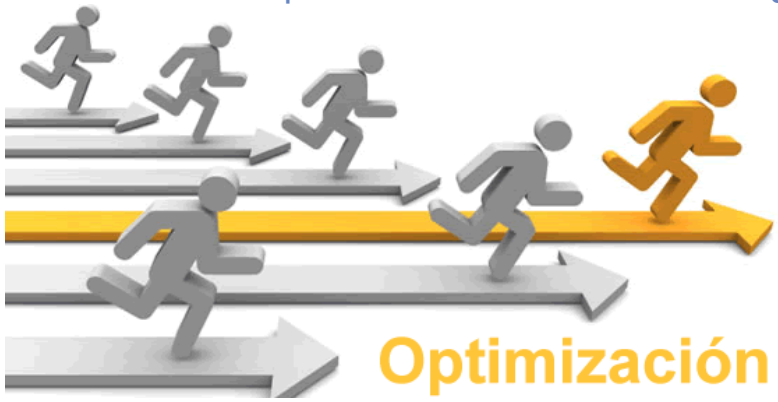
Optimización de consultas SQL

- Conceptos Generales
- Optimización sentencias SQL
- Optimizaciones realizadas
- Herramientas de diagnóstico

01 Conceptos Básicos

Conceptos

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



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 (+).

Tipos de Sentencias SQL

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. Consulta simple: Consulta es otro nombre para la sentencia SELECT.

Sentencias complejas: Una sentencia compleja es un SELECT, INSERT, UPDATE, DELETE que contiene una subconsulta. Equijoins: Joins con condición de igualdad.

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.

02 Optimización de sentencias

everis
an NTT DATA Company

Optimización 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.

Plan de ejecución

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

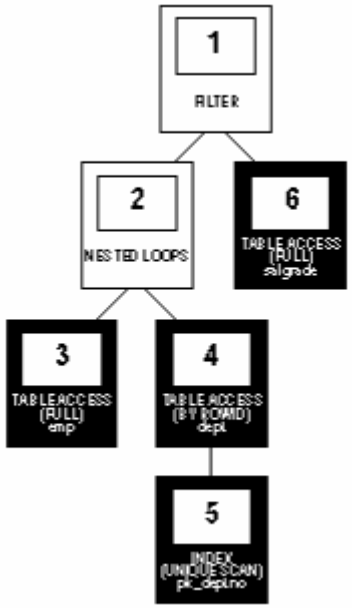
Ejemplo :

```
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);
```

Plan de ejecución

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

```
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);
```

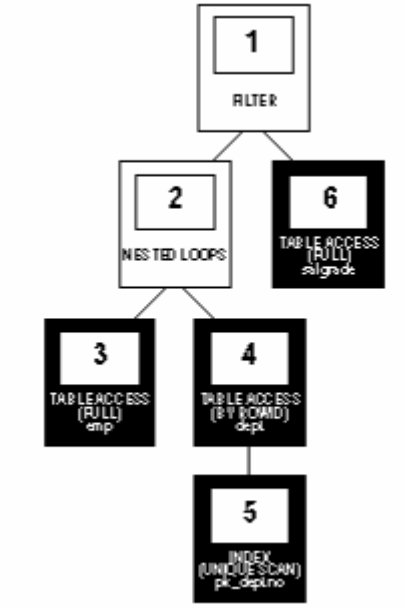


Plan de ejecución

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.

```
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);
```



Plan de ejecución

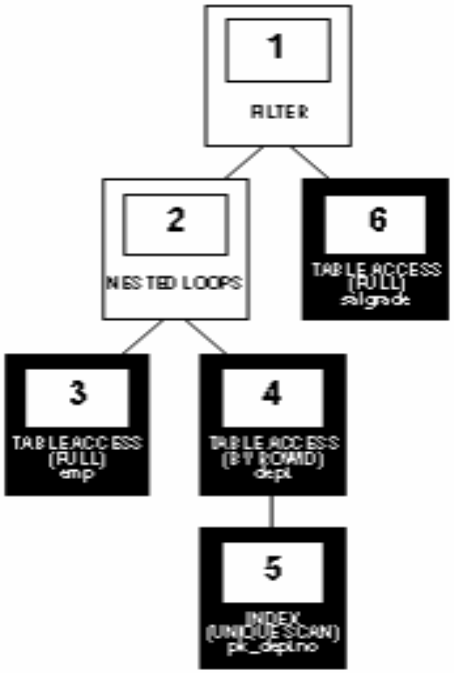
Las cajas sombreadas indican extracción física de datos y objetos de la base de datos.

- 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.

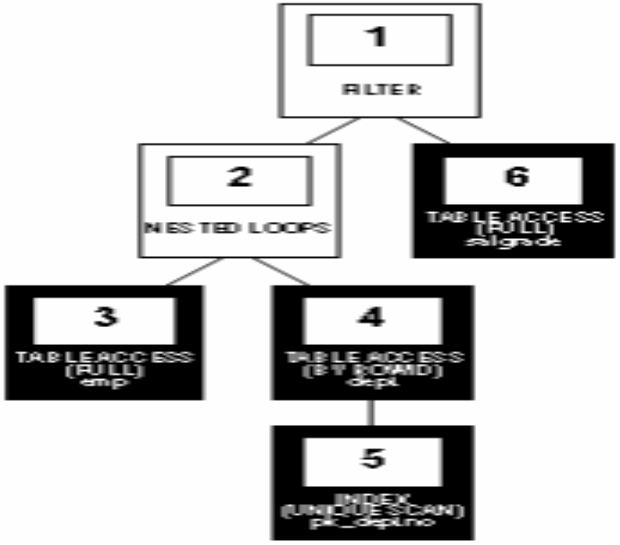
```
SELECT ename, job, sal, dname
FROM emp, dept
WHERE emp.deptno = dept.deptno
AND NOT EXISTS (SELECT *
                FROM salgrade
                WHERE emp.sal BETWEEN losal AND hisal);
```



Plan de ejecución

Orden de realización de las operaciones

-Primero realiza el paso 3 y retorna uno a uno los registros al paso 2
-Para cada registro retornado por el paso 3 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.
-realiza el paso 6 y retorna el registro resultante, si hay alguno al paso 1.
-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



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.

everis
an NTT DATA Company

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

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.

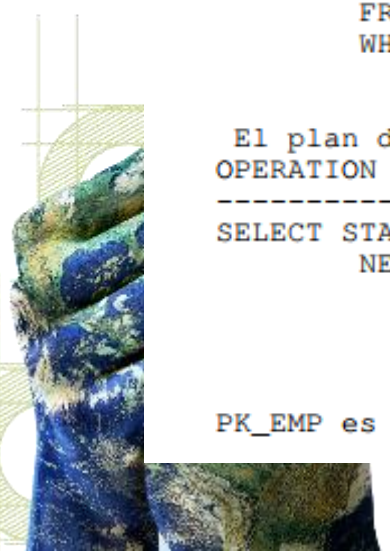
```
Ejemplo :
SELECT * FROM emp WHERE ROWID = '00000DC5.0000.0001';

El plan de ejecución es el siguiente:

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

Tipos de accesos a los datos

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
```



Tipos de accesos a los datos

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
```

Tipos de accesos a los datos

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, 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.
```

Tipos de accesos a los datos

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:

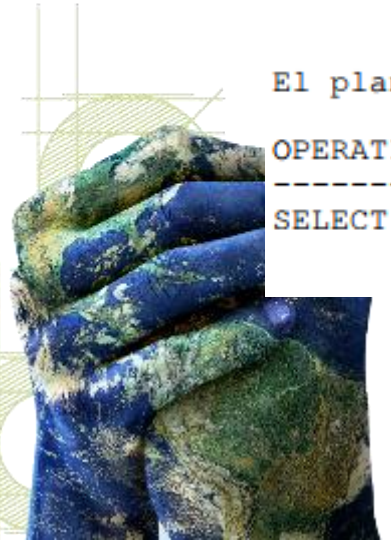
```
OPERATION                          OPTIONS          OBJECT_NAME
-------------------------------------------------------------
SELECT STATEMENT
  NESTED LOOPS
        TABLE ACCESS               FULL               DEPT
        TABLE ACCESS               CLUSTER EMP
```

Donde existe un cluster entre emp y dept por depto.

Tipos de accesos a los datos

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
```



Tipos de accesos a los datos

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.
```

Tipos de accesos a los datos

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.

Tipos de accesos a los datos

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.
```

Tipos de accesos a los datos

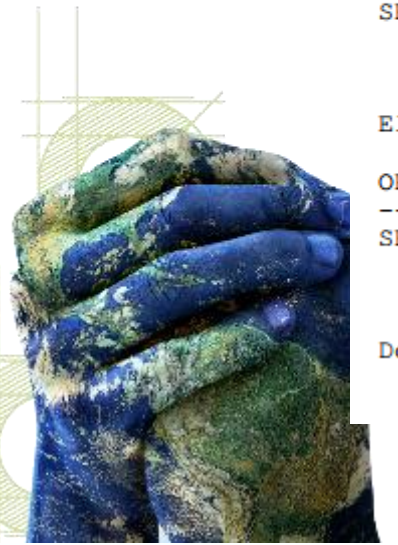
Path 10. – Bounded Range Search on Index Columns. Este acceso es cuando en la cláusula WHERE se encuentran condiciones que usan parcialmente la columna de un índice simple o una o más columnas que forman parte de un índice compuesto. .

```
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.
```

Tipos de accesos a los datos

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.
```



Tipos de accesos a los datos

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
```

Tipos de accesos a los datos

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

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

Tipos de accesos a los datos

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.

```
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.
```

Tipos de accesos a los datos

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
```

Tipos de accesos a los datos

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

```
  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.

03 Optimizaciones Realizadas por Oracle

everis
an NTT DATA Company

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.

everis
an NTT DATA Company

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.
```

everis
an NTT DATA Company

Evaluación de expresiones y condiciones

- 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)
```

everis
an NTT DATA Company

Evaluación de expresiones y condiciones

- 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)
```

Evaluación de expresiones y condiciones

- 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'
```

Evaluación de expresiones y condiciones

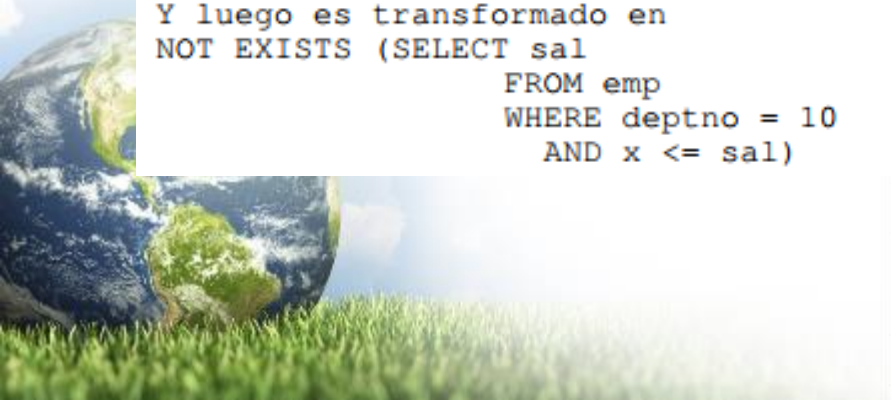
- ANY or SOME: Cuando son usados con constantes los transforma en ors como a continuación:

```
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)
```

Evaluación de expresiones y condiciones

- 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)
```



everis
an NTT DATA Company

Evaluación de expresiones y condiciones

- 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
```

everis
an NTT DATA Company

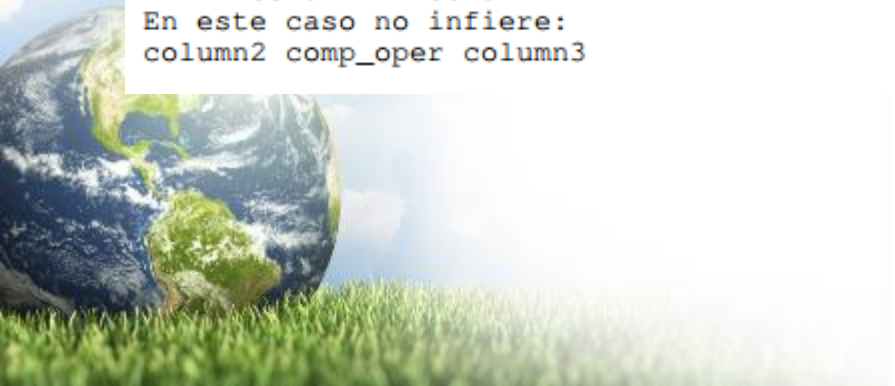
Evaluación de expresiones y condiciones

- 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.

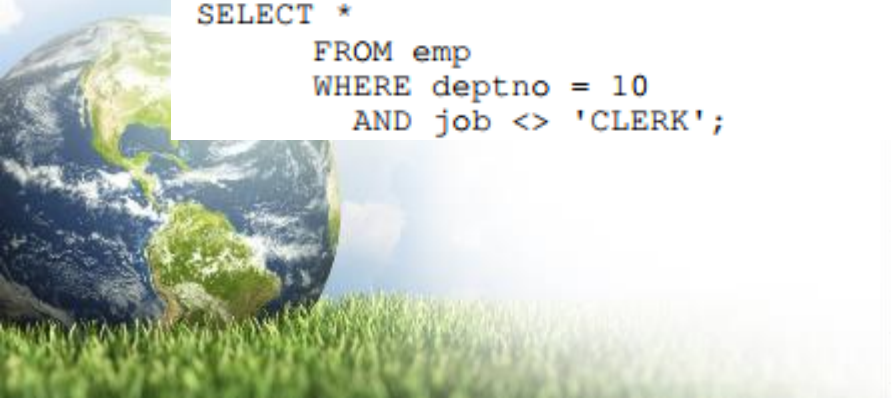
```
Nota: el optimizador solo infiere transitivas de constantes y no de columnas de tablas:
WHERE column1 comp_oper column3
  AND column1 = column2
En este caso no infiere:
column2 comp_oper column3
```



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';
```



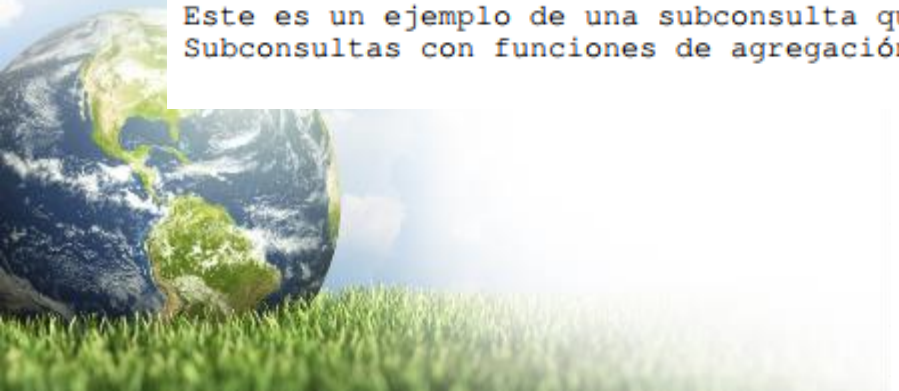
everis
an NTT DATA Company

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;
```

```
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.



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.

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
```

Reglas de optimización:

- **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.

Reglas de optimización:

- **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 %**

.

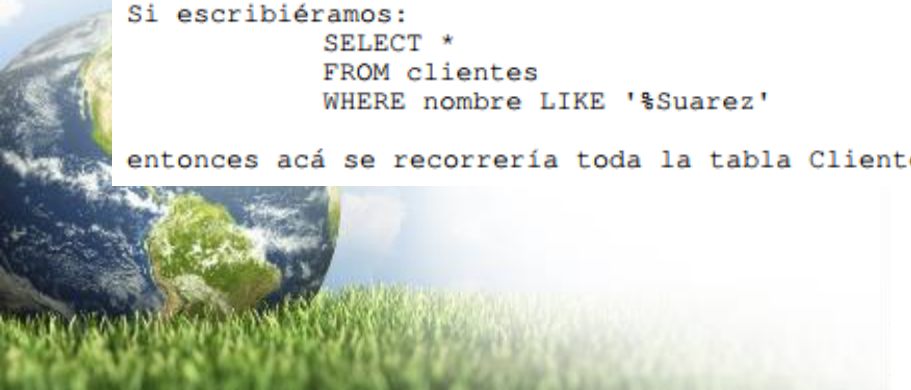
Ejemplo :

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

Si escribiéramos:

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

entonces acá se recorrería toda la tabla Clientes.



Reglas de optimización:

- **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.

Reglas de optimización:

```
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.

Reglas de optimización:

- **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

Reglas de optimización:

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).

everis
an NTT DATA Company

Reglas de optimización:

- **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.

Reglas de optimización:

- **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.

everis
an NTT DATA Company

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';
```

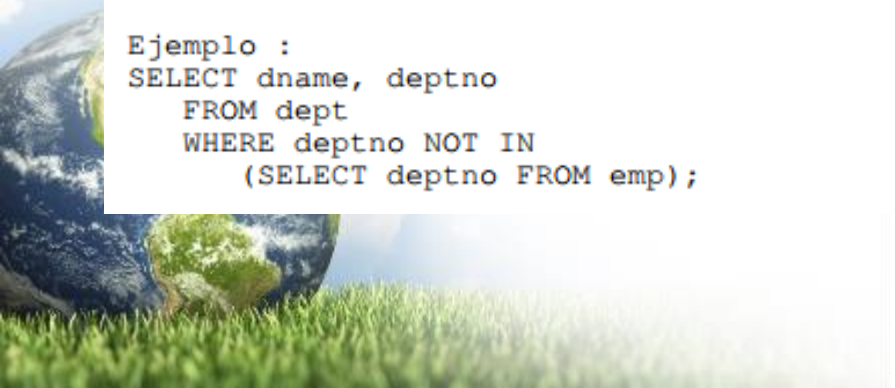
En este caso se nos introduce el índice por estado del recibo y no busca en casi todos los recibos.
Si hacemos el siguiente cambio no se introduce.

```
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);
```



Otras sugerencias:

Podríamos cambiarlo por:

```
SELECT dname, deptno
FROM dept

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

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;
```

Otras sugerencias:

- 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
```

everis
an NTT DATA Company

Índices

Cuando crear Índices

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

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.

Índices

Cuando crear Índices

- 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.

Índices compuestos

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 tabl(x, y, z);
```

Índices compuestos

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 tabl(x, y, z);
```

everis
an NTT DATA Company

Índices compuestos

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.**

everis
an NTT DATA Company

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.

Utilización de “Sugerencias” (HINTS)

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).

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).

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.

Sugerencias de métodos de acceso

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*.

Sugerencias de métodos de acceso

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)
```

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)
```

Sugerencias de métodos de acceso

• **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.

Sugerencias de métodos de acceso

• **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.

everis
an NTT DATA Company

Sugerencias de métodos de acceso

• **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.

Sugerencias para órdenes 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;
```

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...)
```

04 Herramientas de diagnóstico

Herramientas de diagnóstico

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

- La herramienta TRACE
- El comando EXPLAIN PLAN

SQL Trace Facility

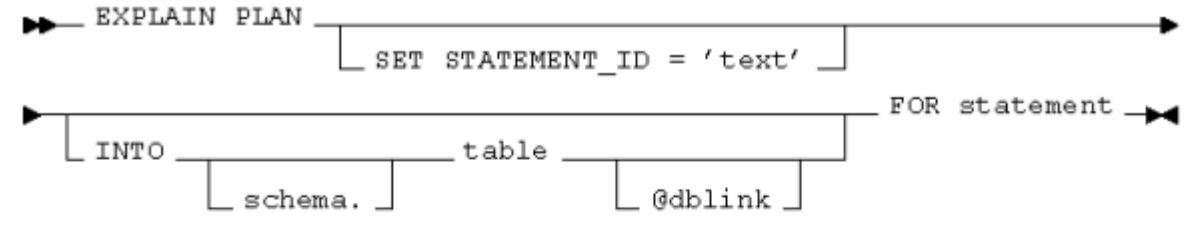
Esta herramienta brinda información de la performance de sentencias SQL individuales. Esta herramienta genera las siguientes estadísticas para cada sentencia:

- 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.

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.

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: