OPTIMIZACIÓN DE CONSULTAS SQL

INDICE

1. CONCEPTOS GENERALES

1.1 Optimización de consultas

Optimización es el proceso de elegir la vía más eficiente para resolver una consulta SQL.

1.2 Tipos de Sentencias SQL

Sentencia simple: Insert, update, delete, select sobre una única tabla.
Consulta simple: Consulta es otro nombre para la sentencia SELECT.
Join: Join es una consulta que selecciona datos de mas de una tabla. Se caracterizan por tener más de una tabla en la cláusula FROM, la condición de la cláusula WHERE es llamada condición del join.
Equijoins: Joins con condición de igualdad.
Nonequijoins: Cuando la condición no tiene igualdades.
Outerjoins: Se caracterizan por tener condiciones con el operador outer join (+).
Producto cartesiano: Son realizados cuando no se tiene una condición en el join. Cada registro de la primera tabla es apareado con todos los de la segunda tabla.
Sentencias complejas: Una sentencia compleja es un SELECT, INSERT, UPDATE, DELETE que contiene una subconsulta.
Consultas compuestas: Una consulta compuesta es aquella que contiene un operador de UNION, UNION ALL, INTERSECT o MINUS para combinar dos sentencias.
Sentencias accediendo a vistas: Se pueden escribir sentencias simples, joins, complejas y compuestas accediendo a vistas, igual que a las tablas.
Sentencias distribuidas: Son aquellas que acceden a datos de bases de datos remotas.

2. OPTIMIZACION DE SENTENCIAS SQL

Para poder realizar una buena optimización de una aplicación se debe conocer que es lo que la aplicación hace:

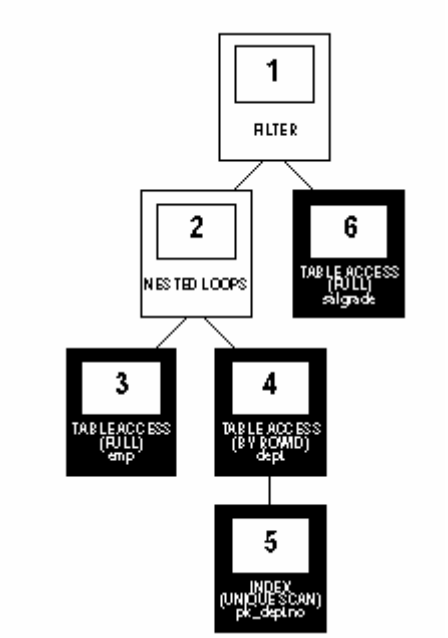
- Cuales son las sentencias SQL que utiliza la aplicación
- Que datos procesa la aplicación
- Cuales son las características y la distribución de esos datos
- Que operaciones la aplicación realiza sobre esos datos.

2.1 Plan de ejecución

Es la forma en la cual oracle accede a los datos para resolver la consulta.
Ejemplo :

```
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 cada paso de ejecución del plan se retornan un conjunto de filas que son usadas por el siguiente paso



Las cajas sombreadas indican extracción física de datos y objetos de la base de datos.

- Los pasos 3 y 6 leen todos los registros de las tablas EMP y SALGRADE, respectivamente.
- El paso 5 busca el ROWID de DEPT con el código de departamento entregado en el paso 3 en el índice PK_DEPTNO.

- El paso 4 trae de la tabla DEPT la fila correspondiente al ROWID obtenido en el paso 5.

Los pasos en las cajas blancas indican operaciones en los registros.

- El paso 2 realiza la operación nested loop (bucle anidado), acepta registros de los pasos 3 y 4, realizando el join y retornando el resultado al paso 1.
- El paso 1 realiza una operación de filtro acepta filas de 2 y 6, elimina las filas del paso 2 que no tengan correspondiente en el paso 6.

Orden de realización de las operaciones

Oracle no realiza las operaciones en el orden que son numeradas. Oracle realiza primero las operaciones correspondientes a las hojas del árbol representado.

Los pasos seguidos por oracle son los siguientes:
- Primero oracle realiza el paso 3 y retorna uno a uno los registros al paso 2
- Para cada registro retornado por el paso 3 oracle realiza los siguientes pasos
- Realiza el paso 5 retornando el rowid al paso 4
- Realiza el paso 4 y retorna el registro al paso 2
- Realiza el paso 2 aceptando un registro del paso 3 y uno del 4 y retornando un registro al paso 1.
- Oracle realiza el paso 6 y retorna el registro resultante, si hay alguno al paso 1.
- Oracle realiza el paso 1, si un registro es retornado por el paso 6 retorna el registro entregado por el paso 2 al usuario que ejecuto el select

Nótese que la ejecución se realiza registro a registro, y no primero un paso y luego el otro.
Se puede consultar el plan de ejecución de una consulta con la sentencia EXPLAIN PLAN.

2.2 Tipos de optimización

Basada en reglas: La optimización se realiza por el camino de acceso disponible.
Basada en costo: La optimización se realiza por el camino de acceso disponible y una información estadística de la tabla que se encuentra en el diccionario de la base de datos.
La información estadística es una forma aproximada de obtener la cantidad de registro que puede retornar la consulta en cada paso.

2.3 Tabla de pesos por tipo de acceso

Rank	Access Path	Método de acceso
1	Single row by ROWID	Una sola fila por ROWID
2	Single row by cluster join	Una sola fila por cluster join
3	Single row by hash cluster key with unique or primary key	Una sola fila por la clave de un hash cluster con clave única o primaria.
4	Single row by unique or primary key	Una sola fila por clave única o primaria
5	Cluster join	Cluster join
6	Hash cluster key	Clave de hash cluster

7	Indexed cluster key	Clave de cluster indexado
8	Composite key	Clave compuesta
9	Single-column indexes	Indices sobre una sola columna
10	Bounded range search on indexed columns	Búsqueda en un rango limitado sobre columnas indexadas
11	Unbounded range search on indexed columns	Búsqueda en un rango sin límites sobre columnas indexadas
12	Sort-merge join	Join sort-merge
13	MAX or MIN of indexed column	MAX o MIN de una columna indexada
14	ORDER BY on indexed columns	ORDER BY en columnas indexadas
15	Full table scan	Búsqueda sobre la tabla completa

2.4 Tipos de accesos a los datos

Path 1. – Single Row by Rowid. Este acceso es solo disponible cuando en la cláusula where se especifica la columna ROWID o la sentencia CURRENT OF CURSOR de SQL embebido.

Ejemplo :
```
SELECT * FROM emp WHERE ROWID = '00000DC5.0000.0001';
```

El plan de ejecución es el siguiente:

```
OPERATION               OPTIONS         OBJECT_NAME
-----------------------------------------------------
SELECT STATEMENT
        TABLE ACCESS                                    BY ROWID                EMP
```

Path 2. – Single Row by Cluster Join. Este acceso esta disponible para sentencias join de tablas grabadas en algún cluster y las condiciones son las siguientes:

- La cláusula WHERE contiene condiciones de igualdad en las columnas de la clave del cluster de ambas tablas.
- La cláusula WHERE contiene condiciones que garantizan que el join retorna solamente una fila. Esto es cuando se tiene condiciones para todos los campos de la clave primaria.

Ejemplo:
```
SELECT *
        FROM emp, dept
        WHERE emp.deptno = dept.deptno
          AND emp.empno = 7900;
```

El plan de ejecución es el siguiente:
```
OPERATION               OPTIONS          OBJECT_NAME
-----------------------------------------------------
SELECT STATEMENT
        NESTED LOOPS
          TABLE ACCESS          BY ROWID        EMP
                INDEX                   UNIQUE SCAN     PK_EMP
          TABLE ACCESS          CLUSTER         DEPT
```

PK_EMP es el nombre del índice de la clave primaria de EMP

Path 3. – Single Row by Hash Cluster Key with Unique or Primary Key. Este acceso se da cuando se cumplen las siguientes condiciones:

- La sentencia WHERE usa todas las columnas del hash cluster key con condiciones de igualdad. Para varias columnas las condiciones están unidas por AND.
- La sentencia garantiza que se retorna una sola fila porque el hash cluster es único o una clave primaria.

```
Ejemplo:
SELECT *
        FROM orders
        WHERE orderno = 65118968;
```

El plan de ejecución es el siguiente:

```
OPERATION                       OPTIONS                 OBJECT_NAME
-----------------------------------------------------
SELECT STATEMENT
        TABLE ACCESS            HASH            ORDERS
```

Path 4. – Single Row by Unique or Primary Key. Este acceso es cuando en la cláusula WHERE se colocan condiciones de igualdad unidas por AND para todos los campos de una clave primaria o un índice único. Para ejecutar la sentencia, Oracle accede al índice para obtener el ROWID y luego con éste accede al registro consultado.

Ejemplo :

```
SELECT *
        FROM emp
        WHERE empno = 7900;
```

El plan de ejecución sería el siguiente:

```
OPERATION               OPTIONS          OBJECT_NAME
-----------------------------------------------------
SELECT STATEMENT
  TABLE ACCESS  BY ROWID        EMP
        INDEX           UNIQUE SCAN     PK_EMP
```

PK_EMP es el nombre de la clave primaria de EMP por EMPNO.

Path 5. – Cluster Join. Este acceso es cuando en la cláusula WHERE se colocan condiciones de igualdad unidas por AND para todos los campos de un cluster.

Ejemplo:

```
SELECT *
        FROM emp, dept
        WHERE emp.deptno = dept.deptno;
```

El plan de ejecución sería el siguiente:

```
OPERATION                       OPTIONS         OBJECT_NAME
-----------------------------------------------------
SELECT STATEMENT
```

```
  NESTED LOOPS
        TABLE ACCESS            FULL            DEPT
        TABLE ACCESS            CLUSTER EMP
```

Donde existe un cluster entre emp y dept por depto.

Path 6. – Hash Cluster Key. Este acceso es cuando en la cláusula WHERE se colocan condiciones de igualdad unidas por AND para todos los campos de un hash cluster key.

Ejemplo:

ORDERS y LINE_ITEMS están almacenadas en un hash cluster con clave orderno.

```
SELECT *
        FROM line_items
        WHERE orderno = 65118968;
```

El plan de ejecución sería el siguiente:

```
OPERATION                       OPTIONS          OBJECT_NAME
-----------------------------------------------------
SELECT STATEMENT
        TABLE ACCESS            HASH            LINE_ITEMS
```

Path 7. – Index Cluster Key. Este acceso es cuando en la cláusula WHERE se colocan condiciones de igualdad unidas por AND para todos los campos del índice de la clave del cluster.

Ejemplo:

Cluster index entre EMP y DEPT, detno es la clave del cluster.

```
SELECT *  FROM emp
        WHERE deptno = 10;
```

El plan de ejecución sería el siguiente:

```
OPERATION                OPTIONS                         OBJECT_NAME
-----------------------------------------------------
SELECT STATEMENT
  TABLE ACCESS  CLUSTER                 EMP
        INDEX           UNIQUE SCAN             PERS_INDEX
```

PERS_INDEX es el nombre del cluster index.

Path 8. – Composite Index. Este acceso es cuando en la cláusula WHERE se colocan condiciones de igualdad unidas por AND para todos los campos del índice compuesto(no único).

Ejemplo:

```
SELECT *
        FROM emp
        WHERE job = 'CLERK'
          AND deptno = 30;
```

El plan de ejecución sería el siguiente:

```
OPERATION                OPTIONS                   OBJECT_NAME
-------------------------------------------------------
SELECT STATEMENT
  TABLE ACCESS  BY ROWID         EMP
        INDEX            RANGE SCAN      JOB_DEPTNO_INDEX
```

JOB_DEPTNO_INDEX es el nombre del índice compuesto con las columnas JOB y DEPTNO.

Path 9. - Single-Column Index. Este acceso es cuando en la cláusula WHERE se usan columnas de uno o más índices de una sola columna en igualdades unidas por ANDs.

Ejemplo:

```
SELECT *
        FROM emp
        WHERE job = 'ANALYST';
```

El plan de ejecución sería el siguiente:

```
OPERATION               OPTION          OBJECT_NAME
-----------------------------------------------------
SELECT STATEMENT
  TABLE ACCESS  BY ROWID         EMP
        INDEX            RANGE SCAN     JOB_INDEX
```

JOB_INDEX es un índice en EMP por JOB.

Oracle puede mezclar hasta 5 índices únicos.
Lo que hace es recorrerse los 5 índices y obtener aquellos rowids que están en todos los índices.

Ejemplo:

```
SELECT *
        FROM emp
        WHERE job = 'ANALYST'
          AND deptno = 20;
```

El plan de ejecución sería el siguiente:

```
OPERATION               OPTIONS                          OBJECT_NAME
-----------------------------------------------------
SELECT STATEMENT
  TABLE ACCESS  BY ROWID                EMP
        AND-EQUAL
          INDEX          RANGE SCAN             JOB_INDEX
          INDEX          RANGE SCAN              DEPTNO_INDEX
```

La operación AND-EQUAL obtiene los ROWIDs comunes entre los índices JOB_INDEX y DEPTNO_INDEX obteniendo los ROWIDs que satisfacen la consulta.

Path 10. - Bounded Range Search on Index Columns. Este acceso es cuando en la cláusula WHERE se encuentran condiciones que usan parcialmente la columna de un índice simple o una o más columnas que forman parte de un índice compuesto.

Condiciones ejemplo:

Columna = expr (para el caso de un índice compuesto)
Columna >[=] expr AND columna <[=] expr
Columna BETWEEN expr AND expr
Columna LIKE 'const%'

Ejemplo:

```
SELECT *
        FROM emp
        WHERE sal BETWEEN 2000 AND 3000;
```

El plan de ejecución sería el siguiente:

```
OPERATION               OPTIONS  OBJECT_NAME
-----------------------------------------------------
SELECT STATEMENT
  TABLE ACCESS  BY ROWID        EMP
        INDEX            RANGE SCAN     SAL_INDEX
```

SAL_INDEX es el nombre del índice en EMP por SAL.

Ejemplo:

```
SELECT *
        FROM emp
        WHERE ename LIKE 'S%';
```

Donde existe un índice por ename.

Path 11. – Unbounded Range Search on Index Columns. Este acceso es cuando en la cláusula WHERE se encuentran condiciones que usan parcialmente la columna de un índice simple o una o más columnas que forman parte de un índice compuesto.

Condiciones ejemplo:

```
WHERE columna >[=] exp

WHERE columna <[=] exp
```

Ejemplo :

```
SELECT *
        FROM emp
        WHERE sal > 2000;
```

El plan de ejecución sería el siguiente:

```
OPERATION               OPTIONS                 OBJECT_NAME
-----------------------------------------------------
SELECT STATEMENT
  TABLE ACCESS  BY ROWID        EMP
        INDEX           RANGE SCAN      SAL_INDEX
```

Donde SAL_INDEX en un índice por SAL en EMP.

Ejemplo:

```
SELECT *
        FROM line_items
        WHERE order > 65118968;
```

Donde existe un índice por ORDER ,LINE en line_items.

Anti-ejemplo:

```
SELECT *
        FROM line_items
        WHERE line < 4;
```

No se accede de esta forma pues line no esta dentro de la primera parte del índice.

Path 12. – Sort-Merge Join. Este acceso esta disponible cuando la sentencia es un join que no está en cluster y en la cláusula WHERE se encuentran condiciones que usan las columnas con igualdades.

Ejemplo:

```
SELECT *
        FROM emp, dept
        WHERE emp.deptno = dept.deptno;
```

El plan de ejecución sería el siguiente:

```
OPERATION                      OPTIONS                OBJECT_NAME
-----------------------------------------------------
SELECT STATEMENT
  MERGE JOIN
        SORT                   JOIN
          TABLE ACCESS         FULL            EMP
        SORT                   JOIN
          TABLE ACCESS         FULL            DEPT
```

Path 13. – MAX o MIN of Indexed Column. Este acceso esta disponible con sentencia en las cuales todas las condiciones siguientes son verdaderas.

- La consulta usa la función MAX o MIN para seleccionar el máximo o el mínimo valor de una columna de un índice simple o la primera columna de un índice compuesto. El argumento de la columna puede tener sumado o concatenado una constante.
- No hay expresiones en la lista de campos seleccionados.
- La sentencia no tiene WHERE o ORDER BY

Oracle hace una búsqueda en el índice por el máximo o el mínimo valor.

Ejemplo:

```
SELECT MAX(sal) FROM emp;
```

El plan de ejecución sería el siguiente:

```
OPERATION                      OPTIONS          OBJECT_NAME
-----------------------------------------------------
SELECT STATEMENT
  AGGREGATE                    GROUP BY
        INDEX                  RANGE SCAN       SAL_INDEX
```

Donde SAL_INDEX es un índice en EMP por SAL

Path 14. - ORDER BY on Indexed Column. Este acceso esta disponible con sentencia en las cuales todas las condiciones siguientes son verdaderas.

- La consulta contiene la cláusula ORDER BY que usa columnas de un índice simple o la primera parte de un índice compuesto. El índice no puede ser un cluster index.
- Deben existir restricciones de integridad que garanticen que no existen valores nulos en los campos que están en el order by(Esto es porque en el índice no se colocan los registros con campos nulos).
- El parámetro NLS_SORT debe estar seteado a BINARY.

Para resolver la consulta oracle recorre el índice y con los rowids accede a la tabla.

Ejemplo:

```
SELECT *
        FROM emp
        ORDER BY empno;
```

El plan de ejecución es el siguiente:

```
OPERATION              OPTIONS                 OBJECT_NAME
-----------------------------------------------------
SELECT STATEMENT
  TABLE ACCESS  BY ROWID        EMP
        INDEX           RANGE SCAN      PK_EMP
```

PK_EMP es la clave primaria de EMP por empno lo que garantiza que empno no tiene nulos.

Path 15. -Full Table Scan. Este acceso esta disponible para cualquier sentencia SQL que no cumple con ninguna de las demás condiciones.

Ejemplo:

```
SELECT *
        FROM emp;
```

El plan de ejecución sería el siguiente:

```
OPERATION              OPTIONS          OBJECT_NAME
-----------------------------------------------------
SELECT STATEMENT
```

```
  TABLE ACCESS  FULL            EMP
```

Estas condiciones no tienen índices disponibles cuando column1 y column2 están en la misma tabla:

```
column1 > column2

column1 < column2

column1 >= column2

column1 <= column2
```

Aunque la columna column esté indexada no se usa ningún índice.

```
column IS NULL

column IS NOT NULL

column NOT IN

column != expr

column LIKE '%pattern'
```

Cuando expr contiene una columna tampoco se utiliza ningún índice.

```
expr = expr2
```

```
NOT EXISTS subquery
```

Una condición que tiene una columna no indexada.

Cualquier sentencia SQL que contenga solamente este tipo de constructores, accederá en forma FULL a la tabla.

3. OPTIMIZACIONES REALIZADAS POR ORACLE

Evaluación de expresiones y condiciones: Primero se evalúan las expresiones y condiciones constantes que son posibles evaluar.

Transformación de sentencias: Se transforman las sentencias, como subconsultas correlativas en simples joins menos complejos.

Mezcla de vistas: Para las sentencias que acceden a vistas, Oracle transforma las vistas en la correspondiente consulta.

Elige el tipo de optimización*:* Escoge entre reglas o costos.

Elige la forma de acceder a los datos: Para cada tabla que accede la consulta escoge todos los diferentes métodos que tiene para acceder a los datos.

Elige el orden del join*:* Cuando hay mas de dos tablas en el join escoge cuales dos serán juntadas primero y cuales luego.

Escoge la operación de join: Para cada join escoge la operación a realizar para resolver el join.

3.1 Evaluación de expresiones y condiciones

El optimizador evalúa lo máximo posible las expresiones y condiciones constantes. Además traslada ciertas estructuras sintácticas en construcciones equivalentes.

Ejemplo :

Si tenemos
```
      sal > 24000/12
```
lo transforma en:
```
      sal > 2000
```
Ténganse en cuenta que no hace lo mismo con
```
      sal*12 > 24000
```
debido a que del lado izquierdo de la desigualdad tiene un campo.
En este caso el programador puede resolver el problema de la desigualdad y escribir la sentencia de forma que oracle la pueda resolver más eficientemente.
En particular si tenemos un índice por sal no será usado en este caso.

LIKE: si uno realiza consultas con like pero sin wilcard, es lo mismo que realizar la igualdad.

Oracle sustituye
```
ename LIKE 'SMITH'
```
por:
```
ename = 'SMITH'
```

Esto lo hace para tipos de largo variable, que no es así con el caso del tipo CHAR(10) que es semánticamente diferente.

IN: Cuando se usa in con constantes oracle transforma los in en ors de la siguiente forma:

```
ename IN ('SMITH', 'KING', 'JONES')
lo sustituye por:
ename = 'SMITH' OR ename = 'KING' OR ename = 'JONES'
```

ANY or SOME: Cuando son usados con constantes los transforma en ors como a continuación:

```
sal > ANY (:first_sal, :second_sal)
lo transforma en:
sal > :first_sal OR sal > :second_sal
```

También transforma ANY or SOME seguidos de subconsulta en EXISTS como sigue:

```
x > ANY (SELECT sal
            FROM emp
            WHERE job = 'ANALYST')
Lo transforma en:
EXISTS (SELECT sal
            FROM emp
            WHERE job = 'ANALYST'
              AND x > sal)
```

ALL: Es transformado con la sentencia and como sigue:

```
sal > ALL (:first_sal, :second_sal)
lo trasforma en:
sal > :first_sal AND sal > :second_sal
```

Cuando ALL es seguido de una subconsulta es transformado en not any como sigue

```
x > ALL (SELECT sal
            FROM emp
            WHERE deptno = 10)
Se trasforma en:
NOT (x <= ANY (SELECT sal
                  FROM emp
                  WHERE deptno = 10) )
Y luego es transformado en
NOT EXISTS (SELECT sal
                  FROM emp
                  WHERE deptno = 10
                    AND x <= sal)
```

BETWEEN: Es transformado en >= and <= como sigue:

```
sal BETWEEN 2000 AND 3000
se transforma en:
sal >= 2000 AND sal <= 3000
```

NOT: Es eliminado si es seguido de un operador que tenga contrario:

```
NOT deptno = (SELECT deptno FROM emp WHERE ename = 'TAYLOR')
Lo transforma en:
deptno <> (SELECT deptno FROM emp WHERE ename = 'TAYLOR')

NOT (sal < 1000 OR comm IS NULL)
Lo transforma en:
NOT sal < 1000 AND comm IS NOT NULL
Y luego en:
sal >= 1000 AND comm IS NOT NULL
```

Transitividad: Oracle puede inferir condiciones transitivas y así poder usar mejor los índices

Ejemplo :

```
WHERE column1 comp_oper constant
  AND column1 = column2
Se puede inferir:
column2 comp_oper constant
donde :
```

Comp_oper	Es un comparador de los siguientes: =,!=, ^=, <, <>, <, >, <=, o >=.
constant	Es una constante.

```
Nota: el optimizador solo infiere transitivas de constantes y no de columnas de tablas:
WHERE column1 comp_oper column3
  AND column1 = column2
En este caso no infiere:
column2 comp_oper column3
```

3.2 Transformación de ors

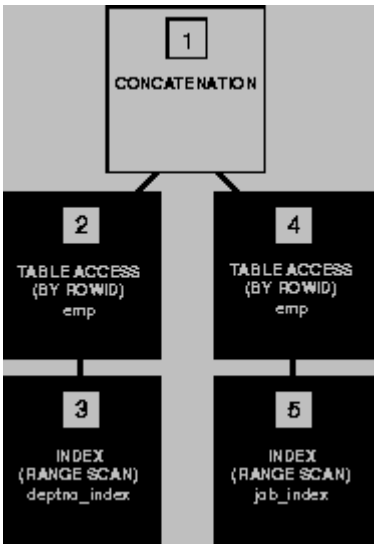
Si existe un índice para dos condiciones unidas por un or, oracle puede separar la consulta en dos consultas independientes unidas por un union all. Si no existe el índice es mejor una sola condición con el operador or.

```
SELECT *
      FROM emp
      WHERE job = 'CLERK'
        OR deptno = 10;
```

Si existe un índice por job y otro por deptno se transforma en:

```
SELECT *
      FROM emp
      WHERE job = 'CLERK'
UNION ALL
SELECT *
      FROM emp
      WHERE deptno = 10
        AND job <> 'CLERK';
```

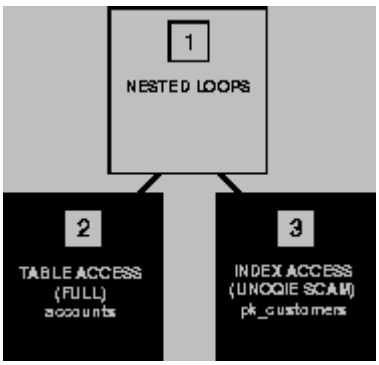
Plan de ejecución



3.3 Transformando sentencias complejas en joins:

```
SELECT *
      FROM accounts
      WHERE custno IN
            (SELECT custno FROM customers);
Lo transforma en:
SELECT accounts.*
      FROM accounts, customers
      WHERE accounts.custno = customers.custno;
```

Plan de ejecución




```
SELECT *
      FROM accounts
      WHERE accounts.balance >
            (SELECT AVG(balance) FROM accounts);
```

Este es un ejemplo de una subconsulta que no puede ser transformada en join.
Subconsultas con funciones de agregación no pueden ser trasformadas en joins.

3.4 Mezcla de vistas:

```
Ejemplo:
CREATE VIEW emp_10
AS SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno
            FROM emp
            WHERE deptno = 10;

SELECT empno
      FROM emp_10
      WHERE empno > 7800;
Se transforma en:
SELECT empno
      FROM emp
      WHERE deptno = 10
        AND empno > 7800;
```

Por lo tanto si existe un índice por deptno y empno puede ser usado, o si existe un índice por empno.

3.5 Algunas reglas de Optimización

- **Regla 1: Cuando la columna indexada está dentro de una expresión o en una función SQL, el índice NO es utilizado.**

Ejemplo:

```
SELECT *
FROM clientes
WHERE RTRIM(nom_cli) = 'Pedro'
```

En este caso el índice sobre 'nom_cli' no será utilizado; por lo que se accederá a toda la tabla Clientes (es decir que se recorrerá toda la tabla).
Del mismo modo:

```
SELECT *
FROM pedidos
WHERE TO_CHAR(fecha_ped, 'DD/MM/YYYY') = '15/02/1991'
```

Aunque exista un índice por el campo fecha_ped igual se accederá secuencialmente a toda la tabla Pedidos; esto es porque se aplicó la función To_char al atributo con índice.
Una forma de evitar la no utilización del índice es modificando la sentencia de la siguiente manera:

```
SELECT *
FROM pedidos
WHERE fecha_ped = TO_DATE('15/02/1991', 'DD/MM/YYYY')
```

Ejemplo:

```
SELECT NIS_RAD
FROM SUMCON
WHERE F_ALTA-1=SYSDATE;
```

Se debe colocar la operación al revés.

```
SELECT NIS_RAD
FROM SUMCON
WHERE F_ALTA=SYSDATE-1
```

- **Regla 2: El índice no será utilizado cuando la columna indexada es comparada con el valor nulo.**

Ejemplo:

```
SELECT *
FROM clientes
WHERE nro_cli IS NOT NULL
```

Para que se pueda utilizar el índice sobre 'num_cli' se debe modificar la sentencia de la siguiente manera:

```
SELECT *
FROM clientes
WHERE nro_cli > -1
```

Observación: esta modificación tendrá sentido si los números de cliente no son negativos.

- **Regla 3: No se utilizará el índice si la columna indexada es comparada por diferencia (!= o NOT)**

Ejemplo :

```
            SELECT *
            FROM clientes
            WHERE nro_cli != 1000
```

se debería de modificar a la siguiente sentencia:

```
            SELECT *
            FROM clientes
            WHERE nro_cli < 1000
            OR nro_cli > 1000
```

- **Regla 4: El índice es utilizado únicamente cuando el primer carácter de la izquierda es distinto de %**

.

Ejemplo :

```
            SELECT *
            FROM clientes
            WHERE nombre LIKE 'Suarez%'
```

Si escribiéramos:

```
            SELECT *
            FROM clientes
            WHERE nombre LIKE '%Suarez'
```

entonces acá se recorrería toda la tabla Clientes.

- **Regla 5: Oracle utiliza hasta cinco índices concurrentes. Si entre ellos hay un índice único, sólo éste será utilizado**

Ejemplo:

```
            SELECT *
            FROM clientes
            WHERE nom_cli LIKE 'Suarez'
            AND direc_cli LIKE 'Bvar%'
            AND país = 'Uruguay'
```

donde nom_cli y país tienen índice asociado.
De esta forma se accede por ROWID a los registros que satisfacen nom_cli like 'Suarez' y del mismo modo con el atributo país. Luego se realiza la intersección de los dos conjuntos y al final se realiza el acceso a la tabla por los ROWIDs de la intersección. Se recomienda no guardar activos nada más que los índices más selectivos, es decir los que reducen al mínimo las líneas; ya que por ejemplo podemos ver en este caso que existirán muchos menos clientes de apellido 'Suarez' que clientes que se encuentran en Uruguay.

- **Regla 6: En el caso del operador 'OR' el optimizador descompone la sentencia del WHERE en tantas partes como condiciones existan separadas por el operador 'OR'**

Basta con que una condición no tenga índice para que el optimizador no utilice ningún índice sobre ninguna de las condiciones.

Ejemplo:

```
SELECT *
FROM clientes
WHERE dir_cli LIKE '%Mercedes%'
OR nom_cli LIKE 'Suarez%'
```

donde existe índice sobre nom_cli.

Se generarán las siguientes partes:

```
SELECT *
FROM clientes
WHERE dir_cli LIKE '%Mercedes%'
```

y

```
SELECT *
FROM clientes
WHERE nom_cli LIKE 'Suarez%'
```

Entonces el optimizador tiene que comprobar la primera condición secuencialmente, por lo que ya no le interesa usar el índice de la segunda condición.

- **Regla 7-a: Si un atributo en común entre dos tablas no está indexado el optimizador ordena cada tabla con un recorrido secuencial y después las fusiona verificando la condición de combinación.**

El inconveniente de este proceso es que es muy largo.
Ejemplo :

```
SELECT *
FROM cliente, pedido
WHERE cliente.nro_cli = pedido.nro_cli
```

donde no existen índices en ninguna de las tablas.

- **Regla 7-b: Cuando uno de los dos campos de las tablas que se combinan está indexado, el optimizador elige como tabla directriz aquella que no disponga de índice. Esta tabla es la que será utilizada para recorrer secuencialmente.**

Ejemplo :

```
SELECT *
FROM clientes, pedido
WHERE cliente.nro_cli = pedido.nro_cli
```

Existe índice sobre el campo nro_cli de la tabla pedido, entonces Oracle recorre la tabla Clientes secuencialmente y con cada nro.de cliente accede por índice a la tabla Pedido.

Regla 7-c: Cuando existen índices sobre los dos campos de la combinación entonces el optimizador selecciona como tabla directriz aquella que aparece en último lugar en la cláusula del FROM

Es conveniente entonces poner la tabla que tenga menos registros al final del FROM.

Tomando como ejemplo la sentencia anterior entonces la tabla directriz será la tabla Pedido.
Esta recomendación es válida también para las sentencias de combinación referenciando

varias tablas.

Ejemplo :

```
SELECT *
FROM clientes C, pedido P, linea_pedido L
WHERE C.nro_cli = P.nro_cli
AND P.nro_ped = C.nro_ped
```

donde existen índices en todas las tablas.
El optimizador entra secuencialmente a la tabla Linea_pedido (tabla directriz) y por cada registro tiene acceso por índice a la tabla Pedido. Luego por cada pedido entra a la tabla Clientes mediante su índice.

Si se sabe que la tabla Cliente es mucho más pequeña que la tabla Pedido y ésta a su vez más chica que la tabla Linea_pedido entonces es mucho más eficiente tomar la tabla Cliente como tabla directriz (por lo que se deberá colocar en la última posición del FROM).

- **Regla 8: Las vistas con GROUP BY pueden empeorar el rendimiento de una consulta.**

Ejemplo :

```
CREATE VIEW V(num_art, cant_total)
AS SELECT num_art, SUM(cant_stock)
FROM stock
GROUP BY num_art
```

Supongamos que se utiliza la vista mediante la sentencia:

```
SELECT *
FROM V
WHERE num_art = 500
```

entonces esta sentencia es evaluada de la siguiente manera:

```
SELECT num_art, SUM(cant_stock)
FROM stock
GROUP BY num_art
HAVING num_art = 500
```

Oracle calcula el total de stock para todos los artículos, luego selecciona por HAVING el total del artículo 500. Entonces se está haciendo un cálculo inútil para todos los demás artículos;
por lo que lo más eficiente sería:

```
SELECT num_art, SUM(cant_stock)
FROM stock
WHERE num_art =500
GROUP BY num_art
```

Podemos decir que es mejor restringir con la cláusula WHERE que con HAVING.

- **Regla 9: Al utilizar en una consulta la cláusula DISTINCT, Oracle siempre tendrá que ordenar los datos de salida.**

```
SELECT DISTINCT num_art
```

```
FROM stock;
```

Oracle tendrá que ordenar la salida para eliminar los valores repetidos, se debe de tratar de evitar esta cláusula.

3.6 Otras sugerencias

- Evitar que ingrese un índice que no es bueno por medio de una operación.

```
Select NIS_RAD, SEC_NIS, F_FACT, SEC_REC
FROM EST_REC
WHERE NIS_RAD=1000001
AND EST_REC='ER020';
```

En este caso se nos introduce el índice por estado del recibo y no busca en casi todos los recibos.
Si hacemos el siguiente cambio no se introduce.

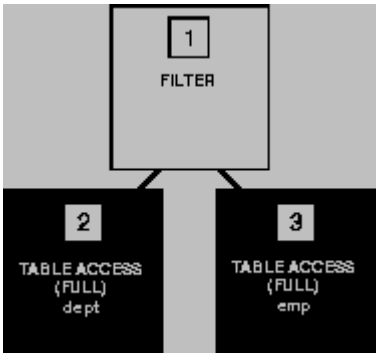
```
Select NIS_RAD, SEC_NIS, F_FACT, SEC_REC
FROM EST_REC
WHERE NIS_RAD=1000001
  AND EST_REC||''='ER020';
```

Para los valores numéricos podemos sumarle 0.

- Eliminar los NOT IN por EXISTS o OUTER JOINS.

```
Ejemplo :
SELECT dname, deptno
   FROM dept
   WHERE deptno NOT IN
      (SELECT deptno FROM emp);
```

El plan de ejecución sería el siguiente:

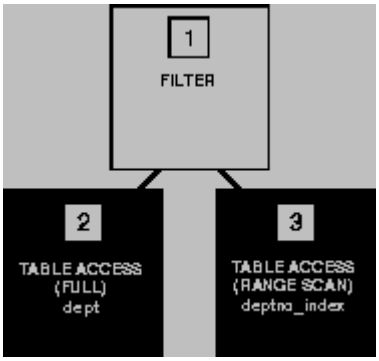


Podríamos cambiarlo por:

```
SELECT dname, deptno
FROM dept
```

```
WHERE NOT EXISTS
      (SELECT deptno
       FROM emp
       WHERE dept.deptno = emp.deptno);
```

El plan de ejecución sería el siguiente:



Nótese que en la segunda consulta se usa un índice por número de departamento.
Con OUTER JOINS la consulta quedaría:

```
SELECT dname, deptno
FROM dept, emp
WHERE dept.deptno = emp.deptno (+)
AND emp.deptno;
```

y tendría el mismo plan de ejecución que la anterior.

- Agregar la columnas que son consultadas como constantes en la cláusula ORDER BY y forman parte de un índice.

Ejemplo:

```
Select NIS_RAD, SEC_NIS, F_FACT
FROM ITIFACT
WHERE IND_FACT=2 AND IND_EMBALSADO=2
ORDER BY NIS_RAD, SEC_NIS, F_FACT
```

Si tenemos un índice por IND_FACT,IND_EMBALSADO, NIS_RAD, SEC_NIS, F_FACT, agregar al ORDER BY los siguientes valores:

```
Select NIS_RAD, SEC_NIS, F_FACT
FROM ITIFACT
WHERE IND_FACT=2 AND IND_EMBALSADO=2
ORDER BY IND_FACT, IND_EMBALSADO, NIS_RAD, SEC_NIS, F_FACT
```

3.7 Índices

3.7.1 Cuando crear Índices

Los índices mejoran la perfomance de las consultas que seleccionan un pequeño porcentaje de los registros de una tabla.

3.7.2 Elección de columnas a indexar

Considerar

- Columnas que son usadas frecuentemente en sentencias WHERE.
- Columnas por las cuales se hacen joins de tablas frecuentemente.
- Solo indexar columnas con buena *selectividad*. La selectividad de un índice es el porcentaje de filas en una tabla que tienen el mismo valor para la columna indexada. La selectividad de un índice es buena si pocas filas tienen el mismo valor.
  Nota: Oracle crea automáticamente índices por las columnas que forman parte de las claves primarias y únicas que se definen como restricciones de integridad. Estos índices son los más efectivos en optimizar la performance.
- No indexar columnas con pocos valores diferentes. Generalmente estas consultas tienen poca selectividad y por lo tanto no optimizan la performance a no ser que las valores usados más frecuentemente sean los que aparecen menos frecuentemente en la columna.
- No indexar columnas que son modificadas frecuentemente. Las sentencias UPDATE, INSERT y DELETE que modifican índices demoran más que si no lo hicieran, ya que deben modificar los datos en la tabla y en el índice.
- No indexar columnas que solamente aparecen en sentencias WHERE con operadores o funciones (que no sean MIN y MAX).
- Indexar claves foráneas en casos en que un gran número de INSERT, UPDATE y DELETE concurrentes acceden las tablas padre e hijo. Estos índices permiten que Oracle modifique los datos en la tabla hijo sin lockear la tabla padre.

Cuando esté evaluando si indexar una columna, considere si lo que se va a ganar en performance de una consulta es mayor que lo que se va a perder en performance al realizar INSERT, UPDATE y DELETE sobre la tabla y además considerar el espacio que se va a perder para almacenar el índice.

3.7.3 Elección de índices compuestos

Un índice compuesto es un índice que está formado por más de una columna. Este tipo de índices puede brindar ventajas adicionales sobre los índices formados por una sola columna.

Mejor selectividad: Algunas veces dos o más columnas, cada una de ellas con poca selectividad, pueden ser combinadas en un índice compuesto con buena selectividad.

Almacenamiento adicional de los datos: Si todas las columnas seleccionadas por una consulta forman parte de un índice compuesto, Oracle puede devolver los valores de estas columnas sin tener la necesidad de acceder a la tabla.

Una sentencia SQL puede utilizar un índice compuesto si la condición de la sentencia utiliza una parte del comienzo del índice (*leading portion*). Se considera una parte del comienzo del índice un conjunto de una o más columnas que fueron especificadas al

principio y consecutivamente en la lista de columnas de la sentencia CREATE INDEX.

Ej:
```
CREATE INDEX comp_ind
      ON tab1(x, y, z);
```

Estas combinaciones de columnas se consideran parte del comienzo del índice: X, XY, XYZ. Las restantes combinaciones no se consideran parte del comienzo del índice.

Considere las siguientes alternativas al elegir columnas para índices compuestos:

- Columnas que son frecuentemente usadas juntas en condiciones WHERE combinadas con operadores AND, especialmente si su selectividad combinada es mejor que la selectividad de cualquiera de las columnas en forma separada.
- Si varias consultas seleccionan el mismo conjunto de columnas basados en uno o más valores, crear un índice compuesto conteniendo todas esas consultas.

Considere las siguientes alternativas para ordenar las columnas en los índices compuestos:

- Crear el índice de forma que las columnas utilizadas en la condición WHERE formen parte del comienzo del índice.
- Si algunas de las columnas se utilizan más frecuentemente en sentencias WHERE, asegúrese de crear el índice de forma que esas columnas más frecuentemente usadas formen parte del comienzo del índice, de manera de permitir que las sentencias que solamente utilizan esas columnas, hagan uso del índice.
- Si todas las columnas se usan en las sentencias WHERE con igual frecuencia, se mejora la performance de la consulta ordenando las columnas en la sentencia CREATE INDEX desde la más selectiva a la menos selectiva.
- Si todas las columnas se usan con la misma frecuencia en la sentencia WHERE pero los datos están físicamente ordenados en una de esas columnas, se debe ubicar esa columna en primer lugar del índice compuesto.

3.7.4 Escribir sentencias que utilicen los índices disponibles

Luego de crear un índice, el optimizador no puede utilizar ese índice simplemente porque este índice existe. El optimizador va a usar ese camino de acceso para resolver una sentencia SQL si esta contiene una construcción que hace disponible ese camino de acceso.

Si se está utilizando la opción de resolución *basada en costo* para resolver las consultas, se deben generar estadísticas para ese índice.

3.7.5 Escribir sentencias que NO utilicen los índices disponibles

En algunas circunstancias se puede querer evitar que una sentencia SQL utilice un índice existente.
Se puede querer hacer esto si se sabe que el índice disponible no es muy selectivo y que una búsqueda en la tabla completa puede ser más eficiente. Se puede forzar que el optimizador utilice una búsqueda completa de la tabla a través de alguno de estos métodos:

- Se puede hacer que no utilice el índice modificando la forma de la sentencia de manera que no cambie su significado, por ej: Concatenando el string nulo cuando se trabaja con caracteres o sumando el 0 cuando se trabaja con números.

- Se puede utilizar la opción FULL para forzar a que el optimizador resuelva la consulta a través de una búsqueda completa en lugar de utilizar el índice.

- Se puede utilizar la opción INDEX o AND_EQUAL para forzar al optimizador a que use un índice o conjunto de índices en lugar de utilizar otro.

3.8 Utilización de "Sugerencias" (HINTS)

Como desarrollador de la aplicación se tiene información acerca de los datos que se manejan, que el optimizador desconoce. Basados en esta información, se puede sugerir una mejor manera de resolver una consulta de lo que el optimizador lo puede hacer. En estos casos se pueden utilizar "sugerencias" para forzar al optimizador para utilizar un determinado plan de ejecución.
Se pueden utilizar sugerencias para especificar:
- El mecanismo de optimización para una sentencia SQL
- El modo del mecanismo basado en costo del optimizador para una sentencia SQL
- El camino de acceso a una tabla especificada en una sentencia.

Las sugerencias solamente se aplican en la sentencia en la cual aparecen y no tienen efecto para las consultas posteriores.
Las sugerencias para una sentencia SQL se envían al optimizador con el formato de comentario. Una sentencia solamente puede tener un comentario con sugerencia. Este comentario solamente puede ir a continuación de las palabras SELECT, UPDATE o DELETE.

3.8.1 Sugerencias para formas de optimización

- **ALL_ROWS:** Selecciona la forma de resolución basada en costo para optimizar la sentencia con un objetivo de *best throughput* (consumo total mínimo de los recursos).

```
SELECT /*+ ALL_ROWS */ empno, ename, sal, job
      FROM emp
      WHERE empno = 7566;
```

- **FIRST_ROWS:** Selecciona la forma de resolución basada en costo para optimizar la sentencia con un objetivo de *mejor tiempo de respuesta* (consumo mínimo de los recursos para devolver el primer registro).

Esta sugerencia hace el que optimizador evalúe las siguientes alternativas:

- Si hay un índice disponible, el optimizador lo va a utilizar antes que una búsqueda por la tabla completa.

- Si una búsqueda por índice esta disponible a través de una cláusula ORDER BY, el optimizador la va a utilizar para evitar una operación de sort.

- Si una búsqueda por índice está disponible, el optimizador va a elegir un join de

loop anidado en vez de un join sort-merge siempre que la tabla asociada sea la potencial tabla interior del loop anidado.

El optimizador ignora **esta** sugerencia en sentencias DELETE y UPDATE y en las sentencias SELECT que contengan alguna de las siguientes sintaxis:

- Operadores de conjunto (UNION, INTERSECT, MINUS, UNION, ALL)
- Cláusulas GROUP BY
- Cláusulas FOR UPDATE
- Funciones de grupo
- Operador DISTINCT

Se debe utilizar el comando ANALYZE para generar estadísticas para las tablas accedidas por sentencias que utilizan la optimización basada en costo.

- **CHOOSE:** Esta sugerencia hace que el optimizador elija entre el modo de resolución basado en costo o el modo de resolución basado en reglas. Esta elección se va a hacer de acuerdo a la existencia de estadísticas sobre las tablas accedidas por la sentencia. Si el diccionario de datos contiene estadísticas de al menos una de las tablas, el optimizador va a utilizar el modo de resolución basado en mejor costo, con el objetivo de best throughput, de lo contrario, si no existen estadísticas para ninguna de las tablas de la consulta, se utiliza el modo de resolución basado en reglas.

- **RULE:** Esta sugerencia selecciona el método de resolución de la consulta basada en reglas.

Nota: la sugerencia RULE junto con el método de resolución basado en reglas, no va a estar disponible en futuras versiones de Oracle.

3.8.2 Sugerencias de métodos de acceso

Al usar estas sugerencias se debe especificar la tabla a ser accedida exactamente como aparece en la sentencia. Si la sentencia utiliza un alias para la tabla, en la sugerencia se debe usar el alias en lugar del nombre de la tabla. El nombre de la tabla dentro de la sugerencia no debe incluir el nombre del esquema aunque el nombre del esquema si aparezca en la sentencia.

- **FULL:** Selecciona una búsqueda completa en la tabla.

La sintaxis es:
FULL(*tabla*)

Donde *tabla* es el nombre o el alias de la tabla donde se debe realizar la búsqueda completa.

```
SELECT /*+ FULL(a) No usar el indice en ACCNO */ accno, bal
      FROM accounts a
      WHERE accno = 7086854;
```

Esta sentencia va a hacer una búsqueda completa en la tabla *accounts* aunque exista un índice sobre el campo *accno*.

- **ROWID:** Hace una búsqueda en la tabla por ROWID para la tabla especificada.

La sintaxis es:
ROWID(*tabla*)

- **INDEX:** Selecciona un índice para la tabla especificada en la sentencia.

La sintaxis es:
INDEX (*tabla indice*)

Donde *tabla* es el nombre o alias de la tabla asociada con el índice sobre el cual se debe hacer la búsqueda, e *índice* es el índice sobre el cual se debe hacer la búsqueda. La sugerencia puede especificar uno o más índices:

- Si la sugerencia especifica un solo índice, el optimizador realiza la búsqueda sobre ese índice. El optimizador no va a considerar una búsqueda en la tabla completa o sobre otro índice que esté definido en la tabla.

- Si la sugerencia especifica una lista de índices, el optimizador va a considerar el costo de búsqueda utilizando cada uno de los índices de la lista y luego va a hacer la búsqueda sobre el de menor costo. También puede elegir hacer la búsqueda sobre múltiples índices y luego juntar los resultados si esta forma de acceso es la de menor costo. El optimizador no va a considerar una búsqueda en la tabla completa o sobre otro índice que no aparezca en la lista de índices.

- Si la sugerencia no especifica índices, el optimizador considera el costo de la búsqueda en cada uno de los índices disponibles para la tabla y luego va a hacer la búsqueda sobre el de menor costo. También puede elegir hacer la búsqueda sobre múltiples índices y luego juntar los resultados si esta forma de acceso es la de menor costo. El optimizador no va a considerar una búsqueda en la tabla completa.

- **INDEX_ASC:** Selecciona una búsqueda por índice para la tabla especificada. Si la sentencia usa una búsqueda por índice, Oracle va a realizar la búsqueda en orden ascendente de los valores del índice.

La sintaxis es:
INDEX_ASC (*tabla índice*)

El modo por defecto de Oracle cuando hace la búsqueda por índices, es recorrer las claves en forma ascendente, por lo que INDEX_ASC actualmente (no se sabe en versiones futuras de Oracle) tiene el mismo comportamiento que la sugerencia INDEX.

- **INDEX_DESC:** Selecciona una búsqueda por índice para la tabla especificada. Si la sentencia usa una búsqueda por índice, Oracle va a realizar la búsqueda en orden descendente de los valores del índice.

La sintaxis es:
INDEX_DESC (*tabla índice*)

Esta sugerencia no tiene efecto en las sentencias que acceden más de una tabla.

- **AND_EQUAL:** Selecciona un plan de ejecución que usa un camino de acceso que hace un merge de las búsquedas en varios índices de una sola columna.

La sintaxis es:
AND_EQUAL (*tabla índice índice...*)

Donde *tabla* especifica el nombre o alias de la tabla asociada con los índices e *índice* especifica un índice donde se debe realizar la búsqueda. Se deben especificar al menos dos índices. No se pueden especificar más de 5 índices.

- **USE_CONCAT:** Esta sugerencia fuerza que condiciones OR combinadas en cláusulas WHERE de una sentencia sean transformadas en una consulta compuesta utilizando el operador de conjuntos UNION ALL. Generalmente esta transformación ocurre solamente si el costo de la consulta con las concatenaciones es mas barato que el costo sin ellas.

3.8.3 Sugerencias para ordenes de Joins

- **ORDERED:** Esta sugerencia hace que Oracle haga el join de las tablas en el orden en el cual aparecen en la cláusula FROM.

Ej:

```
SELECT /*+J ORDERED */ tab1.col1, tab2.col2, tab3.col3
      FROM tab1, tab2, tab3
      WHERE tab1.col1 = tab2.col1
      AND tab2.col1 = tab3.col1;
```

En este ejemplo primero se hace el join entre TAB1 y TAB2 y luego se hace el join del resultado con TAB3.

Si se omite la palabra ORDERED en una sentencia que hace un join, el optimizador selecciona el orden en el cual hacer el join de las tablas.

Se debe utilizar esta sugerencia cuando se conoce algo acerca del número de filas que van a ser devueltas en la operación. Esta información va a permitir seleccionar mejor el orden de las tablas sobre las cuales se va a ejecutar el join, de lo que lo haría el optimizador.

3.8.4 Sugerencias para operaciones de Join

- **USE_NL:** Esta sugerencia hace que Oracle haga el join de cada tabla a otra fila con un loop anidado usando la tabla especificada como la tabla interior del loop.

La sintaxis es:
USE_NL (*tabla*...)

4. HERRAMIENTAS DE DIAGNÓSTICO DE PERFORMANCE

Algunas de las herramientas que tiene Oracle para monitorear y optimizar aplicaciones y sentencias SQL son:

- La herramienta TRACE
- El comando EXPLAIN PLAN

4.1 SQL Trace Facility

Esta herramienta brinda información de la performance de sentencias SQL individuales. Esta herramienta genera las siguientes estadísticas para cada sentencia:
- El número de veces que se hace el parse, ejecución y fetch de cada sentencia SQL.
- El tiempo necesario para ejecutar cada sentencia SQL
- Los accesos a disco y memoria asociados con el procesamiento de cada sentencia SQL
- El número de líneas que procesa cada sentencia SQL.

Esta facilidad se puede habilitar solamente para una sesión o para una instancia. Cuando esta función está habilitada, se recoge información estadística en un archivo de trace.

Se puede utilizar el programa TKPROF para dar formato al contenido del archivo de trace y enviarlo a un archivo de salida en formato leíble.

Como opciones, TKPROF también puede:
- Determinar planes de ejecución para las sentencias SQL
- Crear un script SQL que almacene las estadísticas en la base de datos.

Al ejecutar esta aplicación se aumenta el trabajo del sistema, por lo que solamente se debe utilizar mientras se esté haciendo el tuning de las sentencias SQL y luego inhabilitarlo.

USANDO LA HERRAMIENTA DE TRACE

Estos son los pasos que hay que seguir para habilitar la herramienta de trace:

1. Fijar los parámetros de inicialización para preparar a Oracle para ejecutar la herramienta de trace.
2. Habilitar la herramienta de trace para la sesión y ejecutar la aplicación. Este paso genera un archivo con las estadísticas para las sentencias ejecutadas por la aplicación.
3. Ejecutar TKPROF para traducir el archivo creado en el paso 2 en un archivo de salida leíble. Este paso también de forma opcional puede generar un script que almacene las estadísticas en la base de datos.
4. Interpretar el archivo generado en el paso 3.
5. Opcionalmente, ejecutar el script producido en el paso 3 para almacenar las estadísticas en la base de datos.

A continuación detallamos cada uno de los pasos.

- **Paso 1. Parámetros de inicialización para la herramienta de trace**

Antes de ejecutar la aplicación con la opción de trace habilitada, se deben configurar adecuadamente los siguientes parámetros de inicialización.

  - TIMED_STATISTICS: Habilita e inhabilita la recolección de estadísticas como ser tiempos de CPU y de ejecución, así como también la recolección de otras estadísticas. El valor por omisión es FALSE.
  - MAX_DUMP_FILE_SIZE: Este parámetro determina el máximo tamaño del archivo de trace en cantidad de bloques del sistema operativo. El valor por defecto es 500.
  - USER_DUMP_DEST: Este parámetro indica el destino del archivo de trace. El valor por defecto es el directorio donde se guardan los archivos de dump del sistema operativo.

- **Paso 2. Habilitar la herramienta de trace**

La herramienta de trace se puede habilitar para:
- Una sola sesión
- La instancia

*** Para habilitar la herramienta de trace para una sesión**, se debe dar el siguiente comando SQL:

```
ALTER SESSION
      SET SQL_TRACE = TRUE;
```

*** Para inhabilitar la herramienta de trace para una sesión**, se debe dar el siguiente comando SQL:

```
ALTER SESSION
      SET SQL_TRACE = FALSE;
```

La herramienta de trace también se inhabilita automáticamente de la sesión cuando la aplicación se desconecta de Oracle.

*** Para habilitar la herramienta de trace para una instancia**, se debe cargar el valor del parámetro de inicialización SQL_TRACE con el valor TRUE. Este valor hace que se recojan estadísticas para todas las sesiones.

Una vez que se ha habilitado para una instancia, se puede inhabilitar para una sesión con el comando:

```
ALTER SESSION
      SET SQL_TRACE = FALSE;
```

Una vez que la aplicación de trace ha generado el/los archivos de trace, se puede:

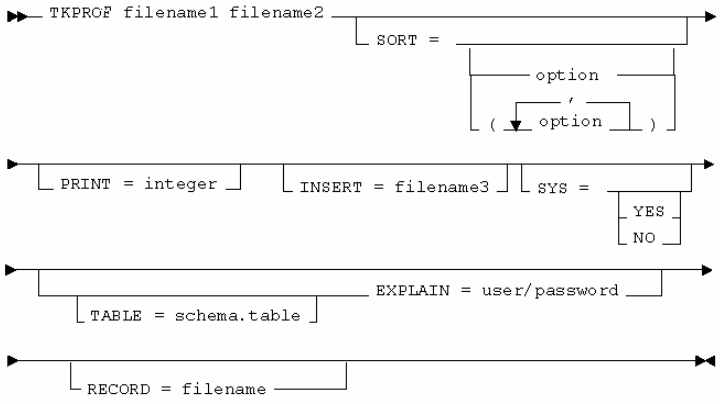
- Ejecutar TKPROF en cada archivo en forma independiente produciendo archivos formateados y leíbles, uno por cada sesión.

- Juntar todos los archivos de trace en un solo archivo y ejecutar TKPROF en el archivo resultante.

- **Paso 3. Ejecutando TKPROF**

TKPROF acepta como entrada un archivo generado por la herramienta de trace de SQL y produce un archivo con formato.

TKPROF también se puede utilizar para generar planes de ejecución. Esta herramienta se invoca usando la siguiente sintaxis:



- Nota:

Cuando se trabaja con el release 7.3.2.2.0 del TKPROF, se debe editar el archivo de trace (.trc) y borrarle la línea que dice APPNAME ..., de lo contrario, al ejecutar el comando TKPROF, se recibe un error de ejecución.

- filename1

  Archivo de entrada, es el archivo de trace que contiene las estadísticas generadas por la herramienta de SQL. Este es un archivo cuyo nombre tiene extensión .trc

- filename2

  Archivo de salida. Especifica el archivo en el cual TKPROF va a generar la salida formateada. Este es un archivo cuyo nombre tiene extensión .prf

- EXPLAIN

  Determina el plan de ejecución para cada sentencia SQL que se encuentra en el archivo de trace y escribe este plan de ejecución en el archivo de salida. TKPROF determina el plan de ejecución dando el comando EXPLAIN PLAN después de conectarse al Oracle con el usuario (*user*) y la contraseña (*password*) especificados en este parámetro. El usuario especificado debe tener permiso de CREATE SESSION

- TABLE

  Especifica el esquema (schema) y el nombre de la tabla (table) en la cual TKPROF va a almacenar temporalmente los planes de ejecución antes de escribirlos en el archivo de salida.
  Si la tabla ya existe, TKPROF borra su contenido, la usa para el almacenamiento temporario del resultado del comando EXPLAIN PLAN y después borra los registros. Si esta tabla no existe, TKPROF la crea, la usa y luego le hace un drop. El usuario especificado debe ser capaz de ejecutar sentencias de INSERT, SELECT y DELETE sobre la tabla. Si la tabla no existe, el usuario también debe ser capaz

de ejecutar comandos CREATE TABLE Y DROP TABLE. Si no se especifica un nombre de tabla, TKPROF utiliza la tabla PROF$PLAN_TABLE.

- INSERT
  Crea un script SQL que almacena las estadísticas del archivo de trace en la base de datos. TKPROF crea este script con el nombre *filename3*. Este script crea una tabla e inserta en la tabla una fila con estadísticas para cada una de las sentencias de SQL.

- SYS
  Habilita e inhabilita el listado de sentencias SQL ejecutadas por el usuario SYS, o sentencias SQL recursivas en el archivo de salida. El valor por omisión es YES.

- SORT
  Ordena las sentencias SQL analizadas en orden descendiente de la opción de orden especificada antes de listarlas en el archivo de salida.
  Si se especifica más de una opción, la salida es ordenada en orden descendiente de la suma de los valores especificados en la opción de orden.
  Las opciones de orden son las siguientes:

```
PRSCNT      PRSCPU      PRSELA      PRSDSK      PRSQRY      PRSCU
PRSMIS      EXECNT      EXECPU      EXEELA      EXEDSK      EXEQRY
EXECU EXEROW      EXEMIS      FCHCNT      FCHELA      FCHDSK
FCHQRY      FCHCU FCHROW
```

  La funcionalidad de cada una de estas opciones aparece listada en una hoja aparte.

- PRINT
  Lista solamente las primeras *integer* sentencias SQL en el archivo de salida. Si se omite este parámetro TKPROF lista todas las sentencias analizadas.

- RECORD
  Crea un script SQL con el nombre *filename* con todas las sentencias SQL no recursivas que se encuentran en el archivo de trace. Esto puede ser utilizado para recrear los eventos del usuario a partir del archivo de trace.

ESTADISTICAS DEL TRACE

TKPROF lista las estadísticas para una sentencia SQL analizada por la herramienta de trace en forma de filas y columnas. Cada fila corresponde a uno de los tres pasos en que se divide el procesamiento de una sentencia SQL.

Pasos en que se divide:

- Parse
  Este paso traduce la sentencia SQL en un plan de ejecución. Este paso incluye el control de las autorizaciones de seguridad, existencia de tablas, columnas y demás objetos a los que se hace referencia.

- Ejecución
  Este paso es la ejecución propiamente dicha de la sentencia por parte de Oracle. Para las sentencias de INSERT, UPDATE y DELETE, este paso modifica los datos. Para las sentencias SELECT, este paso identifica las filas seleccionadas.

- Fetch
  Este paso recupera las filas devueltas por una consulta. Los fetches solamente se hacen para sentencias SELECT.

El paso para el cual la fila contiene las estadísticas esta identificado por el valor en la columna *call*.

Columnas de Salida

Las otras columnas de la salida generada por la herramienta de trace, son estadísticas combinadas para todos los parses, ejecuciones y fetches de una sentencia.

- Count
  Número de veces que se hizo el parse, ejecución o fetch de una sentencia.
- Cpu
  Tiempo total en segundos de la CPU utilizado para todos los parse, ejecución y fetch de la sentencia.
- Elapsed
  Tiempo total completo en segundos para todas las llamadas de parse, ejecución y fetch de la sentencia.
- Disk
  Número total de blocks físicos leídos desde el disco para todas las llamadas de parse, ejecución o fetch.
- Query
  Número total de buffers recuperados en modo consistente para todas las llamadas de parse, ejecución y fetch. Los buffers son recuperados generalmente en modo consistente para las consultas.
- Current
  Número total de buffers recuperados en modo corriente. Los buffers son recuperados generalmente en modo corriente para las sentencias INSERT, UPDATE y DELETE.
  La suma de *query* y *current* es el número total de buffers accedidos.
- Rows
  Número total de filas procesadas por la sentencia SQL. Este total no incluye las filas procesadas por sub-consultas de la sentencia SQL.

  Para las sentencias SELECT, el número de filas devueltas aparece en el paso de fetch.

  Para las sentencias UPDATE, DELETE e INSERT, el número de filas procesadas aparece en el paso de ejecución.

Resolución de estadísticas. Como la toma de estadísticas por parte del sistema tiene un tiempo de resolución mínimo de una centésima de segundo, cualquier operación en un cursor que demore una centésima de segundo o menos puede que no sea medida de forma correcta. Esto es una cosa que se debe tener en cuenta al interpretar las estadísticas.

Llamadas recursivas. Algunas veces para ejecutar una sentencia de un usuario, Oracle debe ejecutar algunas sentencias adicionales. Estas sentencias reciben el nombre de *llamadas recursivas* o sentencias SQL recursivas. El tiempo de ejecución de estas llamadas recursivas debe ser tenido en cuanta al evaluar el tiempo necesario para la ejecución total de la sentencia que genera la llamada recursiva.

Ejemplo de la salida generada por el comando TKPROF

CONSULTA REALIZADA

```
SELECT u.desc_usr, p.nom_perfil, o.objeto
FROM usuarios u, usuario_perfil p, perfil_objeto o
WHERE u.nom_usr=p.nom_usr
      and p.nom_perfil = o.nom_perfil
      and p.nom_perfil in
                  (select nom_perfil from perfil_objeto)
```

COMANDO TKPROF EJECUTADO

TKPROF ora_38014 salida EXPLAIN=usuario/password

Donde **usuario/password** corresponden al usuario y a la contraseña de acceso al esquema de la base de datos donde ejecutamos la consulta.

```
TKPROF: Release 7.3.2.2.0 - Production on Tue Aug 11 09:28:26 1998

Copyright (c) Oracle Corporation 1979, 1994.  All rights reserved.

Trace file: ora_38014.trc
Sort options: default

********************************************************************************
count    = number of times OCI procedure was executed
cpu      = cpu time in seconds executing
elapsed  = elapsed time in seconds executing
disk     = number of physical reads of buffers from disk
query    = number of buffers gotten for consistent read
current  = number of buffers gotten in current mode (usually for update)
rows     = number of rows processed by the fetch or execute call
********************************************************************************

alter session
set SQL_TRACE=TRUE
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        0      0.00       0.00          0          0          0           0
Execute      1      0.00       0.10          0          0          0           0
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        1      0.00       0.10          0          0          0           0

Misses in library cache during parse: 0
Misses in library cache during execute: 1
Optimizer goal: CHOOSE
Parsing user id: 24  (PROD)
********************************************************************************
select parameter, value
from
 v$nls_parameters     where (upper(parameter) in ('NLS_SORT','NLS_CURRENCY',
  'NLS_ISO_CURRENCY',                   'NLS_DATE_LANGUAGE',
  'NLS_NUMERIC_CHARACTERS',                   'NLS_LANGUAGE','NLS_TERRITORY'))
```

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.01       0.01          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.00       0.00          0          0          0           7
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.01       0.01          0          0          0           7

Misses in library cache during parse: 0
Optimizer goal: CHOOSE
Parsing user id: 24  (PROD)
error during parse of EXPLAIN PLAN statement
ORA-01039: insufficient privileges on underlying objects of the view

parse error offset: 101
********************************************************************************

select value
from
 v$nls_parameters          where (upper(parameter) = 'NLS_DATE_FORMAT')


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.01       0.01          0          0          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.01       0.01          0          0          0           1

Misses in library cache during parse: 0
Optimizer goal: CHOOSE
Parsing user id: 24  (PROD)
error during parse of EXPLAIN PLAN statement
ORA-01039: insufficient privileges on underlying objects of the view

parse error offset: 94
```

```
SELECT u.desc_usr, p.nom_perfil, o.objeto
FROM usuarios u, usuario_perfil p, perfil_objeto o
WHERE u.nom_usr=p.nom_usr
      and p.nom_perfil = o.nom_perfil
      and p.nom_perfil in
                  (select nom_perfil from perfil_objeto)
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.09          0          0          0           0
Execute      1      0.01       0.00          0          0          0           0
Fetch      208      1.91       2.36       1080       1312         50        3110
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      210      1.92       2.45       1080       1312         50        3110

Misses in library cache during parse: 0
```

```
Optimizer goal: CHOOSE
Parsing user id: 24  (PROD)

Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   GOAL: CHOOSE
   3110   NESTED LOOPS
     40    HASH JOIN
     45     HASH JOIN
     32      INDEX   GOAL: ANALYZED (FULL SCAN) OF 'PK_USUARIO_PERFIL'
                 (UNIQUE)
     13      VIEW
   1777       SORT (UNIQUE)
   1777        TABLE ACCESS   GOAL: ANALYZED (FULL) OF
                   'PERFIL_OBJETO'
     31     TABLE ACCESS   GOAL: ANALYZED (FULL) OF 'USUARIOS'
  40871    TABLE ACCESS   GOAL: ANALYZED (FULL) OF 'PERFIL_OBJETO'


********************************************************************************

OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        3      0.01       0.10          0          0          0           0
Execute      4      0.01       0.10          0          0          0           0
Fetch      210      1.92       2.37       1080       1312         50        3118
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      217      1.94       2.57       1080       1312         50        3118

Misses in library cache during parse: 0
Misses in library cache during execute: 1
OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        0      0.00       0.00          0          0          0           0
Execute      0      0.00       0.00          0          0          0           0
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        0      0.00       0.00          0          0          0           0

Misses in library cache during parse: 0

    4  user  SQL statements in session.
    0  internal SQL statements in session.
    4  SQL statements in session.
    1  statement EXPLAINed in this session.
********************************************************************************
Trace file: ora_38014.trc
Trace file compatibility: 7.03.02
Sort options: default

       0  session in tracefile.
```

```
       4  user  SQL statements in trace file.
       0  internal SQL statements in trace file.
       4  SQL statements in trace file.
       4  unique SQL statements in trace file.
       1  SQL statements EXPLAINed using schema:
           PROD.prof$plan_table
             Default table was used.
             Table was created.
             Table was dropped.
     270  lines in trace file.
```

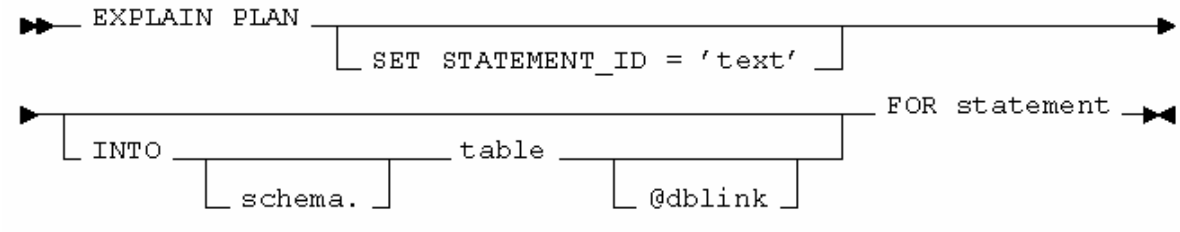
4.1.1 Opciones de Orden

```
PRSCNT      number of times parsed
PRSCPU      CPU time spent parsing
PRSELA      elapsed time spent parsing
PRSDSK      number of physical reads from disk during parse
PRSQRY      number of consistent mode block reads during parse
PRSCU       number of current mode block reads during parse
PRSMIS      number of library cache misses during parse
EXECNT      number of executes
EXECPU      CPU time spent executing
EXEELA      elapsed time spent executing
EXEDSK      number of physical reads from disk during execute
EXEQRY      number of consistent mode block reads during execute
EXECU       number of current mode block reads during execute
EXEROW      number of rows processed during execute
EXEMIS      number of library cache misses during execute
FCHCNT      number of fetches
FCHCPU      CPU time spent fetching
FCHELA      elapsed time spent fetching
FCHDSK      number of physical reads from disk during fetch
FCHQRY      number of consistent mode block reads during fetch
FCHCU       number of current mode block reads during fetch
FCHROW      number of rows fetched
```

4.2 El comando Explain Paln

Este comando nos permite ver el plan de ejecución elegido por Oracle para resolver sentencias de tipo SELECT, UPDATE, INSERT y DELETE.
El plan de ejecución de una sentencia es la secuencia de operaciones que realiza Oracle para ejecutar dicha sentencia.
La utilidad del plan de ejecución es que nos permite ver exactamente como Oracle ejecuta la sentencia, lo que ayuda a ver si la sentencia escrita saca provecho de los índices existentes sobre la/s tabla/s.

La sintaxis es:



- SET STATEMENT_ID
  Especifica el valor de la columna STATEMENT_ID para los registros del plan de ejecución de la tabla de salida.
- INTO
  Especifica el nombre del esquema, tabla y base de datos que va a contener la tabla de salida. Esta tabla debe existir antes de la ejecución del comando. Si se omite este parámetro Oracle considera una tabla de nombre PLAN_TABLE en el esquema del usuario que ejecuta el comando en la base de datos local.
- FOR
  Especifica la sentencia SELECT, UPDATE, INSERT o DELETE para la cual se va a generar el plan de ejecución.

Ejemplo:

```
EXPLAIN PLAN
      SET STATEMENT_ID = 'Actualización de costos'
      INTO salida
      FOR UPDATE precios
            SET art_costo = art_costo * 1,05
            WHERE art_id IN (SELECT art_id
                             FROM origen
                             WHERE pais_origen = 'BRASIL')
```

Antes de ejecutar el comando EXPLAIN PLAN debe existir una tabla en la base de datos donde se va a almacenar la salida, generalmente, el nombre por defecto de esa tabla es PLAN_TABLE.

Esta tabla contiene las siguientes columnas:

1. STATEMENT_ID
Esta columna tiene el valor de la opción STATEMENT_ID especificado en la sentencia EXPLAIN PLAN

2. TIMESTAMP
Fecha y hora en que se ejecutó el comando EXPLAIN PLAN

3. REMARKS
Cualquier comentario de hasta 80 caracteres que se quiera asociar con el paso descripto en el plan de ejecución.

4. OPERATION
El nombre de la operación interna ejecutada en este paso. En la primera línea generada para una sentencia, la columna contiene alguno de estos valores, dependiendo del tipo de sentencia:
'DELETE STATEMENT'

'INSERT STATEMENT'
'SELECT STATEMENT'
'UPDATE STATEMENT'

5. OPTIONS
Una variante de la operación descripta en la columna OPERATION.

6. OBJECT_NODE
El nombre del link de la base de datos usado para referenciar el objeto (un nombre de tabla o un nombre de vista).

7. OBJECT_OWNER
El nombre del usuario dueño del esquema que contiene la tabla o el índice.

8. OBJECT_NAME
El nombre de la tabla o del índice.

9. OBJECT_INSTANCE
Un número correspondiente a la posición del objeto dentro de la sentencia original. La numeración se realiza de izquierda a derecha, desde afuera hacia adentro con respecto al texto original de la sentencia. Hay que tener en cuenta que la expansión de la definición de una vista puede llevar a números impredecibles.

10.OBJECT_TYPE
Este campo tiene información adicional acerca del objeto; por ejemplo: NON-UNIQUE para los índices

11.OPTIMIZER
El modo de funcionamiento del optimizador

12.SEARCH_COLUMNS
No se utiliza

13.ID
Un número asignado a cada paso en el plan de ejecución.

14.PARENT_ID
El ID del próximo paso de ejecución que opera con la salida del paso ID.

15.POSITION
El orden de procesamiento para los pasos que tienen el mismo PARENT_ID.

16.OTHER
Otra información que sea específica al paso en ejecución y que el usuario pueda encontrar útil.

17.OTHER_TAG
Describe el contenido de la columna OTHER

18.COST
El costo de la operación estimado por el optimizador. Para las sentencias que utilizan la resolución basada en reglas, esta columna tiene valor nulo. El costo no está determinado por las operaciones de acceso a las tablas. El valor de esta consulta no tiene una unidad particular de medida, es solamente un valor ponderado que se usa para comparar los costos de los planes de ejecución.

19.CARDINALITY
La estimación que hace la resolución de la consulta basada en costo, del número de filas accedidas por la operación

20.BYTES
La estimación que hace la resolución de la consulta basada en costo, del número de bytes accedidos por la operación

- Lista de las combinaciones de los valores de OPERATION Y OPTIONS producidos por el comando EXPLAIN PLAN y su significado dentro de un plan de ejecución.

OPERATION	**OPTION**	**Descripción**
AND-EQUAL		Una operación que acepta múltiples conjuntos de ROWIDs y devuelve la intersección de los conjuntos, eliminando los duplicados. Esta operación se usa para los accesos a través de índices de una sola columna
CONNECT BY		Una recuperación de filas en orden jerárquico para una sentencia que contiene una cláusula CONNECT BY
CONCATENATION		Una operación que acepta múltiples conjuntos de filas y devuelve la unión de los conjuntos
COUNT		Una operación que cuenta el número de filas seleccionadas de una tabla
	STOPKEY	Una operación count donde el número de filas devueltas está limitado por la expresión ROWNUM en la cláusula WHERE
FILTER		Una operación que acepta un conjunto de filas, elimina alguna de ellas y devuelve el resto.
FIRST ROW		Una recuperación solamente de la primera fila seleccionada por una consulta.
FOR UPDATE		Una operación que recupera y bloquea las filas seleccionadas por una consulta que contiene una cláusula FOR UPDATE
OPERATION	**OPTION**	**Descripción**
INDEX*	UNIQUE SCAN	La recuperación de una sola ROWID desde un índice.
	RANGE SCAN	La recuperación de una o más ROWIDs desde un índice. Los valores indexados son recorridos en orden ascendente
	RANGE SCAN DESCENDING	La recuperación de una o más ROWIDs desde un índice. Los valores indexados son recorridos en orden ascendente
INTERSECTION		Una operación que acepta dos conjuntos de filas y devuelve la intersección de ambos, eliminando duplicados
MERGE JOIN*		Una operación que acepta dos conjuntos de filas, cada uno ordenado por un valor específico, combina cada una de las filas de un conjunto con la fila correspondiente del otro, y devuelve el resultado.
	OUTER	Una operación *merge join* para ejecutar una sentencia *outer join*

MINUS		Una operación que acepta dos conjuntos de filas y devuelve las filas que aparecen en el primer conjunto pero no en el segundo conjunto, eliminando duplicados.
NESTED LOOPS+		Una operación que acepta dos conjuntos de filas, un conjunto exterior y otro conjunto interior. Oracle compara cada fila del conjunto exterior con cada fila del conjunto interior y devuelve aquellas filas que cumplen con una condición.
	OUTER	Una operación de loops anidados para ejecutar una sentencia de *outer join*.
PROJECTION		Una operación interna
REMOTE		Una recuperación de datos desde una base de datos remota.
SEQUENCE		Una operación que involucra el acceso a valores de una secuencia
SORT	AGGREGATE	La recuperación de una sola fila que es el resultado de aplicar una función de grupo a un grupo de filas seleccionadas.
	UNIQUE	Una operación que ordena un conjunto de filas para eliminar duplicados.
	GROUP BY	Una operación que ordena un conjunto de filas en grupos para una consulta con una cláusula GROUP BY
	JOIN	Una operación que ordena un conjunto de filas antes de hacer una operación de *merge-join*
	ORDER BY	Una operación que ordena un conjunto de filas para una consulta con una cláusula ORDER BY
TABLE ACCESS*	FULL	Una recuperación de todas las filas de una tabla
	CLUSTER	Una recuperación de filas de una tabla basada en valores de la clave de un cluster indexado.
	HASH	Una recuperación de filas de una tabla basada en valores de la clave de un hash cluster.
	BY ROWID	Una recuperación de una fila desde una tabla basada en su ROWID
UNION		Una operación que acepta dos conjuntos de filas y devuelve la unión de los conjuntos, eliminando duplicados
VIEW		Una operación que ejecuta una consulta de vistas y luego devuelve las filas resultantes a otra operación.

***** Estas operaciones son métodos de acceso; **+** Estas operaciones son de join.

Ejemplo de la salida del EXPLAIN PLAN.

El siguiente ejemplo muestra una sentencia SQL y su correspondiente plan de ejecución generado por medio del comando EXPLAIN PLAN.

Esta es la sentencia que va a ser analizada:

```
SELECT u.desc_usr, p.nom_perfil, o.objeto
FROM usuarios u, usuario_perfil p, perfil_objeto o
WHERE u.nom_usr=p.nom_usr
      and p.nom_perfil = o.nom_perfil
```

```
      and p.nom_perfil NOT IN
                              (SELECT nom_perfil
                              FROM perfil_objeto);
```

El siguiente comando EXPLAIN PLAN genera el plan de ejecución y coloca la salida en la tabla PLAN_TABLE:

```
EXPLAIN PLAN
SET STATEMENT_ID = 'Ejemplo numero 1'
FOR
SELECT u.desc_usr, p.nom_perfil, o.objeto
FROM usuarios u, usuario_perfil p, perfil_objeto o
WHERE u.nom_usr=p.nom_usr
      and p.nom_perfil = o.nom_perfil
      and p.nom_perfil NOT IN
                              (SELECT nom_perfil
                              FROM perfil_objeto);
```

Esta sentencia SELECT genera la siguiente salida:

```
SELECT operation, options, object_name, id, parent_id, position
FROM plan_table
ORDER BY id;
```

```
OPERATION               OPTIONS     OBJECT_NAME       ID    PARENT_ID   POSITION
-------------------------------------------------------------------------------
SELECT STATEMENT                                      0                     11
HASH JOIN                                             1           0         1
TABLE ACCESS            FULL        USUARIOS          2           1         1
HASH JOIN                                             3           1         2
INDEX                   FULL SCAN   PK_USUARIO_PERFIL 4           3         1
INDEX                   FULL SCAN   PK_PERFIL_OBJETOS 5           4         1
TABLE ACCESS            FULL        PERFIL_OBJETO     6           3         2
```

La cláusula ORDER BY devuelve los pasos del plan de ejecución ordenados secuencialmente por el valor del campo ID, sin embargo, Oracle no ejecuta los pasos en este orden.
Como el paso PARENT_ID recibe la información de la ejecución del paso ID, como se puede ver hay más de un paso ID que vuelcan su resultado en el mismo PARENT_ID.
El valor de la columna POSITION para la primera fila de la salida, indica el costo estimado por el optimizador para la resolución de esta consulta. En el caso de este ejemplo, el costo es 11.

El siguiente tipo de SELECT genera una representación anidada de la salida, mucho más cercana a la forma en que se desarrollan los pasos de procesamiento de la sentencia SQL.
El orden intenta representar una estructura de árbol.

```
SELECT LPAD(' ',2*(LEVEL-1))||operation||' '||options||' '||object_name
       ||' '||DECODE(ID, 0, 'Costo = '||position) "Query Plan"
FROM plan_table
START WITH ID = 0
CONNECT BY PRIOR id = parent_id
```

```
Query Plan
-----------------------------------------
SELECT STATEMENT   Costo = 11
```

```
HASH JOIN
  TABLE ACCESS FULL USUARIOS
  HASH JOIN
    INDEX FULL SCAN PK_USUARIO_PERFIL
      INDEX FULL SCAN PK_PERFIL_OBJETOS
    TABLE ACCESS FULL PERFIL_OBJETO
```

5. HERRAMIENTAS DE MONITOREO UNIX

Se mencionarán algunas herramientas de monitoreo de UNIX, que nos permitirán saber si el servidor está en condiciones de ejecutar una nueva consulta.

5.1 El comando vmstat

A través del comando vmstat (Virtual Memory Statistic) podemos ver información del estado de los procesos, la utilización de la CPU, la utilización de Memoria, el tráfico en los discos, etc.

La sintaxis es:
vmstat <opciones> <intervalo de medición> <n° de muestras>

La primer línea que es una medida de los valores desde el arranque del sistema, no debe tenerse en cuenta.

Ejemplo de una foto tomada durante un período tranquilo.

```
vmstat 1 5
procs     memory            page            disk          faults      cpu
 r b w   swap  free  re  mf pi po fr de sr s6 s2 s2 sd   in   sy   cs us sy id
 1 1 0  10416  6968 154 217 436 162 1332 0 252 0 11 0 2 129 1323 1062 27  9 63
 0 0 0 11031184 879472 78 3 3744 8 2536 0 439 0 2 0  0 1046 2362  623 43  4 53
 1 1 0 11052632 884576 86 118 1832 16 2288 0 430 0 0 0 0 1275 4509 999 41 17 42
 1 3 0 11053200 886960 80 0 1512 0 2168 0 374 0 0 0  0 1060 4620  890 40  8 52
 0 0 0 11053504 887360 282 0 1896 0 2040 0 367 0 0 0 1 1040 3022  744 34  4 62
```

Ejemplo de una foto tomada durante un período de gran actividad.

```
vmstat 1 5
 procs     memory            page            disk          faults      cpu
 r b w   swap  free  re  mf pi po fr de sr s6 s2 s2 sd   in   sy   cs us sy id
 1 1 0   4200   360 153 217 464 162 1332 0 251 0 11 0 2 160   38 1135 27  9 64
 33 10 0 8079936 824936 1041 91 5664 0 3104 0 1166 0 0 0 0 3088 24478 3160 74 26 0
 38 12 0 8072544 824792 1068 16 6264 0 1784 0 1409 0 0 0 0 3891 25055 4316 68 32 0
 19 16 0 8072080 822616 948 3 9216 56 1648 0 1078 0 0 0 0 3371 22528 3987 84 16 0
 30 11 0 8073288 820208 709 19 6152 0 936 0 1145 0 0 0 1 4029 23570 5141 78 22 0
```

Para nuestro análisis, veremos alunas columnas que nos ofrece este comando:

Procesos (procs):
- La columna **r** (ready) indica el número de procesos en cola listos para su ejecución.
- La columna **b** (blocked for resources) indica los procesos en espera de una operación de i/o.
- La columna **w** (swapped) sumariza los procesos que han sido llevados al área de swapping en los últimos segundos.

Cuando la máquina está “bien”, las tres columnas deben de tener un valor cercano a 0. El sistema está critico cuando estos valores tienen dos dígitos.

Memoria (memory):

- La columna **swap** indica la memoria disponible para swapping, expresada en kbytes.
- La columna **free** indica la memoria física disponible, expresada en kbytes.

CPU:
- La columna **us** (user) indica el porcentaje de dedicación de CPU para procesos de usuario.
- La columna **sy** (system) indica el porcentaje de dedicación de CPU para procesos del sistema.
- La columna **id** (idle) indica el porcentaje de dedicación de CPU de esperas o desocupado.

5.2 El comando sar

El comando sar (System Activity Reporter), al igual que el comando vmstat, nos permite ver información sobre los datos de disco, utilización de CPU, utilización de memoria, etc.

La sintaxis es:
sar <opciones> <intervalo de medición> <n° de muestras>

Ejemplo de una foto tomada durante un período tranquilo.

```
sar 1 10
SunOS ufxnic002 5.6 Generic_105181-39 sun4u    04/20/05
18:15:36   %usr   %sys   %wio   %idle
18:15:37      12       2       7      79
18:15:38      12       4       5      79
18:15:39       1       2       0      97
18:15:40       2       0       0      98
18:15:41       2      15       0      83

Average        5       5       2      87
```

Ejemplo de una foto tomada durante un período de gran actividad.

```
sar 1 10
SunOS ufxnic002 5.6 Generic_105181-39 sun4u    04/20/05

09:00:34    %usr    %sys    %wio   %idle
09:00:35      79      21       0       0
09:00:36      78      22       0       0
09:00:37      78      22       0       0
09:00:39      71      29       0       0
09:00:40      74      26       0       0

Average       76      24       0       0
```

La diferencia que tiene con las columnas CPU del vmstat, es que la columna id es separada en dos: **wio** que representa el porcentaje de CPU correspondiente a la espera por i/o, y **idle** que es el porcentaje de CPU desocupada.