unidad especificada por el formato fmt.

Tabla 2.3 Funciones de Fecha

2.5.2 EJEMPLOS

- MONTHS_BETWEEN ('01-SEP-95', '11-JAN-94') → 19.6774194
- ADD_MONTHS ('11-JAN-94',) → '11-JUL-94'
- NEXT_DAY('01-SEP-95','FRIDAY') → '08-SEP-95'
- LAST_DAY('01-SEP-95') →'30-SEP-95'

- ROUND('25-JUL-95','MONTH') → 01-AUG-95
- TRUNC('25-JUL-95','MONTH') → 01-JUL-95

2.6 FUNCIONES DE CONVERSIÓN DE DATOS

2.6.1 DEFINICIÓN

Convierte un valor de un tipo de datos a otro tipo.

2.6.2 TIPOS

CONVERSIONES IMPLICITAS

DE	A
VARCHAR2 o CHAR	NUMBER
VARCHAR2 o CHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

Tabla 2.4 Conversiones Implícitas

CONVERSIONES EXPLICITAS

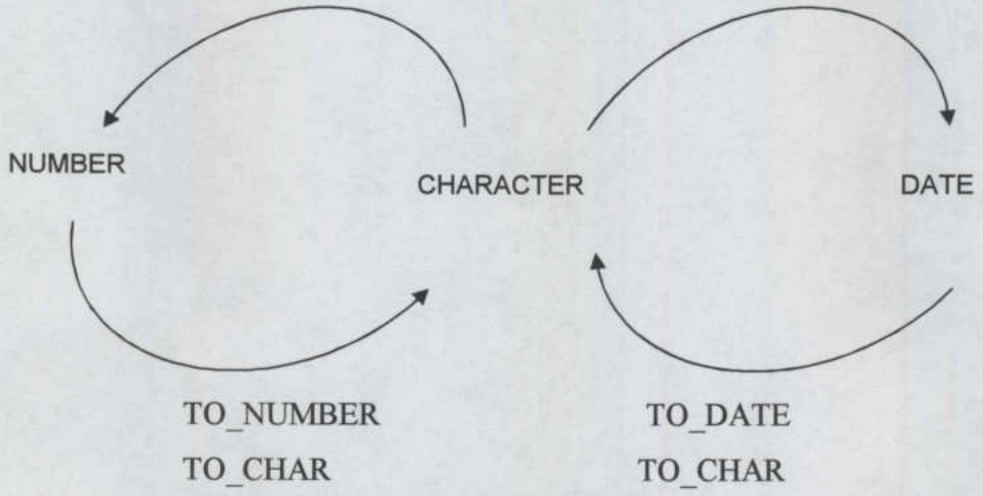


FIG 2.1 Conversiones Explícitas

3. CONSULTAS DE SELECCIÓN

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

3.1 CONSULTAS BÁSICAS

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

```
SELECT Campos FROM Tabla;
```

En donde campos es la lista de campos que se deseen recuperar y tabla es el origen de los mismos, por ejemplo:

```
SELECT Nombre, Telefono FROM Clientes;
```

Esta consulta devuelve un recordset con el campo nombre y teléfono de la tabla clientes.

3.2 ORDENAR LOS REGISTROS

Adicionalmente se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la claúsula ORDER BY Lista de Campos. En donde Lista de campos representa los campos a ordenar. Ejemplo:

```
SELECT CodigoPostal, Nombre, Telefono FROM Clientes
ORDER BY Nombre;
```

Esta consulta devuelve los campos CodigoPostal, Nombre, Telefono de la tabla Clientes ordenados por el campo Nombre. Se pueden ordenar los registros por mas de un campo, como por ejemplo:

```
SELECT CodigoPostal, Nombre, Telefono FROM Clientes
ORDER BY CodigoPostal, Nombre;
```

Incluso se puede especificar el orden de los registros: ascendente mediante la claúsula (ASC -se toma este valor por defecto) ó descendente (DESC) SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY CodigoPostal DESC , Nombre ASC;

3.3 CONSULTAS CON PREDICADO

El predicado se incluye entre la claúsula y el primer nombre del campo a recuperar, los posibles predicados son:

Predicado	Descripción
ALL	Devuelve todos los campos de la tabla
TOP	Devuelve un determinado número de registros de la tabla
DISTINCT	Omite los registros cuyos campos seleccionados coincidan totalmente
DISTINCTROW	Omite los registros duplicados basandose en la totalidad del registro y no sólo en los campos seleccionados.

ALL

Si no se incluye ninguno de los predicados se asume ALL. El Motor de base de datos selecciona todos los registros que cumplen las condiciones de la instrucción SQL. No se conveniente abusar de este predicado ya que obligamos al motor de la base de datos a analizar la estructura de la tabla para averiguar los campos que contiene, es mucho más rápido indicar el listado de campos deseados.

```
SELECT ALL FROM Empleados;
SELECT * FROM Empleados;
```

TOP

Devuelve un cierto número de registros que entran entre al principio o al final de un rango especificado por una cláusula ORDER BY. Supongamos que queremos recuperar los nombres de los 25 primeros estudiantes del curso 2004:

```
SELECT TOP 25 Nombre, Apellido FROM Estudiantes
ORDER BY Nota DESC;
```

Si no se incluye la cláusula ORDER BY, la consulta devolverá un conjunto arbitrario de 25 registros de la tabla Estudiantes .El predicado TOP no elige entre valores iguales. En el ejemplo anterior, si la nota media número 25 y la 26 son iguales, la consulta devolverá 26 registros. Se puede utilizar la palabra reservada PERCENT para devolver un cierto porcentaje de registros que caen al principio o al final de un rango especificado por la cláusula ORDER BY. Supongamos que en lugar de los 25 primeros estudiantes deseamos el 10 por ciento del curso:

```
SELECT TOP 10 PERCENT Nombre, Apellido FROM Estudiantes
ORDER BY Nota DESC;
```

El valor que va a continuación de TOP debe ser un Integer sin signo.TOP no afecta a la posible actualización de la consulta.

DISTINCT

Omite los registros que contienen datos duplicados en los campos seleccionados. Para que los valores de cada campo listado en la instrucción SELECT se incluyan en la consulta deben ser únicos. Por ejemplo, varios empleados listados en la tabla Empleados pueden tener el mismo apellido. Si dos registros contienen López en el campo Apellido, la siguiente instrucción SQL devuelve un único registro:

```
SELECT DISTINCT Apellido FROM Empleados;
```

Con otras palabras el predicado DISTINCT devuelve aquellos registros cuyos campos indicados en la cláusula SELECT posean un contenido diferente. El resultado de una consulta que utiliza DISTINCT no es actualizable y no refleja los cambios subsiguientes realizados por otros usuarios.

DISTINCTROW

Devuelve los registros diferentes de una tabla; a diferencia del predicado anterior que sólo se fijaba en el contenido de los campos seleccionados, éste lo hace en el contenido del registro completo independientemente de los campos indicados en la cláusula SELECT.

```
SELECT DISTINCTROW Apellido FROM Empleados;
```

Si la tabla empleados contiene dos registros: Antonio López y Marta López el ejemplo del predicado DISTINCT devuleve un único registro con el valor López en el campo Apellido ya que busca no duplicados en dicho campo. Este último ejemplo devuelve dos registros con el valor López en el apellido ya que se buscan no duplicados en el registro completo.

3.4 ALIAS

En determinadas circunstancias es necesario asignar un nombre a alguna columna determinada de un conjunto devuelto, otras veces por simple capricho o por otras circunstancias. Para resolver todas ellas tenemos la palabra reservada AS que se encarga de asignar el nombre que deseamos a la columna deseada. Tomado como referencia el ejemplo anterior podemos hacer que la columna devuelta por la consulta, en lugar de llamarse apellido (igual que el campo devuelto) se llame Empleado. En este caso procederíamos de la siguiente forma:

```
SELECT DISTINCTROW Apellido AS Empleado FROM Empleados;
```

3.5 BASES DE DATOS EXTERNAS

Se debe hacer referencia a la recuperación de registros de bases de datos externa. Es ocasiones es necesario la recuperación de información que se encuentra contenida en una tabla que no se encuentra en la base de datos que ejecutará la consulta o que en ese momento no se encuentra abierta, esta situación la podemos salvar con la palabra reservada IN de la siguiente forma:

```
SELECT DISTINCTROW Apellido AS Empleado
FROM Empleados IN
'c:\databases\gestion.mdb';
```

En donde c:\databases\gestion.mdb es la base de datos que contiene la tabla Empleados.

4. CRITERIOS DE SELECCIÓN

Para comenzar hay que recalcar tres detalles de vital importancia. El primero de ellos es que cada vez que se desee establecer una condición referida a un campo de texto la condición de búsqueda debe ir encerrada entre comillas simples; la segunda es que no se posible establecer condiciones de búsqueda en los campos memo y; la tercera y última hace referencia a las fechas.

Las fechas se deben escribir siempre en formato mm-dd-aa en donde mm representa el mes, dd el día y aa el año, hay que prestar atención a los separadores -no sirve la separación habitual de la barra (/), hay que utilizar el guión (-) y además la fecha debe ir encerrada entre almohadillas (#). Por ejemplo si deseamos referirnos al día 3 de Septiembre de 2004 deberemos hacerlo de la siguiente forma;

```
#09-03-95# ó #9-3-95#.
```

4.1 OPERADORES LÓGICOS

Los operadores lógicos soportados por SQL son: AND, OR, XOR, Eqv, Imp, Is y Not. A excepción de los dos últimos todos poseen la siguiente sintaxis:

<expresión1> operador <expresión2>

En donde expresión1 y expresión2 son las condiciones a evaluar, el resultado de la operación varía en función del operador lógico. La tabla adjunta muestra los diferentes posibles resultados:

<expresión1>	Operador	<expresión2>	Resultado
Verdad	AND	Falso	Falso
Verdad	AND	Verdad	Verdad
Falso	AND	Verdad	Falso
Falso	AND	Falso	Falso
Verdad	OR	Falso	Verdad
Verdad	OR	Verdad	Verdad
Falso	OR	Verdad	Verdad
Falso	OR	Falso	Falso
Verdad	XOR	Verdad	Falso
Verdad	XOR	Falso	Verdad
Falso	XOR	Verdad	Verdad
Falso	XOR	Falso	Falso
Verdad	Eqv	Verdad	Verdad
Verdad	Eqv	Falso	Falso
Falso	Eqv	Verdad	Falso
Falso	Eqv	Falso	Verdad
Verdad	Imp	Verdad	Verdad
Verdad	Imp	Falso	Falso
Verdad	Imp	Null	Null
Falso	Imp	Verdad	Verdad
Falso	Imp	Falso	Verdad
Falso	Imp	Null	Verdad
Null	Imp	Verdad	Verdad
Null	Imp	Falso	Null
Null	Imp	Null	Null

Si a cualquiera de las anteriores condiciones le anteponemos el operador NOT el resultado de la operación será el contrario al devuelto sin el operador NOT.

El último operador denominado Is se emplea para comparar dos variables de tipo objeto <Objeto1> Is <Objeto2>. Este operador devuelve verdad si los dos objetos son iguales.

```
SELECT * FROM Empleados WHERE Edad > 25 AND Edad < 50;
```

```
SELECT * FROM Empleados WHERE (Edad > 25 AND Edad < 50)
OR Sueldo = 100; SELECT * FROM Empleados WHERE NOT
Estado = 'Soltero';
SELECT * FROM Empleados WHERE (Sueldo > 100 AND Sueldo <
500) OR (Provincia = 'Madrid' AND Estado = 'Casado');
```

4.2 INTERVALOS DE VALORES

Para indicar que deseamos recuperar los registros según el intervalo de valores de un campo emplearemos el operador Between cuya sintaxis es:

campo [Not] Between valor1 And valor2 (la condición Not es opcional)

En este caso la consulta devolvería los registros que contengan en "campo" un valor incluido en el intervalo valor1, valor2 (ambos inclusive). Si anteponemos la condición Not devolverá aquellos valores no incluidos en el intervalo.

```
SELECT * FROM Pedidos WHERE CodPostal Between 28000 And
28999;
```
(Devuelve los pedidos realizados).

```
SELECT IIf(CodPostal Between 28000 And 28999,
'Provincial', 'Nacional') FROM Editores;
```
(Devuelve el valor 'Provincial' si el código postal se encuentra en el intervalo, 'Nacional' en caso contrario).

4.3 EL OPERADOR LIKE

Se utiliza para comparar una expresión de cadena con un modelo en una expresión SQL. Su sintaxis es:

expresión Like modelo

En donde expresión es una cadena modelo o campo contra el que se compara expresión. Se puede utilizar el operador Like para encontrar valores en los campos que coincidan con el modelo especificado. Por modelo puede especificar un valor completo (Ana María), o se pueden utilizar caracteres comodín como los reconocidos por el sistema operativo para encontrar un rango de valores (Like An*).

El operador Like se puede utilizar en una expresión para comparar un valor de un campo con una expresión de cadena. Por ejemplo, si introduce Like C* en una consulta SQL, la consulta devuelve todos los valores de campo que comiencen por la letra C. En una consulta con parámetros, puede hacer que el usuario escriba el modelo que se va a utilizar.

El ejemplo siguiente devuelve los datos que comienzan con la letra P seguido de cualquier letra entre A y F y de tres dígitos:

```
Like 'P[A-F]###'
```

Este ejemplo devuelve los campos cuyo contenido empiece con una letra de la A a la D seguidas de cualquier cadena.

```
Like '[A-D]*'
```

En la tabla siguiente se muestra cómo utilizar el operador Like para comprobar expresiones con diferentes modelos.

Tipo de coincidencia	Modelo Planteado	Coincide	No coincide
Varios caracteres	'a*a'	'aa', 'aBa', 'aBBBa'	'aBC'
Carácter especial	'a[*]a'	'a*a'	'aaa'
Varios caracteres	'ab*'	'abcdefg', 'abc'	'cab', 'aab'
Un solo carácter	'a?a'	'aaa', 'a3a', 'aBa'	'aBBBa'
Un solo dígito	'a#a'	'a0a', 'a1a', 'a2a'	'aaa', 'a10a'
Rango de caracteres	'[a-z]'	'f', 'p', 'j'	'2', '&'
Fuera de un rango	'[!a-z]'	'9', '&', '%'	'b', 'a'
Distinto de un dígito	'[!0-9]'	'A', 'a', '&', '~'	'0', '1', '9'
Combinada	'a[!b-m]#'	'An9', 'az0', 'a99'	'abc', 'aj0'

4.4 EL OPERADOR IN

Este operador devuelve aquellos registros cuyo campo indicado coincide con alguno de los en una lista. Su sintaxis es:

expresión [Not] In(valor1, valor2, . . .)

```
SELECT * FROM Pedidos WHERE Provincia In ('Queretaro', 'San
Luis Potosi', 'Guanajuato');
```

4.5 LA CLÁUSULA WHERE

La cláusula WHERE puede usarse para determinar qué registros de las tablas enumeradas en la cláusula FROM aparecerán en los resultados de la instrucción SELECT. Si no se

emplea esta cláusula, la consulta devolverá todas las filas de la tabla. WHERE es opcional, pero cuando aparece debe ir a continuación de FROM.

```
SELECT Apellidos, Salario FROM Empleados WHERE Salario >
21000;

SELECT Id_Producto, Existencias FROM Productos
WHERE Existencias <= Nuevo_Pedido;

SELECT * FROM Pedidos WHERE Fecha_Envio = #5/10/94#;

SELECT Apellidos, Nombre FROM Empleados WHERE Apellidos
= 'King';

SELECT Apellidos, Nombre FROM Empleados WHERE Apellidos
Like 'S*';

SELECT Apellidos, Salario FROM Empleados WHERE Salario
Between 200 And
300;

SELECT Apellidos, Salario FROM Empl WHERE Apellidos
Between 'Lon' And 'Tol';

SELECT Id_Pedido, Fecha_Pedido FROM Pedidos WHERE
Fecha_Pedido
Between #1-1-94# And #30-6-94#;

SELECT Apellidos, Nombre, Ciudad FROM Empleados WHERE
Ciudad
In ('Mexico', 'Los Angeles', 'Venezuela');
```

5. AGRUPAMIENTO DE REGISTROS

5.1 LA CLÁUSULA GROUP BY

Combina los registros con valores idénticos, en la lista de campos especificados, en un único registro. Para cada registro se crea un valor sumario si se incluye una función SQL agregada, como por ejemplo Sum o Count, en la instrucción SELECT. Su sintaxis es: SELECT campos FROM tabla WHERE criterio GROUP BY campos del grupo GROUP BY es opcional. Los valores de resumen se omiten si no existe una función SQL agregada en la instrucción SELECT. Los valores Null en los campos GROUP BY se agrupan y no se omiten. No obstante, los valores Null no se evalúan en ninguna de las funciones SQL agregadas.

Se utiliza la cláusula WHERE para excluir aquellas filas que no desea agrupar, y la cláusula HAVING para filtrar los registros una vez agrupados. A menos que contenga un dato Memo u Objeto OLE , un campo de la lista de campos GROUP BY puede referirse a cualquier campo de las tablas que aparecen en la cláusula FROM, incluso si el campo no esta incluido en la instrucción SELECT, siempre y cuando la instrucción SELECT incluya al menos una función SQL agregada. Todos los campos de la lista de campos de SELECT deben o bien incluirse en la cláusula GROUP BY o como argumentos de una función SQL agregada.

```
SELECT Id_Familia, Sum(Stock)
FROM Productos
GROUP BY Id_Familia;
```

Una vez que GROUP BY ha combinado los registros, HAVING muestra cualquier registro agrupado por la cláusula GROUP BY que satisfaga las condiciones de la cláusula HAVING.

HAVING es similar a WHERE, determina qué registros se seleccionan. Una vez que los registros se han agrupado utilizando GROUP BY, HAVING determina cuales de ellos se van a mostrar.

```
SELECT Id_Familia Sum(Stock)
FROM Productos GROUP BY Id_Familia
HAVING Sum(Stock) > 100
AND NombreProducto Like BOS*;
```

5.2 AVG (MEDIA ARITMÉTICA)

Calcula la media aritmética de un conjunto de valores contenidos en un campo especificado de una consulta. Su sintaxis es la siguiente Avg(expr) En donde expr representa el campo que contiene los datos **numéricos** para los que se desea calcular la media o una expresión que realiza un cálculo utilizando los datos de dicho campo. La media calculada por Avg es la media aritmética (la suma de los valores dividido por el número de valores). La función Avg no incluye ningún campo Null en el cálculo.

```
SELECT Avg(Gastos) AS Promedio
FROM Pedidos
WHERE Gastos > 100;
```

5.3 COUNT (CONTAR REGISTROS)

Calcula el número de registros devueltos por una consulta. Su sintaxis es la siguiente Count(expr) En donde expr contiene el nombre del campo que desea contar. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

Puede contar cualquier tipo de datos incluso texto. Aunque expr puede realizar un cálculo sobre un campo, Count simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan en los registros. La función Count no cuenta los registros que tienen campos null a menos que expr sea el carácter comodín asterisco (*). Si utiliza un asterisco, Count calcula el número total de registros, incluyendo aquellos que contienen campos null. Count(*) es considerablemente más rápida que Count(Campo). No se debe poner el asterisco entre dobles comillas ('*').

```
SELECT Count(*) AS Total
FROM Pedidos;
```

Si expr identifica a múltiples campos, la función Count cuenta un registro sólo si al menos uno de los campos no es Null. Si todos los campos especificados son Null, no se cuenta el registro. Hay que separar los nombres de los campos con ampersand (&).

```
SELECT Count(FechaEnvío & Transporte) AS Total
FROM Pedidos;
```

5.4 MAX Y MIN (VALORES MÁXIMOS Y MÍNIMOS)

Devuelven el mínimo o el máximo de un conjunto de valores contenidos en un campo especifico de una consulta. Su sintaxis es:

Min(expr)
Max(expr)

En donde expr es el campo sobre el que se desea realizar el cálculo. Expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

```
SELECT Min(Gastos) AS ElMin
FROM Pedidos
WHERE Pais = 'Mexico';

SELECT Max(Gastos) AS ElMax
FROM Pedidos
WHERE Pais = 'Mexico';
```

5.5 STDEV Y STDEVP (DESVIACIÓN ESTÁNDAR)

Devuelve estimaciones de la desviación estándar para la población (el total de los registros de la tabla) o una muestra de la población representada (muestra aleatoria). Su sintaxis es:

StDev(expr)
StDevP(expr)

En donde expr representa el nombre del campo que contiene los datos que desean evaluarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL) StDevP evalúa una población, y StDev evalúa una muestra de la población. Si la consulta contiene menos de dos registros (o ningún registro para StDevP), estas funciones devuelven un valor Null (el cual indica que la desviación estándar no puede calcularse).

```
SELECT StDev(Gastos) AS Desviacion
FROM Pedidos
WHERE Pais = 'Mexico';

SELECT StDevP(Gastos) AS Desviacion
FROM Pedidos
WHERE Pais= 'Mexico';
```

5.6 SUM (SUMAR VALORES)

Devuelve la suma del conjunto de valores contenido en un campo especifico de una consulta. Su sintaxis es:

Sum(expr)

En donde expr respresenta el nombre del campo que contiene los datos que desean sumarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

```
SELECT Sum(PrecioUnidad * Cantidad) AS Total
FROM DetallePedido;
```

5.7 VAR Y VARP (VARIANZA)

Devuelve una estimación de la varianza de una población (sobre el total de los registros) o una muestra de la población (muestra aleatoria de registros) sobre los valores de un campo. Su sintaxis es:

Var(expr)
VarP(expr)

VarP evalúa una población, y Var evalúa una muestra de la población. Expr el nombre del campo que contiene los datos que desean evaluarse o una expresión que realiza un cálculo utilizando los datos de dichos campos.

Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL) Si la consulta contiene menos de dos registros, Var y VarP devuelven Null (esto indica que la varianza no puede calcularse). Puede utilizar Var y VarP en una expresión de consulta o en una Instrucción SQL.

```
SELECT Var(Gastos) AS Varianza
FROM Pedidos
WHERE Pais = 'Mexico';

SELECT VarP(Gastos) AS Varianza
FROM Pedidos
WHERE Pais = 'Mexico';
```

6. MANIPULACIÓN DE DATOS

Podemos manipular los datos almacenados de una manera flexible y segura porque PL/SQL soporta todos los comandos de manipulación de datos, control de transacciones, funciones y operadores manejados por SQL.

Sin embargo PL/SQL no soporta comandos de definición de datos tales como CREATE, o comandos de control del sistema tal como ALTER SYSTEM.

Existen comandos los cuales facilitan el manejo de la información dentro de una tabla, los comandos de uso mas frecuente son INSERT, UPDATE y DELETE, los cuales se presentan a continuación.

6.1 SENTENCIA INSERT

6.1.1 DEFINICIÓN

Sirve para agregar información en una tabla, especificando el lugar de origen de ese dato.

6.1.2 SINTAXIS

```
SQL> INSERT INTO table ({column [, column ... ]})
     VALUES (value [, value ...]);
```

6.1.3 EJEMPLO

```
INSERT INTO emp (empno, ename, job, sal, comm, deptno)

VALUES (7890, 'JINKS', 'CLERK', 1.2E3, NULL, 40);

INSERT INTO empleado_20 (empno, ename, job, sal, comm,
deptno)
            SELECT empno, ename, job, sal, comm, deptno
            FROM emp
            WHERE job = 'CLERK';
```

6.2 SENTENCIA UPDATE

6.2.1 DEFINICIÓN

Es usado éste comando para la actualización de los datos de la tabla.

6.2.2 SINTAXIS

```
SQL> UPDATE table
     SET column = value [, column = value, ...]
     [WHERE condition];
```

6.2.3 EJEMPLO

```
UPDATE emp
    SET comm = NULL
    WHERE job = 'TRAINEE';

UPDATE emp
    SET job = 'MANAGER', sal = sal + 1000, deptno = 20
    WHERE ename = 'JONES';
```

6.3 SENTENCIA DELETE

6.3.1 DEFINICIÓN

Tiene la función de eliminar datos de una tabla.

6.3.2 SINTAXIS

```
SQL> DELETE [FROM] table
     [WHERE condition];
```

6.3.3 EJEMPLO

```
DELETE FROM emp
    WHERE JOB = 'SALESMAN' AND COMM < 100;
```

7. CREACIÓN Y MANEJO DE TABLAS

Una tabla es un tipo de dato compuesto, donde el tamaño es ilimitado ya que puede ser incrementado de manera dinámica. Es la relación de datos, que conjunta despliega información con mayor detalle.

7.1 SENTENCIA CREATE TABLE

7.1.1 DEFINICIÓN

Sirve para crear una tabla, y definir los campos y sus características de los datos.

7.1.2 SINTAXIS

```
SQL> CREATE TABLE [schema.]table
     (column datatype [default exp] [, ...]);
```

7.1.3 EJEMPLOS

```
CREATE TABLE dept
(deptno    NUMBER (2),
 dname     VARCHAR2(10),
 loc       VARCHAR2(9) );
```

```
CREATE TABLE emp
(name VARCHAR(100),
 salary NUMBER,
 hiredate DATE);
```

Para confirmar la creación de la tabla utilice el comando "DESCRIBE" de el ambiente SQL* Plus: `SQL> DESCRIBE table`

7.1.4 CREANDO UNA TABLA UTILIZANDO UN SUBQUERY

También se pueden utilizar SUBQUERY para crear una tabla como se muestra a continuación.

```
CREATE TABLE dept
 [column, column ...]

AS
   Subquery;
```

7.1.4.1 EJEMPLO

```
CREATE TABLE dept_30
AS
  SELECT *
  FROM  dept
  WHERE deptno = 30;
```

7.2 SENTENCIA ALTER TABLE

7.2.1 DEFINICIÓN

Esta sentencia nos sirve para hacer modificaciones en la tabla.

7.2.2 SINTAXIS

```
SQL> ALTER [schema.]table
     ADD(column datatype [default exp] [, ...]);

SQL> ALTER [schema.]table
     MODIFY(column datatype [default exp] [, ...]);

SQL> ALTER [schema.]table
     DROP COLUMN column);
```

7.2.3 EJEMPLO

```
ALTER TABLE dept_30
   ADD (job VARCHAR2(9));

ALTER TABLE dept_30
   MODIFY (job VARCHAR2(15));

ALTER TABLE dept_30
DROP COLUMN job);
```

7.3 SENTENCIA DROP TABLE

7.3.1 DEFINICIÓN

Esta sentencia nos sirve para dar de baja o eliminar una tabla.

7.3.2 SINTAXIS

```
SQL> DROP TABLE table;
```

7.3.3 EJEMPLO

```
DROP TABLE dept;
```

8. CONSTRAINTS

8.1 DEFINICIÓN

Constraints son reglas de validación creadas a nivel de tabla. Los contraints previenen que se borren instancias de una tabla la cual tiene dependencias.

8.2 TIPOS

En Oracle son válidos los siguientes tipos de constraints:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK

8.3 SINTAXIS

```
SQL> CREATE TABLE [schema.]table
     (column datatype [default exp]
      [column_constraint],
      ...
      [table_constraint][,...]);
```

8.4 EJEMPLO

```
CREATE TABLE dept

(deptno    NUMBER (2), CONSTRAINT deptno_ck
                       CHECK (DEPTNO BETWEEN 10 AND 99),
  dname     VARCHAR2(10),
  loc       VARCHAR2(9),
 CONSTRAINT dept_dname_uk UNIQUE (dname),
 CONSTRAINT dept_dname_pk PRIMARY KEY (deptno));


CREATE TABLE emp
 (empno NUMBER(4),
  name VARCHAR(100),
  salary NUMBER(7,2),
  hiredate DATE
  deptno NUMBER(2) NOT NULL,
  CONSTRAINT emp_deptno_fk FOREIGN KEY (deptno) REFERENCES
dept (deptno)
                                    [ON DELETE CASCADE] );
```

8.5 PARA AÑADIR O BORRAR UN CONSTRAINT EN UNA TABLA

Si la tabla previamente ha sido creada y por alguna razón necesitamos añadir o borrar algún constraint asociado a una tabla, podemos utilizar la sentencia ALTER TABLE. Es importante resaltar que NO podemos modificar un Constraint, solo añadir o borrar.

Para añadir un constraint de NOT NULL a un campo en particular, utilizamos la sentencia ALTER TABLE pero con la cláusula MODIFY

```
SQL> ALTER TABLE table
     ADD [CONSTRAINT constraint] type (column);

SQL> ALTER TABLE table


      DROP [CONSTRAINT constraint]);

SQL> ALTER TABLE table
     MODIFY [CONSTRAINT constraint] type (column) NOT NULL);
```

8.5.1 EJEMPLO

```
SQL> ALTER TABLE emp
 2    ADD CONSTRAINT emp_empno_pk PRIMARY KEY (empno);
Table altered.

SQL> ALTER TABLE dept
 2   DROP CONSTRAINT dept_deptno_pk;
Table altered.
```

Se puede utilizar la tabla USER_CONSTRAINTS que provee el diccionario de datos de Oracle para visualizar todos los constraints definidos.

9. PL/SQL

9.1 ACERCA DE PL/SQL Y SU ESTRUCTURA

PL/SQL (lenguaje procedural) es una extensión de SQL con características de lenguaje programación. PL/SQL no es más que un lenguaje propietario de Oracle Corporation, el cual nos permite realizar bloques de códigos de programación donde a su vez se puede manipular los datos de la BD y realizar consultas utilizando el lenguaje SQL.

9.2 ¿CÓMO ESCRIBIR UN BLOQUE PL?

```
* DECLARE - Optional
   Variables, cursors, user-defined exceptions
* BEGIN - Mandatory
   - SQl statements
   - PL/SQL statements
* EXCEPTION - Optional
   Actions to perform when errors occur
* END; - Mandatory
```

Para poner un nombre al bloque se agrega una etiqueta antes de la palabra DECLARE:

```
<< etiqueta1 >>
DECLARE
   seq number;
BEGIN
    select rubros_seq.nextval into seq from dual;
```

```
        insert into rubros(clave, nombre)
        values(seq, 'Utensilio');
    END; etiqueta1
```

9.2.1 TIPOS DE BLOQUES PL/SQL

Un bloque PL/SQL es la unidad básica de todo programa. Todos los programas PL/SQL están compuestos por bloques, los cuales pueden ser secuenciales o anidados.

Los tipos de bloques son:

- Bloques anónimos: Se construyen generalmente dinámicamente y sólo se ejecutan una vez.
- Bloques con nombre: Son como los bloques anónimos, pero tienen una etiqueta que le da al bloque un nombre.
- Subprogramas : Son procedimientos, funciones y paquetes que son almacenados en base de datos y se ejecutan cuando son invocados.
- Triggers: Son bloques con nombre que son almacenados en la base de datos. Son ejecutados implícitamente cuando cierto evento ocurre.

Se muestran a continuación en la tabla 9.1 de una forma general, la estructura de los anónimos, los procedimientos y las funciones, más adelante mencionaré los triggers, procedimientos y funciones con más detalle.

ANONYMOUS	PROCEDURE	FUNCTION

DECLARE
 	
BEGIN
--statements
 	
EXCEPTION
END;
	PROCEDURE name
	IS
	BEGIN
	-- statements
	
	EXCEPTION
	END;
		FUNTION name
		RETURN datatype
		IS
		BEGIN
		-- statements
		RETURN value;
		EXCEPTION
		END;

Tabla 9.1 Bloques

9.3 TIPOS DE VARIABLES Y SU DECLARACIÓN

9.3.1 VARIABLES PL/SQL

- Scalar (string, booleano, numérico, etc)
- Compuesto (registros, variables de tipo tabla, etc)
- Referencia
- LOB (objetos largos, con capacidad hasta 4GB)

Las variables PL/SQL deben ser declaradas dentro del bloque PL en la sección de "DECLARE" de la siguiente manera:

```
DECLARE
   v_deptno          NUMBER(2) := 10;
   v_ubicacion       VARCHAR2(13):= 'Atlanta';
   v_dname           DEPT.DNAME%TYPE;
```

Todas las definiciones de variables pl/sql deben terminar en (;).

9.3.2 TIPOS PL/SQL

PL/SQL tiene tres categorías de tipos: escalar, compuesto y referencias. Después se definen dos categorías adicionales: los tipos LOB (Large Objects, objetos de gram tamaño) y los tipos de objetos. Los tipos escalares no tienen componentes, mientras que los tipos compuestos sí los tienen. Las referencias, por su parte, son punteros a otros tipos. En la figura 9.1 se muestran todos los tipos PL/SQL.

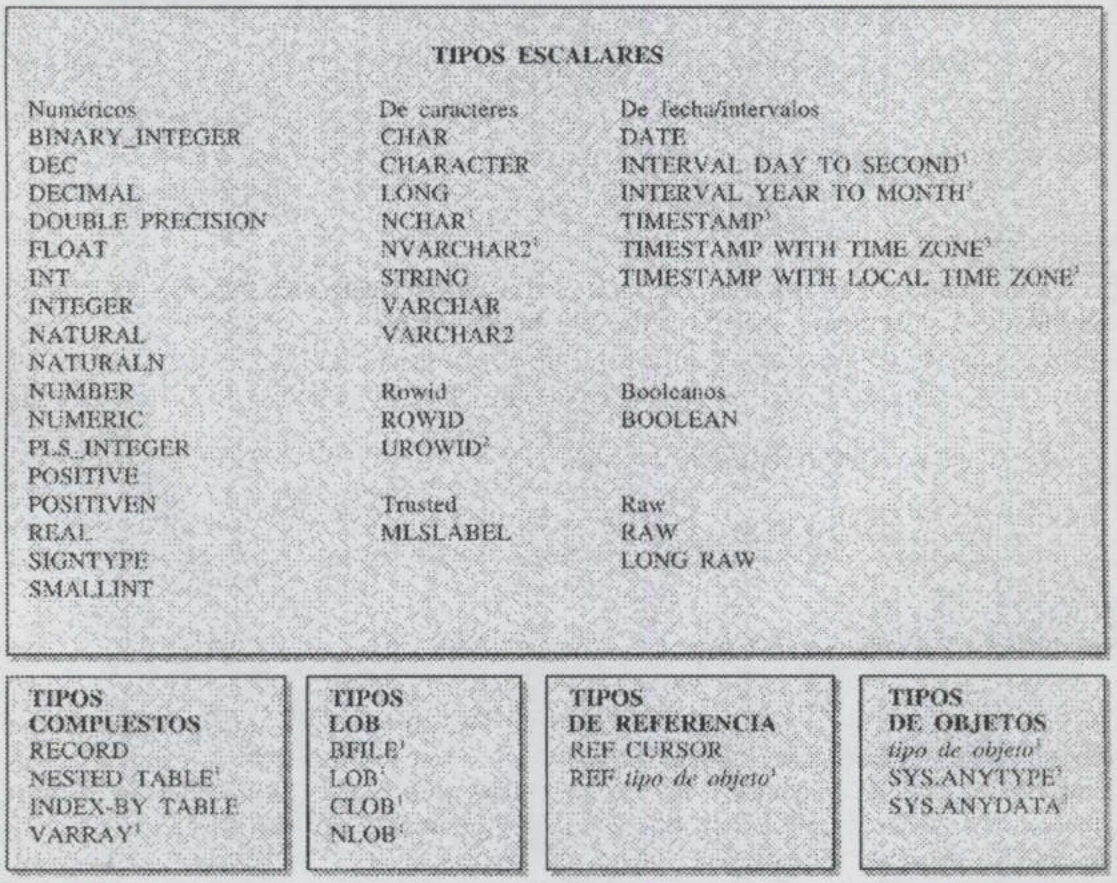


Figura 9.1 Tipos de PL/SQL

Los tipos PL/SQL predefinidos se definen en un paquete denominado STANDARD, cuyos contenidos son accesibles desde cualquier bloque PL/SQL. Además de los tipos, este paquete define las funciones predefinidas SQL y de conversión disponibles en PL/SQL.

TIPOS ESCALARES

Los tipos escalares válidos son los mismos tipos que pueden usarse en una columna de base de datos, más algunos tipos adicionales. Existen siete familias de tipos escalares: numéricos, de carácter, raw, de fecha/intervalo, rowid, voléanos y trusted.

Tipos numéricos

Los tipos de esta familia almacenan valores enteros o reales. Existen tres tipos básicos: NUMBER, PLS_INTEGER y BINARY_INTEGER. Las variables de tipo NUMBER pueden contener valores enteros o reales, mientras que las de los tipos BINARY_INTEGER o PLS_INTEGER sólo pueden contener valores enteros.

NUMBER

Este tipo puede contener un valor numérico entero o de punto flotante. Es igual al tipo NUMBER de la base de datos. La sintaxis para la declaración de una variable de tipo NUMBER es:

- NUMBER (P,S)

Donde P es la operación, y S la escala. La precisión es el número de dígitos en el valor, y la escala es el número de dígitos a la derecha del punto decimal. La escala puede ser negativa, lo que indica que el valor se redondea al número especificado de lugares a la izquierda del punto decimal. Tanto la precisión como la escala son opcionales pero, si se incluye la escala, hay que incluir la precisión también. La tabla 9.2 muestra diversas combinaciones de precisión y escala, junto con su significado.

Declaración	*Valor asignado*	*Valor almacenado*
NUMBER	1234.5678	1234.5678
NUMBER(3)	123	123
NUMBER(3)	1234	Error: excede la precisión
NUMBER(4,3)	123.4567	Error: excede la precisión
NUMBER(4,3)	1.234567	1.235[1]
NUMBER(7,2)	12345.67	12345.67
NUMBER(3,-3)	1234	1000[2]
NUMBER(3, -1)	1234	1230[2]

Tabla 9.2 Valores de Precisión y Escala

La precisión máxima es 38, y la escala puede variar entre -84 y 127.

Un subtipo es un nombre alternativo para un tipo, que puede, adicionalmente, restringir el conjunto de valores legales para una variable de dicho subtipo. Hay una serie de subtipos equivalentes a NUMBER que lo hacen, esencialmente, es renombrar dicho tipo NUMBER, dado que ninguno de ellos introduce restricciones. Pueden utilizarse estos nombres alternativos para facilitar la lectura de un programa, o por cuestiones de compatibilidad con los tipos de datos de otras bases de datos. Los tipos equivalentes son:

- DEC
- DECIMAL
- DOUBLE PRECISION
- FLOAT
- NUMERIC
- REAL

Entre los tipos que pueden almacenar números en punto flotante, DEC, DECIMAL y NUMERIC tienen la precisión máxima, es decir, 38 dígitos decimales. DOUBLE PRECISION y FLOAT tienen una precisión de 126 dígitos binarios, lo que equivale aproximadamente a 38 dígitos decimales. El tipo REAL tiene una precisión de 63 dígitos binarios, es decir, aproximadamente 18 dígitos decimales.

Además de los anteriores subtipos no restringidos, los subtipos INTEGER, INT y SMALLINT pueden almacenar valores enteros con una precisión máxima de 38 dígitos decimales.

BINARY_INTEGER

El tipo NUMBER se almacena en formato decimal, optimizado en cuanto a precisión y eficiencia de almacenamiento. Debido a esto, no es posible realizar operaciones aritméticas de forma directa con el tipo NUMBER. Para poder realizar cálculos utilizando magnitudes numéricas, los valores de tipo NUMBER deben ser convertidos a un tipo binario.
El motor PL/SQL realiza esta conversión de manera automática cuando encuentra valores de tipo NUMBER en una expresión aritmética, y también convierte el resultado a tipo NUMBER cuando sea necesario.

Sin embargo, para valores que sólo se usen durante los cálculos, y no vayan a ser almacenados en la base de datos, se pueden utilizar el tipo BINARY_INTEGER. Este tipo permite almacenar valores enteros con signo en el rango -2147483647 a +2147483647.

Los valores se almacenan en formato binario con complemento a 2 (generalmente, el formato nativo de los números enteros), lo que permite utilizarlos en los cálculos sin necesidad de efectuar ningún tipo de conversión. Los contadores de bucles son, a menudo, del tipo BINARY_INTEGER.

Al igual que sucede con NUMBER, existe una serie de subtipos definidos para BINARY_INTEGER. Sin embargo, a diferencia de la mayoría de subtipos de NUMBER, los subtipos de BINARY_INTEGER sí están restringidos, lo que significa que sólo pueden conectar valores dentro de un rango limitado. Estos subtipos restringidos de BINARY_INTEGER se muestran en la Tabla 9.3.

Subtipo	*Restricción*
NATURAL	0 ... 2147483647
NATURALN	0 ... 2147483647 NOT NULL
POSITIVE	1 ... 2147483647
POSITIVEN	1 ... 2147483647 NOT NULL
SIGNTYPE	-1, 0, 1

Tabla 9.3 Subtipos de BINARY_INTEGER

PLS_INTEGER

El rango de los valores PLS_INTEGER es el mismo que el de los BINARY_INTEGER, es decir, desde -2147483647 a +2147483647, y también se almacenan utilizando el formato en complemento a 2 nativo. Sin embargo, cuando se produce un desbordamiento en un cálculo en el que esté involucrado un valor PLS_INTEGER, se genera un error.

Esto no sucede con los valores del tipo BINARY_INTEGER, con los que, si se produce un desbordamiento, el resultado puede ser asignado sin generar ningún error a una variable de tipo NUMBER (que tiene un rango mayor).

Tipos de carácter

Las variables de tipo carácter permiten almacenar cadenas o datos de tipo carácter. Esta familia de tipos está compuesta por VARCHAR2, CHAR y LONG, junto con NCHAR y NVARCHAR2.

VARCHAR2

Este tipo de se comporta de forma similar al tipo VARCHAR2 de la base de datos. Las variables de tipo VARCHAR2 pueden contener cadenas de caracteres de longitud variable, con una longitud máxima especificada. La sintaxis para la declaración de una variable VARCHAR2 es:

- VARCHAR2(L)

Donde L es la longitud máxima de la variable. Es imprescindible indicar la longitud máxima, no existiendo un valor predeterminado. La longitud máxima que puede definirse para una variable VARCHAR2 es de 32.767 bytes.

La longitud de una variable VARCHAR2 se especifica en bytes, no en caracteres. El valor en cuestión se almacena en el conjunto de caracteres de la base de datos, que podría ser un conjunto de caracteres de un solo byte, como ASCII o EBCDIC Code Page 500, o un conjunto de caracteres multibyte de longitud variable, como Unicote. Si el conjunto de caracteres de la base de datos contiene caracteres multubyte, le máximo número de caracteres que una variable VARCHAR2 puede contener pudiera ser menor que la longitud especificada, debido a que se puede necesitar más un byte para representar un solo carácter. Los subtipos VARCHAR y STRING son equivalentes a VARCHAR2.

TIPOS COMPUESTOS

Los tipos compuestos disponibles en PL/SQL son: registros, tablas y varrays. Un tipo compuesto es aquel que consta de una serie de componentes. Una variable de tipo compuesto contendrá una o más variables escalares (conocidas también como atributos).

TIPOS REFERENCIA

Una vez que se declara en PL/SQL una variable de tipo escalar o compuesto, se asigna el espacio de memoria de dicha variable. La variable se emplea para dar nombre a este espacio de memoria, y para hacer referencia a él en el programa. Sin embargo, no hay ninguna forma de desasignar este espacio de memoria y que la variable continúe estando disponible; la memoria no se libera hasta que no se sale de ámbito de la variable. Los tipos de referencia, que son el equivalente PL/SQL de un puntero en C, no tienen esa restricción. Cuando se declara una variable de un tipo de referencia, ésta puede apuntar a diferentes posiciones de memoria durante la vida del programa.

TIPOS LOB

Los tipos LOB se emplean para almacenar objetos de gran tamaño. Un objeto de gran tamaño es un valor binario o de tipo carácter con un tamaño de hasta 4 Gigabytes. Los objetos de gran tamaño pueden contener datos no estructurados, a los que se puede acceder de forma más eficiente, y con menos restricciones, que a los datos de tipo LONG o LONG RAW.

TIPOS DE OBJETO

Un tipo de objeto es un tipo compuesto que contiene atributos (variables de otros tipos) y métodos (subprogramas).

Se puede observar los siguientes aspectos en lo referente a los tipos de objeto:

- Los tipos de objeto son similares, en cuanto a sintaxis, a los paquetes, en el sentido de que ambos constan de una cabecera y un cuerpo. Como ocurre con los paquetes, el cuerpo depende de la cabecera.
- Los atributos, se declaran del mismo modo que las variables PL/SQL. Los métodos, como ToString, Distance, Plus y Times, se declaran de forma similar a los subprogramas de PL/SQL, con la excepción de que tienen la palabra clave MEMBER.

- Al igual que con un registro PL/SQL, se pueden hacer referencia a los atributos de un objeto utilizando la notación de punto.

9.3.3 VARIABLES NO –PL/SQL

Son variables de ambiente o propietarias de la aplicación que se esté utilizando. Por ejemplo las variables de ambiente SQL*Plus o las de Form applications, etc. Las variables de ambiente se definen fuera del bloque PL antes del bloque de declaración (DECLARE) utilizando la siguiente sintaxis:

- `VARIABLE v_retorno NUMBER`
- `VARIABLE v_salida VARCHAR2(30)`

Las definiciones de variables de ambiente "no" terminan en (;). Además por cada variable debe existir la palabra clave "variable". Por ultimo para referenciar a una variable de ambiente se debe colocar (:)
Adelante del nombre de la variable.

- `: v_retorno:= 10;`

9.4 ESTRUCTURAS DE CONTROL

9.4.1 ESTRUCTURA CONDICIONAL IF

```
IF (expresión) THEN
 Instrucciones

ELSEIF
```

```
    -- Instrucciones
ELSE
    -- Instrucciones
END IF;
```

9.4.2 ESTRUCTURA DE REPETICIÓN: LOOP - EXIT Y WHILE

```
LOOP
    -- Instrucciones
END LOOP;

LOOP
    -- Instrucciones
    IF (expresion) THEN
      -- Instrucciones
      EXIT;
    END IF;
END LOOP;

WHILE (expresion) LOOP
    -- Instrucciones
END LOOP;
```

9.4.3 ESTRUCTURA DE REPETICIÓN: FOR - LOOP

```
FOR contador IN [REVERSE] inicio..final LOOP
          -- Instrucciones
END LOOP;
```

9.5 PROCEDURES

9.5.1 DEFINICIÓN

Un PROCEDURE es un bloque PL/SQL con identificación propia que ejecuta una acción determinada. Un procedimiento es un objeto de la BD, el cual es ejecutado en cualquier momento.

¿Cómo crear un Stored Procedure Usando SQL*Plus?

1. Levante el editor de SQL*Plus y escriba el Procedimiento.
2. Ejecute el script almacenado con extensión .sql desde el SQL *Plus (utilice la instrucción "start *nombre del script*").
3. Utilice el commando SHOW ERRORS para visualizar los errores de complilación.
4. Una vez compilado el procedimiento está listo para su ejecución (utilice el comando EXEC nombre del *procedure*.

9.5.2 SINTAXIS

```
SQL> CREATE [OR REPLACE] PROCEDURE procedure_name
     [(parameter1 [mode1] datatype,
       parameter2 [mode2] datatype,
       . . .)]
      IS/AS
      PL/SQL Block;
```

9.5.3 EJEMPLO

```
CREATE [OR REPLACE] PROCEDURE incluir_dept
     [v_deptno IN dept.deptno%TYPE,
      v_name   IN dept.dname%TYPE,
      v_loc    IN varchar2 (50))

      IS
      BEGIN
        INSERT INTO dept
         VALUES (v_deptno, v_name, v_loc);

      END add_dept;

CREATE PROCEDURE credit (acc_no IN NUMBER, cantidad IN
NUMBER)
AS
   BEGIN
      UPDATE accounts
      SET balance = balance + amount
      WHERE account_id = acc_no;
   END;
```

Para borrar un procedimiento utilice la instruccion: DROP *procedure*

9.6 FUNCIONES

9.6.1 DEFINICIÓN

Una función es un conjunto de instrucciones en PL/SQL, que pueden ser llamados usando el nombre con que se le haya creado. Se diferencian de los procedimientos, en que las funciones retornan un valor al ambiente desde donde fueron llamadas.

9.6.2 SINTAXIS

La sintaxis para crear una función es la siguiente:

```
CREATE [OR REPLACE] FUNCTION name
[(param [IN] datatype) . . .]
RETURN datatype
[IS|AS] pl/sql_subprogram
```

El uso de OR REPLACE permite sobrescribir una función existente. Si se omite, y la función ya existe, se producirá, un error. El único modificador permitido para los parámetros es IN, y si se omite, se tomará por defecto. Es decir, solo se permiten parámetros de entrada.

9.6.3 EJEMPLO

A continuación se presenta un ejemplo de creación de una función:

```
SQL> CREATE FUNCTION get_bal (acc_no IN NUMBER)
  1>    RETURN NUMBER

  2>    IS
  3>    acc_bal NUMBER(11,2);  /* declaración de una variable
*/
  4>  BEGIN
  5>       SELECT balance
  6>        INTO acc_bal        /* asignación */
  7>        FROM accounts
  8>        WHERE account_id = acc_no;
  9>   RETURN(acc_bal);
 10>  END
```

La función *get_bal* retorna el balance de una cuenta dada.

Si se desea eliminar (borrar) una función, se usa la instrucción:

```
SQL> DROP FUNCTION name;
```

9.7 TRIGGERS

9.7.1 DEFINICIÓN

Un trigger (traducción: disparador, desencadenador) es un bloque PL/SQL que se activa implícitamente cuando ocurre un evento (INSERT, UPDATE o DELETE) en una tabla o vista en particular. Existen triggers a nivel de la base de datos como triggers de aplicación.

A nivel de optimización de la base de datos, los triggers son muy útiles, ya que ellos nos ayudan a implementar, reforzar y mantener la integridad referencial, la seguridad de la data, ayudan a manejar problemas de replicación y activación de eventos explícitamente y además nos ayuda a reforzar restricciones complejas y específicas que no son cubiertas por las reglas que nos provee el manejador.

Otras aplicaciones de los TRIGGERS pueden ser las siguientes:

- Generación automática de valores derivados de una columna.
- Prevenir transacciones inválidas.
- Proporcionar auditorias sofisticadas.
- Mantener la sincronía en tablas replicadas.
- Generar estadísticas de acceso.
- Modificar los valores de una vista.
- Publicar información de los eventos generados por la BD, las actividades de los usuarios o de las estructuras SQL que se ha ejecutado.

9.7.2 SINTAXIS

```
SQL> CREATE [OR REPLACE] TRIGGER trigger_name
       timing
          event1 [OR event2 OR event3]
              ON table_name
       Trigger_body;
```

Creando un Trigger por cada fila

```
SQL> CREATE [OR REPLACE] TRIGGER trigger_name

      timing
        event1 [OR event2 OR event3]
            ON table_name
       [REFERENCING OLD as old / NEW as new]
    FOR EACH ROW
      [WHEN condition]
    trigger_body;
```

En la siguiente tabla (9.4 Triggers) muestra los diferentes tipos de manejo de triggers.

Trigger_name	*Es el nombre del trigger*
Timming	Indica cual es el momento en que se disparará el trigger BEFORE AFTER
Event	Identifica la operación DML que causa el levantamiento del trigger: INSERT UPDATE [OF column] DELETE
table_name	Indica la tabla a la cual se le asocia el trigger
REFERENCING	Especifica la correlación entre los valores viejos y nuevos que tiene una fila en particular. (por defectos son OLD y NEW)
FOR EACH ROW	Indica que el trigger será por cada fila de la tabla.

WHEN	Especifica una condición del trigger (Es un predicado condicional que es evaluado por cada fila para determinar si el cuerpo del trigger es ejecutado o no.
Trigger_body	Indica la acción a realizar una vez que el trigger se dispara.

Tabla 9.4 Triggers

Para referenciar los valores de OLD y NEW deben ser antecedidos por (:). No se colocan cuando éstos son referenciados en la cláusula WHEN.

9.7.3 EJEMPLO

```
SQL> CREATE OR REPLACE TRIGGER RESTRICCION_SALARIO
     BEFORE INSERT OR UPDATE OF sal ON emp

     FOR EACH ROW
          BEGIN
          IF NOT (:NEW.JOB IN ('MANAGER', 'PRESIDENT'))
             AND :NEW.SAL > 5000
          THEN
             RAISE_APPLICATION_ERROR
               (-20202, 'EMPLOYEE CANNOT EARN THIS AMOUNT');
          END IF;
          END;

SQL> CREATE TRIGGER scott.salary_check
     BEFORE INSERT OR UPDATE OF sal, job ON scott.emp
     FOR EACH ROW
     WHEN (new.job <> 'PRESIDENT')
     CALL check_sal(:new.job, :new.sal, :new.ename);
```

Para recompilar un trigger utilice la siguiente instrucción:

```
SQL> ALTER TRIGGER trigger_name COMPILE
```

Para borrar un trigger utilice la siguiente instrucción:

```
SQL> DROP TRIGGER trigger_name
```

9.8 CURSORES PL/SQL

El área de contexto es la memoria designada para procesar una instrucción SQL, la cual incluye:

- El número de registros procesados,
- Un apuntador a la representación de la instrucción SQL analizada y,
- En el caso de una consulta, el conjunto de registros que regresan de la consulta.

9.8.1 DEFINICIÓN

Un *cursor* es un manejador o apuntador para el área de contexto. Por medio de éste un programa PL/SQL puede controlar el área de contexto.

Los *cursores implícitos* son creados por Oracle para manejar alguna instrucción SQL y no son declarados por el programador.

Los *cursores explícitos* son aquellos que se declaran, generalmente por medio de una consulta SQL. En la figura 9.2 se muestra las fases del proceso en una instrucción SQL.

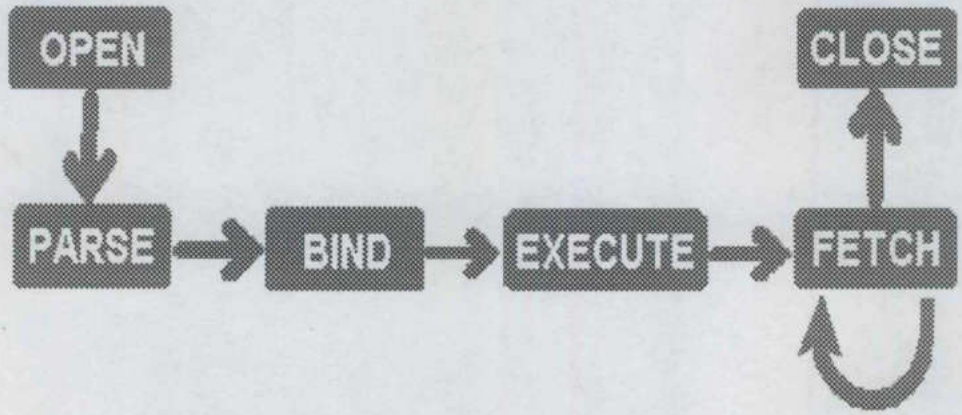


Figura 9.2 Fases para procesar una instrucción SQL

9.8.2 PASOS PARA PROCESAR UN CURSOR EXPLÍCITO

1. Declaración del cursor

   - La consulta no debe contener la cláusula INTO.
   - Se puede hacer referencia a variables dentro de la cláusula WHERE.

   CURSOR *nombre_cursor* IS
   instrucción_SELECT

   También es posible declarar una lista de parámetros:

   CURSOR *nombre_cursor(param1 tipo1, ..., paramN tipoN)* IS
   instrucción_SELECT

2. Abrir el cursor

   - La sintaxis es:

     OPEN nombre_cursor;

   - Se examinan los valores de las variables

   - Se determina el conjunto activo

   - El apuntador para el conjunto activo se establece en el primer registro

3. Recuperar los resultados en variables PL/SQL

   - Tiene dos formas:

     ```
     FETCH cursor_name INTO lista_variables;
     ```

     o

     ```
     FECTH cursor_name INTO registro_PL/SQL;
     ```

4. Cerrar el cursor

   - La sintaxis es:

     ```
     CLOSE nombre_cursor;
     ```

   - Cuando se cierra el cursor, es ilegal tratar de usarlo

   - Es ilegal tratar de cerrar un cursor que ya está cerrado o no ha sido abierto

9.8.3 ATRIBUTOS DE CURSORES EXPLÍCITOS

Toman los valores TRUE, FALSE o NULL dependiendo de la situación:

Atributo	Antes de abrir	Al abrir	Durante la recuperación	Al finalizar la recuperación	Después de cerrar
%NOTFOUND	ORA-1001	NULL	FALSE	TRUE	ORA-1001
%FOUND	ORA-1001	NULL	TRUE	FALSE	ORA-1001
%ISOPEN	FALSE	TRUE	TRUE	TRUE	FALSE
%ROWCOUNT	ORA-1001	0	*	**	ORA-1001

Tabla 9.5 Atributos de los cursores

9.8.4 MANEJO DEL CURSOR POR MEDIO DE CICLOS

Por medio de ciclo LOOP. Debe tenerse cuidado de agregar una condición para salir del ciclo:

```
OPEN nombre_cursor;
LOOP
    FECTH nombre_cursor INTO lista_variables;
    EXIT WHEN nombre_cursor%notfound;
END LOOP;
CLOSE nombre_cursor;
```

Por medio de un ciclo WHILE LOOP. La instrucción FECTH aparece dos veces.

```
OPEN nombre_cursor;
FECTH nombre_cursor INTO lista_variables;
WHILE nombre_cursor%found LOOP
    /* Procesamiento de los registros recuperados */
    FECTH nombre_cursor INTO lista_variables;
END LOOP;
CLOSE nombre_cursor;
```

Por medio de un ciclo FOR LOOP. Es la forma más corta ya que el cursor es implícitamente se ejecutan las instrucciones OPEN, FECTH y CLOSE.

```
FOR variable IN nombre_cursor LOOP
    /* Procesamiento de los registros recuperados */
END LOOP;
```

9.8.4.1 EJEMPLO

```
create or replace package EJEM_CURSOR is
    procedure imprime_facturas;
end;

create or replace package body EJEM_CURSOR is
--
--  Este procedimiento genera un listado de proveedores
--  y las facturas que ha emitido
--
    procedure IMPRIME_FACTURAS is
```

```
    cve_prov        proveedores.clave%type;
    nom_prov        proveedores.nombre%type;

    CURSOR c_proveedores IS
        select clave, nombre from proveedores;
    CURSOR c_facturas IS
        select factura, fecha from facturas where
        proveedor_clave=cve_prov order by fecha;
    fact  c_facturas%rowtype;
begin
     OPEN c_proveedores;
   loop
       FETCH c_proveedores INTO cve_prov, nom_prov;
       exit when c_proveedores%notfound;
       dbms_output.put_line('PROVEEDOR: ' || nom_prov);
       for fact in c_facturas loop
           dbms_output.put_line('           ' ||
         fact.factura || '  (' || fact.fecha || ')');
       end loop;
       dbms_output.put_line(' ');
   end loop;
   CLOSE c_proveedores;
end;
end;
```

CONCLUSIONES

Con la idea de facilitarnos las tareas que debemos de desempeñar los humanos, hemos venido inventado diversas herramientas a lo largo de nuestra historia, que nos permiten tener una mejor calidad de vida.

Las computadoras son uno más de los inventos del hombre, aunque debemos decir que las tecnologías para su fabricación y explotación han tenido un desarrollo sorprendente a partir de la segunda mitad del siglo XX. Esta herramienta por sí sola no es capaz de efectuar ninguna tarea, es tan sólo un conjunto de cables y circuitos que necesitan recibir instrucción por parte de los humanos para desempeñar alguna tarea. El problema entonces, se puede fijar en ¿cómo vamos a poder hacer que un conjunto de circuitos desempeñen una determinada tarea y nos entreguen los resultados que nosotros esperamos?, es decir, ¿de qué manera se puede lograr la comunicación entre el hombre y la computadora?.

Así pues, tratando de dar una solución al problema planteado, surgieron los lenguajes de programación, que son como un lenguaje cualquiera, pero simplificado y con ciertas normas, para poder trasmitir nuestros deseos al ordenador.

El Sistema de Gestión de Bases de Datos (SGBD) Consiste en un conjunto de programas, procedimientos y lenguajes que nos proporcionan las herramientas necesarias para trabajar con una base de datos. Incorporar una serie de funciones que nos permita definir los registros, sus campos, sus relaciones, insertar, suprimir, modificar y consultar los datos.

Por medio de este trabajo pongo en sus manos información de carácter actual, con la cual en estos dias se estan desarrolando multiples aplicaciones, debido a las caracteristicas de las aplicaciones que nos brinda Oracle. Es el manejador de base de datos relacional que hace uso de los recursos del sistema en todas las arquitecturas de hardware, para garantizar su aprovechamiento al máximo en ambientes cargados de información.

Independientemente de si el lector del trabajo presentado es un administrador de bases de datos con experiencia, o sin ella, o un desarrollador de aplicaciones, es necesario conocer cómo funcionan e interactúan las estructuras internas de las bases de datos. Si se gestionan correctamente todos los detalles de la base de datos, se conseguirán dos objetivos: que funcione y que funcione bien. Este trabajo de investigación mostró gran cantidad de información necesaria para conseguir ambos objetivos.

BIBLIOGRAFÍA

En libros:

- Koch, George, *ORACLE 7 Manual de Referencia* Osborne/McGraw-Hill, 1994.

- Koch, George, *ORACLE Manual de Referencia*. Osborne/McGraw-Hill, 1992.

- Cronin, Daniel, *Mastering Oracle*. *Hayden Books*, 1990.

- Oracle 9i Programación PL/SQL Scout Urman, 2002
  McGraw-Hill

En internet:

- Heriberto Zavaleta Morales, Joaquín Peña Acevedo, *Programación en PL/SQL,*
  http://www.lania.mx/biblioteca/seminarios/basedatos/plsql/index.html#triggers, 10/03/2004

- *Introducción a sistemas ORACLE, http://www.bd.cesma.usb.ve/ci3391/manual/,* 10/03/2004