44

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

"A R A G O N"

INGENIERIA EN COMPUTACION

"AFINACION DE APLICACIONES EN SQL * PLUS"

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A :

MARIA DE JESUS MEJIA ROLDAN

ASESOR DE TESIS:
ING. JOSE JUAN RAMON MEJIA ROLDAN

MEXICO, D. F. JUNIO 1996

VNIVERSIDAD NACIONAL
AVFNOMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
CAMPUS ARAGON
JEFATURA DE CARRERA DE INGENIERIA EN COMPUTACION
ICOM./122/96.

LIC. ALBERTO IBARRA ROSAS
JEFE DE LA UNIDAD ACADEMICA
P R E S E N T E .

Por este conducto me permito presentar a usted, nombres de los Profesores que sugiero integren el Sínodo del Exámen Profesional de la alumna, ***MARIA DE JESUS MEJIA ROLDAN***, que presenta el tema de tesis: ***"AFINACION DE APLICACIONES EN SQL-PIUS"***

PRESIDENTE:	***ING. ROBERTO BLANCO BAUTISTA***
VOCAL:	***ING. JOSE JUAN RAMON MEJIA ROLDAN***
SECRETARIO:	***ING. SILVIA VEGA MUTTOY***
SUPLENTE:	***ING. DAVID MOISES TERAN PEREZ***
SUPLENTE:	***ING. ERNESTO PEÑALOZA ROMERO***

Quiero subrayar que el director de tesis es el ***Ing. José Juan Ramón Mejía Roldán,*** el cuál está incluído en base a lo que reza el reglamento de Exámenes Profesionales de esta Escuela.

Sin otro particular, aprovecho la ocasión para enviarle un cordial saludo.

A T E N T A M E N T E
"POR MI RAZA HABLARA EL ESPIRITU"
San Juan de Aragón, Edo. de Méx., [illegible] de 1996.
LA JEFA DE CARRERA

ING. SILVIA VEGA MUTTOY

c.c.p. Ing. Manuel Martinez Ortiz.
Jefe del Departamento de Servicios Escolares.
Ing. José Juan Ramón Mejía Roldán
Director de Tesis.

SVM/srr.

DEDICATORIAS

A Dios y a la Virgen:
Por darme licencia de haberme permitido realizar y culminar mi meta anhelada.

GRACIAS.

A mi padre el Sr. Ramón Mejía Téllez, que a través de su ejemplo y esfuerzo me ha impulsado al éxito de mi vida.

GRACIAS PAPA.

A mi madre la Sra. Remedios Roldán de Mejía, que por su apoyo y dedicación le agradezco lo que ahora soy.

GRACIAS MAMA.

A mis hermanas y hermanos por su apoyo siempre brindado.

GRACIAS.

A Remedios:
Por todo el cariño y motivación que me ha dado siempre para mi superación personal y profesional

GRACIAS BEBA.

A todos los profesores que contribuyeron en mi desarrollo en especial al Ing. Juan Ramón Mejía Roldán, por su interes que ha manifestado para mi formación profesional.

Al Departamento de Informática de la Torre de Pemex, en especial al ING. Efren Garduño Limón y Gabriel Valencia Miranda, por su orientación y ayuda proporcionada para la elaboración de esta tesis.

A la Máxima Casa de Estudios "UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO" por el orgullo de formar parte de ella.

GRACIAS.

INDICE

No.Pág

INTRODUCCION 1

CAPITULO 1 INTRODUCCION A SQL*PLUS

1.1.- SIGNIFICADO 4

1.2.- EQUIPO NECESARIO PARA CORRER SQL*PLUS 5

1.3.- RDBMS 6

1.3.1.- LIMITES EN RDBMS ORACLE 10

1.4.- SINTAXIS EN LOS COMANDOS SQL 10

1.5.- COMANDO DDL 11
1.5.1.- COMANDO CREATE 11
1.5.1.1.- COMANDO CREATE TABLE 11
1.5.1.2.- COMANDO CREATE INDEX 15
1.5.1.3.- COMANDO CREATE SYNONYM 15
1.5.1.4.- COMANDO CREATE VIEW 17
1.5.1.4.1.- SINTAXIS DEL COMANDO CREATE VIEW 18
1.5.1.4.2.- CLAUSULA WHERE 19
1.5.1.4.3.- COMANDO UPDATE 19
1.5.1.4.4.- COMANDO INSERT 20
1.5.1.4.5.- CREACION DE VISTAS EN 2 O MAS TABLAS 21
1.5.2.- COMANDO ALTER 22
1.5.2.1.- MODIFICACION DE CAMPO 23
1.5.2.2.- INSERCION DE CAMPOS 23
1.5.3.- COMANDO DROP 24

1.6.- COMANDO DQL 25
1.6.1.- COMANDO SELECT 25
1.6.1.1.- OPERADORES LOGICOS 27
1.6.1.1.1- OPERADOR IN 28
1.6.1.1.2.- OPERADOR BETWEEN Y NOT BETWEEN 29
1.6.1.1.3.- OPERADOR LIKE 29
1.6.1.1.4.- OPERADOR NULL 30
1.6.1.2.- CONDICIONES MULTIPLES 31

1.7.- COMANDOS DML 33
1.7.1.- COMANDO INSERT 33
1.7.2.- COMANDO UPDATE 35
1.7.3.- COMANDO DELETE 36

1.8.- METODOS PARA RESGUARDAR LA INFORMACION
1.8.1.- COMMIT 37
1.8.2.- ROLLBACK 38

1.9.- TRANSACCIONES LOGICAS 39

1.10.- JOINS 39

1.11.- SUBQUERIES 43

1.12.- INDEXACION DE TABLAS 46

1.13.- INDICES DE UN JOIN 47

CAPITULO 2 HERRAMIENTA DE AFINACION 'EXPLAIN PLAN'

2.1.- REVISION DE APLICACIONES 50

2.2.- HERRAMIENTA DE DIAGNOSTICO 51
2.2.1.- FACILIDAD EN SQL*TRACE 52
2.2.2.- DECLARACION EXPLAIN PLAN 52
2.2.3.- CREACION DE LA TABLA DE SALIDA EXPLAIN PLAN 53
2.2.4.- SINTAXIS DE LA DECLARACION EXPLAIN PLAN 54
2.2.5.- DESCRIPCION DE COLUMNAS DE LA TABLA EXPLAIN PLAN 55
2.2.6.- OPERACIONES USADAS POR EXPLAIN PLAN 57
2.2.7.- OPERACION CON OPCIONES 59
2.2.8.- APLICACION DEL EXPLAIN PLAN ATRAVES DE UN EJEMPLO 61
2.2.9.- FORMATOS DE ANIDACION DE SALIDA PARA EL EXPLAIN PLAN 63
2.2.10.- ARBOL DE ESTRUCTURAS DEL PLAN DE EJECUCION 64

2.3.- MODIFICACION DE ESTADOS SQL EN APLICACIONES 65
2.3.1.- CONSIDERACION DE DATOS 65
2.3.2.- CONSIDERACIONES FUNCIONALES EN ESTRUCTURAS 65

2.4.- OPTIMIZADOR ORACLE 66
2.4.1.- EJEMPLOS DE OPTIMIZACION 67
2.4.2.- FULL TABLE SCAN 70

2.5.- OPTIMIZACION DE DECLARACIONES SQL 71
2.5.1.- OPTIMIZACION DE QUERIES (SELECTs) 71
2.5.2.- OPTIMIZACION NOTs 71
2.5.3.- OPTIMIZACION ORs 72
2.5.4.- OPTMIZACION ORDER BY 73
2.5.5.- OPTIMIZACION GROUP BY 74

2.6.- RENDIMIENTO AUTOMATICO 75
2.6.1.- POR DISTINTIC 75
2.6.2.- POR GROUP BY 75
2.6.3.- POR SUBQUERIES 75
2.6.4.- POR TRAYECTORIAS DIRECTAS 76

CAPITULO 3 HERRAMIENTA DE EJECUCION "TKPROF"

3.1.- PARAMETROS EN EL INIT.ORA 78

3.2.- HABILITACION DE UNA SESION SQL*TRACE 79

3.3.- HABILITACION DEL SQL*TRACE PARA UNA INSTACIA 79

3.4.- ARCHIVOS SQL*TRACE Y SUS VERSIONES 80

3.5.- EJECUCION DEL TKPROF 80
- 3.5.1.- OPCIONES PARA EL PARSE 81
- 3.5.2.- OPCIONES PARA EL EXECUTE 82
- 3.5.3.- OPCIONES PARA EL FETCH 82
- 3.5.4.- PARAMETROS DEL TKPROF 83

3.6.- EJEMPLO 84

3.7.- ESTADISTICAS DE FACILIDAD EN SQL*TRACE 85

3.8.- LLAMADA RECURSIVA 86

CAPITULO 4 METODOS DE AFINACION PARA UN MEJOR RENDIMIENTO

4.1.- CLUSTERS
- 4.1.1.- DEFINICION 88
- 4.1.2.- CARACTERISTICAS SOBRESALIENTES 88
- 4.1.3.- CRITERIO PARA LA SELECCION DE CLUSTERS 90
- 4.1.4.- CREACION DE UN CLUSTER 90
  - 4.1.4.1.- LLAVE PRIMARIA 91
  - 4.1.4.2.- EJEMPLO PARA LA CREACION DE CLUSTER 92
- 4.1.5.- CREACION DE UNA TABLA CLUSTER
  - 4.1.5.1.- CARACTERISTICAS 93
  - 4.1.5.2.- SINTAXIS 93
  - 4.1.5.3.- EJEMPLOS DE LAS 2 FORMAS EXISTENTES PARA LA CREACION DE TABLAS CON CLUSTER 94
- 4.1.6.- CREACION DE INDICES EN UN CLUSTER 96
- 4.1.7.- ELIMINACION DE UNA TABLA CON CLUSTER 97
- 4.1.8.- TABLAS EXISTENTES EN UN CLUSTER 99
- 4.1.9.- ELIMINACION DE UN CLUSTER 100
- 4.1.10.- ORDENAMIENTO DE TABLAS EN UN CLUSTER 100

4.2.- GENERADOR SEQUENCE
- 4.2.1.- PROPOSITO 101
- 4.2.2.- CARACTERISTICAS 101
- 4.2.3.- CREACION DE UNA SECUENCIA 102
- 4.2.4.- VALORES DEFAULT EN UNA SECUENCIA 104

4.2.5.- PARAMETROS EXISTENTES EN UNA SECUENCIA
4.2.5.1.- PARAMETRO NEXTVAL 105
4.2.5.2.- PARAMETRO CURRVAL 105

4.2.6.- COMANDOS EXISTENTES EN UNA SECUENCIA
4.2.6.1.- COMANDO INSERT 106
4.2.6.2.- COMANDO UPDATE 107
4.2.6.3.- COMANDO DROP 108
4.2.6.4.- COMANDO ALTER 109
4.2.7.- SEQUENCE CACHE
4.2.7.1.- CARACTERISTICAS 109
4.2.7.2.- NUMERO DE ENTRADAS EN EL SEQUENCE CACHE 110
4.2.7.3.- NUMERO DE VALORES PARA CADA ENTRADA EN EL SEQUENCE CACHE 110

CAPITULO 5 APLICACIONES

5.1.- INTRODUCCION 112

5.2.- EJEMPLO 1 112
5.2.1.- EXPLAIN PLAN 112
5.2.2.- ANALISIS DEL TKPROF 115
5.2.3.- CREACION DEL INDICE SOBRE LA TABLA EMP 118
5.2.4.- DECLARACION DE LA CLAUSULA WHERE DENTRO DEL SUBQUERY 120

5.3.- EJEMPLO 2 126
5.3.1.- CREACION DE INDICE A LA TABLA EMP 128
5.3.2.- CREACION DE CLUSTER EN LA TABLA EMP 130
5.3.3.- DECLARACION DE LA CLAUSULA WHERE DENTRO DEL SUBQUERY 132

5.4.- EJEMPLO 3 137
5.4.1.- ANALISIS DEL TKPROF 139
5.4.1.1.- CREACION DEL INDICE EN LA TABLA DEPT 141

5.5.- EJEMPLO 4 142
5.5.1.- CAMBIO DE TABLA EN DICHO QUERY 146
5.5.2.- CREACION DEL INDICE EN LA TABLA T_177 148

5.6.- EJEMPLO 5 151

CONCLUSIONES 156

APENDICE 157

BIBLIOGRAFIA 172

GLOSARIO 173

INTRODUCCION

Actualmente y en la gran mayoría de los casos, el ser humano de manera natural sufre de frecuentes temores ante la presencia del inevitable mundo informático basándose sobre argumentos como:

- Desconfianza en las posibles proyecciones sobre el futuro.
- Temor al desplazamiento de recursos humanos por recursos automáticos.
- Resistencia a cambios presentados.

Este tipo de argumentos, afectan de tal magnitud que implican un entorpecimiento a las tareas del personal existente en el Centro de Cómputo.

A medida que las organizaciones dependen cada vez más y más del mundo informático, es más importante lograr un control efectivo tanto en el personal como en la instrumentación usada en dicho centro, debido a que el ambiente económico actual se caracteriza por los frecuentes e importantes cambios sufridos en el desarrollo informático en los cuales para poder ajustarse a tiempo, se debe estar preparado con la información e instrumentación actualizada y disponible en cualquier momento.

Esto implica que estando sobre todo la información al alcance de la organización, las estrategias planeadas pueden ser mejoradas y por lo tanto, ejecución para cualquier operación se tome en forma más eficiente y medible.

En cambio, si se cuenta con instrumentación no actualizada se crearán problemas, que en realidad son de suma importancia trayendo las siguientes consecuencias:

1.- Duplicidad de esfuerzos por parte del usuario y del informático.
2.- Existencia de excesiva redundancia, que provoca duplicidad de información, lo cual genera como resultado final la obtención de errores costosos.
3.- Desintegración en los datos, provocando una incosistencia entre ellos.
4.- La ausencia de aplicaciones capaces de compartir recursos informáticos para producir información general de manera veraz y oportuna.

En el presente trabajo, se desarrolla un análisis completo aplicado a las Bases de Datos en SQL*PLUS, existentes en el Sistema Institucional de Contabilidad (SIC) de la Torre de Pemex, haciéndose una afinación en cada una de las ellas através de los 2 métodos explicados más adelante (Explain Plan y Tkprof), logrando así mejores tiempos de ejecución.

No está de más mencionar que una afinación de aplicación es una parte fundamental para obtener el mejor rendimiento de Base de Datos Oracle.

La afinación de una aplicación debe efectuarse antes de la afinación misma del RDBMS (Sistema de Administración de Base de Datos Relacionales) por las siguientes razones:

1.- Un buen diseño de la aplicación ofrece el mayor control sobre las instrucciones SQL y los datos procesados por el RDBMS.
2.- Se pueden obtener mejoras sustanciales, afinando la aplicación basada en el conocimiento de SQL sin necesidad de estar familiarizado con la estructura interna de Oracle.
3.- Si la aplicación no está afinada correctamente, no se ejecutará adecuadamente aún cuando se corra en un manejador de Base de Datos bien afinado.

El proceso de afinación de una aplicación Oracle incluye:

- Diseño de datos.
- Optimizador Oracle.
- Indices.
- Generador de secuencias.
- Clusters.

En el primer capítulo, se proporciona toda la información necesaria de algunos comandos existentes en SQL*PLUS tal como: su definición, sintáxis y aplicaciones.

En el segundo capítulo, se proporciona toda la información necesaria con la que se debe contar para el análisis del "EXPLAIN PLAN" aplicado a cualquier query seleccionado, obteniéndose de esta herramienta, un análisis diferente en cada una de las columnas existentes en el Explain Plan de acuerdo al query estudiado.

En el tercer capítulo, se proporciona toda la información necesaria para el análisis del TKPROF aplicado a cualquier query seleccionado, obteniéndose de esta herramienta un plan de ejecución donde se analiza:
(el numero de veces en que un procedimiento fue ejecutado, tiempo de CPU, tiempo de enlace, numero de lecturas físicas, concurrentes y consistentes y el numero total de registros procesados).

Dentro de este capítulo es muy importante hacer notar que dentro de esta herramienta de ejecución se puede analizar también el Explain Plan.

En el cuarto capítulo se proporciona la información necesaria para la creación de clusters y generadores de secuencias, siendo estos, dos métodos muy importantes para el logro de una afinación con mejor rendimiento a dicho query seleccionado.

Una vez efectuado todo este estudio, dentro del capítulo cinco se analizarán y se modificaran algunas Bases de Datos, aplicando todo lo visto anteriormente, hasta obtener una afinación adecuada.

CAPITULO 1

INTRODUCCION A SQL*PLUS

1.1.- SIGNIFICADO

SQL (lenguaje de queries estructurados), se ejecuta por medio de RDBMS ORACLE.

Es un producto para trabajar con la Base de Datos Oracle, el cual permite, entre otras de sus funciones:

- Crear tablas en la Base de Datos.
- Almacenar información en tablas.
- Cambiar información en las tablas.
- Recuperar información en la forma seleccionada (conumit, rollback).
- Mantenimiento de la Base de Datos.

Los comandos SQL, son usados para crear, almacenar, cambiar, recuperar y mantener información en una Base de Datos Oracle.

Un comando SQL es almacenado en una parte de memoria llamada Shared Pool Area, donde este permanece hasta que un nuevo comando es introducido.

Los 17 comandos principales en SQL son los siguientes:

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

* Estos comandos no necesitan estar almacenados en el Shared Pool Area.

Existen también comandos que permiten obtener reportes sofisticados, edición de declaraciones SQL y ayuda. Estos comandos son los siguientes:

@	CHANGE	DESCRIBE	INPUT	SET
#	CLEAR	DISCONNECT	LIST	SHOW
$	COLUMN	DOCUMENT	NEWPAGE	SPOOL
/	COMPUTE	EDIT	PAUSE	SQLPLUS
ACCEPT	CONNECT	EXIT	QUIT	START
APPEND	COPY	GET	REMARK	TIMING
BREAK	DEFINE	HELP	RUN	TTITLE
BTITLE	DELETE	HOST	SAVE	UNDEFINE

El lenguaje SQL*PLUS esta diseñado para escribirse y leerse fácilmente. Por ejemplo, para desplegar el nombre, trabajo y salario de cada empleado de la tabla EMP, se podría teclear el siguiente comando:

```
SELECT ENAME, JOB, SAL
FROM EMP;
```

1.2.- EQUIPO NECESARIO PARA CORRER SQL*PLUS

REQUISITOS:

La computadora sobre la cual se corre ORACLE y SQL, es llamada "Host Computer" (computadora anfitriona).

ORACLE y SQL, puede correrse en diferentes tipos de computadoras anfitrionas.

La computadora anfitriona debe tener un sistema operativo que administre los recursos de la computadora entre el hardware y los programas tal como SQL*PLUS, teniéndose en cuenta que toda clase de computadora usa un sistema operativo diferente.

Si el sistema de operación de la computadora anfitriona es usada por varias personas (es decir sistema multiusuario) se necesitará un password y un usuario para obtener el permiso a el sistema de operación y así cada tabla en la Base de Datos se use de forma " apropiada " de acuerdo a un nombre de usuario especifico.

Si la computadora anfitriona, corre en un sistema de tiempo compartido, es necesario tener una persona llamada (Administrador de Base de Datos), para que supervise el uso de el RDBMS ORACLE, es decir, un DBA.

Un DBA es capaz de dar al sistema:

- Nombre de usuario Oracle.
- Password.

1.3.- RDBMS

El RDBMS (RELATIONAL DATABASE MANAGEMENT SYSTEM), nos permite organizar, almacenar, mantener, calcular, combinar y recuperar información.

Es un lenguaje de cuarta generación (4GL), tal como SQL.

Incluye un diccionario de datos.

Entendiéndose por diccionario de datos, a un grupo de tablas y vistas que contienen una descripción de información acerca de:

- Tablas.
- Privilegios al acceso de usuario.
- Otras características de la Base de Datos.

Las tablas del diccionario frecuentemente usadas cuando se accesa datos son:

TAB .- Despliega una lista de tablas, vistas y sinónimos que se han creado.
DTAB .- Despliega las tablas que tiene el diccionario de datos.
COL .- Despliega una lista de todas las características de las columnas para las tablas creadas.
CATALOG.- Despliega una lista de todas las tablas accesadas.

Una manera de utilizarlas es:

SELECT * FROM TAB; (despliega el nombre de todas las tablas, vistas y sinónimos creados).

NOTA :

Todo comando pueden ser escrito con mayúsculas o minúsculas, siendo esto no significante en SQL*PLUS.

No cabe de más mencionar, que una Base de Datos, es una colección organizada de información, donde, una Base de Datos Oracle consiste de tablas, vistas, queries y reportes que están basados sobre las tablas existentes.

En una Base de Datos Oracle, se puede introducir información para incluirse en reporte, tabla o forma. Las entradas podrían ser palabras, números, datos ó algunos textos.

En un sistema de Base de Datos Relacionales, la información es organizada en tablas. Los nombres de las columnas se presentan en la parte de superior de cada tabla, mientras que la información es listada abajo. Por ejemplo:

NOMBRE DE LA COLUMNA

EMPNO	ENAME	JOB	SALARY	COMM	DEPTNO
7328	CHUY	CLERK	800	300	20
7329	JUAN	SALESMAN	100	300	10

CONTENIDO DE CADA COLUMNA

El RDBMS Oracle, nos permite definir las relaciones entre diferentes columnas de la misma o de diferentes tablas.

Una tabla contiene líneas y columnas:

COLUMNA	COLUMNA	COLUMNA
LINEA 1		
LINEA 2		

Donde cada columna (campo) contiene un tipo de información y cada línea (registro) contiene la información adecuada de acuerdo a la columna correspondiente, teniendo presente que cada campo puede tener desde uno a varios valores de información.

Los tipos de campos, que pueden tener las columnas son los siguientes:

CHAR	Secuencia de más de 240 letras, números, puntuaciones y caracteres esenciales tal como +, - , % , y $.
NUMERICO	Contiene un número consistente de dígitos, un signo y un punto decimal.
DATE	Fecha y la hora del día actual.
LONG	Es similar a la columna char, este puede tener más de 65,535 caracteres, pero puede únicamente ser usado en casos limitados.

La relación que puede existir en algunas tablas, es establecer un join como se explicará más adelante, pero por ahora se explicará en forma sencilla por medio de 2 tablas diferentes llamadas EMP y DEPT.

TABLA EMP

EMPNO	ENAME	JOB	SAL	COMM	DEPTNO
					20
					30
					40

TABLA DEPT

DEPTNO	DNAME	LOC
20		
30		
40		
10		

-Cada empleado en EMP tiene un departamento numérico (DEPTNO) que hace referencia a el departamento numérico en DEPT.

-Ambas columnas DEPTNO tienen el mismo tipo de información. Esta información general nos permite entradas en las 2 tablas al ser combinadas o relacionadas a otra tabla.

La habilidad a relacionar una tabla a otra es muy importante, ya que nos permite organizar información en forma separada ó en unidades manejables, por ejemplo:

Cuando las tablas tienen columnas relacionadas, los valores comunes en las tablas permiten operaciones de unión (join).

1.3.1.- LIMITES EN RDBMS ORACLE

ESTRUCTURA	LIMITE
Tablas en una base de datos	No existe límite
Líneas en una tabla	No existe límite
Columnas en una tabla	No existe límite
Caracteres en un campo de tipo caracter	240
Dígitos en un campo numérico	105
Dígitos significativos en un campo numérico	40
Rango de valores en un campo tipo date	1-JAN-4712 BC to 31-DEC-4712 AD
Indices sobre la tabla	No existe límite
Tablas o vistas unidas en un query	No existe límite
Niveles de anidamiento de subquery	255
Caracteres en un nombre	30

1.4.- SINTAXIS DE LOS COMANDOS SQL

SQL*PLUS ofrece ayuda de todos los comandos. Unicamente es necesario teclear HELP y el nombre del comando para obtener la información.
Para esto se clasificarán los comando en DML, DQL, DDL del SQL, como se verán a continuación.

RECOMENDACION:

Cuando se introduce uno o varios comandos SQL*PLUS sobre una línea única y no cabe, no es necesario presionar return, sino que lo más conveniente es seguir escribiendo y ver que el cursor baje automáticamente a la próxima línea, pero eso sí, si no uno termina de escribir es necesario poner (;) y presionar RETURN, ya que el (;) es una señal para que SQL realice su ejecución.

1.5.- COMANDOS DDL

DDL. (Data Definition Languaje) es decir lenguaje para definición de datos y son:

1.- CREATE.
2.- ALTER.
3.- DROP.

1.5.1- COMANDO CREATE.

Este comando se utiliza para crear:

1.- TABLAS.
2.- INDICES.
3.- SYNONYM.
4.- VISTAS.

1.5.1.1.- COMANDO CREATE TABLE

1.- Nombre de la tabla.-

El primer paso en crear una tabla es que tenga un nombre específico, introduciéndose el nombre de la tabla después de la palabra CREATE TABLE.

El nombre que se seleccione para una tabla debe seguir las siguientes reglas estándares para nombrar un objeto en la Base de Datos Oracle, esto es:

- Debe empezar con una letra , A-Z ó a-z.
- Puede contener letras, números, caracteres especiales (como el subrayado), $ y # pero no se recomienda.
- Puede ser escrito con letras mayúsculas o minúsculas, ejemplo: EMP, Emp.
- Tener un tamaño de 30 caracteres.
- No duplicar el nombre de la tabla.

Si el nombre de una tabla es encerrado en comillas no puede ser usada en las 3 primeras reglas, mencionadas arriba, siendo necesario aplicar las siguiente regla:

- El nombre puede contener cualquier combinación de caracteres, excepto que no debe de tener comillas.

No cabe de más mencionar algunas validaciones para nombres de tablas:

NOMBRE	VALIDACION ?
EMP85	Si
85EMP	No, porque no empieza con letra.
Fixed-assets	Sl
Fixedassets	No, porque no contiene un espacio en blanco
"Fixed assets"	Sí , espacio dentro de las comillas
Update	No, porque es una palabra reservada de SQL
TABLA 1	Si, pero se recomienda que el nombre se refiera al tipo de información que se almacenará.

2.- Nombres de las columnas.-

En el comando CREATE TABLE, las columnas son listadas en paréntesis en seguida del nombre de la tablas. Para cada columna, se debe introducir:

- Nombre de la columna.
- Tipo de dato.
- Ancho máximo y otra información descriptiva, dependiendo sobre el tipo de dato.
- Todo valor NULL es permitido.

Cuando se introduce la información es necesario separarlo através de una coma, considerándo que una tabla puede tener hasta 254 columnas únicamente.

3.- Tipo de valores que existirán en cada columna.

(Recordando, que pueden ser de tipo CHAR, NUMERICO,DATE,LONG; si se tiene alguna duda consultar el punto 1.3 de este capitulo, mencionadas anteriormente)

4.- Tamaño máximo de la columna.

Para explicar este punto, se pondrá el siguiente ejemplo:

ESPECIFICACION	SIGNIFICADO
CHAR(12)	La columna puede contener un valor de 12 caracteres.
NUMBER	La columna puede contener un valor de 40 dígitos.
NUMBER (4)	La columna puede contener un valor de 4 dígitos.
NUMBER (8,3)	La columna puede contener un valor de 8 dígitos y 3 dígitos van a la derecha del punto decimal.
DATE	La columna puede contener valores de tipo date.
LONG	La columna puede contener valores tipo long.

NOTA:

Cuando se declara el ancho en cada columna, no se incrementa la memoria en disco, debido a que cada campo en una tabla consume únicamente el espacio de almacenamiento de acuerdo a como sus propios valores lo necesiten. Por ejemplo:

El ancho máximo de la columna ENAME es de 10, únicamente 5 caracteres son usados para almacemar el nombre SMITH en esa columna.

1.5.1.1.1.- EJEMPLO DE COMANDO CREATE TABLE

Sintáxis:

```
SQL> CREATE TABLE Nombre de la tabla
       ( Columna  Tipo de valores );
```

Ejemplo:

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

NOTA:

-Cuando se crea una tabla, puede declararse campos como NOT NULL ó NULL.
-Cuando se crea un campo NOT NULL, considerará que la columna siempre tendrá un valor, pero en cambio si se pone NULL, no se requiere un valor para la columna, pero tampoco es considerado como un cero, sino que se considera como un valor que no es conocido, es decir,que no es aplicable.

RECOMENDACION:

Cuando una tabla ha sido creada, es necesario rectificar si todos los datos que se introdujeron fueron correctos. Para esto es necesario teclear el comando describe, es decir;

SQL > DESC Nombre de la Tabla;

1.5.1.2.- COMANDO CREATE INDEX

SINTAXIS

CREATE INDEX Nombre del indice
ON Nombre de la tabla (columna, columna);

Para tener una información más completa se sugiere ver el tema 1.12. " INDEXACION DE TABLAS "

1.5.1.3.- COMANDO CREATE SYNONYM

Permite anteponer el nombre del usuario junto con el nombre de la tabla existente en él através del comando CREATE SYNONYM, definiéndose de este manera un sinónimo para el nombre de la tabla.

Una vez creado este sinónimo, se puede hacer uso de él, en lugar del nombre de la tabla.

Para referirse a una tabla de usuario, es necesario anteponer el nombre del usuario junto al nombre de la tabla, seguido por un punto (.), es decir:

```
SQL > SELECT *
      FROM SCOTT.DEPT;
```

Donde:

SCOTT.- Nombre del usuario.
DEPT .- Nombre de la tabla.

Resultado:

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

NOTA:

Este resultado sería el mismo, si si tecleara:

```
SELECT *
FROM DEPT;
```

Para la creación de un sinónimo, se seguirá la siguiente sintáxis:

```
SQL>  CREATE SYNONYM  Nombre del sinónimo
        FOR  Nombre del usuario.Nombre de la tabla;
```

EJEMPLO:

```
SQL> CREATE SYNONYM S_DEPT
       FOR SCOTT.DEPT;
```

Una vez creado el sinónimo se podrá ver su contenido, introduciendo:

```
SQL> SELECT *
     FROM S_DEPT;
```

Resultado:

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

NOTA:

Este resultado es igual al que se observó anteriormente en la tabla de usuario, la única diferencia sería que en este último ejemplo se utiliza el nombre del sinónimo en vez de la tabla de usuario.

La creación de un sinónimo, puede ser usada en cualquier cláusula SQL.

1.5.1.4.- COMANDO CREATE VIEW

PROPOSITO:

Una vista es como una ventana, en la cual solamente se puede ver las tablas.

La creación de estas vistas no ocupan espacio de memoria, por lo que algunas veces reciben el nombre de tablas " virtuales ".

Existen 2 principales razones de usar las vistas y son:

1.- **SEGURIDAD.-** Nos permite accesar a ella, toda la información que uno quiera que se despliegue, es decir, el administrador de una compañía, quisiera que el personal de operación tenga acceso a toda la información existente en una tabla pero no a ciertas columnas de la tabla, entonces através de la creación de las vistas, se puede introducir únicamente los campos que se deseen que sean observados.

2.- **CONVENIENCIAS.-** En lugar de usar un query complejo para obtener ciertos datos, se puede crear una vista el cual permita que se obtenga la misma información.

1.5.1.4.1.- SINTAXIS DEL COMANDO CREATE VIEW

```
CREATE VIEW  Nombre de la vista
AS query;
```

EJEMPLO:

Para definir una vista del departamento 10 de la tabla EMP se introduce:

```
SQL> CREATE VIEW EMP10
       AS  SELECT EMPNO, ENAME, JOB
           FROM EMP
           WHERE DEPTNO=10;
```

El query dentro del comando CREATE VIEW selecciona:

- Columna (EMPNO, ENAME, JOB)
- Registros (DEPTNO=10).
- Tabla (EMP).

Después de que haya creado la vista, para conocer su contenido se tecleará :

```
SQL >   SELECT *
        FROM EMP10;          EMP10 = Nombre de la vista
```

Resultado:

EMPNO	ENAME	JOB
7782	CLARK	MANAGER
7839	KING	PRESIDENT
7934	MILLER	CLERK

1.5.1.4.2.- CLAUSULA WHERE

Cuando una vista es creada, se puede hacer uso de la cláusula WHERE, es decir,

```
SQL > SELECT ENAME, JOB
      FROM EMP10
      WHERE EMPNO>7800;
```

Resultado:

ENAME	JOB
KING	PRESIDENT
MILLER	CLERK

1.5.1.4.3.- COMANDO UPDATE

Cuando se cambia la información en una tabla, los cambios serán reflejados en las vistas de una tabla. Por ejemplo, si se actualiza la tabla EMP, los cambios serán visibles en la vista EMP10 y viceversa:

1) Cambiar JOB a MILLER de CLERK a ANALYST en la tabla EMP y observar los cambios en la vista EMP10:

```
SQL > UPDATE EMP
      SET JOB = 'ANALYST '
      WHERE ENAME= ' MILLER ';
```

Para ver los cambios realizados, introducir:

```
SQL> SELECT *
     FROM EMP10;
```

Resultado:

EMPNO	ENAME	JOB
7782	CLARK	MANAGER
7839	KING	PRESIDENT
7934	MILLER	ANALYST

2) Si se creara la actualización en la vista EMP10, los cambios se actualizarán en la tabla EMP:

```
SQL >  UPDATE EMP10
       SET JOB= 'CLERK '
       WHERE ENAME = ' MILLER ';
```

Para ver los cambios realizados, introducir:

```
SQL>  SELECT *
      FROM EMP;
```

Resultado:

EMPNO	ENAME	JOB
7934	MILLER	CLERK

1.5.1.4.4.- COMANDO INSERT

Cuando se insertan registros en una tabla, también son insertados en una vista.

EJEMPLO:

```
SQL >  INSERT INTO EMP
       VALUES ( 7999, 'BROWN', 'CLERK', 7902, '25-JAN-83', 800, NULL,
                10, 102);

       record created
```

Para observar la inserción en una vista, se teclea:

```
SQL > SELECT *
      FROM EMP10
      ORDER BY ENAME;
```

Resultado:

EMPNO	ENAME	JOB
7999	BROMN	CLERK
7782	CLARK	MANAGER
7839	KING	PRESIDENT
7934	MILLER	ANALYST

NOTA:

Dentro de la creación de una vista, puede usarse cualquier query válida a excepción de la cláusula ORDER BY.

Si se quiere ordenar registros, es necesario hacerse después de la creación de la vista.

1.5.1.4.5.- CREACION DE VISTAS EN 2 O MAS TABLAS

Al construir una vista de más de una tabla, se define la vista con un query que contiene el join, por ejemplo:

1.- Creación de una vista llamada PROJSTAFF con las tablas EMP y DEPT.

```
SQL> CREATE VIEW PROJSTAFF ( EMPLOYEE, PROJECT,
                             PROJECT_NUMBER ) AS
     SELECT ENAME, PNAME, EMP1.PROJNO
     FROM EMP1, PROJ
     WHERE EMP1.PROJNO = PROJ.PROJNO;
```

Donde:

EMPLOYEE.- Va hacer el alias para la columna ENAME de la tabla EMP1.
PROJECT .- Va hacer el alias para la columna PNAME de la tabla PROJ.
PROJNO .- Es el campo de unión en ambas tablas conocido como PROJECT_NUMBER).

Para obtener el resultado del join analizado, se tecleará:

```
SQL > COLUMN EMPLOYEE FORMAT A8;
      COLUMN PROJECT  FORMAT A7;
       SELECT EMPLOYEE, PROJECT
       FROM PROJSTAFF
       WHERE PROJECT_NUMBER=101;
```

Resultado:

EMPLOYEE	PROJECT
SMITH	ALPHA
BLAKE	ALPHA
SCOTT	ALPHA
FORD	ALPHA
CHAN	ALPHA
MASON	ALPHA
ADAMS	ALPHA
CLARK	ALPHA
JONES	ALPHA

NOTA:

Para tener un conocimiento más amplio de un join, se sugiere consultar el tema 2 llamado "JOINS".

1.5.2.- COMANDO ALTER

Este comando nos permite hacer alteraciones a la tabla, es decir, nos modifica o añade campos a dicha tabla.

1.5.2.1.- MODIFICACION DE CAMPO

Si se quiere modificar el campo en una tabla, es necesario tener presente lo siguiente:

1.- Nunca se podrá reducir el ancho de la columna si no esta vacía la tabla, es decir, que no contenga datos.
2.- Nunca se podrá cambiar el tipo de datos si la columna no esta vacia.
3.- Nunca se modificará una columna NULL a NOT NULL, al menos que no exista valores nulos en la columna.
4 - Se podrá cambiar la columna de NOT NULL a NULL, añadiendo la claúsula NULL al final de la columna especificada.
5.- Se podrá incrementar el tamaño de la columna, si se desea.

Sintáxis:

```
SQL> ALTER TABLE  Nombre de la Tabla
       MODIFY     Campo   Tipo de valor ;
```

Ejemplo:

```
SQL> ALTER TABLE DEPT                (En este ejemplo se incrementa el
     MODIFY DNAME CHAR (20);          campo DNAME a  20 )
```

1.5.2.2.- INSERCION DE CAMPOS

Si se añade campos a la tabla se considera lo siguiente:

1.- Cuando se añade una nueva columna a la tabla, es necesario notar que su posición será siempre a la derecha de la última columna existente.
2.- Si se tiene datos incluidos en las demás columnas y se añade otra columna, la nueva columna será nula inicialmente.
3.- Para introducir datos a este campo introducido,es necesario utilizar el comando insert.
4.- Se puede definir nuevas columnas como NOT NULL únicamente si la tabla no contiene registros.

Sintáxis:

```
SQL> ALTER TABLE Nombre de la Tabla
       ADD   ( Campo  Tipo de valor  () );
```

Ejemplo:

```
SQL> ALTER TABLE DEPT
       ADD ( Heading Number (3));
```

1.5.3.- COMANDO DROP

Este comando nos permite:

- Borrar todos los registros de la tabla.
- Desocupar espacio en disco.
- Remover la tabla desde la Base de Datos.
- Cualquier índice y privilegios en la tabla, también serán removidas.
- Borrar índices y vistas utilizadas.

Sintáxis:

```
DROP TABLE   Nombre de la tabla;
       ó
DROP INDEX   Nombre del índice;
       ó
DROP VIEW    Nombre de la vista;
       ó
DROP SYNONYM  Nombre del sinónimo;
```

Cuando se tienen datos en una tabla, no serán borrados permanentemente hasta que no se le de Commit, pero si se le da esta instrucción ya no será posible recuperar la información.

1.6.- COMANDO DQL

DQL (Data Query Languaje) es decir, lenguaje de queries de datos, es:" SELECT ".

1.6.1.- COMANDO SELECT

Permite recuperar información de una tabla, ya sea desplegando toda la información de las columnas al mismo tiempo através de un * ó seleccionando cada columna através de su nombre y seguido por comas.

Este query consta de 2 partes:

1.- Cláusula select.- Especifica lo que se va a desplegar.
2.- Cláusula from .- Indica cuales tablas contienen las columnas seleccionadas.

EJEMPLOS:

1) SQL> SELECT *
FROM EMP;

Resultado:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL
7369	SMITH	CLERK	7902	17-DEC-80	800
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600
7521	WARD	SALESMAN	7698	22-FEB-81	1259
7566	JONES	MANAGER	7839	02-APR-81	2975

El * despliega todas las columnas existentes.

2) SQL> SELECT EMPNO,ENAME,JOB
FROM EMP;

Resultado:

EMPNO	ENAME	JOB
7369	SMITH	CLERK
7499	ALLEN	SALESMAN
7521	WARD	SALESMAN
7566	JONES	MANAGER

Este ejemplo despliega únicamente las columnas indicadas.

Dentro de este comando se tiene las siguientes opciones:

1.- Mostrar uno o varios registros almacenados através de la claúsula WHERE, es decir;

SQL> SELECT ENAME
FROM EMP
WHERE DEPTNO=10;

En este ejemplo únicamente se hace referencia al registro o registros cuya columna deptno = 10.

Dentro de esta claúsula se puede seleccionar cualquier registro especifico que uno quiera.

2.- Ordenar en forma ascendente o descendente cualquier campo.

En una forma ascendente se sigue la secuencia (A a Z y no de Z a A), en cambio en forma descendente se despliega la información de acuerdo al campo deseado.

1.- SQL> SELECT *
FROM DEPT
ORDER BY DEPTNO DESC;

2.- SQL> SELECT *
FROM DEPT
ORDER BY DEPTNO
ASC;

1.- Resultado del departamento en forma descendente:

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

2.- Resultado del departamento en forma ascendente:

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

1.6.1.1.- OPERADORES LOGICOS

Existen dentro de la clausula WHERE del comando SELECT algunos operadores lógicos para la selección de registros y son:

OPERADORES DE IGUALDAD Y DESIGUALDAD	
IGUAL A	=
DIFERENTE	!= ó <>
MAYOR QUE	>
MAYOR QUE o IGUAL A	>=
MENOR QUE	<
MENOR QUE o IGUAL A	<=

Otros operadores son:

OTROS OPERADORES	
Igual a cualquier miembro de la lista	IN
Mayor que ó igual a un valor, Menor que ó igual a otro	BETWEEN, LOW Y HIGH
Selecciona a un registro	LIKE
Cadena de cero ó más caracteres	%
Cadena de un caracter	-
Nulos	IS NULL
Negación de algunos operadores	NOT

1.6.1.1.1.- OPERADOR IN

El operador IN permite seleccionar registros que corresponden a uno de los valores de la lista (al igual que el operador "OR").

EJEMPLO:

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

ENAME	JOB
SMITH	CLERK
SCOTT	ANALYST
ADAMS	CLERK
JAMES	CLERK
FORD	ANALYST
MILLER	CLERK

Este ejemplo, muestra todos lo registros que tengan en la columna JOB los valores de CLERK ó ANALYST, es por eso que se dice que el IN actúa como el operador "OR".

NOTA:

Para observar valores contrarios de acuerdo a las columnas declaradas en la cláusula IN, únicamente se antepone el operador NOT a dicha cláusula, por ejemplo:

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

1.6.1.1.2.- OPERADOR BETWEEN Y NOT BETWEEN

El operador BETWEEN permite seleccionar registros que contienen valores comprendidos dentro de un rango, por ejemplo:

```
SQL> SELECT ENAME, JOB , SAL
     FROM EMP
     WHERE SAL BETWEEN 2000 AND 3000;
```

Esta declaración desplegará la columna ENAME, JOB y SAL junto con los registros que tengan un salario entre 2000 y 3000.

En cambio el operador NOT BETWEEN, es todo lo contrario al BETWEEN, desplegando todos los registros que no están comprendidos en 2000 y 3000, por ejemplo:

```
SQL> SELECT ENAME, JOB, SAL
     FROM EMP
     WHERE SAL NOT BETWEEN 2000 AND 3000;
```

1.6.1.1.3.- OPERADOR LIKE

Dentro de este operador se puede seleccionar una cadena de cero o más caracteres a través de un % o una cadena de un caracter através de un "-", por ejemplo:

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

Muestra todos los registros cuyo nombre con S no importanto la parte final del string.

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

Muestra todos los registros cuyo nombre termine con S no importanto la parte inicial del string.

```
SQL> SELECT ENAME, DEPTNO
     FROM EMP
     WHERE ENAME LIKE 'W- - -';
```

Muestra todos los registros cuyo nombre empiece con W seguida de 3 caracteres más.

NOTA:

Para observar valores contrarios de acuerdo a la columna declarada en la cláusula LIKE, únicamente se antepone el operador NOT a dicha cláusula, por ejemplo:

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

Muestra todos los valores del campo "JOB" que no sean SALESMAN.

1.6.1.1.4.-OPERADOR NULL

Un valor NULL es un campo de datos sin valor, es decir, inexistente.

Un campo numérico que contiene un valor nulo es diferente a cuando contiene un valor de cero. Los valores nulos son normalmente mostrados como blancos y los valores cero como un cero numérico (0), por ejemplo:

```
SQL> SELECT *
     FROM EMP
     WHERE COMM IS NULL;
```

Resultado:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7844	TURNER	SALESMAN	7698	8-SEP-81	1500	0	30
7900	JAMES	CLERK	7698	3-DEC-81	950		30
7698	BLAKE	MANAGER	7839	1-MAY-81	2850		30
7499	ALLEN	SALESMAN	7698	2-FEB-81	1600	300	30

Cuando se lista todos los empleados del departamento 30 de la misma compañía,se observan algunos espacios en blanco en la columna (COMM).

En esta ocasión, BLAKE y JAMES tienen un valor nulo en el campo COMM, en cambio TURNER, tiene un valor cero, siendo muy diferente a la que tiene JAMES y BLAKE.

NOTA:

Para observar valores contrarios de acuerdo a la columna declarada en la cláusula NULL, únicamente se antepone el operador NOT a dicha clánsula, por ejemplo:

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

En este caso desplegará todos los valores de COMM, que no sean nulos.

1.6.1.2.- CONDICIONES MULTIPLES

La claúsula WHERE, puede regresar más de un valor de acuerdo a la condición que tenga que satisfacer el query, y esto se logra usando los operadores "AND" y "OR".

EJEMPLOS:

1)

```
SQL>  SELECT ENAME,JOB
      FROM EMP
      WHERE DEPTNO=20
      AND JOB!= CLERK;
```

Este ejemplo, muestra todos aquellos registros cuyo DEPTNO=20 y JOB no sea igual a CLERK.

Resultado:

ENAME	JOB
JONES	MANAGER
SCOTT	ANALYST
FORD	ANALYST

El operador AND añade dentro de la claúsula WHERE, otro criterio de selección para encontrar registros.

2)

```
SQL>  SELECT ENAME,JOB
      FROM EMP
      WHERE DEPTNO=20
      OR JOB!= CLERK;
```

Este ejemplo, muestra todos aquellos registros cuyo DEPTNO=20 ó JOB no sea igual a CLERK.

Resultado:

ENAME	JOB
SMITH	CLERK
ALLEN	SALESMAN
WORD	SALESMAN
JONES	MANAGER
MARTIN	SALESMAN
BLAKE	MANAGER
CLARK	MANAGER
SCOTT	ANALYST

1.7.- COMANDOS DML

Los comandos DML (Data Manipulation Languaje), son:

1 - INSERT.
2 - UPDATE.
3.- DELETE.

1.7.1.- COMANDO INSERT

Este comando permite la inserción de un valor para cada columna en la tabla.

SINTAXIS:

INSERT INTO Nombre de la Tabla
VALUES (Lista los valores de los datos):

Para hacer uso de este comando se recomienda lo siguiente:

1.- Tener creada una tabla antes de que se inserten datos.
2.- Separar los valores con comas.
3.- Usar el comando "DESCRIBE", que muestra el orden y el tipo de las columnas.
4.-Cada valor introducido debe tener el tipo de datos de la columna correspondiente.
5.- Cuando se trata de valores tipo CHAR o DATE,es necesario ponerlo entre apóstrofes, ejemplo:

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

NOTA:

1.- Cada valor debe introducirse de acuerdo al orden especificado por la cláusula insert ó en el orden en que las columnas están cuando se creó la tabla.

2.- Si no se incluye una columna en la cláusula insert, el valor para esa columna por default es NULL.

3.- Dentro de la cláusula VALUES, se puede poner la palabra NULL (al menos de que no se haya especificado NOT NULL para esa columna).

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

4.- Cuando se insertan valores en la columna tipo DATE es necesario considerar:

a) El formato default para dichas fechas:

' DD-MON-YY' Abreviación de MON:

JAN	MAY	SEP
FEB	JUN	OCT
MAR	JUL	NOV
APR	AGO	DEC

b) Introducción automática del tiempo y del día correspondiente através de la cláusula " SYSDATE".

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

5.- Para asegurar que lo que se introdujo es correcto, se hace uso del comando SELECT.

1.7.2.- COMANDO UPDATE

SINTAXIS:

UPDATE Nombre de la tabla
SET Campo=Valor
WHERE Expresión lógica.

Este comando permite realizar una o varias actualizaciones a las columnas correspondientes, considerando que:

1.- Una cláusula SET, permite acuatizar múltiples columnas, especificando los cambios en dicha cláusula, es decir (las columnas y los nuevos valores correspondientes).
2.- Si se omite la cláusula WHERE, todos los valores en la columna cambiaran a el valor declarado en la cláusula SET.
3.- Si no se omite la cláusula WHERE, se cambiarán únicamente los valores de acuerdo a las columnas declaradas en dicha cláusula.

EJEMPLOS:

1) SQL > UPDATE EMP
SET JOB = 'MANAGER'
WHERE ENAME= 'MARTIN';

Resultado:

EMPNO	ENAME	JOB
7369	SMITH	CLERK
7499	ALLEN	SALEMAN
7521	WARD	MANAGER
7566	JONES	SALESMAN
7654	MARTIN	----- SALESMAN-----

En este ejemplo **JOB** va a cambiar de **SALESMAN** a **MANAGER**

2) SQL > UPDATE EMP
SET JOB = 'MARKET REP'
WHERE JOB= 'SALESMAN';

Resultado:

EMPNO	ENAME	JOB
7369	SMITH	CLERK
7499	ALLEN	-----SALESMAN-----
7521	WARD	MANAGER
7566	JONES	-----SALESMAN-----
7654	MARTIN	-----SALESMAN-----

En este ejemplo JOB va a cambiar de SALESMAN a MARKET REP.

3) SQL > UPDATE EMP
SET EMPNO = 7740, JOB = 'MARKET REP '
WHERE JOB= 'SALESMAN';

Resultado:

EMPNO	ENAME	JOB
7369	SMITH	CLERK
---7499---	ALLEN	-----SALESMAN-----
7521	WARD	MANAGER
---7566---	JONES	-----SALESMAN-----
---7654---	MARTIN	-----SALESMAN-----

En este ejemplo todos los EMPNO van a cambiar a 7740 y JOB de SALESMAN a MARKET REP.

1.7.3.- COMANDO DELETE

SINTAXIS:

DELETE FROM Nombre de la tabla
WHERE expresión lógica;

Permite:

1.- Borrar registros parcialmente.
2.- Através de la cláusula WHERE, determina cuales registros serán borrados, siempre y cuando se haya salvado la información.

SQL> DELETE FROM EMP WHERE EMPNO=7654;	SQL> DELETE FROM EMP;
En este caso, se esta especificando que únicamente se borrará el registro que tiene el número 7654 en la columna EMPNO.	Esta instrucción borrará todos registros existentes en la tabla EMP.

1.8.- METODOS PARA RESGUARDAR LA INFORMACION

1.8.1.- COMMIT

Permite que todas las actualizaciones (insert, delete, update) a tablas se realicen en forma permanente.

Cuando un trabajo aún no se ha salvado por medio del COMMIT, únicamente el usuario quien ha hecho estos cambios a las tablas, pueden verlos, pero para que los demás vean los últimos cambios correspondientes, se tiene que dar COMMIT.

Un COMMIT se puede declarar de 3 tipos diferentes:

1.- **COMMIT EXPLICITO.**- Especifica el comando COMMIT en SQL, para que todos los cambios pendientes se hagan permanentes.

SQL> COMMIT;

2.- **COMMIT IMPLICITO.**- En este comando se encuentran una serie de comandos SQL que causan un COMMIT implicito y son:

ALTER	DISCONECT	QUIT
AUDIT	DROP	REVOKE
COMMENT	EXIT	RENAME
CONNECT	GRANT	
CREATE	NO AUDIT	

3.- **COMMIT AUTOMATICO.**-Salva cada una de las actualizaciones realizadas inmediatamente después de ejecutar un insert, update o delete, es decir, que tenga:

SQL> SET AUTOCOMMIT ON ó SET AUTOCOMMIT INMEDIATE

Para apagarlo se teclea:

SQL> SET AUTOCOMMIT OFF (Siendo el default)

1.8.2.- ROLLBACK

Permite cancelar todos los cambios de los trabajos pendientes de salvar y la Base de Datos es restablecida al estado inicial correspondiente antes de dar el último Commit.

Oracle automáticamente realiza ROLLBACK cuando:

1.- El programa es anormal.
2.- Fallas en el sistema, tal como el reseteo de un sistema ó falta de abastecimiento de energía eléctrica.
3.- Errores en los comandos como equivocación de una operación inadecuada.

1.9.- TRANSACCIONES LOGICAS

- Todas las actualizaciones efectuadas através de operaciones sucesivas entre un COMMIT de una Base de Datos son llamadas transacciones, es decir, dentro de esta transacción se puede realizar un insert, update y/o delete.

- Cuando una transacción es interrumpida como una falla en el sistema, la transacción completa volverá a su estado actual como si se le hubiera dado el Rollback.

- Toda transacción lógica prevee todo tipo de error antes de ser salvada la información.

1.10.- JOINS

Con una condición joins, se especifican las relaciones entre diversas tablas.

Un join puede ser de 2 tipos:

1.- EQUI-JOIN .- Porque el operador de comparación en la condición de join es (=).
2.- NON-EQUI-JOIN.- Porque puede especificar cualquier relación entre columnas que no usan el operador (=) como por ejemplo:

WHERE X.SAL > Y.SAL

EJEMPLO:

Tabla EMP

EMPNO	ENAME	JOB	DEPTNO
7369	SMITH	CLERK	20
7499	ALLEN	SALESMAN	30
7521	WARD	SALESMAN	30
7566	JONES	MANAGER	20
7654	MARTIN	SALESMAN	30
7698	BLAKE	MANAGER	30

Tabla DEPT

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

Através de:

```
SQL > SELECT EMPNO, ENAME, JOB, EMP.DEPTNO, DNAME
      FROM EMP, DEPT
      WHERE EMP.DEPTNO= DEPT.DEPTNO;
```

se realizará un join en las columnas DEPTNO de ambas tablas, es decir, desplegará como resultado aquellas columnas que sean iguales en dichas tablas, obteniéndose lo siguiente:

Resultado:

EMPNO	ENAME	JOB	DEPTNO	DNAME
7369	SMITH	CLERK	20	RESEARCH
7499	ALLEN	SALESMAN	30	SALESMAN
7521	WARD	SALESMAN	30	SALESMAN
7566	JONES	MANAGER	20	RESEARCH
7654	MARTIN	SALESMAN	30	SALESMAN
7698	BLAKE	MANAGER	30	SALESMAN

Cuando en algunas ocasiones se llegara a olvidar la cláusula WHERE, desplegará todos los registros existentes tanto de la tabla EMP como DEPT, de acuerdo al join seleccionado, como:

```
SQL> SELECT ENAME, EMP.DEPTNO,LOC
     FROM EMP, DEPT;
```

Un join puede tener las siguientes características:

1.- Con un símbolo (+) puesto en él, causará que los valores nulos, aparezcan en la información desplegada.

```
SQL > SELECT ENAME, DEPT.DEPTNO, LOC
      FROM EMP, DEPT
      WHERE EMP.DEPTNO(+) = DEPT.DEPTNO;
```

Es importante visualizar que el signo (+), se coloca en la tabla donde no existen valores nulos.

Resultado:

ENAME	DEPTNO	LOC
CLARK	10	NEW YORK
MILLER	10	NEW YORK
KING	10	NEW YORK
SMITH	20	DALLAS
SCOTT	20	DALLAS
JONES	20	DALLAS
ADAMS	20	DALLAS
ALLEN	30	CHICAGO
BLAKE	30	CHICAGO
TURNER	30	CHICAGO
JAMES	30	CHICAGO
MARTIN	30	CHICAGO
WARD	30	CHICAGO
	40	BOSTON

2.- Se puede hacer join a la misma tabla cuando se quiere unir un registro en una tabla con otro registro de la misma tabla, es decir se ejecuta un SELF-JOIN.

```
SQL > SELECT WORKER.ENAME, MANAGER.ENAME   MANAGER
      FROM EMP WORKER, EMP MANAGER          (alias)
      WHERE WORKER.MGR=MANAGER.EMPNO;
```

Resultado:

ENAME	MANAGER
SCOTT	JONES
FORD	JONES
ALLEN	BLAKE
WARD	BLAKE
MARTIN	BLAKE
TURNER	BLAKE
JAMES	BLAKE
MILLER	CLARK
ADAMS	SCOTT
JONES	KING
BLAKE	KING
CLARK	KING
SMITH	FORD

Un alias debe ser usado cuando se quiere extraer información de la misma tabla.

En el ejemplo anterior es necesario destacar lo siguiente:

1.- Se declara através del alias WORKER y MANAGER a la misma tabla EMP, entonces cuando se realiza el WHERE, en vez de EMP se pone WORKER. MGR, donde MGR es la columna que se va a comparar con MANAGER. EMPNO, donde MANAGER como se mencionó anteriormente es igual a que si pusieramos EMP.

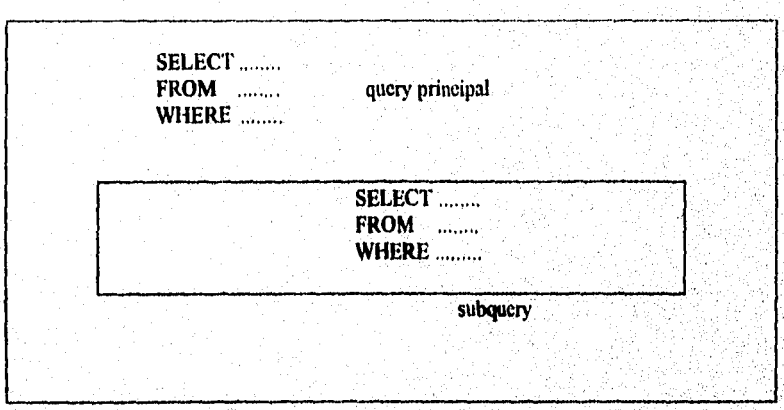
2.- Se puede unir una tabla a si mismo, através de 2 tablas creadas en forma separada, es decir, como se vió en el ejemplo, EMP WORKER y EMP MANAGER , sabiendo que se hace referencia a esa misma tabla.

3.- Es necesario usar un alias para no confundirse en la cláusula FROM, permi - tiendo de esta manera distinguir los nombres de una y otra columna.

1.11.- SUBQUERIES

Un subquery es un query contenido dentro de la cláusula where.

SINTAXIS:



Todos los resultados de un subquery se incluirán en el query principal.

Dependiendo de los subqueries obtenidos se hará siempre una ejecución de abajo hacia arriba, es decir, se ejecutará primero el subquery más interno y después los demás.

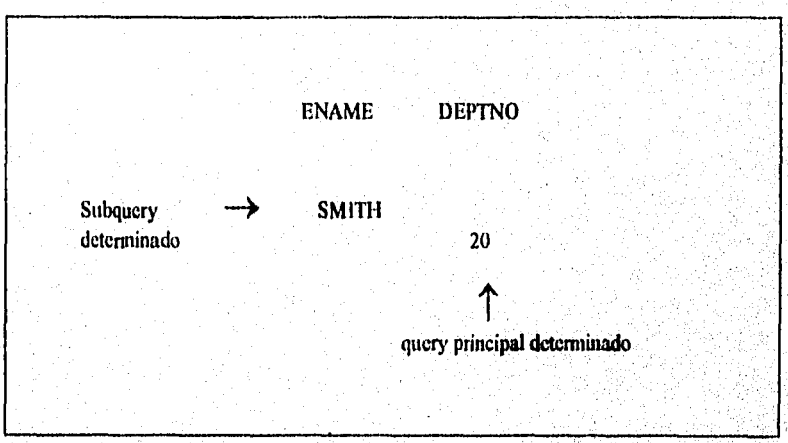
EJEMPLOS:

1) SQL > SELECT ENAME, DEPTNO
FROM EMP
WHERE DEPTNO=
(SELECT DEPTNO
FROM EMP
WHERE ENAME= ' SMITH');

1.- Se hará la búsqueda de la columna DEPTNO dentro de la tabla EMP, cuyo nombre sea igual a 'SMITH'.

2.- Una vez encontrado, desplagará en la pantalla la columna ENAME y DEPTNO.

Resultado:



Existen también subqueries múltiples, que incluyen 2 ó más subqueries.

2) SQL > SELECT ENAME, JOB, SAL
FROM EMP
WHERE JOB= (SELECT JOB
FROM EMP
WHERE ENAME = 'CLARK')
OR SAL >
(SELECT SAL
FROM EMP
WHERE ENAME= 'CLARK'));

Resultado:

ENAME	JOB	SAL
JONES	MANAGER	2975
BLAKE	MANAGER	2850
CLARK	MANAGER	2450
SCOTT	ANALYST	3000
KING	PRESIDENT	5000
FORD	ANALYST	3000

En este ejemplo se realizó lo siguiente:

1.- Se analiza el último subquery cuyo salario sea mayor al que tiene 'CLARK' dentro de la tabla EMP; en este caso se desplegará todos aquellos que tengan un salario mayor de 2450.
2.- El query de arriba de SAL, nos dice, que se despliegue todos los JOB que son iguales al que tienen ' CLARK', en este caso será JOB = 'MANAGER '.
3.- Se continua con el select , desplegando la columna ENAME, JOB y SAL, haciendo incapie de que este ejemplo esta usando el operador OR, es decir, que se puede usar una condición u otra.

NOTA:

Nunca se puede hacer uso de la cláusula order by dentro de un subquery.

1.12.- INDEXACION DE TABLAS

El propósito de un índice es:

Ayudar a las tablas del query más grande en RDBMS a que se ejecute más rápido.

SINTAXIS:

```
SQL > CREATE INDEX Index_name
        ON Nombre de la Tabla ( columna, columna );
```

EJEMPLO:

```
SQL > CREATE INDEX EMP_ENAME
        ON EMP (ENAME);
              ↑       ↑
            tabla   columna
```

La sintáxis para borrar un índice es:

```
SQL > DROP INDEX EMP_ENAME;
```

Un índice tiene las siguientes características:

1.- Se indexa únicamente en tablas grandes (mínimo 50 registros existentes);
2.- Se insertan datos antes de indexar.
3.- Una tabla tiene cualquier número de índices.
4.- Se puede indexar la columna que unicamente identifique líneas (llave primaria).
5.- Los índices son actualizados automáticamente.

Para mejorar la ejecución en un índice, se puede hacer uso de los índices " UNICOS " para garantizar que el valor en una columna sea único.

Ejemplo:

Para estar seguro de que cada nombre de empleado en la tabla EMP aparezca una sola vez se teclea:

```
SQL > CREATE UNIQUE INDEX EMP_EMPNO
      ON EMP (EMPNO );
```

Una vez creado este índice único, se obtendrá un mensaje de error si se quiere insertar o actualizar un registro con el mismo nombre de EMPNO en cualquier registro de la tabla, es decir:

Cuando se trata de insertar un nuevo empleado con la misma columna EMPNO de Miller (7934) o sea:

```
SQL > INSERT INTO EMP
        VALUES ( 7934, 'PHILLIPS', 'CLERK', 7782, ' 30-JAN-84 ', 1500,
                 NULL,10, NULL );
```

Marcará:

ERROR AT LINE 1: ORA-0001: Valor duplicado en el índice .

1.13.- INDICES EN UN JOIN

Los índices son especialmente importantes en los queries que usan joins, si se tienen 2 tablas con joins, es necesario crear un índice sobre la columna que se involucre con el join o en caso necesario en ambos de ellos.

Por ejemplo, si se usa un join en la tabla DEPT y EMP, se podria crear un índice sobre la columna DEPTNO de EMP, debido a que esa columna también hace referencia a DEPT.

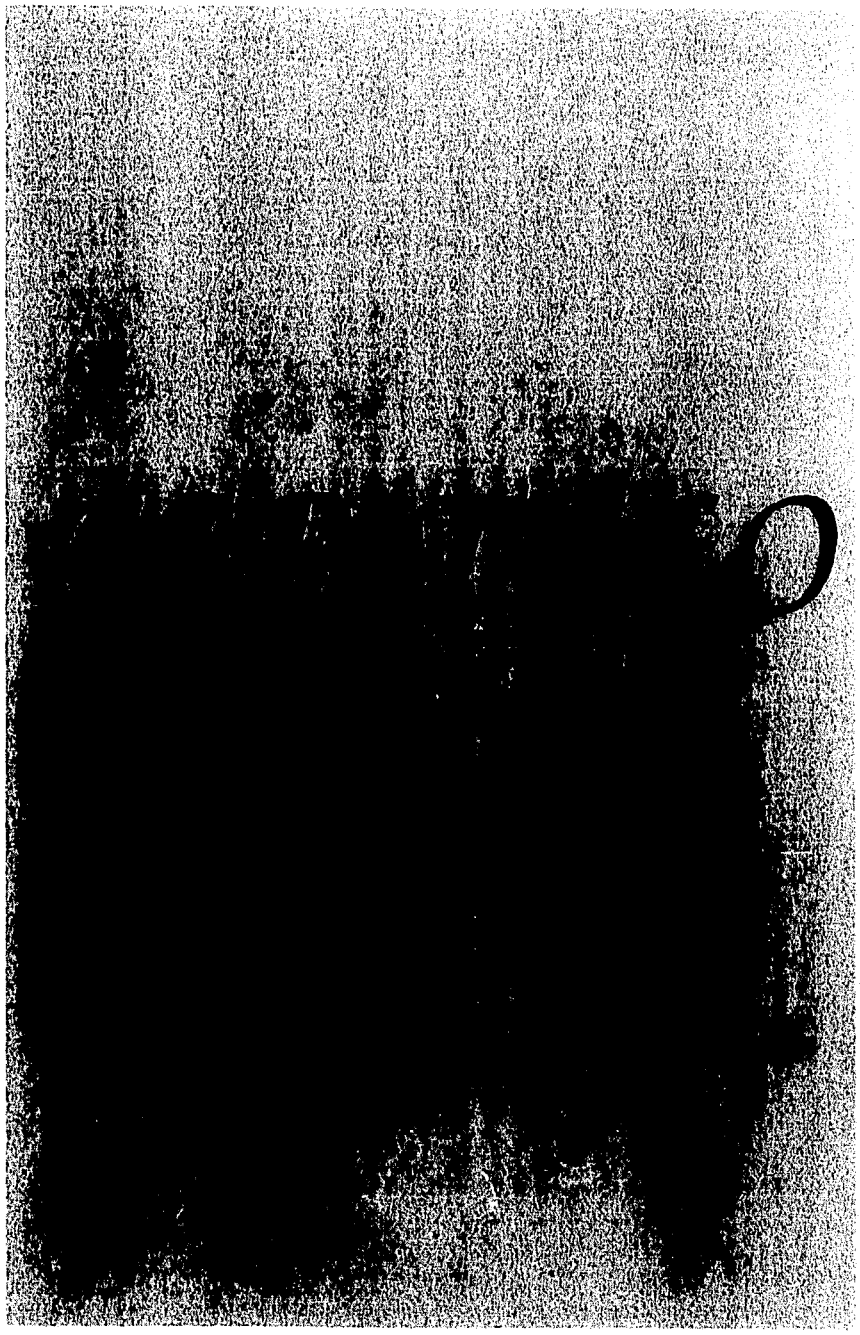
```
SQL > CREATE INDEX EMP_EMPNO
      ON EMP ( DEPTNO);

SQL > SELECT ENAME, LOC
      FROM EMP, DEPT
      WHERE EMP.DEPTNO= DEPT.DEPTNO;
```

Resultado:

ENAME	LOC
CLARK	NEW YORK
KING	NEW YORK
MILLER	NEW YORK
BROWN	NEW YORK
SMITH	DALLAS
JONES	DALLAS
SCOTT	DALLAS
ADAMS	DALLAS
FORD	DALLAS
MASON	DALLAS
CHAN	DALLAS
ALLEN	CHICAGO
WARD	CHICAGO
MARTIN	CHICAGO

El índice sobre DEPTNO es usado para encontrar más rápido los registros de EMP que van hacer join con los registros seleccionados de DEPT.



CAPITULO 2

HERRAMIENTA DE AFINACION
" EXPLAIN PLAN "

OBJETIVO:

La afinación de aplicaciones es parte importante para obtener mejores tiempos de ejecución.

INTRODUCCION:

DEFINICION DEL PROBLEMA.- Detección del mal uso de comandos en el diseño de programas.

PROPOSITO DE ESTUDIO.- Ayudar a los diseñadores de sistemas a mejorar el rendimiento de sus aplicaciones.

2.1.- REVISION DE APLICACIONES

Descubrir que hace la aplicación:

- Que declaraciones SQL utiliza la aplicación.
- Que datos procesa la aplicación.
- Que operaciones ejecuta la aplicación y sobre que datos.

Conocer los datos es el primer paso para medir el rendimiento para su afinación. Las instrucciones de ejecución no pueden contrarrestar los efectos de un mal diseño de base de datos. El rendimiento a adquirir en una aplicación se puede lograr dependiendo de como las bases de datos son definidas y almacenadas desde el diseño inicial.

El buen o mal diseño de sus tablas en el modelo relacional puede ser ventaja para obtencion de mejores rendimientos en sus tiempos de respuesta ó llevarlo a una trampa, creyendo que este tipo de herramientas no sean tan efectivas.

El entender los conceptos de la teoria relacional como son, normalización, atomicidad, etc. En aplicaciones prácticas ayudará a diseñar y a tener mayores logros en la flexibilidad y en el gran rendimiento en una aplicación de base de datos relacional.

Una de las ventajas de las base de datos relacionales es que puede efectuarse cambios rápidamente.

Por otro lado, desde el diseño de sus tablas, los usuarios pueden adquirir mejores rendimientos usando estructuras con índices y clusters.

2.2.- HERRAMIENTAS PARA DIAGNOSTICO

ORACLE proporciona herramientas de diagnóstico, que ayudan a la afinación de aplicaciones, como :

1.- Facilidad SQL*TRACE.
2.- Declaración EXPLAIN PLAN.
3.- Declaración TKPROF.

El SQL*TRACE genera estadísticas por cada declaración procesada en SQL.

El EXPLAIN PLAN muestra el plan de ejecución elegido por el optimizador ORACLE por las declaraciones SELECT, UPDATE, INSERT y DELETE.

El TKPROF, es una herramienta para generar reportes estadísticos, para cada sentencia SQL, indicando el tiempo utilizado para PARSE, EXECUTE y FETCH.

2.2.1.- FACILIDAD EN SQL*TRACE

Estadisticas SQL*TRACE:

- El número de veces que cada declaración SQL, se ha ejecutado, parseado o leído.
- El tiempo necesario para procesar cada declaración SQL.
- La memoria y el acceso a disco asociado con cada declaración SQL.
- El número de líneas de cada ejecución SQL procesada.

2.2.2.- DECLARACION EXPLAIN PLAN

El EXPLAIN PLAN es una herramienta que:

-Despliega el plan de ejecución seleccionado por el optimizador de ORACLE para sentencias "SELECT", "DELETE","UPDATE" y "INSERT".

-Através de estos datos se puede determinar si se está haciendo uso óptimo de los indices.

Los pasos para generar EXPLAIN PLAN son:

-Creación de la tabla EXPLAIN PLAN en la base de datos que es la que va a contener el resultado del explain plan.

-Correr el "EXPLAIN PLAN".

2.2.3.- CREACION DE LA TABLA DE SALIDA EXPLAIN PLAN

Antes de correr la declaración EXPLAIN PLAN, se debe, crear una tabla que contenga las características de salida, requeridas por el EXPLAIN PLAN.

La tabla default usada por el EXPLAIN PLAN es "PLAN_TABLE", sin embargo puede crearla con cualquier otro nombre siempre y cuando contenga las mismas columnas y tipo de datos:

```
CREATE TABLE PLAN_TABLE (
STATEMENT_ID          CHAR(30),
TIMESTAMP             DATE,
REMARKS               CHAR(80),
OPERATION             CHAR(30),
OPTIONS               CHAR(30),
OBJECT_NODE           CHAR(30),
OBJECT_OWNER          CHAR(30),
OBJECT_INSTANCE       CHAR(30),
OBJECT_TYPE           CHAR(30),
SEARCH_COLUMNS        NUMERIC,
ID                    NUMERIC,
PARENT_ID             NUMERIC,
POSITION              NUMERIC,
OTHER                 LONG);
```

2.2.4.- SINTAXIS DE LA DECLARACION EXPLAIN PLAN

La sintáxis de la declaración EXPLAIN PLAN es:

```
EXPLAIN PLAN
( SET STATEMENT_ID = 'description')
( INTO  table_name)
  FOR sql_statement
```

Donde:

description	Un identificador opcional para la declaración. Si no se pone el STATEMENT_ID, la descripción será NULA.Es necesario que ponga un identificador, ya que otros usuarios podrían usarlo y así diferenciaría su información.
table_name	Esta cláusula es opcional ya que especifica el nombre de la tabla de salida dentro la cual usted desea guardar sus resultados.La tabla debe conformar el nombre de la columna, tipo y longitud como se describe en la tabla Plan_Table.

sql_statement	Es donde se pone la sentencia INSERT, DELETE, UPDATE ó SELECT con la cual se ejecutará EXPLAIN PLAN.

2.2.5.- DESCRIPCION DE COLUMNAS DE LA TABLA PARA EL EXPLAIN_PLAN

La tabla PLAN_TABLE usada por la declaración default para EXPLAIN PLAN, contiene las siguientes columnas:

STATEMENT_ID	Identificador opcional que se especifica en la declaración EXPLAIN PLAN.
TIMESTAMP	Fecha y hora, cuando la declaración fué analizada.
REMARKS	Cualquier comentario(máximo de 80 caracteres) que se quiere asociar a cada paso de el plan explicado.
OPERATION	Nombre de la operación "interna" ejecutada en ese paso.

OPTIONS	Adicionalmente describe la salida OPERATION llevada a cada paso.
OBJECT_NODE	Nombre de el database link usado, al referenciar el objeto.
OBJECT_OWNER	Nombre de el usuario propietario de las tablas o indices.
OBJECT_NAME	Nombre de los objetos ó indices.
OBJECT_INSTANCE	Número correspondiente a la posición ordinal de los objetos como aparecieron en la declaración original.
OBJECT_TYPE	Modificador que proporciona la información que describe caracteristicas del objeto de la base de datos, por ejemplo NON_UNIQUE para indices.
SEARCH_COLUMNS	No usada.
ID	Es un número asignado por cada paso en el plan de ejecución.
PARENT_ID	El identificador hijo ligado a los pasos de la operación.

POSITION	El orden de los pasos para el mismo PARENT_ID.
OTHER	Es la información adicional que proporciona el EXPLAIN PLAN, por ejemplo, para queries distribuidos.

2.2.6.- OPERACIONES USADAS POR EXPLAIN PLAN

AND-EQUAL	Recuperación utilizando intersecciones con ROWIDS para las búsquedas con indices. Esta operación se usa con la cláusula WHERE, que contengan comparaciones iguales y AND a la vez,donde cada comparación incluye una columna indexada non-unique.
CONNECT BY	Recuperación, basada sobre un camino jerárquico, se usa al ejecutar la cláusula CONNECT BY en las declaraciones SELECT.
CONCATENATION	Recuperación, desde un grupo de tablas con operación UNION ALL.
COUNTING	Operación que cuenta el número de registros regresados desde la tabla.

FILTER	Restricción, para registros regresados desde la tabla.
FIRST ROW	Recuperación, de el primer registro del resultado de un query.
FOR UPDATE	Recuperación, que ocupa candados sobre los registros seleccionados.
INDEX	Recuperación desde un índice.
INTERSECTION	Solo son almacenados los primeros registros.
MERGE JOIN	La ejecución de un join que mezcla más de 2 operandos.
MINUS	Recuperación de registros en la tabla 1 pero no clasificada en la tabla 2.
NESTED LOOPS	Operaciones joins dependientes ejecutadas sobre 2 operaciones (tipo-hijo).
PROJECTION	Recuperación, de un subset de columnas de una tabla.
REMOTE	Recuperación, desde la base de datos a otra base de datos concurrente.
SEQUENCE	Operación que involucre un generador SEQUENCE.

SORT	Recuperación, de registros ordenados sobre una o más columnas.
TABLE ACCESS	Recuperación, desde una base de datos.
UNION	Recuperación, de registros únicos entre 2 tablas.Los duplicados no son tomados en cuenta.

2.2.7.- OPERACIONES CON OPCIONES

Son operaciones ejecutadas por el RDBMS cuando se procesa una declaración SQL, siendo analizadas dentro del EXPLAIN PLAN.

OPERACION	OPCION	DESCRIPCION
INDEX	**UNIQUE SCAN**	Indice que considera valores únicos.
	RANGE SCAN	Indice que considera un rango de valores recuperados con el operador lógico BETWEEN.
MERGE JOIN	**OUTER**	Unión externa.
NESTED LOOPS	**OUTER**	Unión externa.

SORT	**UNIQUE**	Clasificación que produce valores únicos.
	GROUP BY	Clasificación de las operaciones de agrupación.
	ORDER BY	Forma de agrupación.
	JOIN	Clasificación por merge joln.
TABLE ACCESS	**JOIN**	Clasificación para operaciones JOINS.
	ORDER BY	Clasificación para operaciones de ordenamiento.
	BY ROWID	Tabla accesadas por ROWID. El ROWID es recuperado desde el índice, que son observados desde los renglones de la tabla asociada al índice.
	FULL	Tablas accesadas por el método FULL TABLE SCAN.
	CLUSTER	Tabla accesada por la llave del cluster.

2.2.8.-APLICACION DEL EXPLAIN PLAN A TRAVÉS DE UN EJEMPLO

En este ejemplo, es necesario hacer notar que siempre la sintáxis declarada anteriormente para el análisis del EXPLAIN PLAN debe ser escrita antes de cualquier declaración SQL.

```
SQL> explain plan
  1 set statement_id = 'EMP_SAL'
  2 for
  3 select ename,job,sal,dname
  4 from emp,dept
  5 where emp.deptno= dept.deptno
  6 and not exists
  7     (select * from salgrade
  8     where emp.sal between losal and
                hisal)
  9 /
```

NOTA:

La declaración set statement_id dentro del EXPLAIN PLAN, no precisamente debe ser especificada, debido a que se considera como un identificador opcional.
Dentro de este ejemplo, no se hizo referencia a la declaración INTO (donde se especifica el nombre de la tabla), debido a que la tabla existente, tiene por nombre"PLAN_TABLE", en cambio si tuviera otro nombre si se tendria que especificar dicha opción.

Para analizar los resultados es necesario accesar la tabla referida para el plan de ejecución y obtener la información necesaria:

```
sql>select operation,options,object_name,id,parent_id,position
    from plan_table
    where statement_id='EMP_SAL'
    order by id
    /
```

OPERATION	OPTIONS	OBJECT_NAME	ID	PARENT_ID	POSITION
FILTER			1		0
MERGE JOIN			2	1	1
SORT	JOIN	DEPT	3	2	1
TABLE ACCESS	FULL		4	3	1
SORT	JOIN		5	2	2
TABLE ACCESS	FULL	EMP	6	5	1
TABLE ACCESS	FULL	SALGRADE	7	1	2

2.2.9.-FORMATOS DE ANIDACION DE SALIDA PARA EXPLAIN PLAN

No cabe de más mencionar, que tanto este punto como el siguiente, explicarán el orden en que las operaciones son realizadas, basándose en el ejemplo anterior.

Con el siguiente SELECT se representa una salida anidada, ordenando los pasos del proceso. Se usa la función LPAD junto con el comando CONNECT BY PRIOR, para construir un reporte que nos muestra los niveles de exploración para la construcción de una estructura arbolada como se muestra en la figura del tema 2.2.10.

```
SELECT LPAD( ' ', 2*LEVEL) || ' '|| OPTIONS|| ' ' || OBJECT_NAME
       QUERY_PLAN
FROM PLAN_TABLE  WHERE STATEMENT_ID= 'EMP_SAL'
CONNECT BY PRIOR ID= PARENT_ID AND STATEMENT_ID='EMP_SAL'
START WITH ID=1
```

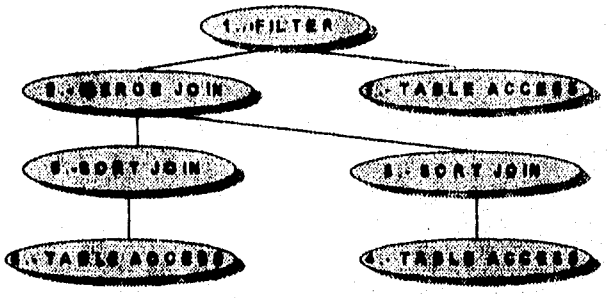
```
QUERY PLAN
----------

FILTER
  MERGE JOIN
    SORT JOIN
      TABLE ACCESS FULL DEPT
    SORT JOIN
      TABLE ACCESS FULL EMP
TABLE ACCESS FULL SALGRADE
```

NOTA:

La declaración 2*level, nos indica que existirán 2 espacios entre un nivel y otro.

2.2.10.-ARBOL DE ESTRUCTURAS DEL PLAN DE EJECUCION



Este diagrama de estructura de árbol ilustra como las operaciones ocurren durante la ejecución de SQL. Cada paso en el plan de ejecución se asigna a un número de la columna ID de la tabla PLAN_TABLE y es representado por un "nodo". El resultado de cada operación de nodo es pasado a el nodo cercano más alto en el árbol.

Analizando el plan de ejecución, se puede identificar las declaraciones ineficientes en la aplicación.

En el plan de ejecución se describen las operaciones realizadas por ORACLE al momento de ejecutar una declaración en SQL.

Las herramientas utilizadas en esta declaración, pueden ayudar a identificar las operaciones **SQL** más lentas en la aplicación.

2.3.- MODIFICACION DE ESTADOS SQL EN APLICACIONES

La afinación de procesos depende grandemente de las estructuras usadas en las aplicaciones.
Cada estructura empleada se beneficia obteniendo mejores rendimientos por cada transacción en cada sesión.

2.3.1.- CONSIDERACION EN DATOS

La consideración sobre el manejo de los datos, es el primer paso al escribir cualquier aplicación. Al determinar la estructura de las tablas que contendran la parte esencial de la aplicación, considérese:

- La flexibilidad como fácil modificador de aplicación.
- Las bases para una ejecución óptima de la aplicación.

2.3.2.- CONSIDERACIONES FUNCIONALES EN ESTRUCTURAS

- Algunas estructuras ORACLE pueden mejorar la funcionalidad de cada aplicación. Entendiendo que estas estructuras ayudan a escribir óptimas aplicaciones en SQL que concuerden con:

  - Optimizador de ORACLE.
  - Indices.
  - Candados a nivel registro.

- Otras estructuras ORACLE mejoran la operación y la ejecución de aplicaciones de la base de datos, no importando que tipo de declaraciones se ejecute:

  - PL/SQL.
  - GENERADOR DE SECUENCIAS.
  - CLUSTERS.
  - PROCESAMIENTO DE ARRAYS.

2.4.- OPTIMIZADOR ORACLE

El optimizador ORACLE selecciona un plan de ejecución basado sobre el siguiente criterio:

- La sintáxis de las declaraciones SQL.
- Los predicados o condiciones de la claúsula WHERE de la declaración SQL.
- Las estructuras y definiciones de los objetos de la base de datos en la declaración SQL.
- Cualquier índice que exista sobre estos objetos de la base de datos.

Cada declaración en SQL procesada por el manejador, primero debe ser examinada por el optimizador ORACLE.

El optimizador ORACLE selecciona el plan de ejecución por cada sentencia SQL.

El plan de ejecución es una lista de los pasos u operaciones de la base de datos, que el RDBMS debe ejecutar en cada declaración SQL.

2.4.1.- EJEMPLOS DE OPTIMIZACION

El uso de "exist" y "not exist" aceleran a forzar el uso del indice al accesar los registros de la tabla, como se muestra en estos 2 ejemplos:

EJEMPLO No.1

```
select saldo, no_cuenta
from cartera_vencida
where no_cuenta not in
     ( select no_cuenta
          from died )
```

```
OPERATION       OPTIONS     OBJECT_NAME
----------      --------    ------------
FILTER
TABLE_ACCESS     FULL       Cartera_vencida
TABLE_ACCESS     FULL       Died
```

EJEMPLO NO.2

```
select saldo, no_cuenta
from cartera_vencida cv
where  not exists
     ( select no_cuenta
     from died m
          where m.no_cuenta=cv.no_cuenta)
```

```
OPERATION        OPTIONS        OBJECT_NAME
-------------    --------       ---------------
FILTER
TABLE_ACCESS     FULL           CARTERA_VENCIDA
TABLE_ACCESS     RANGE SCAN     DIED_IDX
```

EJEMPLO 3:

```
SELECT DNAME,DEPTNO
FROM DEPT
WHERE DEPTNO NOT IN
    (SELECT DEPTNO
      FROM EMP);
```

```
OPERATION          OPTIONS          OBJECT_NAME
---------------    -----------      --------------
FILTER
TABLE ACCESS       FULL             DEPT
TABLE ACCESS       FULL             EMP
```

-Este ejemplo muestra, la sintáxis seleccionada para la declaración SQL que puede tener impacto sobre como se ejecuta en ORACLE.

-Este muestra los planes de ejecuciones para dos declaraciones SQL que se ejecutan en un solo estado.

-Ambas declaraciones, regresan todos los departamentos en la tabla DEPT de los empleados que no se encuentran en la tabla EMP, donde cada declaración busca la tabla EMP como subquery.

-Se asume un índice, que puede acelerar el query de la tabla EMP, DEPTNO_INDEX,que ha sido creado sobre la columna DEPTNO de la tabla EMP.

-Una declaración SQL no puede usar un índice si la columna indexada no es llamada por la cláusula WHERE de la declaración.

-La tercera línea del plan de ejecución indica que esta declaración SQL ejecuta una búsqueda completa (FULL TABLE SCAN) en la tabla EMP. Por lo que la tabla EMP es leída en su totalidad a pesar de la existencia del subquery sobre la columna DEPTNO. Asi que se observa que el FULL TABLE SCAN es una operación que consume mucho tiempo.

EJEMPLO 4:

```
SELECT DNAME, DEPTNO
  FROM DEPT
    WHERE NOT EXISTS
        (SELECT DEPTNO
            FROM EMP
            WHERE DEPT.DEPTNO= EMP.DEPTNO)
```

OPERATION	OPTIONS	OBJECT_NAME
FILTER		
TABLE ACCESS	FULL	DEPT
INDEX	RANGE SCAN	DEPTNO_INDEX

- La cláusula WHERE NOT EXIST en el subquery, jala la columna DEPTNO de la tabla EMP y así el índice DEPTNO_INDEX es usado.

- En la tercera línea del plan de ejecución se observa el uso del índice.

-El índice DEPTNO_INDEX hace un RANGE SCAN que toma menos tiempo que el FULL TABLE SCAN de la tabla EMP que en la primera declaración. Además, en la primera parte de la ejecución, el FULL TABLE SCAN se ejecuta a la tabla EMP por cada DEPTNO en la tabla DEPT. Por estas razones, la segunda declaración SQL es más rápida que la primera.

2.4.2.- FULL TABLE SCAN

> Un query que selecciona más del 10% ó 15% de los registros de la tabla, puede ser ejecutado más rápido através de un FULL TABLE SCAN.

- Si un query no usa un índice, ORACLE necesita ejecutar una búsqueda completa de los registros sobre la tabla.

- ORACLE examina cada registro para determinar el criterio de la cláusula WHERE. Al buscar un solo registro con un índice puede ser considerablemente más rápido que buscarlo con FULL TABLE SCAN, dependiendo de la cantidad de registros almacenados en ella.

- Un FULL TABLE SCAN comprende la lectura de todas los registros de una tabla.

- Al ejecutar un FULL TABLE SCAN, ORACLE lee todos los bloques de la tabla aumentando el overhead.

- Cada bloque lee cada uno de los registros almacenados en el.

- Cada bloque únicamente se lee una vez.

- En acceso a las tablas con pocos bloques, un FULL TABLE SCAN, requiere menos I/O que un query indexado.

2.5.- OPTIMIZACION EN DECLARACIONES SQL

Los diferentes tipos de optimización son:

1.- POR QUERIES (SELECTS).
2.- NOTs.
3.- ORs.
4.- ORDER BY.
5.- GROUP BY.
6.- JOINS.

2.5.1.- OPTIMIZACION DE QUERIES (SELECTS)

Sobre una tabla pequeña, el optimizador de ORACLE ejecutará razonablemente más rápido un acceso secuencial, decidiendo no usar los índices.

Sin embargo, cuando se recuperan al menos el 25 % de los registros de una tabla muy grande, entonces el uso de índices o clusters pueden reducir considerablemente el tiempo del query.

2.5.2.- OPTIMIZACION NOTs

ORACLE no usa los índices cuando los predicados contienen una cláusula no equitativa como en (!= o NOT =) así que esto permite usar un índice sobre otro predicado,

Por ejemplo en:

```
WHERE X!=7   AND Y = 8
```

El índice no es usado por la columna X, pero si podría ser usado por la columna Y. Usualmente en queries con "NOT=" el número de registros regresados es grande. ORACLE lee el índice y entonces las tablas no requerirán I/O's extras.

Sin embargo, cuando el predicado contiene un NOT a la par con otros operadores, ORACLE transforma el predicado con lo cual los índices pueden ser usados en las cláusulas, como se indica:

OPERADORES EXISTENTES	OPERADORES TRANSFORMADOS
NOT >	<=
NOT >=	<
NOT <	>=
NOT <=	>

2.5.3.- OPTIMIZACION ORs

En las declaraciones SQL con cláusula OR puede ser que el optimizador use los índices bajo ciertas condiciones. Generalmente, la optimización ocurrirá si, las columnas en el predicado OR son indexadas, pero la optimización no puede ocurrir si algunas columnas no están indexadas, como:

```
SELECT *
 FROM EMP
     WHERE DEPTNO= 10   OR  JOB = 'CLERK'
```

Los índices existen en ambas columnas DEPTNO y JOB, entonces ORACLE usará ambos índices y ejecutará algo similar a la unión de los 2 queries:

QUERY 1:	QUERY 2:
SELECT * FROM EMP	**SELECT * FROM EMP**
WHERE DEPTNO=10	**WHERE JOB = 'CLERK'**
	AND DEPTNO ! = 10

Los predicados OR no son usados por el optimizador en los siguientes casos:

- Cuando la declaración SQL contienen un CONNECT BY.
- Cuando la declaración SQL contiene un outer-join.
- Cuando ORACLE determina que usos de indices no optimizarán el query.

La optimización también se aplica a cláusulas IN en la cual se traslada a ORs:

DEPTNO IN (X,Y,Z)

significará lo mismo como:

DEPTNO= X OR DEPTNO = Y OR DEPTNO = Z

2.5.4.- OPTIMIZACION ORDER BY

Aunque se tienen diferentes caminos para clasificar declaraciones SQL, es únicamente garantizado el uso de la cláusula ORDER BY al usarla en el final de las declaraciones SQL.

Para un query con cláusula WHERE, primero el dato que este encuentra en el criterio de selección es extraido, y entonces este es ordenado usando uno de los índices ó en su defecto la rutina sort/merge.

2.5.5.- OPTIMIZACION GROUP BY

Cuando se agrupan datos, se podrá eliminar registros no encontrados en el criterio de selección, sí usa una cláusula WHERE, los registros extras no podrán incurrir en el procesamiento. Por ejemplo:

```
SELECT JOB, AVG(SAL)
  FROM EMP
      WHERE JOB!='PRESIDENT'
      AND JOB !='MANAGER'
   GROUP BY JOB
```

pero no en:

```
SELECT JOB, AVG(SAL)
  FROM EMP
   GROUP BY JOB
       HAVING JOB!='PRESIDENT'
        AND JOB !='MANAGER'
```

por 2 razones importantes:

1.- Porque el QUERY 2 no tienen la cláusula WHERE, todos los registros son agrupados e incurren en el procesamiento HAVING. Este incluye registros los cuales puedan ser eliminados con la cláusula WHERE y así obtener más rápidez.

2.- La cláusula HAVING intenta eliminar datos agrupados basados sobre el criterio de grupo.Aunque su sintáxis es similar a la cláusula WHERE, HAVING no puede intercambiarse, ni hacer uso de los índices.

2.5.6.- OPTIMIZACION JOINS

Las reglas generales para optimizar el rendimiento de un join son:

1.- Las columnas usadas en las cláusulas join deben pertenecer a un índice.

2.- Los joins deben manejar tablas grandes que regresen pocos registros.

2.6.- RENDIMIENTO AUTOMATICO

1.- DISTINCT.
2.- GROUP BY.
3.- SUBQUERIES.
4.- RECUPERACION DE UN SOLO RENGLON CON TRAYECTORIAS DIRECTAS.

2.6.1.- POR DISTINCT

Dada una declaración SQL e incluyendo un query con uno o más DISTINCTS, ORACLE ordena los registros, descartando los duplicados. Los índices no son usados al procesarse con esta cláusula.

2.6.2.-POR GROUP BY

Los queries con GROUP BY se procesan similarmente en los queries con DISTINCT. ORACLE ordena los registros, agrupando durante el ordenamiento, y descartando los duplicados que se encuentran. También los índices no son usados.

2.6.3.-POR SUBQUERIES

Ciertos subqueries son transformados automáticamente transformados dentro del join, al usar la claúsula "IN" como en:

```
SELECT * FROM EMP
     WHERE EMP.DEPTNO IN
          (SELECT DEPTNO FROM EMP)
```

el cual se procesa como el siguiente query lógico (en la cual no es válida su sintáxis en SQL*PLUS).

```
SELECT EMP.A
  FROM EMP, D:
  ( SELECT DISTINCT DEPTNO FROM DEPT)
      WHERE D.DEPTNO= EMP.DEPTNO
```

2.6.4.-POR TRAYECTORIAS DIRECTAS

Cuando una tabla en la cláusula FROM se garantiza que apunte a un solo registro, esta tabla se procesa desde el primer join. Ejemplos:

- Trayectorias con índices unique en la cláusula WHERE = constante.
- Obtención sobre vistas simples (tal como SELECT AVG(SAL) FROM EMP).
- Con acceso a trayectorias directas ROWID.

CAPITULO

CAPITULO 3

HERRAMIENTA DE EJECUCION " TKPROF "

FACILIDAD SQL*TRACE

SQL*TRACE proporciona información de la ejecución de las declaraciones SQL y genera las siguientes estadísticas:

- EXECUTE, PARSE Y FETCH.
- Tiempos de enlace en el CPU.
- Lecturas físicas y lógicas.
- Número de registros procesados.

- **SQL*TRACE puede habilitarse para todas las declaraciones SQL ejecutadas, por sesión de usuario ó por instancia y se localizan dentro de un archivo TRACE específico.**

- **Para trasladar el archivo TRACE a una forma legible, ejecute el programa TKPROF.**

- **Opcionalmente, TKPROF también muestra el plan de ejecución de una declaración SQL, especificando esto con el argumento EXPLAIN PLAN sobre TKPROF.**

- **Considerar que al usar la facilidad del SQL*TRACE se incrementa el overhead.**

3.- "TKPROF" AFINACION DE APLICACIONES EN SQL*PLUS

Forma de uso:

1.- Poner los parámetros en el INIT.ORA.

2.- Correr la aplicación habilitando el TRACE.

3.- Usar TKPROF para reportar las estadisticas en forma legible.

3.1.- PARAMETROS EN EL INIT.ORA

(Habilitar o deshabilitar la facilidad SQL*TRACE)

sql_trace = false
sql_trace = true

Al poner este parámetro en TRUE se proporciona la información de afinación para todas las sesiones, ya que causa mucho overhead, se debe de levantar la base con el valor de TRUE solo si su propósito es coleccionar estadisticas a nivel base de datos.

Si lo que se desea es coleccionar estadísticas a nivel de usuario, deje el comando con el valor de FALSE.

Antes de correr la aplicación con SQL*TRACE, se declara los siguientes parámetros a nivel instancia:

TIMED_STATISTICS = TRUE

Proporciona el tiempo ofrecido por el SQL*TRACE, tal como el tiempo de enlace en el CPU. Habilitando TIMED_STATISTICS causa un tiempo extra de llamadas para operaciones de nivel bajo. Este parámetro también colecciona ciertas estadísticas en el monitoreo desde SQL*DBA.

MAX_DUMP_FILE_SIZE = n

Determina el máximo número de los bloques en disco para obtención de la salida del archivo TRACE. Si se trunca la salida, es necesario incrementar el número de bloques especificados por este parámetro y empiezar el TRACE nuevamente.

USER_DUMP_DEST= directory

El archivo TRACE es escrito sobre el directorio default identificado por el manejador (ORACLE- HOME/rdbms/log), para modificar el destino usemos este parámetro.

3.2.- HABILITANDO UNA SESION SQL*TRACE

Para habilitar SQL*TRACE para una sesión, se ejecuta:

ALTER SESSION SET SQL_TRACE TRUE;

Para deshabilitar:

ALTER SESSION SET SQL_TRACE FALSE;

SQL*TRACE se deshabilita automáticamente cuando se abandona la sesión.

3.3.- HABILITANDO EL SQL*TRACE PARA UNA INSTANCIA

Para habilitar SQL*TRACE para una instancia, se declara en el INIT.ORA, todos los parámetros vistos en el punto "Parámetros del INIT.ORA" y el parámetro "sql_trace=true".

Se coleccionan estadísticas de todos los usuarios de todas las sesiones. Sin embargo se puede,deshabilitar el TRACE de una sesión individual ejecutando en SQL:

ALTER SESSION SET SQL_TRACE FALSE;

Cuando el SQL*TRACE se habilita para una instancia, ORACLE crea archivos TRACE cada vez que el usuario se conecte y desconecte. Antes de correr TKPROF, se pueden añadir todos los archivos TRACE dentro de un solo archivo.

NOTA:

Después de arrancar la base de datos, los archivos TRACE creados, contienen datos que reflejan la actividad de los procesos de arranque, de manera individual contienen estadísticas que reflejan un aumento no proporcional de actividades de I/O así como varias actividades cache completas de la base de datos.

3.4.- ARCHIVOS SQL*TRACE Y SUS VERSIONES

Al habilitar la facilidad SQL*TRACE a nivel de instancia,ORACLE escribe un archivo TRACE cada vez que un usuario se conecte o desconecte. Si la operación del sistema mantiene múltiples versiones de archivos, asegura que los suficientes límites superiores para la numeración de archivos TRACE se puedan expresar y generar fácilmente en los archivos TRACE.

3.5.- EJECUCION DEL TKPROF

```
Sintáxis:

TKPROF tracefile outputfile
(SORT={SORTOPTION,SORTOPTION...})
[PRINT = n]
[EXPLAIN = USERNAME/PASSWORD]
```

3.- "TKPROF" AFINACION DE APLICACIONES EN SQL*PLUS

- TKPROF interpreta los archivos TRACE producidos por SQL*TRACE en formatos que se puedan interpretar fácilmente.

- TKPROF también puede ser usado para generar salidas EXPLAIN PLAN.

Para obtener información acerca del uso de este comando, teclee desde el prompt de UNIX simplemente " tkprof" sin ningun parámetro, donde:

tracefile	Este argumento es el nombre de el archivo SQL*TRACE donde las estadisticas SQL*TRACE fueron grabadas, al habilitar el trace.
outputfile	Este argumento es el nombre del archivo donde se escribirá la salida del TKPROF.
SORT	Este argumento opcional causa que las estadisticas en el archivo de salida se obtengan en el orden solicitado. Si se especifica más de una opción de clasificación, la salida es ordenada en forma descendente por la suma de los valores especificados en las opciones de orden.

3.5.1.-OPCIONES PARA EL "PARSE"

PRSCNT	Número de ocurrencias en el análisis sintáctico "parse".
PRSCPU	Tiempo del "CPU" utilizado en el "parse".
PRSELA	Tiempo de enlace utilizado en el "parse".

PRSPHR	Número de lecturas físicas durante el "parse".
PRSCR	Número de lecturas a los bloques de modo consistente durante el "parse".
PRSCU	Número de lecturas a los bloques de modo concurrentes durante el "parse".

3.5.2.-OPCIONES PARA EL "EXECUTE"

EXECNT	Número de ejecuciones ("execute").
EXECPU	Tiempo del "CPU" utilizado en el "execute".
EXEELA	Tiempo de enlace utilizado en el "execute".
EXEPHR	Número de lecturas físicas durante el "execute".
EXECR	Número de lecturas a los bloques de modo consistente durante el "execute".
EXECU	Número de lecturas a los bloques de modo concurrentes durante el "execute".
EXEROW	Número de líneas procesadas durante el execute.

3.5.3.-OPCIONES PARA EL "FETCH"

FCHCNT	Número de lecturas (fetchs).
FCHCPU	Tiempo del "CPU" utilizado en el "fetch".

FCHELA	Tiempo de enlace utilizado en el "fetch".
FCHPHR	Número de lecturas físicas durante el "fetch".
FCHCR	Número de lecturas a los bloques de modo consistente durante el "fetch".
FCHROW	Número de registros atraídos durante el fetch.

3.5.4.-PARAMETROS DEL TKPROF

PARAMETRO PRINT=n	Este argumento opcional causa la primera clasificación "n" (entera)de las operaciones SQL que fueron analizadas en la sesión TRACE ó en la sesión por instancia, impresas en el archivo de salida.
PARAMETRO EXPLAIN=username/password	Este argumento opcional, origina que TKPROF corra la declaración EXPLAIN PLAN sobre todas las declaraciones en el archivo TRACE, la declaración EXPLAIN PLAN produce planes de ejecución en SQL. Los algoritmos EXPLAIN PLAN con el username/password que se mejora en este argumento, se crea en la tabla PLAN TABLE, y borra el PLAN TABLE cuando termina la ejecución.

Los procesos del usuario deben tener los privilegios RESOURCE para usar el argumento EXPLAIN con el TKPROF.

3.6.- EJEMPLO

1.-Para analizar el TKPROF, es necesario checar que dentro del query a analizar, no exista, la sintáxis de la declaración EXPLAIN PLAN, es decir, debe mostrarse como a continuación se presenta:

```
SELECT * FROM EMP,DEPT  WHERE EMP.DEPTNO=DEPT.DEPTNO
```

2.- En este ejemplo, el archivo "tracefile" es 12_1005.TRC.

3.- Para obtener los resultados del TKPROF, debe de introducirse lo siguiente:

```
tkprof 12_1005.trc  PRU.SAL  SORT=(EXECPU,
FCHCPU) EXPLAIN= scott/tiger
```

Las estadísticas son agrupadas en el archivo de salida, PRU.SAL, clasificando los tiempos de CPU utilizados en el "execute" y los registros del "fetch" también por CPU. Todas las declaraciones de los registros TRACE y su plan de ejecución para cada declaración también se obtienen en PRU.SAL.

4.-Muestra de una salida TKPROF de la declaración SQL anterior con EXPLAIN ejecutada como comando opcional.

```
        count    cpu    elap     phys     cr    cur    rows
        -----    -----  -----    -----    ----  ----   -----

PARSE       1     88      98        3     37     0
EXECUTE     1      5      12        2      2     2       0
FETCH       1     10      18        2      2     2      14

  EXECUCION PLAN:
        MERGE JOIN
          SORT JOIN
            TABLE ACCESS (FULL) OF 'DEPT'
          SORT JOIN
            TABLE ACCESS (FULL) OF 'EMP'
```

NOTA:

En el capítulo 5, se explicará con más detalle este tema.

3.7.- ESTADISTICAS DE FACILIDAD EN SQL*TRACE

Los registros en SQL*TRACE corresponden a las declaraciones SQL que efectúan lecturas de los registros procesados, donde:

PARSE	El parse traslada la declaración SQL dentro de un plan de ejecución. Este paso incluye chequeos, como pudieran ser, autorización de entradas, seguridad, chequeos para la existencia de tablas, nombre de columnas y otros objetos referenciados.
EXECUTE	Los pasos de ejecución de las corridas de los comandos en el manejador.
FETCH	Los pasos para la recuperación de los registros que satisfacen el QUERY.

A continuación se describen las columnas que se proporcionan para una salida SQL*TRACE:

COUNT	El número de veces que una declaración SQL, es registrada (parse), ó ejecutada (execute)de el número de lecturas (fetch) emitidas por la declaración, y en el orden en el que la operación puede mejorar.
CPU	Es el tiempo total para el parse, fetch ó execute en milisegundos.
ELAP	Es el tiempo total de enlace para cada paso, en milisegundos.

PHYS	Es el número total de bloques de lectura,desde una columna de base de datos sobre el disco, para el paso de procesamiento.
CR	Es el número de buffers recuperados de manera consistente. La suma de lecturas consistentes y concurrentes, es igual al número total de bloques de datos accesados.
ROWS	Es el número total de registros procesados por la declaración SQL (no incluye registros procesados por subqueries).

- Para las declaraciones SELECT, el número de registros procesados aparecen en la línea de el fetch.
- Para UPDATE, DELETE e INSERT , el número de registros procesados aparecen en la línea de execute.

NOTA:

Las estadísticas tienen tiempo de resolución de milisegundos, esto significa que cualquier operación sobre un cursor que tome un milisegundo ó menos causaría que no se tomará exactamente.

3.8.- LLAMADA RECURSIVA

Para el orden de ejecución de un comando SQL, adicionalmente ORACLE ejecuta uno o más estados SQL.

Estos estados son nombrados " llamadas recursivas".

3.- "TKPROF" AFINACION DE APLICACIONES EN SQL*PLUS

Las llamadas recursivas son generadas cuando la información del diccionario de datos no se encuentra disponible en el diccionario de datos cache, pero que tendrá que ser recuperada de disco.

Si se trata de insertar un registro dentro de una tabla que no tenga suficiente espacio, ORACLE hace la llamada recursiva localizando el espacio dinámicamente.

Si una llamada recursiva ocurre cuando el SQL*TRACE esta habilitado, el TKPROF mostrará estadísticas sobre las operaciones SQL, así como sus tiempos de inicio.

Las estadísticas recursivas son claramente marcadas como declaraciones SQL recursivas. Nótese, que sin embargo, el tiempo de "CPU" , el tiempo enlazado "ELAP", las lecturas físicas "PHYS", las lecturas consistentes (CR) y las lecturas concurrentes (CUR), son incluídas en las estadísticas de las declaraciones SQL causadas por la ejecución recursiva.

CAPITULO

CAPITULO 4

" METODOS DE AFINACION PARA UN MEJOR RENDIMIENTO "

4.1.- CLUSTERS

4.1.1- DEFINICION

Un cluster es una estructura de base de datos que mejora la ejecución de las declaraciones SQL, en la unión de datos en múltiples tablas.

Un cluster produce un beneficio, si la aplicación frecuentemente selecciona el mismo grupo de registros desde una o más tablas. Por ejemplo, si siempre se procesan empleados por el número de departamentos y se crea un cluster por el número de departamento de la tabla de empleados, beneficiará el tiempo de respuesta, ya que un cluster permite que dicha relación de tablas tengan la misma extensión de espacio en disco, debibo a que los registros que realicen el join sean almacenados permanentemente.

4.1.2.- CARACTERISTICAS SOBRESALIENTES

En la creación de un cluster es necesario considerar lo siguiente:

- Se debe tener una columna con el mismo tipo, tamaño y nombre, es decir, si se tiene una columna llamada **DEPTNO** en las tablas **EMP** y **DEPT**, es necesario checar que esa columna sea numérica y que tenga el valor default en ambas tablas, si esto es correcto, esa columna recibirá el nombre de columna cluster.

-Todos los registros de las tablas que tengan el mismo valor en las columnas cluster, van hacer almacenados en la misma página(s) del disco, si es que no se encuentran en una sola.

-Todo valor distinto que aparezca en la columna cluster, será almacenado solamente una vez en la base de datos.

Por ejemplo, si en la tabla EMP y DEPT se realizó un cluster sobre la columna DEPTNO, los registros de EMP cuyo DEPTNO=10 y los registros para DEPT cuyo DEPTNO=10, serán almacenados en la misma página del disco, debido a que tienen el valor 10 en la columna cluster. El valor 10 podría ser almacenado solamente una vez, aunque existiera 3, 4 ó 5 registros en la tabla EMP ó DEPT que tengan DEPTNO=10.

Un ejemplo acerca de como son almacenados algunos registros en la base de datos de acuerdo a las tablas EMP y DEPT, haciéndose referencia a DEPTNO=10 sería la siguiente:

```
7782  CLARK    MANAGER   7839  09-JUN-81   2450   10 .....
7839  KING     PRESIDENT 7832  17-NOV-81   5000   10 .....
7934  MILLER   CLERK     7782  23-JAN-82   1300   10 .....
ACCOUNTING     ..........................................  10
```

En cambio cuando se crea un cluster quedaría:

```
10....  7782 CLARK   MANAGER  7839  09-JUN-81  2450      7839 KING
        PRESIDENT  7832  17-NOV-81  5000        7934 MILLER CLERK
        7782  23-JAN- 82  1300          ACCOUNTING  ..... .....
```

NOTA:

Es importante observar que en el segundo caso cuando se creó el cluster, se afecta la forma en que las tablas son físicamente almacenadas en disco, pero esto no quiere decir,que afecte la apariencia lógica de la tablas que se tiene originalmente.

4.1.3.- CRITERIO PARA LA SELECCION DE CLUSTERS

Para la creación de cluster en diversas tablas es necesario tener presente lo siguiente:

- Los clusters, mejoran la ejecución de la declaración SQL en la unión de tablas sobre la llave cluster. Tales declaraciones requieren menos I/O que si las tablas fuesen almacenadas individualmente.

- Los clusters también mejoran la ejecución de la declaraciones SQL de las tablas con ciertos queries basados sobre los valores de la llave cluster.

- Las declaraciones SQL que ejecuten un FULL TABLE SCAN únicamente de una tabla en un cluster, el espacio almacenado por dicho cluster puede ser más grande que si la tabla fuese almacenada individualmente. Un FULL TABLE SCAN de una tabla que tenga cluster requiere del acceso de todos los bloques y un cluster usualmente ocupa más bloques que cualquier tabla única que podría ser almacenada individualmente.

- Las declaraciones SQL que se inserten en un registro dentro de tablas con cluster ó valores actualizados de la llave cluster, pueden ser más grandes, si se insertan ó se actualiza sobre tablas únicas, teniendo en cuenta de que tales declaraciones requieren de un overhead para mantenimiento de la llave cluster.

- Para elegir si en las tablas se crea un cluster o no, es necesario considerar los beneficios e intercambios de los clusters de acuerdo a las necesidades de la aplicación.Por ejemplo, se crearán tablas con cluster en donde siempre existan aplicaciones cuya finalidad sea de unir algunas declaraciones, es decir, como se mencionó anteriormente si siempre se procesan empleados por el número de departamentos, podrá ser beneficó, si se crea un cluster por el número de departamento de la tabla de empleados.

- No se podrá crear cluster en las tablas, en donde se inserten datos constantemente, debido a que dicha inserción producirá una modificación en el cluster.

4.1.4.- CREACION DE UN CLUSTER

Para su creación, se tiene la siguiente sintáxis:

```
CREATE CLUSTER cluster
( column spec, column spec );
```

donde:

cluster	Es el nombre del CLUSTER que va tener incluida la columna(s) cluster especificada.
columna	Especifica el nombre de la columna de la tabla que corresponda a las columnas cluster, del cluster creado. Generalmente, la colum- na(s) cluster de una tabla será la columna (s) que correspondan a la llave primaria ó a una porción de la llave primaria.
spec	Describe el tipo y el tamaño de la columna precedida.

El valor de una llave cluster está comprendida de:

- ROWID (19 bytes).
- Almacenamiento interno para la columna (s) con cluster; es decir, a el tamaño de cada columna en bytes se le agrega 1 byte de overhead por cada columna.

Por ejemplo, si se tiene la siguiente declaración:

CREATE CLUSTER XYZ(ABC CHAR(5), MNO CHAR (7));

se creará un valor de llave cluster de 33 bytes porque:

- 19 bytes son del ROWID.
- 5 y 7 bytes son de las columnas ABC y MNO.
- 2 bytes son del overhead, debido a que existen 2 columnas.

por lo tanto 19 + 5 + 7 + 2 = 31.

4.1.4.1.- LLAVE PRIMARIA

Una llave primaria de una tabla es usada para identificar únicamente cada registro en la tabla y está comprendida por una o más columnas de una tabla. El valor de la llave primaria únicamente identifica un registro de una tabla, así que para cada valor de la llave primaria hay exactamente un registro y para cada registro hay exactamente un valor de llave primaria.

Una llave primaria puede comprender cualquier columna de la tabla. Generalmente, la llave primaria es listada primero dentro de la declaración CREATE TABLE. Por ejemplo:

Una tabla no lleva un orden específico, así que uno por si solo no puede disponer del registro número 5 de la tabla, sino que se debe hacer uso de la llave primaria para almacenar el registro especifico, es decir, si DEPTNO (nombre de la columna) es la llave primaria de la tabla DEPT, entonces al realizar una investigación de número de departamento 30 siempre se retomará un registro correspondiente al departamento 30.

EJEMPLO:

```
CREATE TABLE DEPT
( DEPTNO number primary key ) ;
```

ó

```
CREATE TABLE DEPT
( DEPTNO number
  primary key );
```

Toda llave primaria es almacenada en el diccionario de datos.

NOTA:

Aunque ORACLE no requiere que se especifique una llave primaria a una tabla, todas las tablas pueden tener una (y únicamente una) llave primaria. Si una tabla no tiene, esta puede contener registros duplicados.

La única función de una llave primaria es la ejecución de identificar únicamente un registro, siendo esta identificación buena como cualquier otra.

4.1.4.2.- EJEMPLO DE LA CREACION DE CLUSTER

```
SQL > CREATE CLUSTER EMP_DEPT
        ( DEPTNO number );
```

donde:

EMP_DEPT	**Es el nombre de el cluster.**
DEPTNO	**Hace referencia a la columna del cluster.**
number	**Tipo de dato.**

NOTA:

Cuando se crea un cluster, recordar siempre que si el tipo de dato es:

- **CHAR, siempre debe llevar el tamaño en paréntesis.**
- **DATE, no es necesario especificar el tamaño.**
- **NUMBER si se quiere especificar el tamaño, sino la máquina lo da por default.**

4.1.5.- CREACION DE UNA TABLA EN UN CLUSTER

4.1.5.1.- CARACTERISTICAS

-Todo cluster específico, dentro de la creación de una tabla, requerirá de su creación primeramente.

-Las tablas son añadidas a el cluster através de la cláusula cluster en la declaración CREATE TABLE.

-Un cluster puede contener como máximo, 16 tablas, aunque se adquiera de una ejecución negada en los clusters cuando se tenga un cluster con más de 4 ó 5 tablas juntas.

-Las tablas deben tener columnas del cluster que hayan sido descritas en el comando CREATE CLUSTER.

Una creación de una tabla con cluster, es como si se creara cualquier tabla ordinaria excepto que:

- **Se debe usar la cláusula CLUSTER para identificar las columnas cluster de la tabla y el nombre del cluster , en el cual la tabla formará parte de el.**

- **Al menos una de las columnas del cluster de la tabla debe de ser NOT NULL.**

4.1.5.2.- SINTAXIS

Para la creación de una tabla con un cluster específico, existen 2 formas:

1) **CREATE TABLE table (column spec, column spec)**
   CLUSTER cluster (cluster_column, cluster_column);

donde:

table	Especifica el nombre de la tabla.
column spec	Especifica el nombre de una columna junto con el tipo de valor declarado.
cluster	Es el nombre del cluster.
cluster_column	Identifica una columna cluster en la tabla. La columna(s) cluster deben tener el mismo tipo y tamaño de acuerdo a las columnas correspondientes, descritas en el comando CREATE CLUSTER.

2) CREATE TABLE table (column, column)
CLUSTER cluster (cluster_column, cluster_column)
as query;

donde:

column	Asume el tipo y el tamaño de la columna retornado por el query.
query	Retoma una serie de registros la cual son insertados en la tabla creada nuevamente. El query también determina el tipo y el tamaño de las líneas de la tabla. El número de columnas retornadas por el query deben ser las mismas, así como el número de columnas nombradas en paréntesis después del nombre de la tabla.

4.1.5.3.-EJEMPLOS DE LAS 2 FORMAS EXISTENTES PARA LA CREACION DE TABLAS CON UN CLUSTER

1) Creación de una tabla nombrada CL_DEPT, tal como la tabla DEPT , pero es un miembro de el cluster EMP_DEPT:

```
SQL > CREATE TABLE CL_DEPT ( DEPTNO number (2) not null,
                             DNAME  char (14),
                             LOC    char (13))
      CLUSTER EMP_DEPT (DEPTNO );
```

```
SQL > INSERT INTO CL_DEPT
      SELECT *
      FROM DEPT;
```

En este ejemplo se hace notar lo siguiente:

1.-Se crea primero una tabla con nombre **CL_DEPT**, junto con sus nombres, tipo y tamaño de las columnas en paréntesis en este caso, teniendo la propiedad NOT NULL la columna DEPTNO.

2.- Es necesario que al menos una columna en cualquier tabla con cluster tenga la propiedad de ser NOT NULL.

3.- Através del comando insert se van a introducir todos los registros existentes en **DEPT** a la tabla **CL_DEPT**.

2) Otra forma de crear una tabla con cluster es:

```
SQL > CREATE TABLE CL_EMP
      CLUSTER EMP_DEPT (DEPTNO)
      AS SELECT *
      FROM EMP;
```

Haciéndose notar que:

1.- En el comando **CLUSTER**, no es necesario que se ponga el tipo de valor y el tamaño en paréntesis, ya que este se declara en el momento en que se cree el **CLUSTER**.

2.- La declaración **AS**, nos va a permitr tener una copia de todo lo existente de la tabla **EMP** a la tabla **CL_EMP**.

NOTA:

Si una tabla contiene muchos registros con el mismo valor en la columna cluster, el cluster puede producir una reducción substancial en el almacenamiento en disco.

Si una tabla contiene muchos valores en la columna cluster y únicamente pocos registros tienen el mismo valor, el cluster puede incrementar su valor al almacenarse en disco.

3) Tener presente siempre que para la creación de cualquier tabla con un cluster, como en este caso las tablas EMP y DEPT , es necesario crear primero el cluster llamado PERSONNEL introduciéndose:

```
CREATE CLUSTER PERSONNEL
( department_ number      number );
```

después se teclea:

```
CREATE  TABLE EMP                    CREATE TABLE DEPT
( EMPNO nuber not null,              ( DEPTNO    number   not null,
  DEPTNO number not null )             DNAME     char (9),
CLUSTER personnel ( DEPTNO);           LOC       char (9))
                                     CLUSTER personnel (DEPTNO);
```

NOTA:

Cuando se crea un tabla, no es necesario que se declare el nombre de la columna en la creación del cluster, igual a cuando se hace mención del cluster dentro de la declaración CREATE TABLE, sino que lo más importante es que correspondan los tipos y los tamaños de los datos.

4.1.6.- CREACION DE INDICES EN UN CLUSTER

La creación de un índice sobre un cluster permite que se tenga un acceso más rápido a los registros dentro de un cluster.

Esta creación es requerida antes de que cualquier operación DML sea ejecutada sobre tablas en el cluster.

SINTAXIS

CREATE INDEX nombre del índice ON CLUSTER nombre del cluster.

EJEMPLO:

```
CREATE INDEX iname ON CLUSTER cname;
```

NOTA:

Todas las columnas no son declaradas en el enunciado CREATE INDEX, debido a que el índice es construido sobre las columnas que se especifican dentro de la llave cluster, es decir, aquellas que son definidas en la declaración CREATE CLUSTER.

EJEMPLO:

```
CREATE CLUSTER PERSONNEL ( DEPARTMENT_NUMBER  NUMBER );
```

Esta declaración define un cluster nombrado PERSONNEL que contiene las tablas EMP y DEPT.

Para la creación de dichas tablas EMP y DEPT se teclea lo siguiente:

```
CREATE  TABLE EMP                CREATE TABLE DEPT
( EMPNO nuber not null,          ( DEPTNO   number  not null,
  DEPTNO number not null )         DNAME    char (9),
CLUSTER personnel ( DEPTNO);       LOC      char (9) )
                                 CLUSTER personnel ( DEPTNO);
```

Para la creación en el índice se introduce lo siguiente:

```
CREATE INDEX  IDX_PERSONNEL  ON CLUSTER  PERSONNEL;
```

NOTA:

Teniendo creado el índice cluster, se puede insertar registros dentro de la tabla EMP ó DEPT.

Cuando se tiene un índice, el orden de las columnas en las llaves cluster afecta la estructura de el índice en el cluster.

4.1.7.- ELIMINACION DE UNA TABLA CON CLUSTER

Para poder hacer una eliminación de una tabla con un cluster, se realizan los siguientes pasos:

1.- Determinar los índices que son definidos sobre la tabla original.

2.- Crear una nueva tabla fuera del cluster con la misma columna de acuerdo a la tabla que se quiera eliminar y copear el original de los datos de la tabla en este,eliminando después con el comando drop la tabla original y automáticamente SQL*PLUS borrará los índices de la tabla.

3.- Renombrar la nueva tabla con el mismo nombre de la tabla original.

EJEMPLO:

```
SQL > CREATE TABLE CL_DEPT
      CLUSTER EMP_DEPT ( DEPTNO )
      AS SELECT *
      FROM DEPT;
```

1.- Para eliminar la tabla CL_DEPT desde el cluster EMP_DEPT se debe primero determinar que indices están definidos sobre CL_DEPT.

Para conocer esto, es necesario explicar lo siguiente:

Una tabla llamada INDEXES, lista los indices que hayan sido creados en las tablas existentes, para tener una información más completa acerca de este índice, es necesario que se consideren las siguientes columnas:

INAME	Despliega el nombre del indice.
ICREATOR	Despliega el creador del indice.
TNAME	Despliega el nombre de la tabla indexada.
CREATOR	Despliega el nombre del creador de la tabla indexada.
COLNAMES	Lista el nombre de la columna(s) indexadas.
INDEXTYPE	Despliega el tipo de indice que se trate, es decir, NON UNIQUE ó UNIQUE.

EJEMPLO:

```
SQL > COLUMN  INAME      FORMAT  A10;
SQL > COLUMN  ICREATOR   FORMAT  A10;
SQL > COLUMN  TNAME      FORMAT  A10;
SQL > COLUMN  CREATOR    FORMAT  A10;
SQL > COLUMN  COLNAMES FORMAT  A10;
SQL > SELECT *
      FROM INDEXES;
```

Resultado:

INAME	ICREATOR	TNAME	CREATOR	COLNAMES	INDEXTYPE
DEPT_DEPTN	JON	DEPT	JON	DEPTNO	NON UNIQUE
EMP_ENAME	JON	EMP	JON	ENAME	NON UNIQUE
EMP_DEPTNO	JON	EMP	JON	DEPTNO	NON UNIQUE

NOTA:

El nombre que aparezca en la columna ICREATOR y CREATOR no siempre es el mismo.
La cláusula COLUMN, mencionada anteriormente, nos permite tener una visión explicita.
La tabla INDEXES, es una tabla del diccionario de datos.

2.- Crear una copia de la tabla con cluster y después proceder a la eliminación de la tabla original, es decir,

```
SQL > CREATE TABLE UNCL_ EMP
        AS  ( SELECT *
                FROM CL_EMP );

SQL > DROP TABLE CL_EMP;
```

3.- Renombrar la nueva tabla

```
SQL > RENAME  UNCL_EMP  TO CL_EMP;
```

En este caso se renombro con el nombre de la tabla que fue borrada, pero se puede nombrar con otro nombre, haciéndose incapie en el nombramiento, se sugiere que se nombre con el mismo de la tabla borrada para evitar confusiones.

4.1.8.- TABLAS EXISTENTES EN UN CLUSTER

Es fácil conocer que tablas pertenecen a un clusters tecleando únicamente:

```
SQL > SELECT CLNAME, TNAME
      FROM CLUSTERS;
```

donde:

CLNAME	**Despliega el nombre del cluster.**
TNAME	**Despliega el nombre de las tablas.**

EJEMPLO:

CLNAME	TNAME
DEPT_EMP	CL_EMP
DEPT_EMP	CL_DEPT

Este ejemplo establece, un cluster con nombre (DEPT_EMP), conteniendo 2 tablas nombradas CL_EMP y CL_DEPT.

NOTA:

Conociendo en este caso el nombre de las tablas, se puede conocer los índices que han sido definidos en dicha tabla, através del siguiente comando:

```
SELECT INAME, COLNAMES
FROM INDEXES
WHERE TNAME = 'CL_EMP';
```

Recordar que las tablas INDEXES y CLUSTERS, son tablas del diccionario de datos.

4.1.9.- ELIMINACION DE UN CLUSTER

Para poder efectuar esta acción es necesario tener presente de que antes de que se borre un cluster, debe primero ser borrado todas las tablas que son miembros de el cluster con el comando DROP TABLE.

SINTAXIS:

DROP TABLE nombre de la tabla;

Una vez realizado esto, se podría borrar el cluster con el siguiente comando:

SINTAXIS:

DROP CLUSTER nombre del cluster;

4.1.10.- ORDENAMIENTO DE TABLAS EN UN CLUSTER

Este punto es muy importante, ya que nos especifica el orden en la cual las tablas con clusters son creadas, llamadas o referenciadas en las declaraciones SQL que efectúan la ejecución. Para esto, es necesario seguir un orden en las tablas con cluster:

1.-Crear tablas con cluster en orden ascendente de acuerdo al tamaño, es decir, la tabla más pequeña con menos registros en cada valor de llave cluster, debe ser creada primero.

La tabla más grande con más registros para cada valor de llave cluster debe ser creada al último.

2.-Es necesario llamar las tablas con cluster en orden ascendente de acuerdo al tamaño.

4.2.- GENERADOR SEQUENCE

4.2.1.- PROPOSITO

Establece un objeto desde la base de datos, permitiendo que múltiples usuarios puedan generar una secuencia como identificación única, eliminando la serialización y mejorando la concurrencia de la aplicación.

La secuencia de números puede ser usada para generar una llave primaria automáticamente.

4.2.2.- CARACTERISTICAS

- La creación de secuencias, es nueva característica para ORACLE V.6, ya que anteriormente los valores secuenciales únicamente podían ser originados através de programas, debido a que un nuevo valor de llave primaria podría ser obteni - do, seleccionando el valor originado más reciente e incrementando esto. Este método requería de un cierre durante la transacción y esto producía que múltiples usarios esperaran a el valor próximo de la llave primaria, conocido como serialización.

- Si una aplicación usa valores únicos o secuenciales para cualquier identificación única, se puede generar através del generados SEQUENCE.

- Toda definición de secuencias es disponible a cualquier usuario.

- Una secuencia puede ser accesada e incrementada por múltiples usuarios sin esperar ningún tiempo para poder hacerlo.

- Un generador SEQUENCE no espera que una transacción que haya terminado una secuencia termine, antes de que otra secuencia pueda ser incrementada otra vez.

- Una secuencia puede ser alterada, borrada o actualizada.

- Puede contener un sinónimo.

- El número de secuencias ORACLE para cualquier secuencia creada es automáticamente generada por ORACLE.

- Los generadores de números de secuencias ORACLE, permiten la generación simultánea de múltiples números de secuencia garantizando que cada número de secuencia sea único.

- Para observar las secuencias creadas, se tiene que accesar a USER_SEQUENCES y ALL_SEQUENCES através de:

```
SQL > SELECT * FROM USER_SEQUENCES;
```

4.2.3.- CREACION DE UNA SECUENCIA

SINTAXIS:

```
CREATE SEQUENCE sequence
INCREMENT BY n
START WITH n
MAXVALUE n
MINVALUE n
CYCLE | NOCYCLE
CACHE | NOCACHE
```

Donde:

sequence .- Nombre de la secuencia.

INCREMENT BY .- Determina el intervalo entre el número de secuencias. Si el incremento es negativo, entonces la secuencia descenderá; pero si es positivo, entonces ascenderá. Su valor default es 1.

START WITH .- Es el primer número de secuencia a ser creado. El valor default es el valor declarado en MINVALUE para el ascenso de secuencias y el valor declarado en MAXVALUE es para el descenso de secuencias. Esta cláusula se usa cuando una secuencia ascendente no empiece con el valor declarado en MINVALUE o cuando una secuencia descendente no empiece con el valor de MAXVALUE.

MINVALUE .- Es el valor mínimo de la secuencia generada.

MAXVALUE .- Es el valor máximo de la secuencia generada.

CACHE .- Localiza el número de secuencia para que sea almacenado en memoria, permitiendo tener una generación más rápida en los números de secuencias.
Esta cláusula es útil para aplicaciones que se requieren continuamente en una aplicación. Su valor default es 20. El valor especificado para CACHE debe ser menor al que se tenga en (MAXVALUE-MINVALUE).

NOCACHE .- No permite ningún almacenamiento en memoria de secuencuencias.

CYCLE - Especifica el número de secuencias adicionales que podrían ser generados una vez que la secuencia es terminada, es decir, si se especifica esta claúsula entonces la secuencia retomará el valor a MINVALUE después de investigar el valor de MAXVALUE para secuencias ascendentes, en cambio para secuencias descendentes, la secuencia retomará el valor de MAXVALUE después de investigar el de MIN - VALUE.

NOCYCLE .- No especifica el número de secuencias adicionales que podrían ser generadas una vez que la secuencia termine.

EJEMPLO:

1) CREATE SEQUENCE SEQ9
   INCREMENT BY 10
   START WITH 50
   MINVALUE 1
   MAXVALUE 150
   CACHE 15
   CYCLE;

En este ejemplo se hará lo siguiente:

1.- Se creará una secuencia cuyo nombre es SEQ9.
2.- Tendrá un incremento en cada secuencia de 10.
3.- El primer valor de secuencia será primero de uno y después de 61.
4.- En este caso no se toma en cuenta el valor de MINVALUE debido a que se puso la claúsula START WITH.
5.- El valor máximo de secuencias es de 150.

6.- Serán almacenados en CACHE 15 valores.
7.- Entrará en un ciclo, es decir, cuando se quiera empezar una secuencia después de que se haya alcanzado el límite predefinido, con CYCLE se vuelve a empezar a partir de los valores declarados MAXVALUE y MINVALUE.

NOTA:

Más adelante se darán más ejemplos y explicación acerca de los claúsulas mencionadas.

4.2.4.- VALORES DEFAULT EN UNA SECUENCIA

Los valores default en una secuencia son designados automáticamente, por ejemplo, si no se especifica ninguno de los parámetros mencionados anteriormente, se incrementará una secuencia ascendente que empiece con 1 y se incremente de 1 en 1 hasta que no supere el límite. Pero si únicamente se especifica INCREMENT BY -1 , se creará una secuencia descendente que empiece con -1 y se decremente hasta no ser igual al valor mínimo, es decir, (10 e 27) -1.

Una secuencia creciente permite:

- Tener un crecimiento sin saltos.
- Detención en un límite predefinido.

Cuando se crea una secuencia que crece sin saltar, no se especifica un valor para MAXVALUE para secuencias ascendentes ó un valor para MINVALUE para secuencias descendentes. Sino que se especifica NOCYCLE.

Al crear una secuencia que se detenga en un límite predeterminado, es necesario especificar un valor MAXVALUE para un límte de secuencias ascendentes ó un valor MINVALUE para secuencias descendentes ó especificar NOCYCLE.

Cualquier intento que se haga para generar un número de secuencia una vez que se ha alcanzado el límite, retornará un error.

Cuando se quiera empezar una secuencia después de que se haya alcanzado el límite predefinido, es necesario especificar valores para MAXVALUE y MINVALUE y especificar CYCLE.

La cláusula START WITH únicamente afecta al primer número de secuencia.

4.2.5.- PARAMETROS EXISTENTES EN UNA SECUENCIA

4.2.5.1.- PARAMETRO NEXTVAL

Permite generar un número de secuencias de una secuencia especificada. Una referencia a NEXTVAL causa un número de secuencias a ser generadas.

Su sintáxis es:

sequence.NEXTVAL

Donde:

sequence.- Es el nombre de la secuencia creada.

4.2.5.2.- PARAMETRO CURRVAL

Permite conocer el valor actual de la secuencia. Una referencia a NEXTVAL para una secuencia dada, muestra el número de secuencia actual a ser acupado en CURRVAL.

NEXTVAL debe ser usado a generar un número de secuencias en la sesión actual antes de consultar a CURRVAL.

Su sintáxis es:

sequence.CURRVAL

Donde:

sequence.- Es el nombre de la secuencia creada.

Ambos parámetros CURRVAL Y NEXTVAL pueden ser usados dentro:

- La cláusula SELECT de una declaración SELECT (excepto vistas).
- La lista de valores de una declaración INSERT.
- La expresión SET de una declaración UPDATE.

Pero no son usados:

- Dentro de subqueries.
- Dentro de una lista SELECT de una vista.

- En la cláusula DISTINCT.
- En la cláusula ORDER BY, GROUP BY y HAVING de la declaración SELECT.
- En operadores como (UNION, INTERSECT, MINUS).

EJEMPLO:

Después de que una secuencia se haya creado, este puede ser usado a generar números de secuencias únicos, por ejemplo:

1) El usuario ELLY crea una secuencia nombrada ESEQ através al introducir:

```
CREATE SEQUENCE ESEQ INCREMENT BY 10;
```

La primera referencia a ESEQ.NEXTVAL retornará 1 y la siguiente referencia retornará 11, ya que cada referencia subsecuente se incrementará de 10 en 10 de acuerdo al último valor obtenido.

Siempre el parámetro NEXTVAL debe ser colocado después del nombre de la secuencia precedido por un punto (.), es decir:

```
ESEQ.NEXTVAL
```

NOTA:

No se puede referenciar a NEXTVAL más de una vez en una misma declaración para una secuencia dada. Las múltiples referencias a NEXTVAL en una misma declaración SQL causará que todas las referencias usen el mismo número de secuencias.

2) Elizabeth puede determinar el valor actual de SEQ1, existente en la tabla DUAL, usando el parámetro CURRVAL, es decir:

```
SELECT SEQ1.CURRVAL
FROM DUAL;
```

4.2.6.- COMANDOS EXISTENTES EN UN SECUENCIA

4.2.6.1.- COMANDO INSERT.

Dentro de una inserción, se puede hacer uso del parámetro NEXTVAL, una vez creada la secuencia, es decir:

1) INSERT INTO EMP
VALUES (ESEQ.NEXTVAL, ' LEWIS', 'TIMOTH');

asumiendo que la tabla EMP tiene 3 columnas únicamente (EMPNO, LAST_NAME, FIRST_NAME).

Es necesario considerar, que el valor de EMPNO dependerá de:

- El valor existente en la cláusula INCREMENT BY, cuando se haya creado la secuencia.
- Si no existe la cláusula INCREMENT BY, habrá un incremento de 1.

Cuando un número de secuencia es generado, la secuencia es incrementada, independientemente de que la transacción se le haya dado Commit ó Rollback.

Si 2 usuarios están accesando a la misma secuencia al mismo tiempo, el número de secuencias de uno u otro usuario puede tener espacio, debido a que el número de secuencia está siendo generado por el otro usuario.

Dos usuario nunca tendrán el mismo número de secuencia generado por la misma secuencia.

2) Considerando que EMP tendrá únicamente 2 columnas (EMPNO, PROJNO).

INSERT INTO EMP_PROJ
VALUES (ESEQ.CURRVAL, 101);

EMPNO, es este caso tendrá el valor actual existente de acuerdo al último NEXTVAL asignado.

Para observar el último número de secuencia que se genere dentro de una sesión se teclea:

SELECT ESEQ.CURRVAL
FROM any_table;

4.2.6.2.- COMANDO UPDATE

Para explicar este comando dentro de una secuencia se analizará el siguiente ejemplo:

Si se quiere manufacturar una planta que hace televisiones, es necesario crear un generador de número de secuencias para construir números seriales por cada televisión

producida. Un número de televisiónes al azar son probadas y cada grabación de prueba es actualizada con el número serial de dicha televisión.

Usando una secuencia TV_SERIAL, la inserción de los registros dentro de la tabla TV y la actualización de la tabla TEST_RESULT podría ser escrita como:

```
INSERT INTO TV
VALUES( TV_SERIAL.NEXTVAL, SYSDATE, '19 INCH ');

UPDATE TEST_RESULT
SET TV_NUMBER= TV_SERIAL.CURRVAL
WHERE TEST_NUMBER =1015;
```

Todos los registros que tengan la columna TEST_NUMBER =1015 cambiará el valor de la columna TV_NUMBER al valor que tenga TV_SERIAL.CURRVAL.

4.2.6.3.- COMANDO DROP

Permite eliminar la secuencia especificada.

SINTAXIS:

```
DROP SEQUENCE usuario.nombre de la secuencia;
```

EJEMPLO:

```
DROP SEQUENCE ELLY.ESEQ;
```

Donde:

ELLY.- Es el usuario.
ESEQ.- Nombre de la secuencia.

Un método para empezar una secuencia, es haciendo una eliminación y recreación de este nuevamente. Por ejemplo, si se tiene una secuencia de 150 y nos gustaría empezar en la secuencia 25, se podrá:

1.- Borrar la secuencia.
2.- Crear esto con el mismo nombre y empezar con el valor de 25 a traves de la cláusula START WITH, es decir:

```
CREATE SEQUENCE ESEQ
START WITH 25;
```

4.2.6.4.- COMANDO ALTER

Permite cambiar las opciones que tenga el generador SEQUENCE, es decir;

- Cambia el incremento futuro entre el número de una secuencia.
- Cambia el valor de MINVALUE ó MAXVALUE ó elimina uno ó ambos.
- Cambia el parámetro CACHE ó lo elimina.

SINTAXIS:

ALTER SEQUENCE nombre de la secuencia;

EJEMPLO:

1) Para obtener un valor máximo y nuevo para la secuencia ESEQ se introduce:

ALTER SEQUENCE ESEQ MAXVALUES 1500;

2) Para añadir CYCLE y CACHE se introduce:

ALTER SEQUENCE ESEQ CYCLE CACHE 5;

4.2.7.- SEQUENCE CACHE

4.2.7.1.- CARACTERISTICAS

- Un número de secuencias pueden ser almacenada en el SEQUENCE CACHE en el SGA (system global área).
- Un número de secuencias pueden ser accesados más rapidamente através del SEQUENCE CACHE que son leídas desde el disco.
- Consiste de entradas. Cada entrada puede tener números de secuencias para una secuencia única.
- Puede tener todos las secuencias usadas concurrentemente en la aplicación.
- Incrementa el número de valores para cada secuencia dada en el SEQUENCE CACHE.

4.2.7.2.- NUMERO DE ENTRADAS EN EL CACHE SEQUENCE

-Cuando una aplicación accesa una secuencia en el SEQUENCE CACHE, los números de secuencias son leidos rapidamente. Sin embargo, si una aplicación accesa una secuencia que no está en el CACHE, la secuencia debe ser leida desde el disco a el cache antes de que los números de secuencias sean usadas.

-Si la aplicación usa muchas secuencias concurrentes, el SEQUENCE CACHE no debe ser grande para tener todas las secuencias, en este caso, se debe accesar números de secuencias que requieren de lecturas de disco frecuentemente.

-Para tener un acceso rápido de todas las secuencias, se debe estar seguro de que el CACHE tenga entradas grandes para leer todas las secuencias usadas concurrentemente en las aplicaciones.

-El número de entradas en el SEQUENCE CACHE es determinado por el parámetro SEQUENCE_CACHE_ENTRIES del INIT.ORA.

-El valor default para este parámetro es de 10 entradas.

-Cada entrada en el CACHE tiene una secuencia única, necesitándose una entrada CACHE para cada secuencia en uso concurrente.

-Si el valor para el SEQUENCE_CACHE_ENTRIES es bajo, I/O ocurrirán.

-La declaración CREATE SEQUENCE CACHE define la cantidad de números para una secuencia CACHE, que será usada antes de que se vaya al disco a localizar el próximo número.

-Las secuencias que son creadas con el CREATE SEQUENCE NOCACHE no reside en el SEQUENCE CACHE , asi que estas secuencias tienen ser escritas a el diccionario cada vez que se use.

-El rango de entradas que se pueden declarar en el SEQUENCE_ CACHE_ ENTRIES es de 10 a 32,000.

4.2.7.3.- NUMERO DE VALORES PARA CADA ENTRADA EN EL SEQUENCE CACHE

-Cuando una secuencia es leida dentro de un SEQUENCE CACHE, los valores de secuencia son generados y almacenados en una entrada CACHE. Estos valores pueden ser almacenados rápidamente.

-El número de valores de secuencias almacenados en el cache es determinado por el parámetro CACHE, en la declaración CREATE SEQUENCE. El valor default para este parámetro es 20.

EJEMPLO:

Esta declaración CREATE SEQUENCE, crea la secuencia SEQ2 con 50 valores de secuencia que son almacenados en el SEQUENCE CACHE:

```
1)  CREATE SEQUENCE SEQ2
    CACHE 50;
```

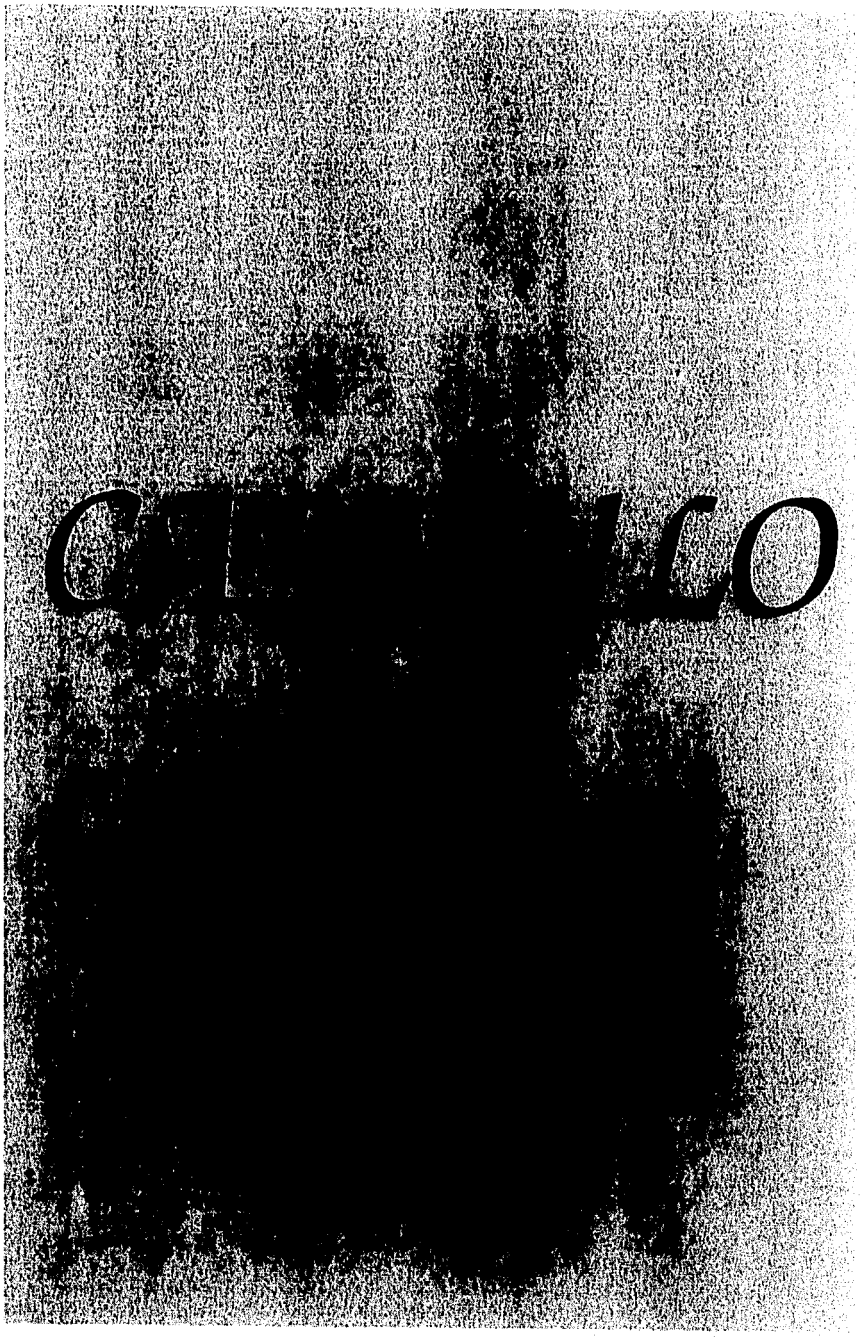
Los primeros 50 valores de SEQ2 pueden ser leídos desde el CACHE. Cuando el valor 51 es accesado, los próximos 50 valores serán leídos desde el disco.

En cambio, si se selecciona un valor más alto para CACHE, permitirá que se accesen más números de secuencias sucesivos con pocas lecturas de disco a el SEQUENCE CACHE.

En caso de que exista una falla por ejemplo en ORACLE, todos los valores secuenciales en el CACHE serán perdidos.

```
2)  CREATE SEQUENCE SEQ3
    NOCACHE;
```

Se puede usar la opción NOCACHE en la declaración CREATE SEQUENCE para no almacenar datos de una secuencia particular en el SEQUENCE CACHE. En este caso cada acceso a la secuencia requerirá una lectura a disco, por supuesto estas lecturas serán más lentas a comparación del CACHE.



CAPITULO 5

" APLICACIONES "

5.1.- INTRODUCCION

Dentro de este capitulo, se analizarán las 2 herramientas estudiadas anteriormente, "EXPLAIN PLAN" y "TKPROF", aplicadas a 5 programas realizados en SQL*PLUS.

Através de ellas se analizará que todos los objetos en dichos programas sean los correctos, tales como índices y clusters, en caso de no serlo, proceder a su creación.

5.2.- EJEMPLO NO.1

```
SELECT DNAME, DEPTNO
FROM DEPT
WHERE DEPTNO NOT IN (
                      SELECT DEPTNO
                      FROM EMP )
```

En este ejemplo pequeño, se explicará con detalle todos los pasos para analizar el EXPLAIN PLAN y el TKPROF, dentro de él se hará uso de 2 tablas llamadas EMP y DEPT, se observará que la claúsula NOT IN, no forza a usar el índice, siempre y cuando no exista la cláusula WHERE (como se verá más adelante).

5.2.1.- EXPLAIN PLAN

Para analizar esta herramienta, es necesario proceder a los siguientes pasos:

1.- Introducir la sintáxis del EXPLAIN PLAN como se muestra a continuación:

```
EXPLAIN PLAN
SET STATEMENT_ID='EJEMPLO1'
INTO PRU1
FOR
  SELECT DNAME, DEPTNO
  FROM DEPT
  WHERE DEPTNO NOT IN (
                        SELECT DEPTNO
                        FROM EMP )
```

Para ver el análisis de dicho ejemplo, es necesario que se introduzca en otro archivo , con extensión sql lo siguiente:

```
SELECT OPERATION,OPTIONS,OBJECT_NAME,ID,PARENT_ID, POSITION
FROM PRU1
WHERE STATEMENT_ID=' EJEMPLO1'
ORDER BY ID
```

Una vez teniendo estos 2 archivos, es necesario ejecutar dentro del prompt de SQL, lo siguiente:

1) El nombre del archivo donde fue creada la tabla PLAN_TABLE (recordando que no es necesario llamarla de esta forma, en este caso es llamada PRU1), a-través de esta instrucción:

```
SQL> START TABLE;
```

donde:

TABLE.- Es el nombre del archivo que contiene la tabla.

2) Ejecutar el archivo donde esta declarada la sintáxis del EXPLAIN PLAN, introduciéndose:

```
SQL> START EJEM1;
```

Cuando esta declaración es correcta deberá aparecer en la parte final la palabra:

"explained"

3) Ejecutar el archivo, donde están declaradas las opciones que aparecerán como resultado, introduciéndose:

```
SQL> START DESP;
```

Teniendo como resultado lo siguiente:

OPERATION	OPTIONS	OBJECT_NAME	ID	PARENT_ID	POSITION
FILTER			1		
TABLE ACCESS	FULL	DEPT	2	1	1
TABLE ACCESS	FULL	EMP	3	1	2

Para explicar con detalle estos resultados de salida, existen 2 formatos de anidación como se mencionó anteriormente através de:

1.- Comando LPAD.
2.- Arbol estructurado.

Unicamente en este ejemplo se analizarán los 2 formatos:

1.- COMANDO LPAD

Este comando nos mostrará los niveles de las operaciones de acuerdo a como fueron realizándose, para esto se introduce:

```
SELECT LPAD (' ', 2*LEVEL)|| ' ' ||OPTIONS|| ' ' || OBJECT_NAME  QUERY PLAN
FROM PRU1
WHERE STATEMENT_ID= ' EJEMPLO1'
CONECCT BY PRIOR ID= PARENT_ID  AND STATEMENT_ID= 'EJEMPLO1'
START WHIT ID=1;
```

Resultado:

```
 QUERY  PLAN
----------------------
   FILTER
     TABLE ACCESS    FULL    DEPT
     TABLE ACCESS    FULL    EMP
```

En este caso, primero se tuvo que hacer un FULL TABLE SCAN a las 2 tablas y después un FILTER.

2.- ARBOL ESTRUCTURADO

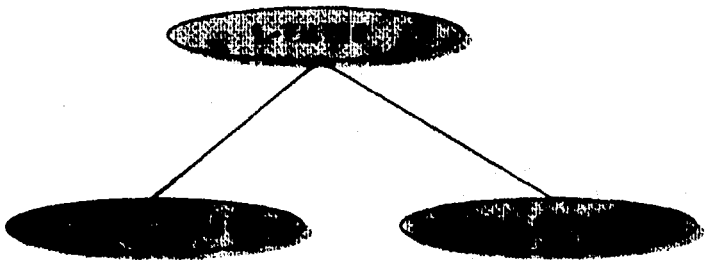
Nos mostrará como las operaciones fueron realizándose, es decir, se considera de abajo a arriba, tomando en cuenta lo números que aparecen en el ID y PARENT_ID.

El árbol mostrado abajo, se construye de la siguiente manera:

1.-FILTER contiene dentro del ID el número 1, por lo tanto se considera como padre del árbol.

2.-TABLE ACCESS de la tabla DEPT, contiene dentro de ID el número 2, pero dentro del PARENT_ID el número 1, por lo tanto, se considera como hijo y FILTER como padre.

3.-TABLE ACCESS de la tabla EMP, contiene dentro de ID el número 3, pero dentro del PARENT_ID el número 1, por lo tanto, se considera como hijo y FILTER como padre.



En este caso se realizó primero un TABLE ACCESS a las 2 tablas y luego un FILTER (es decir una restriccion de datos).

La opción POSITION dentro del análisis del EXPLAIN PLAN, nos señala que primero se realizará aquella opción que tenga el número más grande existente en dicha tabla, siendo en este caso el número 2 realizando un FULL TABLE SCAN a la tabla EMP, después a la tabla DEPT y por último se hace una restricción de datos.

NOTA:

Para concluir esta parte del análisis del EXPLAIN PLAN, es necesario aclarar, como se dijo anteriormente, se realizó FULL TABLE SCAN a dichas tablas debido a que no se hizo acceso al indice al no usar la cláusula WHERE y la declaración del join en ella.

5.2.2.- ANALISIS DEL TKPROF

Para realizar el análisis de esta herramienta se recomienda chocar antes de todo, los parámetros localizados dentro del INIT.ORA como son:

```
sql_trace           = false
sql_trace           = true
timed_statistics    = true
max_dump_file_size  = n
user_dump_dest      = destino
```

donde:

n .- Es el número de bloques en disco.
destino.- Lugar donde serán almacenados dichos traces.

Inmediatamente se procede a los siguientes pasos:

1.- Ver que dentro del archivo donde se encuentre el query, no este declarado la sintáxis del EXPLAIN PLAN, es decir, que únicamente exista el query correspondiente.

2.- Una vez realizado el paso anterior, dentro de SQL, para alterar la sesión se introduce:

```
SQL> ALTER SESSION SET SQL_TRACE TRUE;
```

3.- Se ejecuta el archivo que contiene el query, através de:

```
SQL> START EJEM1;
```

4.- Se deshabilita la sesión através de:

```
SQL> ALTER SESSION SET SQL_TRACE FALSE;
```

5.- Una vez realizado los pasos anteriores, es necesario salir y dirigirse a la ruta de almacenamiento para ver el trace especificado en la declaración USER_ DUMP _DEST.

6.- Anotar el nombre del archivo trace, teniendo como extensión trc.

7.- Una vez conociendo el archivo trace se introduce:

```
TKPROF 45_2557.trc CHUY.DOC PRINT=1 EXPLAIN=SCOTT/TIGER;
```

donde:

PRINT — Es el parámetro que nos especifica el número de salidas que se quieren imprimir, es decir, si es 1, únicamente nos mandará el select, su análisis y el resultado total.
En caso de ser 2 sería el select, EXPLAIN PLAN(si es que se declara), y el resultado total
En caso de 3 sería select, EXPLAIN PLAN, ALTER SESSION y el resultado total.

EXPLAIN — Permitirá la obtención de un análisis dentro del resultado obtenido del TKPROF, para ser uso de esta instrucción únicamente se deberá conocer el usuario y el password corespondiente a la tabla usada.

8.- Por último, se ve el resultado a través de introducir únicamente:

```
> vi ejemp.doc
```

count = Número de veces en que un procedimiento es ejecutado.
cpu = Tiempo del CPU, dado en milisegundos.
elap = Tiempo de enlace, ejecutado en milisegundos.
phys = Número de lecturas físicas.
cr = Número de lecturas consistentes.
cur = Número de lecturas concurrentes.
rows = Número total de registros procesados.

cursor number: 1

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	0	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

SELECT DNAME,DEPTNO FROM DEPT WHERE DEPTNO NOT IN (SELECT DEPTNO FROM EMP)

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	1	1	0	0	0	
EXECUTE:	1	4	4	1	0	2	0
FETCH:	1	0	6	3	9	8	1

EXECUTION PLAN:
FILTER
TABLE ACCESS (FULL) OF 'DEPT'
TABLE ACCESS (FULL) OF 'EMP'

ALTER SESSION SET SQL_TRACE FALSE

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	0	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

EXECUTION PLAN:

OVERALL TOTALS FOR ALL STATEMENTS

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	2	1	1	0	0	0	
EXECUTE:	2	4	4	1	0	2	0
FETCH:	1	0	6	3	9	8	1

TOTAL NUMBER OF SQL STATEMENTS: 3
TRACE FILE: 46_15710.TRC

Dentro de este ejemplo, hay que aclarar que el query seleccionado únicamente analiza un select, por lo tanto el número de registros será declarado dentro de la columna FETCH.

Analizando los resultados anteriores, se obtiene:

1) Dentro de la columna PARSE, se observó que se tuvo 2 análisis de dicho select, 1 acceso de un milisegundo al CPU,tiempo de enlace de 1 milisegundo y ninguna lectura física,consistente y concurrente (es decir, cuando se tiene cero todo esta en disco).

2) Dentro de la columna EXECUTE, se observó que se realizó 2 ejecuciones de dicho select,un tiempo de 4 milisegundos en el CPU, tiempo de enlace de 4 milisegundos, una lectura física para traer 2 lecturas concurrentes y cero registros.

3) Dentro de la columna FETCH,se observó que se realizó 1 lectura,tiempo de enlace de 6 milisegundos, 3 lecturas físicas para traer 9 lecturas consistentes y 8 lecturas concurrentes originando un solo registro.

CONCLUSION:

En este caso, se observa que el número de lecturas físicas, consistentes y concurrentes son demasiadas grandes para originar un solo registro, por lo tanto se recurre a la creación de índices, para comprobar que exista una disminución en dichas lecturas.

5.2.3.- CREACION DEL INDICE SOBRE LA TABLA EMP

Originando un índice sobre la tabla EMP respecto a la columna DEPTNO, obtendrá como resultado del EXPLAIN PLAN lo siguiente:

OPERATION	OPTIONS	OBJECT_NAME	ID	PARENT_ID	POSITION
FILTER			1		
TABLE_ACCESS	FULL	DEPT	2	1	1
TABLE_ACCESS	FULL	EMP	3	1	2

se observará que el resultado es el mismo que el anterior, debido a la cláusula where.

Veamos ahora el análisis del TKPROF:

count = Número de veces en que un procedimiento es ejecutado.
cpu = Tiempo del CPU, dado en milisegundos.
elap = Tiempo de enlace, ejecutado en milisegundos.
phys = Número de lecturas físicas.
cr = Número de lecturas consistentes.
cur = Número de lecturas concurrentes.
rows = Número total de registros procesados.

---

cursor number: 1

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	0	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

SELECT DNAME,DEPTNO FROM DEPT WHERE DEPTNO NOT IN (SELECT DEPTNO FROM EMP)

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	2	0
FETCH:	1	5	8	2	9	8	1

EXECUTION PLAN:
FILTER
TABLE ACCESS (FULL) OF 'DEPT'
TABLE ACCESS (FULL) OF 'EMP'

ALTER SESSION SET SQL_TRACE FALSE

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	0	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

EXECUTION PLAN:

OVERALL TOTALS FOR ALL STATEMENTS

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	2	0	0	0	0	0	
EXECUTE:	2	0	0	0	0	2	0
FETCH:	1	5	8	2	9	8	1

TOTAL NUMBER OF SQL STATEMENTS: 3
TRACE FILE: 25_22141.TRC

CONCLUSION FINAL:

En el resultado anterior, se pudo observar, que efectivamente hubo una disminución del tiempo del CPU y en de enlace tanto para el PARSE como para el EXECUTE, pero resultó que hubo un incremento de tiempo en el FETCH, debido a la creación de los índices.
No se forza al uso del índice únicamente si existe la cláusula NOT IN , por lo tanto se procede a la utilización de la cláusula WHERE dentro del subquery, que es donde va a ir especificado dicho join, como se verá en el pró ximo punto.

5.2.4.-DECLARACION DE LA CLAUSULA WHERE DENTRO DEL SUBQUERY

En este ejemplo, se pretende obtener resultados de lecturas menores a los observados anteriormente, a través de la cláusula WHERE dentro del subquery, analizándose primero sin la creación de índices y después con dicha creación.

1.- Análisis del EXPLAIN PLAN y TKPROF, sin creación de índices en el query:

```
EXPLAIN PLAN
SET STATEMENT_ID='EJEMPLO2'
INTO PRU1
FOR
   SELECT DNAME, DEPTNO
   FROM DEPT
   WHERE DEPTNO NOT IN (
                        SELECT DEPTNO
                        FROM EMP
                        WHERE DEPT.DEPTNO=EMP.DEPTNO)
```

Resultado:

OPERATION	OPTIONS	OBJECT_NAME	ID	PARENT_ID	POSITION
FILTER			1		
TABLE ACCESS	FULL	DEPT	2	1	1
TABLE ACCESS	FULL	EMP	3	1	2

Como se observa, este resultado es el mismo al que se obtuvo inicialmente, debido a que no se ha procedido a la creación del índice.

Veamos ahora, el análisis del TKPROF:

count = Número de veces en que un procedimiento es ejecutado.
cpu = Tiempo del CPU, dado en milisegundos.
elap = Tiempo de enlace, ejecutado en milisegundos.
phys = Número de lecturas físicas.
cr = Número de lecturas consistentes.
cur = Número de lecturas concurrentes.
rows = Número total de registros procesados.

---

cursor number: 1

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	0	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

SELECT DNAME,DEPTNO FROM DEPT WHERE DEPTNO NOT IN (SELECT DEPTNO FROM EMP WHERE DEPT.DEPTNO=EMP.DEPTNO)

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	1	0	2	0	0	2	0
FETCH:	1	1	1	0	9	8	1

EXECUTION PLAN:

FILTER
TABLE ACCESS (FULL) OF 'DEPT'
TABLE ACCESS (FULL) OF 'EMP'

ALTER SESSION SET SQL_TRACE FALSE

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	0	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

EXECUTION PLAN:

OVERALL TOTALS FOR ALL STATEMENTS

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	2	0	0	0	0	0	
EXECUTE:	2	0	2	0	0	2	0
FETCH:	1	1	1	0	9	8	1

TOTAL NUMBER OF SQL STATEMENTS: 3
TRACE FILE: 50_17124.TRC

En este ejemplo, se analizó lo siguiente:

1) Dentro de la columna PARSE, se observó que se tuvo 2 análisis de dicho select.

2) Dentro de la columna EXECUTE, se observó que se realizó 2 ejecuciones de dicho select, tiempo de enlace de 2 milisegundos, 2 lecturas concurrentes y cero registros.

3) Dentro de la columna FETCH, se observó que se realizó 1 lectura, tiempo de 1 milisegundo en el CPU, tiempo de enlace de 1 milisegundo, 9 lecturas consis - tes y 8 lecturas concurrentes originando un solo registro.

CONCLUSION:

Como se vió, hubo una disminución en el CPU, ELAP de la columna PARSE y de la columna EXECUTE a comparación de los resultados obtenidos ante - riormente, sin la cláusula WHERE, pero de todos modos el número de lectu - ras concurrentes y consistentes aún siguen siendo muy grandes, por lo tanto se recurre a la creación de los índices.

2.- Análisis del EXPLAIN PLAN y TKPROF, con la creación del índice hacia la tabla EMP en la columna DEPTNO (que es donde se realiza dicho join) , se obtiene como resultado del EXPLAIN PLAN, lo siguiente:

OPERATION	OPTIONS	OBJECT_NAME	ID	PARENT_ID	POSITION
FILTER			1		
TABLE_ACCESS	FULL	DEPT	2	1	1
INDEX	RANGE_SCAN	EMP_IND	3	1	2

Como se observará aquí, se procede al uso del índice.

Veamos ahora, el análisis del TKPROF:

count = Número de veces en que un procedimiento es ejecutado.
cpu = Tiempo del CPU, dado en milisegundos.
elap = Tiempo de enlace, ejecutado en milisegundos.
phys = Número de lecturas físicas.
cr = Número de lecturas consistentes.
cur = Número de lecturas concurrentes.
rows = Número total de registros procesados.

cursor number: 1

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	0	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

SELECT DNAME,DEPTNO FROM DEPT WHERE DEPTNO NOT IN (SELECT DEPTNO FROM EMP WHERE DEPT.DEPTNO=EMP.DEPTNO)

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	2	0
FETCH:	1	2	4	1	9	0	1

EXECUTION PLAN:

FILTER
TABLE ACCESS (FULL) OF 'DEPT'
INDEX (RANGE SCAN) OF 'EMP_IND' (NON-UNIQUE)

ALTER SESSION SET SQL_TRACE FALSE

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	1	1	0	0	0	
EXECUTE:	0	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

EXECUTION PLAN:

OVERALL TOTALS FOR ALL STATEMENTS

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	2	2	1	0	0	0	
EXECUTE:	2	0	0	0	0	2	0
FETCH:	1	2	4	1	9	0	1

TOTAL NUMBER OF SQL STATEMENTS: 3
TRACE FILE: 12_22946.TRC

CONCLUSION:

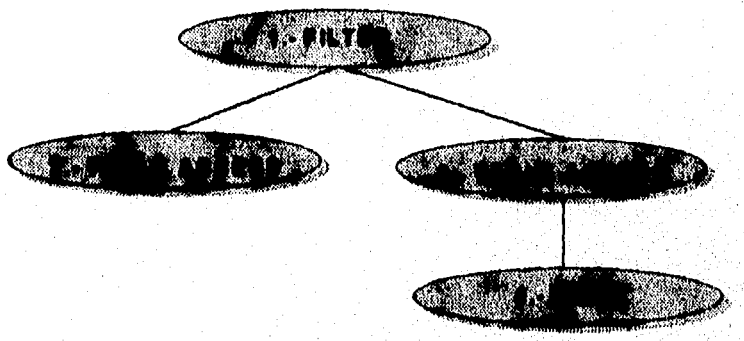
El número de lecturas físicas disminuyó de 1 a 0, el número de lecturas concurrentes disminuyó de 9 a 0, aunque el número de tiempo tanto del CPU, como de enlace, aumento debido a la creación de índices, pero el problema fué que el número de lecturas consistentes fué otra vez de 9, por lo tanto se prosigue a la última comparación de resultados, que es con la creación de clusters.

3.- Análisis del EXPLAIN PLAN y TKPROF, con creación de cluster en la tabla EMP hacia la columna DEPTNO, obtiende como resultado del EXPLAIN PLAN, lo siguiente:

OPERATION	OPTIONS	OBJECT_NAME	ID	PARENT_ID	POSITION
FILTER			1		
TABLE ACCESS	FULL	DEPT	2	1	1
TABLE ACCESS	CLUSTER	EMP	3	1	2
INDEX	UNIQUE SCAN	MOMI	4	3	1

Como se observará, aquí se procede al uso del cluster y del índice.

Analizando con el árbol estructurado obtenemos:



De acuerdo a lo anterior, se realizó lo siguiente:

1.- Se realiza primero el cluster.
2.- Inmediatamente se procede al índice.
3.- Se compara los resultados con los de la tabla DEPT, realizando un FULL TABLE SCAN y,
4.- Se obtienen los registros correspondientes.

Veamos ahora, el análisis del TKPROF:

count = Número de veces en que un procedimiento es ejecutado.
cpu = Tiempo del CPU, dado en milisegundos.
elap = Tiempo de enlace, ejecutado en milisegundos.
phys = Número de lecturas físicas.
cr = Número de lecturas consistentes.
cur = Número de lecturas concurrentes.
rows = Número total de registros procesados.

cursor number: 1

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	0	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

SELECT DNAME,DEPTNO FROM DEPT WHERE DEPTNO NOT IN (SELECT DEPTNO FROM EMP WHERE DEPT.DEPTNO=EMP.DEPTNO)

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	1	1	0	0	0	
EXECUTE:	1	0	0	0	0	2	0
FETCH:	1	0	0	0	12	0	1

EXECUTION PLAN:
FILTER
TABLE ACCESS (FULL) OF 'DEPT'
TABLE ACCESS (CLUSTER) OF 'EMP'
INDEX (UNIQUE SCAN) OF 'MOMP (CLUSTER)

ALTER SESSION SET SQL_TRACE FALSE

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	0	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

EXECUTION PLAN:

OVERALL TOTALS FOR ALL STATEMENTS

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	2	1	1	0	0	0	
EXECUTE:	2	0	0	0	0	2	0
FETCH:	1	0	0	0	12	0	1

TOTAL NUMBER OF SQL STATEMENTS: 3
TRACE FILE: 43_27979.TRC

CONCLUSION FINAL:

Como se pudo observar en dichos resultados,está es la solución correcta aunque el número de lecturas consistentes fué mayor, debido a que cuando se hace creación de un cluster, siempre el número de lecturas a bloques es mayor debido a que si no encuentra el dato correcto,se va a otro bloque y así sucesivamente, por lo tanto siempre que se haga un análisis como el anterior se debe ver todos los tiempos de ejecución,si son mayores se procede entonces a la creación de índices ó clusters, aunque algunas veces como se verá más adelante,cuando las tablas son muy grandes no es conveniente la creación de clusters.

5.3.- EJEMPLO NO.2

```
SELECT DNAME, DEPTNO
FROM DEPT
WHERE DEPTNO NOT EXISTS(
                        SELECT DEPTNO
                        FROM EMP );
```

En este ejemplo, se pretende ver como la claúsula NOT EXISTS, no tiene el mismo comportamiento que la claúsula NOT IN, debido a que en algunas ocasiones con la cláusula NOT EXISTS, el número tanto de lecturas, es decir (FISICA, CONCURRENTES y CONSISTENTES) y por supuesto el número de tiempo(CPU, ENLACE) es menor, como se observará a continuación.

No cabe de más mencionar con respecto a la cláusula WHERE, sino existe dentro del subquery, no se podrá forzar el uso del índice.

Analizando el EXPLAIN PLAN de este query, se obtiene lo siguiente:

OPERATION	OPTIONS	OBJECT_NAME	ID	PARENT_ID	POSITION
FILTER			1		
TABLE ACCESS	FULL	DEPT	2	1	1

La pregunta que se haría, respecto a este resultado sería la siguiente:

¿ Porque únicamente se realiza un FULL TABLE SCAN a la tabla DEPT ?

Para poder responderla,analicemos los campos y los registros que contiene la tabla DEPT:

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

Aquí se observa que los registros que contiene DEPTNO son (10,20,30,40), si nos fijamos en la tabla EMP, únicamente los registros que contiene son (10,20,30).

Entonces, observando el query, nosotros veremos que este nos dice lo siguiente:

-Muestra el nombre y el departamento que exista en la tabla DEPT, pero que no exista en la tabla EMP.

Analizando el primer query, vemos que se refiere a los registros de DEPTNO (10,20,30,40), por lo tanto le pondremos un 1 por ser verdadero, pero entrando al subquery vemos que 40 no existe en la tabla EMP, por lo tanto pondremos un cero.
Entonces, analizándose como una compuerta AND, siempre y cuando no exista una claúsula WHERE dentro del subquery, nos dará como resultado FALSE, por lo tanto se verá en el TKPROF que el número de registros retornados va hacer cero, es por esta razón que únicamente accesa a la tabla tabla DEPT.

Procediendo al análisis del TKPROF, obtendremos lo siguiente:

count = Número de veces en que un procedimiento es ejecutado.
cpu = Tiempo del CPU, dado en milisegundos.
elap = Tiempo de enlace, ejecutado en milisegundos.
phys = Número de lecturas físicas.
cr = Número de lecturas consistentes.
cur = Número de lecturas concurrentes.
rows = Número total de registros procesados.

---

cursor number: 1

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	0	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

SELECT DNAME,DEPTNO FROM DEPT WHERE NOT EXISTS (SELECT DEPTNO FROM EMP)

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	1	1	1	0	2	2	0
FETCH:	1	0	0	0	0	0	0

EXECUTION PLAN:
FILTER
TABLE ACCESS (FULL) OF 'DEPT'

ALTER SESSION SET SQL_TRACE FALSE

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	0	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

EXECUTION PLAN:

OVERALL TOTALS FOR ALL STATEMENTS

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	2	0	0	0	0	0	
EXECUTE:	2	1	1	0	2	2	0
FETCH:	1	0	0	0	0	0	0

TOTAL NUMBER OF SQL STATEMENTS: 3
TRACE FILE: 42_3235.TRC

-Con respecto a la columna PARSE, únicamente se realizó 2 análisis en dicho select.

-Con respecto a la columna EXECUTE, se tendrá 2 ejecuciones, 1 tiempo de CPU 1 tiempo de enlace, 2 lecturas consistentes y concurrentes para originar cero registros.

-Con respecto a la columna FETCH, se analizó una sola lectura para originar cero registros.

CONCLUSION:

Efectivamente, se cumplio lo que se había dicho al principio de este ejemplo, que tanto el número de lecturas como el tiempo iban a ser más pequeños debido a que la claúsula NOT EXISTS se comporta como una compuerta AND siempre y cuando no exista una cláusula WHERE dentro del subquery, observando también que el número de lecturas analizadas en el EXECUTE son grandes para no traer ningún registro, por lo tanto se procede a la creación de índices como se verá em el pròximo punto.

5.3.1.- CREACION DE INDICE A LA TABLA EMP

Al crear un índice a la tabla EMP respecto al campo DEPTNO y analizando el EXPLAIN PLAN se observará que el resultado es el mismo al anterior,debido a que no existe la cláusula WHERE.

OPERATION	OPTIONS	OBJECT_NAME	ID	PARENT_ID	POSITION
FILTER			1		
TABLE ACCESS	FULL	DEPT	2	1	1

Pero si analizamos el TKPROF, veremos lo siguiente:

count = Número de veces en que un procedimiento es ejecutado.
cpu = Tiempo del CPU, dado en milisegundos.
elap = Tiempo de enlace, ejecutado en milisegundos.
phys = Número de lecturas físicas.
cr = Número de lecturas consistentes.
cur = Número de lecturas concurrentes.
rows = Número total de registros procesados.

cursor number: 1

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	0	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

SELECT DNAME,DEPTNO FROM DEPT WHERE NOT EXISTS (SELECT DEPTNO FROM EMP)

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	1	0	0	0	1	2	0
FETCH:	1	0	0	0	0	0	0

EXECUTION PLAN:
FILTER
TABLE ACCESS (FULL) OF 'DEPT'

ALTER SESSION SET SQL_TRACE FALSE

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	0	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

EXECUTION PLAN:

OVERALL TOTALS FOR ALL STATEMENTS

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	2	0	0	0	0	0	
EXECUTE:	2	0	0	0	1	2	0
FETCH:	1	0	0	0	0	0	0

TOTAL NUMBER OF SQL STATEMENTS: 3
TRACE FILE: 60_2992.TRC

-Con respecto a la columna PARSE,únicamente se realizó 2 análisis en dicho select.

-Con respecto a la columna EXECUTE, se tendrá 2 ejecuciones, 1 lectura consistente y 2 concurrentes para originar cero registros.

-Con respecto a la columna FETCH, se analizó una sola lectura para originar cero registros.

CONCLUSION:

Como se vió en esta ocasión, no hubo realización de tiempos tanto del CPU como de enlace, debido a que la creación del índice en la tabla EMP permitió una acceso más rápido.
Este resultado es adecuado, pero veremos a continuación que pasa con la creación de clusters sobre la tabla EMP.

5.3.2.- CREACION DE CLUSTER EN LA TABLA EMP

Con respecto al EXPLAIN PLAN, el resultado será el mismo a los anteriores, pero con el TKPROF será diferente:

count = Número de veces en que un procedimiento es ejecutado.
cpu = Tiempo del CPU, dado en milisegundos.
elap = Tiempo de enlace, ejecutado en milisegundos.
phys = Número de lecturas físicas.
cr = Número de lecturas consistentes.
cur = Número de lecturas concurrentes.
rows = Número total de registros procesados.

cursor number: 1

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	0	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

SELECT DNAME,DEPTNO FROM DEPT WHERE NOT EXISTS (SELECT DEPTNO FROM EMP)

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	1	0	0	0	1	2	0
FETCH:	1	0	1	0	0	0	0

EXECUTION PLAN:
FILTER
TABLE ACCESS (FULL) OF 'DEPT'

ALTER SESSION SET SQL_TRACE FALSE

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	0	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

EXECUTION PLAN:

OVERALL TOTALS FOR ALL STATEMENTS

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	2	0	0	0	0	0	
EXECUTE:	2	0	0	0	1	2	0
FETCH:	1	0	1	0	0	0	0

TOTAL NUMBER OF SQL STATEMENTS: 3
TRACE FILE: 8_2795.TRC

-Con respecto a la columna PARSE,únicamente se realizó 2 análisis en dicho select.

-Con respecto a la columna EXECUTE, se tendrá 2 ejecuciones, 1 lectura consistente y 2 concurrentes para originar cero registros.

-Con respecto a la columna FETCH, se analizó una sola lectura,un tiempo de enlace para originar cero registros.

CONCLUSION FINAL:

Dentro de la creación del cluster en la tabla EMP con respecto al campo DEPTNO, se observa que existe un tiempo de enlace en la columna FETCH, precisamente por el acceso a los bloques.
Con respecto a los 3 puntos anteriores, se concluye que los resultados más adecuados son los obtenidos cuando se hace creación del índice aunque también podría ser cuando se crea un cluster, debido a que permite un almacenamiento menor en el disco.

5.3.4.- DECLARACION DE LA CLAUSULA WHERE DENTRO DEL SUBQUERY

1.- Analizando el EXPLAIN PLAN del siguiente query, sin creación de índice, se obtiene lo siguiente:

```
SELECT DNAME, DEPTNO
FROM DEPT
WHERE NOT EXISTS
      ( SELECT DEPTNO
        FROM EMP
        WHERE DEPT.DEPTNO= EMP.DEPTNO);
```

OPERATION	OPTIONS	OBJECT_NAME	ID	PARENT_ID	POSITION
FILTER			1		
TABLE_ACCESS	FULL	DEPT	2	1	1
TABLE_ACCESS	FULL	EMP	3	2	1

Lo que se observa aquí, es que se hace acceso por FULL TABLE SCAN a la tabla EMP, debido a que no sea ha procedido a la creación del índice.

Respecto al análisis del TKPROF, se tendrá lo siguiente:

count = Número de veces en que un procedimiento es ejecutado.
cpu = Tiempo del CPU, dado en milisegundos.
elap = Tiempo de enlace, ejecutado en milisegundos.
phys = Número de lecturas físicas.
cr = Número de lecturas consistentes.
cur = Número de lecturas concurrentes.
rows = Número total de registros procesados.

cursor number: 1

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	0	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

SELECT DNAME,DEPTNO FROM DEPT WHERE NOT EXISTS (SELECT DEPTNO FROM EMP WHERE DEPT.DEPTNO = EMP.DEPTNO)

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	2	0
FETCH:	1	1	1	0	9	8	1

EXECUTION PLAN:
FILTER
TABLE ACCESS (FULL) OF 'DEPT'
TABLE ACCESS (FULL) OF 'EMP'

ALTER SESSION SET SQL_TRACE FALSE

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	0	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

EXECUTION PLAN:

OVERALL TOTALS FOR ALL STATEMENTS

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	2	0	0	0	0	0	
EXECUTE:	2	0	0	0	0	2	0
FETCH:	1	1	1	0	9	8	1

TOTAL NUMBER OF SQL STATEMENTS: 3
TRACE FILE: 28_3547.TRC

Lo que se observa aquí, es lo siguiente:

1.- Existe ahora sí un registro retornado, debido precisamente al join existente en el WHERE del subquery.
2.- Con respecto a la columna PARSE,únicamente se realizó 2 análisis en dicho select.
3.- Con respecto a la columna EXECUTE, se tendrá 2 ejecuciones, 2 lecturas concurrentes para originar cero registros.
4.- Con respecto a la columna FETCH,se analizó una sola lectura, un tiempo tanto de CPU como de enlace, 9 lecturas consistentes y 8 concurrentes para originar un registro.

CONCLUSION:

El problema aquí, es el aumento grande en el número de lecturas consistentes y concurrentes de la columna FECHT, precisamente porque existe la obtención de un solo registro y además porque no se ha originado un índice.

2.-Procediendo a la creación del índice en la tabla EMP respecto al campo DEPT-NO, se obtiene como resultado del EXPLAIN PLAN lo siguiente:

OPERATION	OPTIONS	OBJECT_NAME	ID	PARENT_ID	POSITION
FILTER			1		
TABLE ACCESS	FULL	DEPT	2	1	1
INDEX	RANGE SCAN	EMP	3	2	1

Aquí observamos, que efectivamente se procedió al acceso del índice, originando como resultado del **TKPROF** lo siguiente:

count = Número de veces en que un procedimiento es ejecutado.
cpu = Tiempo del CPU, dado en milisegundos.
elap = Tiempo de enlace, ejecutado en milisegundos.
phys = Número de lecturas físicas.
cr = Número de lecturas consistentes.
cur = Número de lecturas concurrentes.
rows = Número total de registros procesados.

---

cursor number: 1

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	0	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

SELECT DNAME,DEPTNO FROM DEPT WHERE NOT EXISTS (SELECT DEPTNO FROM EMP WHERE DEPT.DEPTNO = EMP.DEPTNO)

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	1	1	0	0	0	
EXECUTE:	1	1	1	0	0	2	0
FETCH:	1	0	0	0	9	0	1

EXECUTION PLAN:
FILTER
TABLE ACCESS (FULL) OF 'DEPT'
INDEX (RANGE SCAN) OF 'EMP_IND' (NON-UNIQUE)

ALTER SESSION SET SQL_TRACE FALSE

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	0	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

EXECUTION PLAN:

OVERALL TOTALS FOR ALL STATEMENTS

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	2	0	0	0	0	0	
EXECUTE:	2	0	0	0	0	2	0
FETCH:	1	1	1	0	9	8	1

TOTAL NUMBER OF SQL STATEMENTS: 3
TRACE FILE: 31_4886.TRC

-Con respecto a la columna PARSE, únicamente se realizó 2 análisis en dicho select y un tiempo tanto de CPU como de enlace.

-Con respecto a la columna EXECUTE, se tendrá 2 ejecuciones, un tiempo tanto de CPU como de enlace y 2 lecturas concurrentes para originar cero registros.

-Con respecto a la columna FETCH, se analizó una sola lectura y 9 lecturas consistentes para originar un registro.

CONCLUSION:

Hubo una disminución de 8 a 0 lecturas concurrentes en la columna FETCH para originar un solo registro, aunque hubo un aumento de tiempos (CPU, enlace), tanto para la columna PARSE como en el EXECUTE, considerándolo normal, pero el problema sigue siendo el número grande de lectura consistente para un solo registro originado, por lo tanto se procede a la creación de clusters.

3.- Analizando el TKPROF, con la creación del cluster sobre la tabla EMP respecto a la columna DEPTNO, se obtiene lo siguiente:

count = Número de veces en que un procedimiento es ejecutado.
cpu = Tiempo del CPU, dado en milisegundos.
elap = Tiempo de enlace, ejecutado en millisegundos.
phys = Número de lecturas físicas.
cr = Número de lecturas consistentes.
cur = Número de lecturas concurrentes.
rows = Número total de registros procesados.

cursor number: 1

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	0	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

SELECT DNAME,DEPTNO FROM DEPT WHERE NOT EXISTS (SELECT DEPTNO FROM EMP WHERE DEPT.DEPTNO = EMP.DEPTNO)

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	2	0
FETCH:	1	1	1	0	12	0	1

EXECUTION PLAN:
FILTER
TABLE ACCESS (FULL) OF 'DEPT'
TABLE ACCESS (CLUSTER) OF 'EMP'
INDEX (UNIQUE SCAN) OF 'MOMI' (CLUSTER)

ALTER SESSION SET SQL_TRACE FALSE

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	0	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

EXECUTION PLAN:

OVERALL TOTALS FOR ALL STATEMENTS

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	3	0	4	0	0	0	
EXECUTE:	3	0	0	0	0	2	0
FETCH:	1	1	1	0	12	0	1

TOTAL NUMBER OF SQL STATEMENTS: 3
TRACE FILE: 32_4886.TRC

-Con respecto a la columna **PARSE**, se realizó 3 análisis en dicho select y 4 lecturas físicas.

-Con respecto a la columna **EXECUTE**, se tendrá 3 ejecuciones, y 2 lecturas concurrentes para originar cero registros.

-Con respecto a la columna **FETCH**, se analizó una sola lectura, un tiempo de (CPU,enlace) y 12 lecturas consistentes para originar un registro.

CONCLUSION FINAL:

El número de lecturas consistentes aumenta debido a la búsqueda de bloques, originando un aumento de tiempo tanto para CPU como de enlace. Concluyéndose finalmente que cuando se utiliza la cláusula NOT EXISTS, sin la cláusula WHERE, los resultados obtenidos dentro del TKPROF son muy diferentes.

5.4.- EJEMPLO NO.3

```
SELECT ENAME, JOB, SAL, DNAME
FROM EMP, DEPT
WHERE EMP.DEPTNO=. DEPT.DEPTNO
AND NOT EXISTS
      ( SELECT *
        FROM SALGRADE
        WHERE EMP.SAL BETWEEN LOSAL AND HISAL );
```

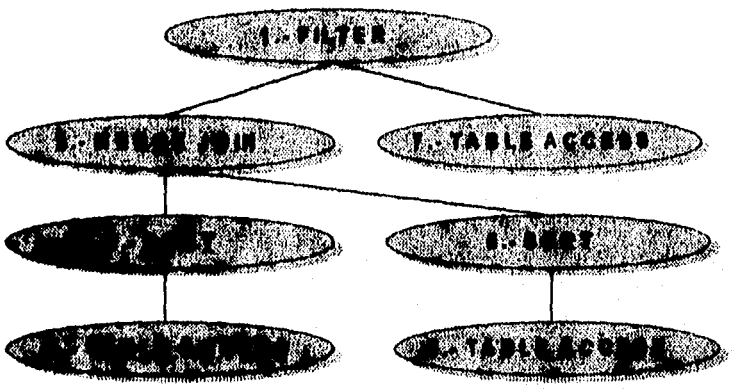
En este ejemplo se analizará la cláusula NOT EXISTS acompañado de la cláusula WHERE dentro del subquery utilizando el operador BETWEEN.

Se observa, que el número de registros retornados será igual a cero debido a que todos los valores de SAL caen dentro del rango del LOSAL y HISAL.

Procediendo al análisis del EXPLAIN PLAN de este query, se obtiene lo siguiente:

OPERATION	OPTIONS	OBJECT_NAME	ID	PARENT_ID	POSITION
FILTER			1		
MERGE JOIN			2	1	1
SORT	JOIN		3	2	1
TABLE ACCESS	FULL	DEPT	4	3	1
SORT	JOIN		5	2	2
TABLE ACCESS	FULL	EMP	6	5	1
TABLE ACCESS	FULL	SALGRADE	7	1	2

Ilustrando este resultado con el árbol estructurado, obtenemos lo siguiente:



Observando con detalle el árbol estructurado de abajo hacia arriba, obtendremos lo siguiente:

1.- Se realizan 2 TABLE_ACCESS, uno de la tabla EMP y otro de la DEPT, identificándose con el número 6 y 4.

2.- Después se prosigue con el SORT de la tabla EMP y DEPT, identificándose con el número 5 y 3.

3.- Se procede con el MERGE JOIN de la tabla DEPT y el TABLE ACCESS de la tabla DEPT, identificándose con el número 2 y 7.

4.- Se realiza un FILTER (restricción), identificándose con el número 1.

NOTA:

Es importante especificar que este análisis anterior, es que el se obtiene de la columna ID y PARENT_ID.

La columna **POSITION**, nos muestra los pasos del procedimiento de la columna **PARENT_ID** que aplicado a este ejemplo nos dice lo siguiente:

1.- Recordemos que siempre se analiza aquel valor mayor que aparezca en la columna **POSITION**, en este caso veremos que existen dos números 2.

2.- El número 2 ubicado en la parte final del EXPLAIN PLAN, realiza un FULL TABLE SCAN a la tabla SALGRADE, lo que quiere decir que primero se analiza esta tabla y luego los resultados se comparan con la tabla EMP, realizándose otro FULL TABLE SCAN.

3.- Una vez realizado este FULL TABLE, se procede a relizarse un SORT a la tabla EMP (especificado con otro número 2 dentro del análisis del EXPLAIN PLAN) y luego se compara con la tabla DEPT realizándose otro FULL TABLE.

4.-Realizado dicha comparación, se realiza un SORT a la tabla DEPT y luego se procede a un MERGE JOIN para obtener al final los resultados correspondientes através de un FILTER.

5.4.1.- ANALISIS DEL TKPROF

Procediendo al análisis de este query, se obtendrán los siguientes resultados:

count = Número de veces en que un procedimiento es ejecutado.
cpu = Tiempo del CPU, dado en milisegundos.
elap = Tiempo de enlace, ejecutado en milisegundos.
phys = Número de lecturas físicas.
cr = Número de lecturas consistentes.
cur = Número de lecturas concurrentes.
rows = Número total de registros procesados.

---

cursor number: 1

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	0	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

ALTER SESSION SET SQL_TRACE TRUE

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

SELECT ENAME, JOB, SAL, DNAME FROM EMP,DEPT WHERE EMP.DEPTNO = DEPT.DEPTNO AND NOT EXISTS (SELECT * FROM SALGRADE WHERE EMP.SAL BETWEEN LOSAL AND HISAL)

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	1	1	0	0	0	
EXECUTE:	1	71	78	2	1	2	0
FETCH:	1	1	1	0	1	2	0

EXECUTION PLAN:

```
FILTER
  MERGE JOIN
    SORT (JOIN)
      TABLE ACCESS (FULL) OF 'DEPT'
    SORT (JOIN)
      TABLE ACCESS (FULL) OF 'EMP'
  TABLE ACCESS (FULL) OF 'SALGRADE'
```

ALTER SESSION SET SQL_TRACE FALSE

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	0	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

EXECUTION PLAN:

OVERALL TOTALS FOR ALL STATEMENTS

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	3	1	1	0	0	0	
EXECUTE:	3	71	78	2	1	2	0
FETCH:	1	1	1	0	1	2	0

TOTAL NUMBER OF SQL STATEMENTS: 4
TRACE FILE: 25_22931.TRC

-Con respecto a la columna **PARSE**, se realizó 3 análisis en dicho select y un tiempo de (CPU,enlace).

-Con respecto a la columna **EXECUTE**, se tendrá 3 ejecuciones, 71 tiempos en el CPU, 78 tiempos de enlace, 2 lecturas físicas, 1 lectura consistente y 2 lecturas concurrentes para originar cero registros.

-Con respecto a la columna **FETCH**, se analizó una sola lectura, un tiempo de (CPU,enlace), 1 lectura consistente y 2 concurrentes para originar un registro.

CONCLUSION:

Se observa que existe un tiempo exageradamente grande, debido a que no existe creado ningún índice, por lo tanto se procede a su creación para ver si el número de tiempo tanto del CPU como de enlace disminuye.

Respecto al número de procedimientos que fueron ejecutados tanto en el PARSE como en el EXECUTE, son adecuados debido a que se tiene que accesar a 3 tablas diferentes.

5.4.1.1.-CREACION DEL INDICE EN LA TABLA DEPT

Al crear un índice a la tabla DEPT, respecto al campo DEPTNO y al analizar el TKPROF, obtenemos lo siguiente:

count = Número de veces en que un procedimiento es ejecutado.
cpu = Tiempo del CPU, dado en milisegundos.
elap = Tiempo de enlace, ejecutado en milisegundos.
phys = Número de lecturas físicas.
cr = Número de lecturas consistentes.
cur = Número de lecturas concurrentes.
rows = Número total de registros procesados.

cursor number: 1

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	0	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

SELECT ENAME, JOB, SAL, DNAME FROM EMP,DEPT WHERE EMP.DEPTNO= DEPT.DEPTNO AND NOT EXISTS (SELECT * FROM SALGRADE WHERE EMP.SAL BETWEEN LOSAL AND HISAL)

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	2	0
FETCH:	1	0	0	0	1	0	0

EXECUTION PLAN:
FILTER
NESTED LOOPS
TABLE ACCESS (FULL) OF 'EMP'
TABLE ACCESS (BY ROWID) OF ' DEPT'
INDEX (RANGE SCAN) OF ' IN_DEPT' (NON- UNIQUE)
TABLE ACCESS (FULL) OF 'SALGRADE'

ALTER SESSION SET SQL_TRACE FALSE

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	1	0	0	0	0	
EXECUTE:	0	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

EXECUTION PLAN:

OVERALL TOTALS FOR ALL STATEMENTS

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	2	1	0	0	0	0	
EXECUTE:	2	0	0	0	0	2	0
FETCH:	1	0	0	0	1	0	0

TOTAL NUMBER OF SQL STATEMENTS: 4
TRACE FILE: 44_848.TRC

-Con respecto a la columna PARSE, se realizó 2 análisis en dicho select y un tiempo de CPU.

-Con respecto a la columna EXECUTE, se tendrá 2 ejecuciones y 2 lecturas concurrentes para originar cero registros.

-Con respecto a la columna FETCH, se analizó una sola lectura, y 1 lectura consistente para originar cero registro.

CONCLUSION FINAL:

Efectivamente, la creación de índice, originó una disminución de tiempo tanto en el CPU como en el de enlace, por lo tanto esta es la solución correcta, para la extracción de cero registros.

5.5.- EJEMPLO NO.4

El siguiente programa, presenta un reporte general de avisos existentes en diversas tablas.

```
SELECT T_404.N_CONTEMI, T_404.CV_AVI, NOM_CONT, N_DIV, NOM_
DIV, T_177.N_DOC, T_177.T_DOC, T_177.R_SEC, N_CENAFO, N_DEP, CV_
PROY, CV_AUT, N_CONORI, T_177.DIG_AVI, T_177.I_OP, SUM(DECODE
(T_177.DIG_AVI, 'A', T_177.I_OP, 'B', T_177.I_OP)) SUMIPO,CV_BEN,NOM_
BEN. N_CONEX, DES_CONEX, N_CENDEST, N_ CENADS,N_DEPADS, EN-
TALM, N_REQ, N_PED, N_CONTRAT, R_FEDCAU, F_VENC, N_CHEQ,
ORD_PAG,FOL_TES, CIC_COMP,DECODE(T_177.DIG_AVI, 'A', 'D', 'B', 'H')
ID_MOVDH,DECODE(T_177.DIG_AVI,'A',T_006.SCTA_DB,'B',T_006.SCTA_
CP)SUBCTA, T_041.N_ CONT ,N_FACT,DESC_REG , T_177.N_CTACONF

FROM T_041, T_059, T_177, T_404, T_408, T_006

WHERE SUBSTR(T_404.CV_AVI,1,2) SUBSTR(T_404.CV_AVI,3,1) BETWEEN
'95K'  AND SUMREG=NVL(REGCONF,0)
AND T_006.N_DOC = T_177.N_POLCONF
AND T_006.R_SEC = T_177.R_SECONF
AND T_059.N_CONT = T_404.N_CONTEMI
AND T_408.CV_DIV = T_059.N_DIV
AND T_177.N_CONTEMI = T_404.N_CONTEMI
AND T_177.CV_AVI = T_404.CV_AVI
AND T_177.N_POLCONF IS NOT NULL
AND (T_177.DIG_AVI = 'A' OR T_177.DIG_AVI = 'B')

GROUP BY T_404.N_CONTEMI,T_404.CV_AVI, NOM_CONT, N_DIV, NOM_
DIV, T_177.N_DOC, T_177.T_DOC, T_177.R_SEC, N_CENAFO, N_DEP, CV_
PROY, CV_AUT, N_CONORI, DECODE(T_177.DIG_AVI, 'A ', T_006.SCTA_
DB,'B',T_006.SCTA _CR), DECODE(T_177.DIG_AVI,'A','D','B','H'), CV_BEN,
NOM_BEN, N_CONEX, DES_CONEX, N_CENDEST, N_CENADS, N_DEPADS,
FOL_TES, CIC_COMP, T_041.N_CONT, N_FACT, DES_REG, T_177.N_CTA
CONF, T_177.I_OP, T_177.DIG_AVI
```

Antes de proceder al análisis del EXPLAIN PLAN de este ejemplo, es importante dar a conocer los índices, columnas y tipo de datos así como el número de registros que contiene cada tabla.

TABLA	INDICE	TIPO	COLUMNAS	REGISTRO
T_006	T006IND1 T006IND2	UNIQUE NON_UNIQUE	N_DOC, R_SEC N_DOC	1,336,554
T_041				1
T_059	T059IND1	UNIQUE	N_CONT	74

T_177	T177IND1 T177IND2 T177IND3	UNIQUE NON_UNIQUE NON_UNIQUE	CV_AVI,N_CONTEMI, N_DOC, R_SEC CV_VAL,N_DOC,R_SEC, N_CENAFO, N_DEP, N_CONORI N_POLCONF	145973
T_404	T404IND1	UNIQUE	N_CONTEMI, CV_AVI	6061
T_408	T408IND1	UNIQUE	CV_AVI	6

El resultado del EXPLAIN PLAN es el siguiente:

OPERATION	OPTIONS	OBJECT_NAME	ID	PARENT_ID	POSITION
SORT	GROUP BY		1		
NESTED LOOPS			2	1	1
NESTED LOOPS			3	2	1
NESTED LOOPS			4	3	1
NESTED LOOPS			5	4	1
NESTED LOOPS			6	5	1
TABLE ACCESS	FULL	T_006	7	6	1
TABLE ACCESS	BY ROWID	T_177	8	6	2
INDEX	RANGE SCAN	T177IND3	9	8	1
TABLE ACCESS	BY ROWID	T_404	10	5	2
INDEX	UNIQUE SCAN	T404IND1	11	10	1
TABLE ACCESS	BY ROWID	T_059	12	4	2
INDEX	UNIQUE SCAN	T059IND1	13	12	1
TABLE ACCESS	BY ROWID	T_408	14	3	2
INDEX	UNIQUE SCAN	T408IND1	15	14	1
TABLE ACCESS	FULL	T_041	16	2	2

Aquí observamos que la tabla T_006, realiza un FULL TABLE SCAN, por lo tanto ocasionará un tiempo muy grande tanto en el CPU como en el de enlace, como se verá en los resultados del TKPROF:

count = Número de veces en que un procedimiento es ejecutado.
cpu = Tiempo del CPU, dado en milisegundos.
elap = Tiempo de enlace, ejecutado en milisegundos.
phys = Número de lecturas físicas.
cr = Número de lecturas consistentes.
cur = Número de lecturas concurrentes.
rows = Número total de registros procesados.

cursor number: 1

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	0	0	0	0	0	0	
EXECUTE:	1	1	1	0	0	0	0
FETCH:	0	0	0	0	0	0	0

SELECT T_404.N_CONTEMI, T_404.CV_AVI, NOM_CONT, N_DIV, NOM_DIV, T_177.N_DOC, T_177.T_DOC, T_177.R_SEC, N_CENAFO, N_DEP, CV_PROV, CV_AUT, N_CONORI, T_177.DIG_AVI, T_177.I_OP, SUM(DECODE(T_177.DIG_AVI, 'A', T_177.I_OP, 'B', T_177.I_OP)) SUMIPO,CV_BEN, NOM_BEN, N_CONEX, DES_CONEX, N_CENDEST, N_CENADS, N_DEPADS, ENTALM, N_REQ, N_PED, N_CONTRAT, R_FEDCAU, F_VENC, N_CHEQ, ORD_PAG, FOL_TES, CIC_COMP, DECODE (T_177. DIG_ AVI,'A','D','B','H')ID_MOVDH,DECODE(T_177.DIG_AVI,'A',T_006.SCTA_DB,'B',T_006.SCTA_CP) SUBCTA, T_041.N_CONT,N_FACT, DESC_REG, T_177.N_CTACONF FROM T_041, T_059, T_177, T_404, T_408, T_006 WHERE SUBSTR(T_404.CV_AVI,1,2) SUBSTR(T_404.CV_AVI,3,1) BETWEEN '95K' AND SUMREG= NVL (REGCONF,0) AND T_006.N_DOC = T_177.N_POLCONF AND T_006.R_SEC = T_177.R_SECONF AND T_059.N_CONT = T_404.N_CONTEMI AND T_408.CV_DIV = T_059.N_DIV AND T_177.N_CONTEMI = T_404.N_CONTEMI AND T_177.CV_AVI= T_404.CV_AVI AND T_177.N_POLCONF IS NOT NULL AND (T_177.DIG_AVI = 'A' OR T_177.DIG_AVI = 'B') GROUP BY T_404.N_CONTEMI, T_404.CV_AVI, NOM_CONT, N_DIV, NOM_DIV, T_177.N_DOC, T_177.T_DOC, T_177.R_SEC, N_CENAFO, N_DEP, CV_PROV, CV_AUT, N_CONORI, DECODE(T_177.DIG_AVI, 'A', T_006. SCTA_ DB, 'B', T_006.SCTA_CR), DECODE (T_177.DIG_AVI,'A','D','B','H'), CV_BEN, NOM_BEN, N_CONEX, DES_CONEX, N_CENDEST, N_CENADS, N_DEPADS, FOL_TES, CIC_COMP, T_041.N_CONT, N_FACT, DES_REG, T_177.N_CTACONF, T_177.I_OP, T_177.DIG_AVI

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	8	16	0	0	0	
EXECUTE:	1	8095851	8108478	8574396	6438004	10462	0
FETCH:	349	489	661	0	0	0	5230

EXECUTION PLAN:

```
SORT(GROUP BY)
  NESTED LOOPS
    NESTED LOOPS
      NESTED LOOPS
        NESTED LOOPS
          NESTED LOOPS
            TABLE ACCESS (FULL) OF 'T_006'
              TABLE ACCESS ( BY ROWID ) OF ' T_177'
                INDEX ( RANGE SCAN ) OF 'T177IND3' (NON- UNIQUE)
            TABLE ACCESS (BY ROWID) OF 'T_404'
              INDEX ( UNIQUE SCAN ) OF 'T4041ND1' (UNIQUE)
          TABLE ACCESS (BY ROWID) OF 'T_059'
            INDEX ( UNIQUE SCAN ) OF 'T059IND1' (UNIQUE)
        TABLE ACCESS (BY ROWID) OF 'T_408'
          INDEX ( UNIQUE SCAN ) OF 'T408IND1' (UNIQUE)
      TABLE ACCESS (FULL) OF 'T_041'
```

OVERALL TOTALS FOR ALL STATEMENTS

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	8	16	0	0	0	
EXECUTE:	2	8095852	8108479	8574396	6438004	10462	0
FETCH:	349	489	661	0	0	0	5230

TOTAL NUMBER OF SQL STATEMENTS: 2
TRACE FILE: 12_16412.TRC

CONCLUSION:

El problema que se tuvo en este ejemplo, es precisamente la colocación de las tablas dentro del FROM;en este caso nosotros observamos que la tabla T_041 esta al principio de la claúsula FROM y la tabla T_006 que es la que contiene más registros se encuentra al final de dicho FROM, por lo tanto no se forza a usar el índice.

Es por esta razón que la corrida de este query tardó 2 días completos para originar únicamente 5230 registros.

5.5.1.- CAMBIO DE TABLA EN DICHO QUERY

Antes de analizar el EXPLAIN PLAN, es necesario realizar un cambio a la claúsula FROM como a continuación se presenta:

FROM T_041, T_059, T_177, T_404, T_408, T_006
a
FROM T_006, T_059, T_177, T_404, T_408, T_041

Originando como resultado del EXPLAIN PLAN lo siguiente:

OPERATION	OPTIONS	OBJECT_NAME	ID	PARENT_ID	POSITION
SORT	GROUP BY		1		
NESTED LOOPS			2	1	1
NESTED LOOPS			3	2	1
NESTED LOOPS			4	3	1
NESTED LOOPS			5	4	1
NESTED LOOPS			6	5	1
TABLE ACCESS	FULL	T_404	7	6	1
TABLE ACCESS	BY ROWID	T_059	8	6	2
INDEX	UNIQUE SCAN	T059IND1	9	8	1
TABLE ACCESS	BY ROWID	T_408	10	5	2
INDEX	UNIQUE SCAN	T408IND1	11	10	1
TABLE ACCESS	BY ROWID	T_177	12	4	2
INDEX	RANGE SCAN	T177IND1	13	12	1
TABLE ACCESS	BY ROWID	T_006	14	3	2
INDEX	UNIQUE SCAN	T006IND1	15	14	1
TABLE ACCESS	FULL	T_041	16	2	2

Analizando el EXPLAIN PLAN, se observa que efectivamente se acceso el índice a la tabla T_006, aunque se produce un FULL TABLE SCAN a la tabla T_404.

Procediendo al análisis del TKPROF, se observa lo siguiente:

count = Número de veces en que un procedimiento es ejecutado.
cpu = Tiempo del CPU, dado en milisegundos.
elap = Tiempo de enlace, ejecutado en milisegundos.
phys = Número de lecturas físicas.
cr = Número de lecturas consistentes.
cur = Número de lecturas concurrentes.
rows = Número total de registros procesados.

cursor number: 1

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	0	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

SELECT T_404.N_CONTEMI, T_404.CV_AVI, NOM_CONT, N_DIV, NOM_DIV, T_177.N_DOC, T_177.T_DOC, T_177.R_SEC, N_CENAFO, N_DEP, CV_PROV, CV_AUT, N_CONORI, T_177.DIG_AVI, T_177.I_OP, SUM(DECODE(T_177.DIG_AVI, 'A', T_177.I_OP, 'B', T_177.I_OP)) SUMIPO,CV_BEN, NOM_BEN, N_CONEX, DES_CONEX, N_CENDEST, N_CENADS, N_DEPADS, ENTALM, N_REQ, N_PED, N_CONTRAT, R_FEDCAU, F_VENC, N_CHEQ, ORD_PAG, FOL_TES, CIC_COMP, DECODE(T_177.DIG_AVI,'A','D','B','H')ID_MOVDH,DECODE(T_177.DIG_AVI,'A', T_006.SCTA_DB, 'B',T_006.SCTA_CP) SUBCTA,T_041.N_CONT,N_FACT, DESC_REG, T_177.N_CTACONF FROM T_006, T_059, T_177, T_404, T_408, T_041 WHERE SUBSTR(T_404.CV_AVI,1,2) SUBSTR(T_404.CV_AVI,3,1) BETWEEN '95K' AND SUMREG=NVL(REGCONF,0) AND T_006.N_DOC=T_177.N_POLCONF AND T_006.R_SEC=T_177.R_SECONF AND T_059.N_CONT=T_404.N_CONTEMI AND T_408.CV_DIV= T_059.N_DIV AND T_177.N_CONTEMI = T_404.N_CONTEMI AND T_177.CV_AVI = T_404.CV_AVI AND T_177.N_POLCONF IS NOT NULL AND (T_177.DIG_AVI = 'A' OR T_177.DIG_AVI = 'B') GROUP BY T_404.N_CONTEMI, T_404.CV_AVI, NOM_CONT, N_DIV, NOM_DIV, T_177.N_DOC, T_177.T_DOC, T_177.R_SEC, N_CENAFO, N_DEP, CV_PROV, CV_AUT, N_CONORI, DECODE(T_177.DIG_AVI, 'A',T_006.SCTA_DB,'B',T_006.SCTA_CR), DECODE (T_177.DIG_AVI,'A','D','B','H'), CV_BEN, NOM_BEN, N_CONEX, DES_CONEX, N_CENDEST, N_CENADS, N_DEPADS, FOL_TES, CIC_COMP, T_041.N_CONT, N_FACT, DES_REG, T_177.N_CTACONF, T_177.I_OP, T_177.DIG_AVI

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	8	10	0	0	0	
EXECUTE:	1	2202	4425	1733	65079	10462	0
FETCH:	349	587	1632	0	0	0	5230

EXECUTION PLAN:
SORT(GROUP BY)
NESTED LOOPS
NESTED LOOPS
NESTED LOOPS
NESTED LOOPS
NESTED LOOPS
TABLE ACCESS (FULL) OF 'T_404'
TABLE ACCESS (BY ROWID) OF ' T_059'
INDEX (UNIQUE SCAN) OF 'T059IND1' (UNIQUE)
TABLE ACCESS (BY ROWID) OF 'T_408'
INDEX (UNIQUE SCAN) OF 'T408IND1' (UNIQUE)
TABLE ACCESS (BY ROWID) OF 'T_177'
INDEX (UNIQUE SCAN) OF 'T177IND1' (UNIQUE)
TABLE ACCESS (BY ROWID) OF 'T_006'
INDEX (UNIQUE SCAN) OF 'T006IND1' (UNIQUE)
TABLE ACCESS (FULL) OF 'T_041'

OVERALL TOTALS FOR ALL STATEMENTS

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	2	9	14	0	0	0	
EXECUTE:	2	2202	4425	1733	45079	10462	0
FETCH:	349	587	1632	0	0	0	5230

TOTAL NUMBER OF SQL STATEMENTS: 3
TRACE FILE: 28_14299.TRC

CONCLUSION:

Vemos que efectivamente hubo una gran disminución muy grande tanto en el tiempo del CPU, como en el de enlace, en el número de lecturas (FISICAS, CONCURRENTES y CONSISTENTES).
Notando además que el número de lecturas concurrentes fue igual al anterior debido a que este ocasiona que se traigan los mismos registros de resultado, es decir (5230).
Respecto a la duración, fué de 5 horas para originar 5230 registros.
Por lo tanto, se analizó otra vez con detalle y se pretendió crear un cluster a la tabla T_006 respecto al campo N_DOC, pero no se pudo debido a la cantidad de memoria ocupada.

5.5.2.- CREACION DEL INDICE EN LA TABLA T_177

Procediendo a la creación de un índice a la tabla T_177 respecto al campo R_SECONF se obtiene como resultado del **TKPROF** lo siguiente:

count = Número de veces en que un procedimiento es ejecutado.
cpu = Tiempo del CPU, dado en milisegundos.
elap = Tiempo de enlace, ejecutado en milisegundos.
phys = Número de lecturas físicas.
cr = Número de lecturas consistentes.
cur = Número de lecturas concurrentes.
rows = Número total de registros procesados.

cursor number: 1

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	0	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

```
SELECT T_404.N_CONTEMI, T_404.CV_AVI, NOM_CONT, N_DIV, NOM_DIV, T_177.N_DOC,
T_177.T_DOC, T_177.R_SEC, N_CENAFO, N_DEP, CV_PROV, CV_AUT, N_CONORI, T_177.DIG_AVI,
T_177.I_OP, SUM(DECODE( T_177.DIG_AVI, 'A', T_177.I_OP, 'B', T_177.I_OP)) SUMIPO,CV_BEN,
NOM_BEN. N_CONEX, DES_CONEX, N_CENDEST, N_CENADS, N_DEPADS, ENTALM, N_REQ, N_PED,
N_CONTRAT, R_FEDCAU, F_VENC, N_CHEQ, ORD_PAG, FOL_TES, CIC_COMP, DECODE(T_177.DIG_
AVI,'A','D','B','H')ID_MOVDH,DECODE(T_177.DIG_AVI,'A', T_006.SCTA_DB, 'B',T_006.SCTA_CR)
SUBCTA,T_041.N_CONT,N_FACT, DES_REG, T_177.N_CTACONF FROM T_006, T_059, T_177, T_404,
T_408, T_041 WHERE SUBSTR(T_404.CV_AVI,1,2) SUBSTR(T_404.CV_AVI,3,1) BETWEEN '95K' AND
SUMREG=NVL(REGCONF,0) AND T_006.N_DOC=T_177.N_POLCONF AND T_006.R_SEC= T_177. R_SE -
CONF AND T_059.N_CONT=T_404.N_CONTEMI AND T_408.CV_DIV= T_059.N_DIV AND
T_177.N_CONTEMI = T_404.N_CONTEMI AND T_177.CV_AVI = T_404.CV_AVI AND T_177.N_POLCONF
IS NOT NULL AND (T_177.DIG_AVI = 'A' OR T_177.DIG_AVI = 'B') GROUP BY T_404.N_CONTEMI,
T_404.CV_AVI, NOM_CONT, N_DIV, NOM_DIV, T_177.N_DOC, T_177.T_DOC, T_177.R_SEC, N_CENA -
FO,N_DEP, CV_PROV, CV_AUT, N_CONORI, DECODE(T_177.DIG_AVI, 'A',T_006.SCTA_ DB, 'B',
T_006.SCTA_CR), DECODE (T_177.DIG_AVI,'A','D','B','H'), CV_BEN, NOM_BEN, N_CONEX, DES_CONEX,
N_CENDEST, N_CENADS, N_DEPADS, FOL_TES, CIC_COMP, T_041.N_CONT, N_FACT, DES_REG,
T_177.N_CTACONF, T_177.I_OP, T_177.DIG_AVI
```

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	6	6	0	0	0	
EXECUTE:	1	1697	2164	1742	45079	10462	0
FETCH:	349	497	617	0	0	0	5230

EXECUTION PLAN:

```
SORT(GROUP BY)
  NESTED LOOPS
    NESTED LOOPS
      NESTED LOOPS
        NESTED LOOPS
          NESTED LOOPS
            TABLE ACCESS (FULL) OF 'T_404'
              TABLE ACCESS ( BY ROWID ) OF ' T_059'
                INDEX ( UNIQUE SCAN) OF 'T059IND1' (UNIQUE)
          TABLE ACCESS (BY ROWID) OF 'T_408'
            INDEX ( UNIQUE SCAN ) OF 'T408IND1' (UNIQUE)
        TABLE ACCESS (BY ROWID) OF 'T_177'
          INDEX ( UNIQUE SCAN ) OF 'T177IND1' (UNIQUE)
      TABLE ACCESS (BY ROWID) OF 'T_006'
        INDEX ( UNIQUE SCAN ) OF 'T006IND1' (UNIQUE)
    TABLE ACCESS (FULL) OF 'T_041'
```

OVERALL TOTALS FOR ALL STATEMENTS

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	6	6	0	0	0	
EXECUTE:	2	1697	2164	1742	45090	10462	0
FETCH:	349	497	617	0	0	0	5230

TOTAL NUMBER OF SQL STATEMENTS: 2
TRACE FILE: 11_20454.TRC

Hubo una disminución en los tiempos (CPU y enlace) de la columna EXECUTE y FETCH, debido a la creación de este índice, pero aún se sigue viendo valores bastantes grandes, por lo tanto se procede a la creación de cluster de la tabla T_059 respecto al campo N_CONT y N_DIV. obteniéndose los siguientes resultados en el TKPROF:

count = Número de veces en que un procedimiento es ejecutado.
cpu = Tiempo del CPU, dado en milisegundos.
elap = Tiempo de enlace, ejecutado en milisegundos.
phys = Número de lecturas físicas.
cr = Número de lecturas consistentes.
cur = Número de lecturas concurrentes.
rows = Número total de registros procesados.

cursor number: 1

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	0	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	0	0
FETCH:	0	0	0	0	0	0	0

SELECT T_404.N_CONTEMI, T_404.CV_AVI, NOM_CONT, N_DIV, NOM_DIV, T_177.N_DOC, T_177.T_DOC, T_177.R_SEC, N_CENAFO, N_DEP, CV_PROV, CV_AUT, N_CONORI, T_177.DIG_AVI, T_177.I_OP, SUM(DECODE(T_177.DIG_AVI, 'A', T_177.I_OP, 'B', T_177.I_OP)) SUMIPO,CV_BEN, NOM_BEN, N_CONEX, DES_CONEX, N_CENDEST, N_CENADS, N_DEPADS, ENTALM, N_REQ, N_PED, N_CONTRAT, R_FEDCAU, F_VENC, N_CHEQ, ORD_PAG, FOL_TES, CIC_COMP, DECODE(T_177.DIG_AVI,'A',ID,'B',IP)ID_MOVDH,DECODE(T_177.DIG_AVI,'A', T_006.SCTA_DB, 'B',T_006.SCTA_CP) SUBCTA,T_041.N_CONT,N_FACT, DESC_REG, T_177.N_CTACONF FROM T_006, T_059, T_177, T_404, T_408, T_041 WHERE SUBSTR(T_404.CV_AVI,1,2) SUBSTR(T_404.CV_AVI,3,1) BETWEEN '95K' AND SUMIREG=NVL(REGCONF,0) AND T_006.N_DOC=T_177.N_POLCONF AND T_006.R_SEC=T_177.R_SECONF AND T_059.N_CONT=T_404.N_CONTEMI AND T_408.CV_DIV= T_059.N_DIV AND T_177.N_CONTEMI = T_404.N_CONTEMI AND T_177.CV_AVI = T_404.CV_AVI AND T_177.N_POLCONF IS NOT NULL AND (T_177.DIG_AVI = 'A' OR T_177.DIG_AVI = 'B') GROUP BY T_404.N_CONTEMI, T_404.CV_AVI, NOM_CONT, N_DIV, NOM_DIV, T_177.N_DOC, T_177.T_DOC, T_177.R_SEC, N_CENAFO, N_DEP, CV_PROV, CV_AUT, N_CONORI, DECODE(T_177.DIG_AVI, 'A',T_006.SCTA_DB,'B',T_006.SCTA_CR), DECODE (T_177.DIG_AVI,'A','D','B','H'), CV_BEN, NOM_BEN, N_CONEX, DES_CONEX, N_CENDEST, N_CENADS, N_DEPADS, FOL_TES, CIC_COMP, T_041.N_CONT, N_FACT, DES_REG, T_177.N_CTACONF, T_177.I_OP, T_177.DIG_AVI

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	2	11	18	0	0	0	
EXECUTE:	2	2910	3942	1828	89348	20942	0
FETCH:	351	530	623	0	0	0	5230

EXECUTION PLAN:
SORT(GROUP BY)
NESTED LOOPS
NESTED LOOPS
NESTED LOOPS
NESTED LOOPS
NESTED LOOPS
TABLE ACCESS (FULL) OF 'T_408'
TABLE ACCESS (CLUSTER) OF 'T_059'
INDEX (UNIQUE SCAN) OF 'POP2' (CLUSTER)
TABLE ACCESS (CLUSTER) OF 'MOMI'
INDEX (RANGE SCAN) OF 'POP1' (CLUSTER)
TABLE ACCESS (BY ROWID) OF 'T_177'
INDEX (RANGE SCAN) OF 'T177IND1' (NON-UNIQUE)
TABLE ACCESS (BY ROWID) OF 'T_006'
INDEX (UNIQUE SCAN) OF 'T006IND1' (UNIQUE)
TABLE ACCESS (FULL) OF 'T_041'

OVERALL TOTALS FOR ALL STATEMENTS

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	2	11	22	0	0	0	
EXECUTE:	3	2910	3942	1828	89348	20924	0
FETCH:	351	530	623	0	0	0	5230

TOTAL NUMBER OF SQL STATEMENTS: 2
TRACE FILE: 49_29542.TRC

CONCLUSION FINAL:

Con la creación del cluster en la tabla T_059, no se pudo obtener resultados adecuados, sino al contrario se vio que hubo aumento en algunos, por lo tanto los resultados más adecuados de acuerdo a los obtenidos en los casos anteriores, fué cuando se accesa el índice a la tabla T_006.

El objetivo de analizar este ejemplo, fué en observar como una mala colocación de tablas dentro de la cláusula FROM produce tanto un aumento de lecturas y tiempos muy exagerados.

5.6.- EJEMPLO NO. 5

En el siguiente ejemplo, se pretende mostrar lo dicho anteriormente acerca del comando insert, update y delete, que el número de registros retornados serán presentados en la columna EXECUTE dentro del análisis del TKPROF.

Por otro lado se analizarán diferentes selects y através de sus resultados, se tendrá que detectar cual es su problema, es decir, creación de índices o cambios de cláusulas ó creación de cluster.

Antes de proceder al análisis de estos ejemplos, no cabe de más mencionar que un análisis del TKPROF puede realizarse de 2 formas:

1.- A NIVEL DE INSTANCIA.
2.- A NIVEL DE USUARIO.

1.-Nivel de instancia, se refiere al análisis de un programa completo, es decir, este programa puede contener comandos como (insert, delete, update, select). Cuando se analiza un programa completo, es necesario que dentro del INIT. ORA, se cambie el parámetro SQL_TRACE=TRUE y sobre todo se el programa es muy grande, es necesario cambiar el parámetro MAX_DUMP_ FILE _ SIZE.

2.-Nivel de usuario, se refiere a un análisis sencillo, es decir, si se tiene un programa completo y se quiere analizar únicamente aquel query que contenga un comando como (insert, delete, update, select), se puede hacer.
Dicho de otra manera, dentro de este nivel nunca se va analizar un programa completo.
Cuando se procede a este análisis, es necesario checar dentro del INIT.ORA , el parámetro SQL_TRACE=FALSE.

Procedamos al análisis de varios ejemplos:

1)

= =

SELECT N_SCTACT,1 INTO :b1, :b2 FROM T_052 WHERE N_CTA=:b3 AND CV_SCTA=:b4

Resultado:

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	118	0	0	5	40	0	
EXECUTE:	1204	0	0	0	0	0	0
FETCH:	1204	0	0	144	4818	0	1240

NOTA:

En este ejemplo, se hace notar lo siguiente:

1. **Cuando se tiene un número cero, significa que todo esta en memoria.**
2. **Las lecturas que se obtuvieron en el parse son muy grandes, para analizar únicamente un select, por lo tanto se procede a la modificación del parámetro HOLD_CURSOR=YES (explicado con más detalle, en el apéndice D).**
3. **Se realizaron 4 lecturas consistentes, por cada registro obtenido, es decir, hubo una escala de 4:1**

2)

UPDATE T_006 SET CV_VAL='1' WHERE N_DOC=:b1 AND CTA_DBB=:b2 AND SCTA_DB=:b3 AND (CV_VAL IS NULL OR CV_VAL=' ')

Resultado:

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	115	0	0	0	0	0	
EXECUTE:	899	0	0	0	351592	35109	11703
FETCH:	0	0	0	0	0	0	0

NOTA:

En este ejemplo, se hace notar lo siguiente:

1.- **El número de lecturas consistentes exageradamente grande, indica que es necesario proceder a la creación del índice.**
2.- **Se realizaron 30 lecturas consistentes, por cada registro obtenido, es decir, hubo una escala de 30:1.**

SELECT CTA_CRB, SCTA_CR, SUM (I_OP), COUNT(*) FROM T_006 WHERE N_DOC=:b1 AND CV_VAL='1' GROUP BY CTA_CRB, SCTA_CR

Resultado:

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	115	0	0	0	0	0	
EXECUTE:	115	0	0	0	24088	0	0
FETCH:	420	0	0	0	0	0	305

NOTA:

En este ejemplo, se hace notar lo siguiente:

1.- **Lectura consistente muy grande, debido a la cláusula count(*).**
2.- **Se registra más de una lectura FETCH por registro obtenido.**

3)

UPDATE T_006 SET CV_VAL='2' WHERE N_DOC=:b1 AND CTA_CRB=:b2 AND SCTA_CR=:b3 AND CV_VAL='1'

Resultado:

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	115	0	0	0	0	0	
EXECUTE:	305	0	0	0	106752	35109	11703
FETCH:	0	0	0	0	0	0	0

NOTA:

En este ejemplo, se hace notar lo siguiente:

1. **Número de lectura consistente exageradamente grande, por lo tanto se procede a la creación del índice.**
2. **Se realizaron 9 lecturas consistentes, por cada registro obtenido, es decir, hubo una escala de 9:1.**

4)

= =

SELECT N_DOC, T_DOC,F_ELAB, TO_NUMBER (TO_CHAR(F_ELAB,'MMYY')), TO_NUMBER(TO_CHAR(ADD_MONTHS(F_ELAB,-1), 'MMYY')) FROM T_004 WHERE AR_FUN=:b1 AND F_ELAB> =:b2 AND F_ELAB<= :b3 AND CV_ACT='3'

Resultado:

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	1	0	0	0	0	0	0
FETCH:	116	0	0	15	400	0	115

NOTA:

En este ejemplo, no se detecta ningún problema, por lo tanto se dice que son correctas todas sus declaraciones.

5)

= =

INSERT INTO T_013 SELECT :b1 , :b2, T_006.R_SEC, :b3, :b4, T_007.N_DEP, :b5, T_007.N_CENAFO, T_007.CV_AUT, T_007.CV_PROY, T_007.FOL_ADE, T_007.F_VENC, T0_006.I_ME, T0.004,.T_MON, T0_004.T_CAMOP, T_007. N_ CONORI, T_006.I_OP, T_007.DOC_FTE, T_007.DIG_ADE, T_006.CTA_DBB, T_006.CTA_CRB, T_006.CTA_CRO, T_006.DIG_ESP, T_006.DIG_AJ FROM T_006, T_003, T_007, T_004, T0_004, T0_006 WHERE T_003.N_DOC=:b6 AND T0_006.N_DOC=T_003.N_DOCORI AND T0_006.T_DOC=T_003.T_DOCORI

AND T0_006.R_SEC=T_003.R_SECORI AND T_007.N_DOC=T_003.N_DOCORI AND T_007.R_SEC=T_003.R_SECORI AND T0_004.N_DOC=T0_006.N_DOC AND T_006.CTA_CRO IN (7306,7309,7314,7315,7316,7317,7318, 7319, 7326, 73-28, 7329,7330,7336,7339,7381,7383) AND T_006.N_DOC=T_003.N_DOC AND T_006.R_SEC=T_003.R_SEC AND T_006.CV_VAL='4'

Resultado:

	COUNT	CPU	ELAP	PHYS	CR	CUR	ROWS
PARSE:	1	0	0	0	0	0	
EXECUTE:	391	0	0	9083	887269	0	0
FETCH:	0	0	0	0	0	0	0

NOTA;

En este ejemplo, se detecta un número muy grande de lecturas físicas dentro de la columna EXECUTE, originando un número de lecturas consistentes demasiado grande,por lo tanto la solución esta en cambiar la cláusula IN por la cláusula EXISTS.

CONCLUSIONES

La afinación de aplicaciones es un paso muy importante que debe ser consultado en la realización de cualquier Base de Datos, para ver si efectivamente se está utilizando los objetos correctos de dicha Base, estableciéndose bases correctas para el intercambio de datos entre dicho personal, promoviendo con esto un ambiente cooperativo para un fin común.

La existencia de ambientes cooperativos en cada departamento, contribuye a disminuir ó evitar la duplicidad de información, así como la multiplicación de esfuerzos que impactan directamente en los costos y calidad de información proporcionada por los sistemas informáticos, siempre y cuando se cuente con elementos de software y hardware necesarios para dar paso a las construcciones de aplicaciones amigables capaces de conjugar las capacidades de las Bases de Datos formales existentes en equipos mayores, dando como resultado la información correcta en el lugar y en el momento oportuno.

Finalmente la calidad de información, debe ser clara para todo usuario final ya sea através de un formato escrito ó gráfico que permita visualizar en forma rápida y clara los puntos de operación en donde existen desviaciones y apartir de ellos llevar a cabo la toma de decisiones para lograr optimizar los recursos existentes y planear las necesidades futuras.

APENDICE A

"DISEÑO DE DATOS"

A.1.-CONSIDERACIONES DEL DISEÑO RELACIONAL

- Datos que se identifican para alcanzar el diseño relacional.
- Ventajas y desventajas que se obtendrían,al eliminar redundancia.
- Determinar el mejor diseño determinado por su modelo entidad relación.

- Uno de los primeros pasos al diseñar una aplicación, es determinar la estructura de las tablas que contendrán los datos.

- Almacenar el mismo dato en diferentes formas,seleccionando una condición relacional diseñada para tener muchos beneficios.

- Cualquier aplicación diseñada por primera vez deberá acercarse a la teoría relacional y sus estructuras de normalización.

A.2.- SELECCION DEL DISEÑO DE DATOS

- Un diseño relacional que no almacena datos redundantes, es optimamente el más apropiado para las aplicaciones.

- Los diseños relacionales ofrecen gran flexibilidad,con buen rendimiento para declaraciones SQL.

- Si el rendimiento es el principal objetivo se puede encontrar un prototipo de un diseño relacional que proporcione mayor rendimiento a las aplicaciones.

- Si el rendimiento de la aplicación disminuye debido a el diseño de las tablas en la base de datos, se puede considerar la reestructuración de los datos.

-La reestructuración de los datos puede involucrar modificaciones considerables a la aplicación.Considerando que un buen diseño de la base de datos servirá para aplicar mejores rendimientos.

APENDICE B

"INDICES"

Los indices mejoran la ejecución de querys, al seleccionar un pequeño porcentaje de los registros de una tabla.

Como una pista usan indices entre un 10% ó 15% de los registros de la tabla.

Los tipos de indices existentes son:

1.- INDICES UNIQUE
2.- INDICES CONCATENADOS
3.- INDICES MULTIPLES

Los indices son estructuras de la base de datos que aumentan el rendimiento del query. Los indices son usados en conjunto con las columnas de las tablas. Un índice asocia cada valor distinto de una columna con los registros en la tabla. La columna que tiene el índice es llamada columna "LLAVE". Para crear un índice use la cláusula CREATE INDEX.

B.1.-INDICES UNIQUE

Algunos índices a columnas pueden imponer registros especiales entre los valores de las diferentes columnas, este índice es llamado "UNIQUE ". El índice unique no se crea, por lo menos si existieran 2 líneas en la tabla que contengan el mismo valor en la columna indexada.

B.2.-INDICES CONCATENADOS

Un índice que puede generarse con más de una columna, es llamado "INDICE CONCATENADO".

Los índices concatenados mejoran el rendimiento en :

- Alta selectividad.
- Almacenamiento de datos adicionales.

Los índices concatenados son útiles en los SELECT's.

Algunas veces 2 columnas con baja selectividad pueden ser candidatos a producir un índice concatenado.

Los índices concatenados también pueden ser usados para retornar datos adicionales de las columnas del query. Si un query selecciona una columna que no forma parte del índice, ORACLE debe descubrir el ROWID en el índice y entonces recupera el registro correspondiente de la tabla.

B.3.-SELECCION DE COLUMNAS PARA INDICES CONCATENADOS

- Las columnas que incorporan un índice concatenado son referidas como " LLAVE CONCATENADA".

- Un SQL usa un índice concatenado, al determinar el contenido de las columnas en la cláusula WHERE y el orden de las columnas al momento del "CREATE INDEX ".

- Un query puede, únicamente usar un índice concatenado si este hace referencia a la parte principal del WHERE.

- La porción principal de el índice concatenado se refiere a las primeras columnas especificadas en el comando CREATE INDEX.

B.4.- INDICES MULTIPLES

Un query con 2 o más predicados sobre la misma tabla, puede usar índices múltiples si:

- Los índices son índices de columnas no-unique.
- Los predicados son equitativos.

Los datos regresados por cada acceso del índice, se unirán con resultados previos, hasta obtener el resultado final. Los índices múltiples no son usados para predicados tal como (DEPTNO != 10) como en:

```
SELECT * FROM EMP
WHERE JOB='MANAGER'
AND DEPTNO != 10;
```

El índice para la igualdad (JOB='MANAGER') lo manejará el query.

B.5.- SELECCION ENTRE INDICES MULTIPLES

- En queries con índices múltiples disponibles pero no iguales, ORACLE selecciona el manejo de índices basado sobre los tipos de índices y características de las columnas.

- Cuando ambos índices,unique y no-unique en un query, se pueden disponer de ellos, ORACLE usa el índice unique e ignora el índice no-unique,evitando el "merge".

Por ejemplo, en el siguiente query, un índice unique sobre EMPNO y un índice no-unique sobre SAL;

```
SELECT ENAME FROM EMP
  WHERE SAL = 3000
    AND EMPNO = 7902;
```

ORACLE usará únicamente el índice EMPNO. Si un registro es descubierto con EMPNO 7902, el actual campo SAL con el valor de 3000, es probable que el optimizador use el índice sobre SAL.

Cuando se requiere del uso de unidades léxicas, en el criterio de selección de columnas que pertenecen a un índice, este no podrá usarse, como en los siguientes predicados:

```
WHERE SAL*12=24000
WHERE SAL+0 =500
WHERE '' || ENAME = 'SMITH'
WHERE SUBSTR(ENAME,1,1) = ' S '
```

Así mismo, cuando use cláusulas de excepción en columnas indexadas (que no se han alterado con unidades léxicas), el optimizador podrá utilizar el índice, solo si en la cláusula SELECT, se usa una expresión como la siguiente:

{ MIN | MAX } (COL { + | - } CONSTANTE)

B.6.- CUANDO USAR UN INDICE

Un índice puede ser usado si:

- Este es referenciado en un predicado.
- Un predicado es una porción del criterio de selección usado al incluir o excluir líneas desde un resultado.
- La columna indexada no es modificada por una función u operación aritmética.

Un índice será usado si el optimizador decido que es el momento apropiado.

B.7.- CUANDO NO USAR UN INDICE

Un índice no será usado si:

- No hay una cláusula **WHERE**.
- El predicado modifica las posiciones de las columnas indexadas en cualquier paso.
- En el predicado se usa explícitamente, para la búsqueda de registros, los valores **NULL** ó **NOT NULL** en la columna indexada.

B.8.- CARACTERISTICAS

- Los índices se usan para optimizar querys.
- Cuando un query que hace referencia a una columna indexada por medio de la cláusula WHERE. ORACLE investiga si el índice esta de acuerdo a los valores especificados en la cláusula WHERE.
- Si el query contiene las columnas relecionadas con el índice, leerá los valores directamente por medio del índice, en lugar de leerlos desde la tabla.
- Por cada valor, el índice tambien identifica las localizaciones físicas con el "ROWID" de la líneas en la tabla.
- Si el query selecciona datos relacionados con los valores indexados, ORACLE encuentra los registros en la tabla por medio del ROWID.
- La búsqueda por ROWID es el camino más rápido para que ORACLE localice un registro.

B.9.- VALORES NULL EN EL INDICE

Cuando se definen columnas de tablas como NOT NULL, se permitirá usar un índice sobre un insignificante número de queries. Los índices no son usados si el predicado contiene una de las 2 frases:

IS NULL
IS NOT NULL

Sin embargo, si quisieramos obtener los valores NOT NULL, podríamos escribir la declaración SQL, para usar el índice. Por ejemplo, para ver todos los empleados que tengan comisión (donde la comisión es NOT NULL), se podría usar uno de los siguientes queries:

QUERY1:

```
SELECT * FROM EMP
WHERE COMM IS NOT NULL
```

QUERY2:

```
SELECT * FROM EMP
WHERE COMM>=0
```

Asumiendo que un índice existe sobre la columna COMM, este no se usa en el QUERY 1, pero si por el QUERY 2 sin hacer un FULL TABLE SCAN. Sin embargo, en la tabla EMP la mayoría de los registros contienen valor NULL, es más conveniente usar el QUERY 1 que el QUERY 2.

B.10.- SELECCION DE COLUMNAS A INDEXAR

Pistas:

- Indexar columnas comúnmente usadas en la cláusula WHERE.
- Indexar columnas en la cual, los valores máximos ó mínimos son frecuentemente seleccionados.
- Indexar columnas que son usadas frecuentemente uniendo tablas.
- Indexar columnas con mayor selectividad.
- No hacer índices de columnas con pocos valores distintos, en donde las columnas tienen baja selectividad.
- No hacer índices de columnas en tablas pequeñas.
- No hacer índices de columnas que son altamente modificadas.

La " SELECTIVIDAD " es alta si son pocos registros extraidos, con el valor de la columna llave. Para esto, los índices unique son los más efectivos.

Si una tabla usa menos de 5 bloques de datos, un FULL TABLE SCAN puede regresar los registros más rápidamente que un query indexado. Se puede determinar cuantos bloques de datos son usados examinando el ROWID.

El UPDATE actualiza columnas indexadas, pero el INSERT y DELETE modifican tablas indexadas que afectan el tiempo de respuesta al rearmar el índice.

Cuando seleccionamos las columnas para un índice, consideremos la ganancia en tiempo de ejecución para el query en las instrucciones INSERT, UPDATE y DELETE, evaluando el tiempo de procesamiento con y sin índices. El tiempo se puede medir con "SET TIME ON".

APENDICE C

"CANDADOS"

C.1.- ADMINISTRACION DE CANDADOS A NIVEL REGISTRO

Caracteristicas:

- Como se administran los candados a registros al aumentar el rendimiento.
- Como toma ventajas el manejador de los candados a registros.
- Como monitorear actividades de los candados.

C.2.-COMO MEJORAR EL RENDIMIENTO CON CANDADOS A NIVEL REGISTRO

Si se instala ORACLE con la opción de procesamiento de transacciones de datos a nivel registro (llamado TPO):

Permite a usuarios accesar a diferentes registros de la misma sin poner en contención la Base de Datos.

Las aplicaciones que corren en el OLTP (Online Trasaction Processing) se benefician con la ganancia en los rendimientos y tiempos de respuesta.

Las aplicaciones al OLTP se caracterizan porque múltiples usuarios consultan y/o actualizan concurrentemente diferentes registros de la misma tabla al mismo tiempo.

Tómese en cuenta que los lectores nunca esperan a los escritores y los escritores nunca esperan a los lectores.

C.3.- VENTAJAS EN LOS CANDADOS A NIVEL REGISTRO

El candado a nivel registro no requiere de implementaciones especiales sobre la parte de el desarrollo de aplicaciones, el único requisito, es que instale el manejador con esta opción. Considérese que los candados a nivel tabla, no permite que los candados a nivel registro se puedan usar.

El candado a nivel de tabla reduce los beneficios y el rendimiento de la opción de procesamiento de transacción (TPO).

Los candados a nivel de tabla se establecen explícitamente al solicitarse, como se observa en las siguientes instrucciones:

```
LOCK TABLE EMP IN EXCLUSIVE MODE
LOCK TABLE DEPT IN SHARE MODE
```

C.4.- MONITOREO

Para observar la actividad entre los candados a nivel Tabla o Registro, así como la detección de contención (dead locks) entre a:

```
$sqldba
 sqldba > connect internal
 sqldba > monitor
```

APENDICE D

"HOLD-CURSOR Y REBIND"

Correspondientes a los parámetros del precompilador, que afectan a las áreas de contexto (ó cursores) usadas por los programas y procesos que usan la PGA (Program Global Area) y el SPA (Shared Poll Area).

El parámetro HOLD_CURSOR, permite que cada instrucción DML especifica como usar los bloques en memoria cache.

HOLD_CURSOR= NO (DEFAULT)	Una vez ejecutado el estado DML el área de contexto es liberado para ser usado por otros cursores.
HOLD_CURSOR= YES	Si se requiere hacer ejecuciones continuas de un mismo DML y no perder la liga entre el área de contexto y el cursor.

El REBIND especifica la manera en que las variables host son manejadas.

REBIND= YES (DEFAULT)	No mantienen las direcciones de las variables host entre una ejecución y otra, como en las subrutinas ó user exists.
REBIND= NO	Mejora el rendimiento de los procesos de las variables host al mantenerlas desde el primer parse.

Para usarse es necesario declararlos en el makefile, en la parte de las opciones del precompilador.

EJEMPLO:

```
PCCFLAGS = ireaclen=256        oreclen = 256
           select_error=no     sqlcheck=none
           hold_cursor=yes     rebind=no
```

APENDICE E

"LECTURAS Y ESCRITURAS"

La memoria cache, es la localidad que almacena copias de datos residentes en disco para que puedan ser accesados rápidamente.

Los procesos de los usuarios se ven beneficiados al efectuar las operaciones de lectura y escritura sin tener que hacer intercambio de bloques de datos a disco ("SWAP") ó lo que se conoce como operaciones de I/O a disco.

Cuando el proceso de usuario accesa directamente todos los bloques de datos que se encuentran almacenados en memoria cache, sin tener que ir a disco, se llama " CACHE HIT ". Por el contrario, cuando el proceso del usuario requiere de operaciones I/O y "SWAP" al disco, lo llamamos "CACHE MISS". Obviamente que los datos con " CACHE HITS " proporcionan un porcentaje mucho menor en el tiempo de respuesta que los que hacen " CACHE MISSES".

El tamaño de la memoria cache, nos da la probabilidad de que los bloques de datos residan o no en estas.

El porcentaje total de los datos que son accesados a memoria se le conoce como HIT RADIO.

ORACLE, calcula la estadística del HIT RADIO, desde las estadísticas de lectura física y lógica con la siguiente fórmula:

HIT RADIO= (LECTURA LOGICA - LECTURA FISICA) / (LECTURA LOGICA)

El monitor de I/O en SQL*DBA contiene 2 partes:

INTERVAL.- La mitad izquierda de la pantalla es llamada intervalo. Las estadísticas en esta mitad de pantalla reflejan las I/O que han ocurrido desde la última actualización en ORACLE.

CUMULATIVE.-La mitad derecha de la pantalla es llamada acumulativa. Las estadísticas en esta mitad de pantalla reflejan las I/O que han ocurrido desde que se empieza a observar la pantalla.

Si el HIT RADIO es bajo, es decir menor del 60% ó 70%, entonces se puede incrementar el número de buffers en memoria para mejorar los tiempos de ejecución. Para hacer el cache buffer más largo, se incrementa el valor en el parámetro DB_BLOCK_BUFFERS de el INIT.ORA.

APENDICE F

"ROWID"

Es una representación hexadecimal de las direcciones del registro. La dirección puede ser obtenida através de un SELECT seguido por la palabra ROWID.

EJEMPLO:

```
SELECT ENAME, ROWID
FROM EMP
```

Resultado:

ENAME	ROWID
SMITH	0000054A.0000.0001
ALLEN	0000054A.0001.0001
WARD	0000054A.0002.0001
JONES	0000054A.0003.0001
MARTIN	0000054A.0004.0001

El ROWID, puede ser usado para:

- Obtener información del espacio utilizado sobre una tabla.
- Para tener un acceso rápido de registro.

Tiene el siguiente formato:

0000054A	0000	0001
⇓	⇓	⇓
Número de bloque lógico dentro del archivo	Número de registro dentro del bloque	Número de archivo en la B.D.

-Todo número de bloque, es relacionado de acuerdo al archivo.
-Todo número de bloque es distinto dentro del archivo.
-Todo número de registro, siempre empieza con cero.
-Todo número de archivo, es distinto dentro de una base de datos.

Para obtener una información clara y completa del ROWID, se puede usar algunas funciones del SQL, por ejemplo:

- Usando la función SUBSTR, permite analizar el ROWID,dentro de 3 componentes FILE, BLOCK y ROW, como a continuación se muestra:

```
SELECT ROWID, SUBSTR(ROWID,15,4) "FILE", SUBSTR( ROWID,1,8) "BLOCK",
              SUBSTR (ROWID,10,4) "ROW"
FROM EMP
```

Resultado:

ROWID	FILE	BLOCK	ROW
0000054A.0000.0001	0001	0000054A	0000
0000054A.0001.0001	0001	0000054A	0001
0000054A.0002.0001	0001	0000054A	0002

Obteniéndose este resultado de la siguiente manera:

1.-El 15,4 localizado dentro del primer SUBSTR, nos indica que desde el primer número que aparezca en la columna ROWID, se contaran 15, los próximos 4 números después, son los que aparecerán en la columna "FILE".

2.-El 1,8 localizado dentro del segundo SUBSTR, nos indica que desde el primer número que aparezca en la columna ROWID hasta el octavo, son los que aparecerán en la columna "BLOCK".

3.-El 10, 4 localizado dentro del tercer SUBSTR, nos indica que desde el primer número que aparezca en la columna ROWID, se contaran 10, los próximos 4 números después, son los que aparecerán en la columna "ROW".

APENDICE G

"TABLAS"

TABLA EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	07-JUN-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7834	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	11-JUL-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

TABLA DEPT

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

TABLA SALGRADE

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

BIBLIOGRAFIA

1.- Branca Bill
"ORACLE TOOLS FOR UNIX"
Oracle, México 1993.

2.- Cheu Dwight
"SQL LANGUAJE"
Oracle, México 1990.

3.- Dimmick Shelly
"ORACLE RDBMS"
Oracle,México 1990.

4.- Faden Elly
"ORACLE RDBMS"
Oracle, México 1991.

5.- Jernigan Kevin
"SQL*PLUS"
Oracle, México 1988.

6.- Kock George
"ORACLE 6"
Mc Graw-Hill, México 1992.

7.- Linden Brian
"ORACLE RDBMS"
Oracle, México 1990.

8.- Linden Brian
"SQL LANGUAJE"
Oracle, México 1989.

9.- Oxford University Press
"DICCIONARIO DE INFORMATICA"
Díaz de Santos, España, 1993.

10.- Pemex
"BOLETIN DE INFORMATICA"
Pemex,México 1990.

11.- Talavera,J.Carlos
"BASES DE DATOS "
ITAM, México 1993.

GLOSARIO

CACHE.- Es una localidad de memoria que permite el acceso de datos residentes en disco.

DATABASE LINK.- Es un objeto de la Base de Datos, que contiene la información necesaria del usuario, equipo y Base de Datos para establecer una conexión remota.

INSTANCIA.- El área de memoria, reservada por el manejador de Base de Datos llamada SGA (System Global Area), junto con sus procesos asíncronos que permiten la comunicación entre el manejador y la extracción y/o grabación a las áreas reservadas en disco.

LECTURAS CONSISTENTES Son lecturas originadas por el propio manejador, teniendo la función de checar de que todos los datos existentes en una Base de Datos sean correctos.

LECTURAS CONCURRENTES Cuando múltiples usuarios compiten por el uso de un mismo recurso ó registro. La suma de estas lecturas junto con las consistentes, es igual al número de bloques de datos que fueron accesados en el momento de la consulta. Para resolver este problema, el manejador de la Base de Datos es el que decide el orden en que los usuarios establecerán dicha lectura.

LECTURAS FISICAS.- Son bloques extraídos desde el disco para ser depositadas en la memoria cache.

LECTURAS LOGICAS.- La lectura de los bloques en memoria previa a la lectura física.

QUERY.- Es un tipo de declaración SQL, usado para extraer datos. Un query frecuentemente empieza con la palabra reservada " SELECT " de SQL.

SCRIPTS.- Archivo que contiene un procedimiento de información.

SEQUENCE.- Objeto de la Base de Datos que nos otorga secuencias ascendentes, tales como folios.

SGA.- Es la memoria compartida por todos los usuarios de la Base de Datos. Mejora la comunicación entre el usuario y el proceso.

SUBSET.- Cuando existe un JOIN, que usa subqueries en la cláusula WHERE.