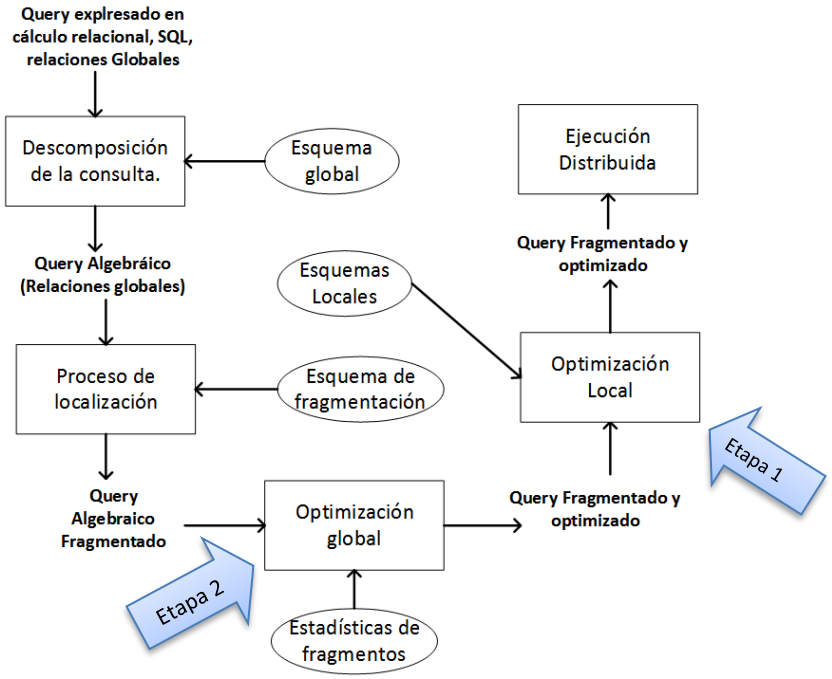
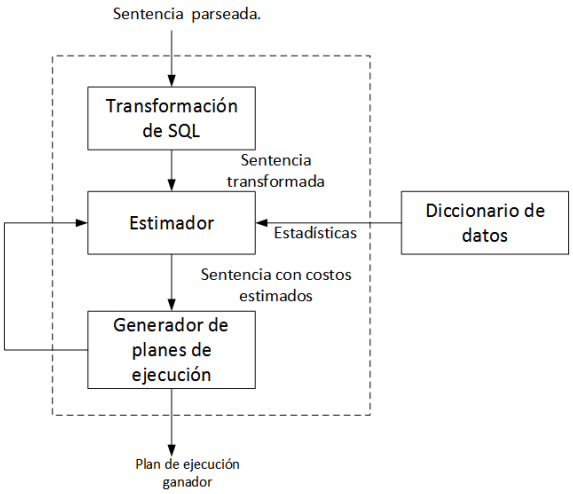
TEMA 5
OPTIMIZACIÓN DE CONSULTAS DISTRIBUIDAS – PARTE 1

5.1. INTRODUCCIÓN

- Las optimizaciones del tema anterior se realizaron considerando el esquema global y los esquemas de fragmentación.
- Mediante la aplicación de propiedades, reglas de transformación y equivalencia se realizó la simplificación de una consulta distribuida eliminando expresiones que producen conjuntos de datos vacíos.
- Sin embargo, este tipo de optimización no considera aspectos importantes como:
  - Encontrar una mejor solución al intercambiar el orden de ejecución de operaciones: Árboles de operadores equivalentes.
  - Características de los fragmentos. Por ejemplo: Cardinalidad, selectividad, etc.
- Por lo anterior, se requiere de una siguiente etapa encargada de generar los llamados *planes de ejecución* y seleccionar aquel que represente la opción más óptima, o por lo menos la más cercana a lo óptimo.
- Esta etapa es representada por el llamado *Optimizador global*.
- En una BDD el optimizador debe realizar tareas adicionales a las de un optimizador centralizado. Dichas tareas están representadas por una serie de algoritmos que consideran la naturaleza de distribución de la BD.
- Para contar con una mejor comprensión de estos algoritmos, el capítulo se divide en 2 etapas:
  - 1. Revisión de las características de un optimizador centralizado
  - 2. Los conceptos aprendidos en la etapa 1 permitirán comprender los algoritmos empleados en un optimizador global.



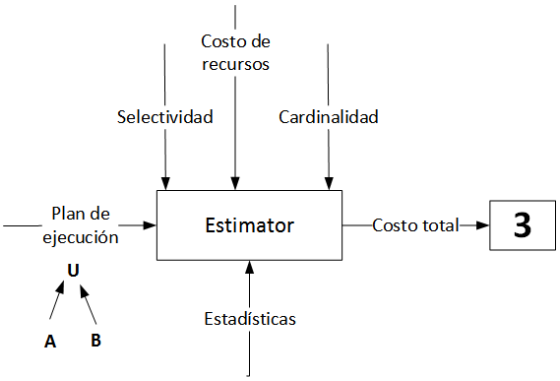
5.2. CARACTERÍSTICAS DEL OPTIMIZADOR CENTRALIZADO:



Formado por los siguientes componentes:

5.2.1. Query transformer

- El optimizador determina si es necesario re-escribir la sentencia SQL de tal forma que sea posible obtener mejores planes de ejecución.
- Debido a que SQL no es un lenguaje procedimental, el optimizador es libre de dividir, reorganizar o reescribir una sentencia y procesarla en cualquier orden.
- Transformaciones de sentencias SQL: (Ejemplos de transformaciones en Oracle).
    - OR Expansion
    - View Merging
    - Predicate Pushing
    - Subquery Unnesting
    - Query Rewrite with Materialized Views
    - Star Transformation
    - In-Memory Aggregation
    - Table Expansion
    - Join Factorization



5.2.2. Estimador.

- Su principal función es determinar el costo total (valor numérico) para un determinado plan de ejecución.
- Este cálculo se realiza considerando principalmente variables de:
    - Costo de procesamiento
    - Costo de operaciones I/O
    - Costos de comunicación
- Emplea principalmente 3 medidas para determinar el costo total:
    - *Selectividad*: Porcentaje estimado de registros que se obtendrían de una fuente de datos (una tabla, una vista, o el resultado de un join) al aplicarle un predicado. La selectividad es expresada por un número decimal en el rango [0,1]
    - *Cardinalidad*: Número de registros obtenido en cada operación contenida en el plan de ejecución.
    - *Costo de recursos*: Unidades de trabajo de recursos empleados: Uso de disco, procesador y memoria.
- La estimación de los valores de selectividad y cardinalidad se obtienen a través de:
    - Estadísticas
    - Valores por default en caso de no contar con estadísticas.

- Histogramas, en especial para distribuciones heterogéneas de los datos.

5.2.3. Generador de planes de ejecución.

- Encargado de generar y analizar posibles planes de ejecución por cada ***bloque SQL***. Una de las principales actividades es la ejecución de operaciones JOIN.
- El plan generado es enviado al estimador para determinar su costo total de ejecución.
- Cuando en una sentencia SQL existen múltiples tablas, el generador debe determinar la forma más eficiente de combinar cada pareja de tablas. Para ello, debe realizar una adecuada selección de los siguientes conceptos:
  - Selección del método de acceso a datos (Access Paths)
  - Selección de la estrategia de Ordenamiento de operaciones JOIN
  - El tipo de JOIN (inner, outer, etc.)
  - Selección de la estrategia para ejecutar una operación JOIN

5.2.3.1. Selección del método de acceso a datos (Access Paths).

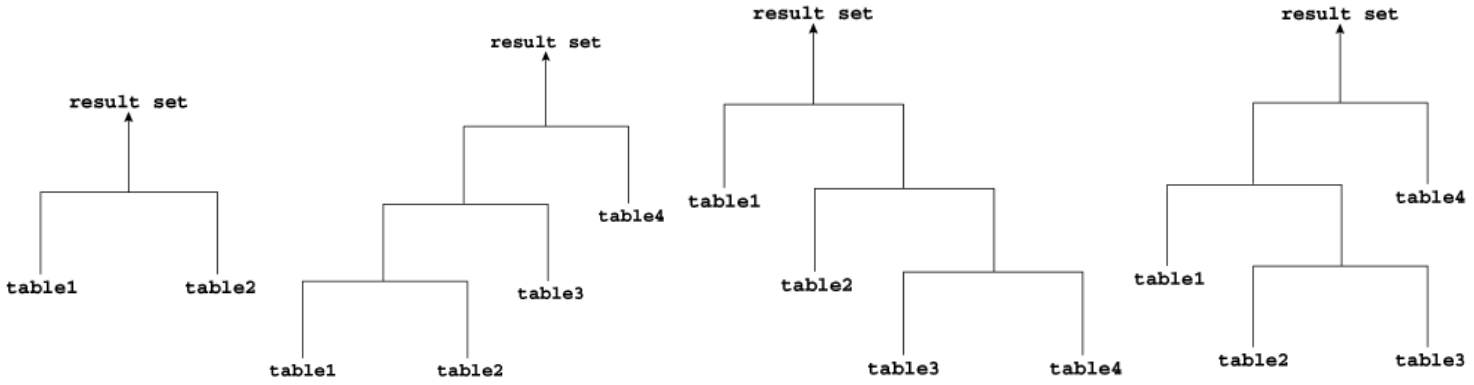
Ejemplos de métodos de acceso a datos en Oracle.

Access Path	Tablas	Índices B-Tree	Índices BitMap	Cluster Tables
Full Table Scans	X			
Table Access by Rowid	X			
Sample Table Scans	X			
Index Unique Scans		X		
Index Range Scans		X		
Index Full Scans		X		
Index Fast full scans		X		
Index Skip Scans		X		
Index Join scans			X	
BitMap index single value			X	
BitMap index Range Scans			X	
BitMap merge				
Cluster scans				X
Hash scans				X

- Un clúster de tablas está formado por un conjunto de tablas que comparten columnas en común. Los datos de dichas columnas se almacenan en el mismo bloque de datos. Esta condición permite que se requiere leer un solo bloque para obtener datos de diferentes tablas.
- Los Access paths BitMap y Hash scan hacen referencia a índices de tipo BitMap y al uso de funciones hash. Para efectos del curso se revisará únicamente Access paths para índices B – Tree.

5.2.3.2. Selección de la estrategia de Ordenamiento de operaciones JOIN.

- Join Tree
- Left Deep Join Tree
- Right Deep Join Tree
- Bushy (tupido) Join Tree



5.2.3.3. Selección del Método para ejecutar operaciones JOIN

- Nested Loops Joins
- Hash Joins
- Sort Merge Joins
- Cartesian Joins

5.2.3.4. Tipos de Joins.

El tipo de join especificado en la sentencia SQL puede influenciar al optimizador en especial para determinar el orden correcto de ejecución:

- Inner Joins
- Outer Joins
- Semijoins
- Antijoins

Una de los principales objetivos en cuanto a la ejecución de operaciones JOIN es la reducción de registros desde las primas operaciones de tal forma que los pasos siguientes sean menos costosos. Ejemplo:

- Tratar de ejecutar primero operaciones JOIN que generen como resultado un solo registro a lo más, es decir, detectar predicados asociados con restricciones `UNIQUE`, `PRIMARY KEY`: `where empleado_id = 10`

5.3. OPTIMIZACIÓN DE CONSULTAS CENTRALIZADAS – ORACLE.

- La entrada del optimizador es una consulta SQL parseada, formada por un conjunto de “bloques”. Cada instrucción `select` representa un bloque.

Ejemplo:

```
select first_name, last_name
from hr.employees
where department_id
    in (select department_id
          from hr.departments
          where location_id = 1800
    );
```

- En el ejemplo anterior existen 2 bloques SQL. Por cada bloque el optimizador genera un "Sub-plan" de ejecución. Cada sub-plan es optimizado de forma separada.
- El número posible de "Sub-planes" que pueden existir es proporcional al número de tablas listadas en la cláusula "FROM".

La selección del mejor plan de ejecución depende de varios factores:

- La forma en la que se escribe la consulta.
- El tamaño de los conjuntos de datos
- Las estructuras de acceso a datos existentes.
- Los recursos del sistema existentes: memoria, procesador.
- Las estadísticas existentes.

Ejemplo:

Suponer que se desea obtener a todos los empleados que son managers.

- Si las estadísticas indican que el 80% de los empleados son managers, el optimizador seleccionaría un ***full table scan***.
- Si las estadísticas indican que el 5% de los empleados son managers, el optimizador seleccionaría un ***index scan***.

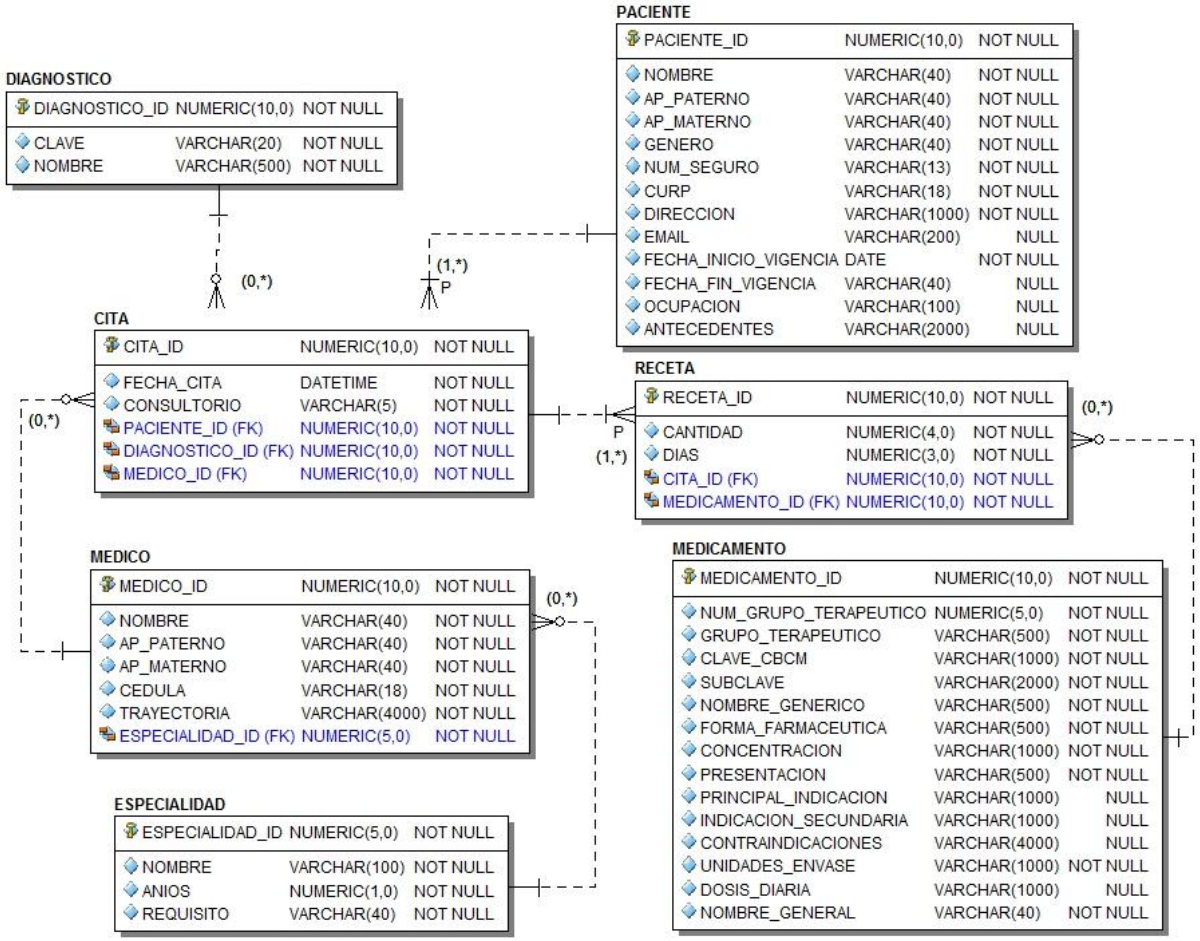
5.3.1. Planes de ejecución

- Un plan de ejecución está formado por la combinación de pasos que la BD realiza para ejecutar una sentencia SQL.
- En cada paso se obtienen datos físicamente.
- El plan de ejecución incluye:
  - "Access Paths" para cada tabla incluida en la sentencia
  - El orden en el que se ejecutan las operaciones JOIN
  - El método seleccionado para ejecutar un JOIN.
  - Operaciones sobre los datos como son: `filter, sort, aggregation`.
  - Costo y cardinalidad de cada operación.
  - Información de particionamiento en caso de tablas particionadas.
  - Procesamiento en paralelo.
- Para mostrar un plan de ejecución se emplea la sentencia `explain plan`.
- Al ejecutar la sentencia anterior, el optimizador elige el plan de ejecución e inserta los datos que los describen en una tabla llamada `plan_table`

- En caso de no existir, la tabla se puede crear ejecutando `@$ORACLE_HOME/rdbms/admin/catplan.sql`
- Posterior a la ejecución de la sentencia `explain plain` es posible emplear algún script o paquete proporcionado por Oracle para mostrar el plan de ejecución más reciente almacenado en `plan_table`, todos ellos ubicados en $ORACLE_HOME/rdbms/admin
  - `utlxpls.sql`
  - `utlxplp.sql`
  - También existe esta función para personalizar la salida del plan: `dbms_xplan.display`, por ejemplo, para definir el detalle de la salida se especifica: `basic`, `serial`, `typical`, y `all`

Ejemplo:

Considerar el siguiente modelo relacional para ilustrar los ejemplos que corresponde con el control de citas y médicos de un hospital.



- Obtener el plan de ejecución para la siguiente consulta:

Mostrar el nombre de los médicos y las fechas de sus citas programadas en el consultorio C-593

```
SQL>
explain plan
set statement_id ='s1' for
select m.nombre,c.fecha_cita
from medico m, cita c
where m.medico_id = c.medico_id
and consultorio='C-593'
```

Mostrar el plan de ejecución:

```
SQL> set linesize 100
SQL> select plan_table_output
     from table(dbms_xplan.display('PLAN_TABLE','s1','TYPICAL'));
```

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
Plan hash value: 2487834737

-------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |           |    17 |  1105 |    51   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                 |           |    17 |  1105 |    51   (0)| 00:00:01 |
|   2 |   NESTED LOOPS                |           |    17 |  1105 |    51   (0)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL          | CITA      |    17 |   510 |    34   (0)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN          | MEDICO_PK |     1 |       |     0   (0)| 00:00:01 |
|   5 |   TABLE ACCESS BY INDEX ROWID| MEDICO    |     1 |    35 |     1   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("CONSULTORIO"='C-593')
   4 - access("M"."MEDICO_ID"="C"."MEDICO_ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
   - this is an adaptive plan
```

A nivel básico un plan de ejecución típicamente está formado por los siguientes elementos (mostrados en la figura anterior).

Columna Id:

- Representa el identificador de la operación. Este valor NO indica el orden de ejecución. Se emplea únicamente para hacer referencia a cada operación.
- Observar los "*" que se muestran del lado izquierdo. Este carácter indica la existencia de información adicional para la operación. Observar en la imagen anterior, en la sección "Predicate Information" aparece el detalle de cada operación empleando su identificador.

Operation:

- Indica el tipo de operación que se va a realizar. Existen diversos tipos de operación, pero a nivel básico, esta columna incluye el "Access Path" o método de acceso a datos que se empleará para obtener datos.
- Observar que existe una indentación entre operaciones. La operación con la mayor indentación se ejecuta primero. En este ejemplo, las operaciones 3 y 4 se ejecutarán primero de forma independiente. El resultado de ambas sirve como entrada para ejecutar la siguiente operación, que en este caso es la operación 2. Observar que el resultado de la operación 2 junto con el resultado

de la operación 5 se encuentran en el mismo nivel, por lo tanto, ambas operaciones serán empleadas para ejecutar la operación 1. Finalmente, el resultado de la operación 1 será empleado por la operación 0 para mostrar el resultado final.

Rows:
- Indica el número de registros estimados que obtendría cada una de las operaciones ejecutadas.

Bytes:
- Memoria estimada en bytes que se requiere para ejecutar cada una de las operaciones del plan

Cost:
- Porcentaje estimado de procesamiento requerido para ejecutar cada una de las operaciones del plan.

Time:
- Tiempo estimado de procesamiento de cada una de las operaciones del plan.

5.3.2. Métodos de acceso a datos (Access Paths)

Este concepto es fundamental para entender y generar planes de ejecución eficientes. Como se mencionó anteriormente, a nivel básico existen 2 tipos de métodos de acceso:
- Acceso a índices.
- Acceso a tablas.

5.3.2.1. Acceso a Índices

Para el caso de los índices, es importante recordar la información que contienen. Para efectos de un plan de ejecución, el índice puede ser visualizado como una tabla de datos con 2 columnas: `ROW_ID` y `etiqueta`. Por ejemplo, si la columna `email` de la tabla `cliente` está indexada, los datos que contiene el índice se verán así:

ROW_ID	etiqueta
01	juan@mail.com
02	paco@mail.com
03	hugo@mail.com
…	….

- Recordando, un ROW_ID representa un puntero a disco donde se encuentra almacenado cada registro de la tabla y representa el método de acceso más eficiente para recuperar un dato. Por simplicidad, se emplean los valores 01,02,03, pero en realidad los valores de un ROW_ID en el caso de Oracle son 18 caracteres. Para mayores detalles en cuanto al funcionamiento de índices B tree y ROW_IDs, revisar tema 3 del curso de BD, o en su defecto https://docs.oracle.com/cloud/latest/db112/CNCPT/indexiot.htm#CNCPT721
- El objetivo de leer un índice es la obtención de una lista de ROW_IDs que serán empleados para acceder a los datos de una tabla de forma eficiente.

5.3.2.2. Acceso a tablas.

En esta estrategia se realiza un acceso directo a los registros de una tabla. Generalmente un acceso a un índice requiere menos recursos ya que su estructura es de un tamaño mucho menor que el de una tabla. Sin embargo, en algunos casos un acceso a una tabla puede ser mejor opción que un acceso a disco.
La manera en la que la tabla organiza el almacenamiento de sus datos también influye en la selección del tipo de acceso:

- ***Heap Organized table (Default):*** Los registros no son almacenados en algún orden en particular. Cada registro se almacena en el primer espacio disponible dentro del segmento de datos.
- ***Index Organized table***: Los registros se ordena de acuerdo al valor de la PK.
- ***External table:*** Tabla de solo lectura, los metadatos de la tabla se almacenan en la BD mientras que los datos son almacenados de forma externa.

En las siguientes secciones se revisan estos 2 tipos de accesos a detalle.

5.3.3. Table Access Paths.

5.3.3.1. Table Access Full.

En general el optimizador selecciona esta estrategia en las siguientes situaciones:

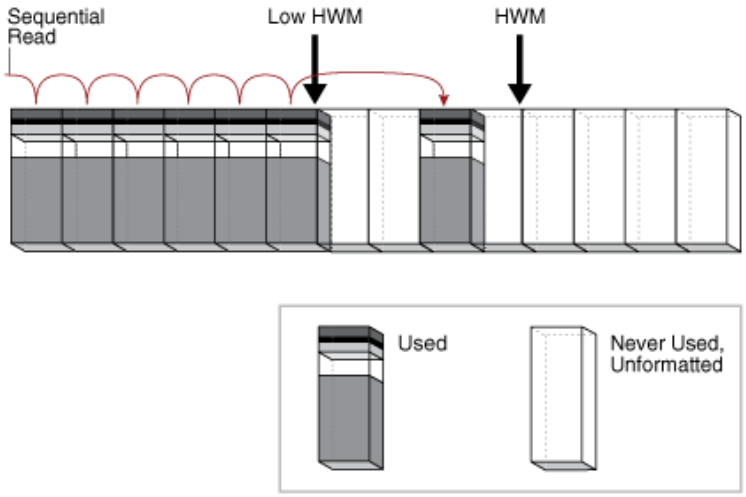
- No existe un índice
- El predicado del query invoca funciones y no existe un índice basado en dicha función.
- Se lanza un `count(*)` , el índice existe, pero existen valores nulos en la columna indexada.
- El número de registros a obtener es grande (selectividad baja).
  - Es más eficiente hacer Table Access Full (leer cantidades grandes de datos pocas veces) que leer pocos datos con alta frecuencia.
- Las estadísticas están desactualizadas.
  - Al inicio la tabla es pequeña, después crece. Dicho crecimiento permitiría el uso de un índice para mejorar el desempeño. Sin embargo, al no actualizar dicho crecimiento en las estadísticas, el optimizador no podrá determinar que el índice es más adecuado.
- Tablas pequeñas.
  - Si la tabla contiene menos de N bloques de datos donde N es un umbral definido por el parámetro
    `DB_FILE_MULTIBLOCK_READ_COUNT`

```
SQL> show parameter DB_FILE_MULTIBLOCK_READ_COUNT

NAME                 TYPE        VALUE
------------------------------------ ----------- ---------
db_file_multiblock_read_count          integer     128
```

- La consulta usa un Hint para forzar un full table scan : `FULL(table alias)`

Para realizar un full table scan, el manejador leerá todos los bloques de datos que contiene datos (bloques con formato) hasta encontrar a la llamada “Marca de Agua”. Bloques sin formato son saltados. Cada bloque se lee una sola vez.



Ejemplo:

```
explain plan for
select * from paciente;

select plan_table_output
from table(dbms_xplan.display);
```

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 588422155

--------------------------------------------------------------------------------
| Id  | Operation         | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          | 15302 |    26M|   477   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| PACIENTE | 15302 |    26M|   477   (0)| 00:00:01 |
--------------------------------------------------------------------------------
```

5.3.3.2. Table Access by rowid

- La forma más rápida de acceder a un registro es a través de su ROW_ID.
- Generalmente se emplea este método de acceso posterior al escaneo de un índice.
- Si el índice contiene todas las columnas requeridas se omite realizar esta operación.

Ejemplo:

```
explain plan for
select nombre
from paciente
where paciente_id = 3;

select plan_table_output
from table(dbms_xplan.display);
```

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
Plan hash value: 3341724136

-------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |     1 |    35 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| PACIENTE    |     1 |    35 |     2   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN          | PACIENTE_PK |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
------------------------------------------------

   2 - access("PACIENTE_ID"=3)
```

Ejemplo:

```
explain plan for
select nombre
from paciente
where paciente_id >18500;

select plan_table_output
from table(dbms_xplan.display);
```

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
Plan hash value: 2503382986

-------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |             |     1 |    35 |     1   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PACIENTE    |     1 |    35 |     1   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | PACIENTE_PK |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
------------------------------------------------

   2 - access("PACIENTE_ID">18500)
```

- En este ejemplo, la lectura del índice produce pocos ROW_IDs, pero el manejador lee por bloque para minimizar el número de accesos a un mismo bloque. Esto se indica con la palabra BATCHED.

5.3.3.3. Sample table scan

- En esta estrategia se obtiene una muestra de registros de una tabla o de una sentencia `SELECT` compleja que contiene diversos joins y “Vistas”.
- La muestra deseada se expresa en porcentaje [0.000001,100)
- Obtiene un porcentaje aproximado del contenido de una tabla. El porcentaje se aplica a los bloques, no a los registros.

Ejemplo:

```
explain plan for
select *
from cita sample block(10);

select plan_table_output
from table(dbms_xplan.display);
```

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 1426444112

-------------------------------------------------------------------------------
| Id  | Operation           | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |   208 | 16848 |     5   (0)| 00:00:01 |
|   1 |  TABLE ACCESS SAMPLE| CITA |   208 | 16848 |     5   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

5.3.3.4. In Memory table scan

- Obtiene registros de una tabla con atributos almacenados en memoria de forma Columnar: *IM Column Store*.
- IM Column Store es un área de memoria ubicada en la SGA que almacena copias de tablas en un formato columnar especial que permite agilizar el escaneo de datos.

Ejemplo:

```
SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR);

SQL_ID  2mb4h57x8pabw, child number 0
-------------------------------------
select * from oe.product_information where list_price > 10 order byproduct_id

Plan hash value: 2256295385
-----------------------------------------------------------------------------------------------
|Id| Operation                   | Name                |Rows|Bytes |TempSpc|Cost(%CPU)|Time|
-----------------------------------------------------------------------------------------------
| 0| SELECT STATEMENT            |                     |    |      |       |21 (100)|        |
| 1|  SORT ORDER BY              |                     | 285| 62415|82000|21   (5)|00:00:01|
|*2|   TABLE ACCESS INMEMORY FULL| PRODUCT_INFORMATION | 285| 62415|       | 5   (0)|00:00:01|
-----------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - inmemory("LIST_PRICE">10)
       filter("LIST_PRICE">10)
```

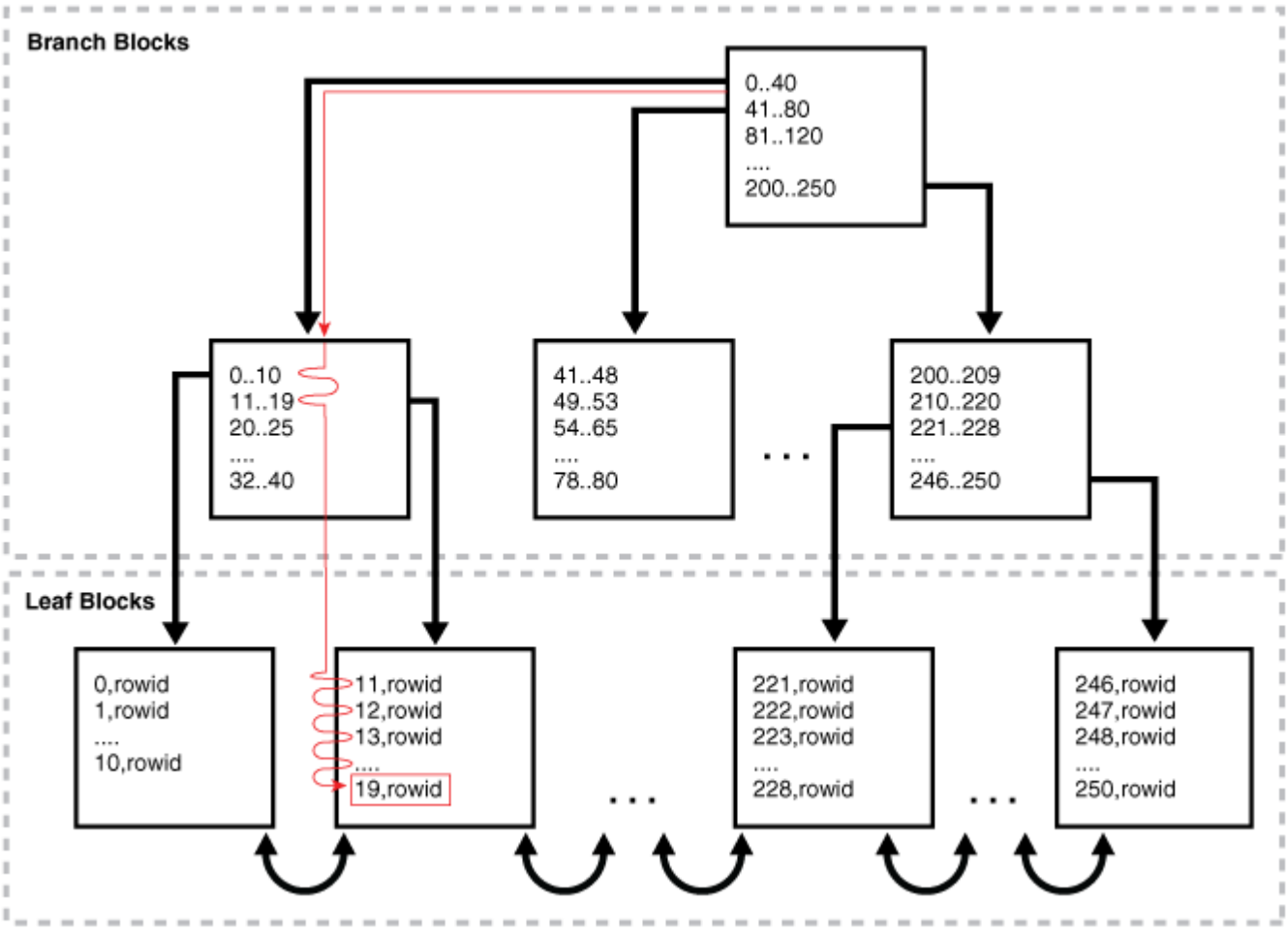
5.3.4. B-Tree Index Access Paths.

Para efectos del curso, solo se consideran los índices tipo B-Tree.

- Index Unique Scans
- Index Range Scans
- Index Full Scans
- Index Fast Full Scans
- Index Skip Scans
- Index Join Scans

5.3.4.1. Index Unique scan.

- Regresa a lo más un ROW_ID al realizar el escaneo del índice.
- Empleado cuando se incluye un operador de igualdad en una columna indexada
- El índice se recorre hasta encontrar la primera ocurrencia debido a que se trata de un Índice de tipo UNIQUE y por lo tanto, no es necesario continuar con el escaneo.
- En la siguiente figura se muestra la forma en la que se realiza el escaneo para encontrar el ROW_ID cuyo valor de la columna es 19.



Ejemplo:

```
create unique index paciente_email_idx
on paciente(email);

explain plan for
select email
from paciente
where email =' jonh@mail.com';

select plan_table_output
from table(dbms_xplan.display);
```

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------------------------
Plan hash value: 839293805

-------------------------------------------------------------------------------------
| Id  | Operation         | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |                    |     1 |   102 |     1   (0)| 00:00:01 |
|*  1 |  INDEX UNIQUE SCAN| PACIENTE_EMAIL_IDX |     1 |   102 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------------------------

   1 - access("EMAIL"=' jonh@mail.com')
```

Ejemplo:

Observar que el siguiente ejemplo no se hace uso del índice. ¿Por qué razón?

```
explain plan for
select email
from paciente;


select plan_table_output
from table(dbms_xplan.display);
```

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------------------------
Plan hash value: 588422155

-------------------------------------------------------------------------------
| Id  | Operation         | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          | 16584 |  1651K|   477   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| PACIENTE | 16584 |  1651K|   477   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

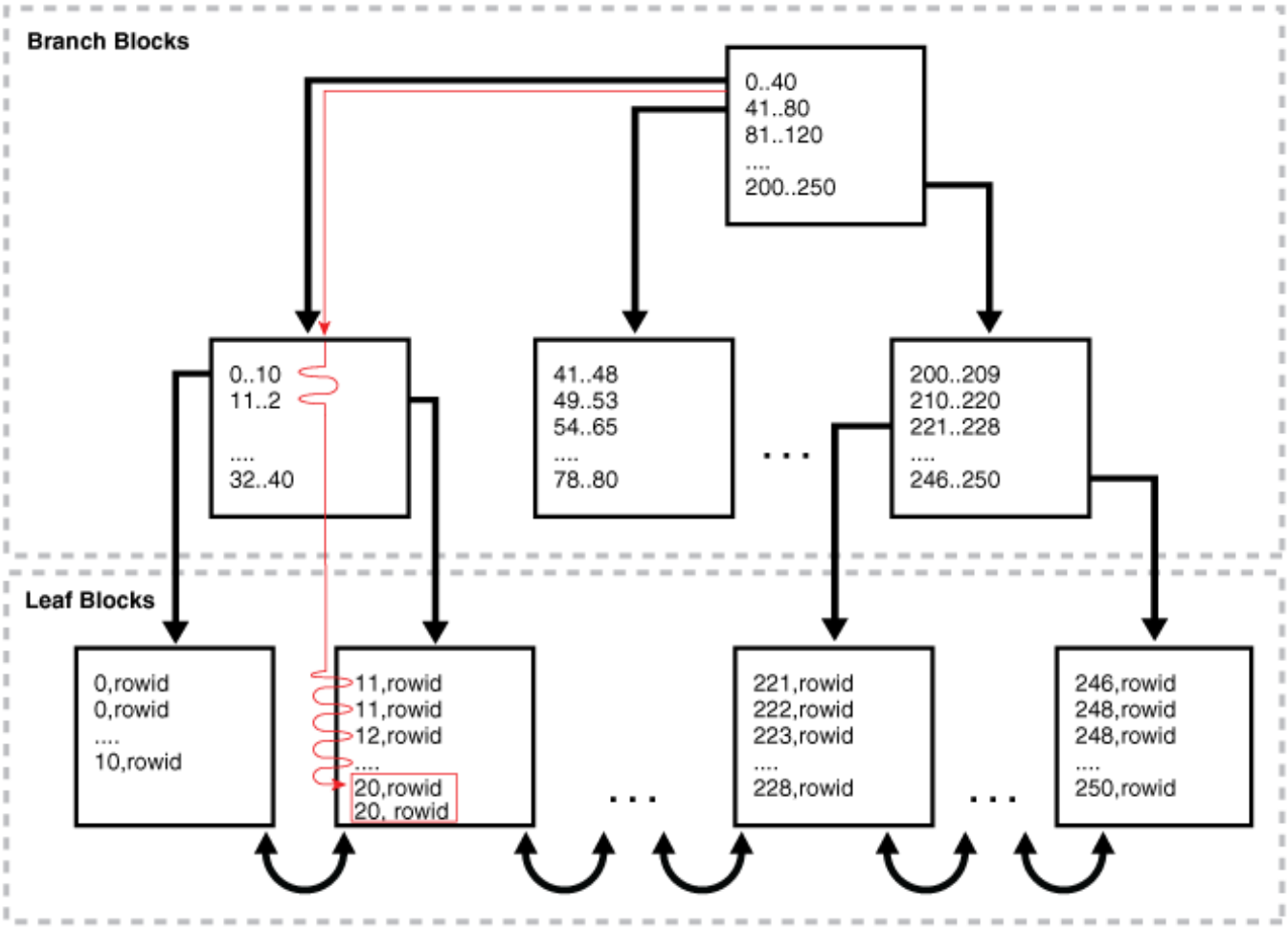
Respuesta:

La tabla contiene valores nulos.

5.3.4.2. Index Range scans.

- Realiza un escaneo de valores ordenados a un índice B-tree
- Por default, los valores en un índice se almacenan de forma ascendente y se escanean en el mismo orden.
- Se emplea en casos como:
  - Predicados en la columna indexada con operadores >=, <=, etc.
  - El índice es `non unique`. Significa que pueden obtenerse más de un valor.

En la siguiente imagen se muestra el funcionamiento de un escaneo del índice por rango. Observar que el escaneo puede realizarse de forma horizontal empleando los punteros horizontales entre los nodos hojas. Por ejemplo, se escanean todos los ROW_IDs entre 20 y 40.



Ejemplo:

```
explain plan for
select email
from paciente
where email  like 'bob@%'
order by email desc;

select plan_table_output
from table(dbms_xplan.display);
```

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------
------------------------------------------------
Plan hash value: 3471813995

-------------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                   |     1 |   102 |     1   (0)| 00:00:01 |
|*  1 |  INDEX RANGE SCAN DESCENDING| PACIENTE_EMAIL_IDX |     1 |   102 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
------------------------------------------------

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------
------------------------------------------------

   1 - access("EMAIL" LIKE 'bob@%')
       filter("EMAIL" LIKE 'bob@%')
```

Ejemplo:

- ¿Por qué razón no se emplea índice?

```
create index paciente_nombre_idx
on paciente(nombre);

explain plan for
select nombre
from paciente
order by paciente_id desc;

select plan_table_output
from table(dbms_xplan.display);
```

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------
------------------------------------------------
Plan hash value: 375339519

-------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          | 16584 |   566K|       |   635   (1)| 00:00:01 |
|   1 |  SORT ORDER BY     |          | 16584 |   566K|   728K|   635   (1)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| PACIENTE | 16584 |   566K|       |   477   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------
```

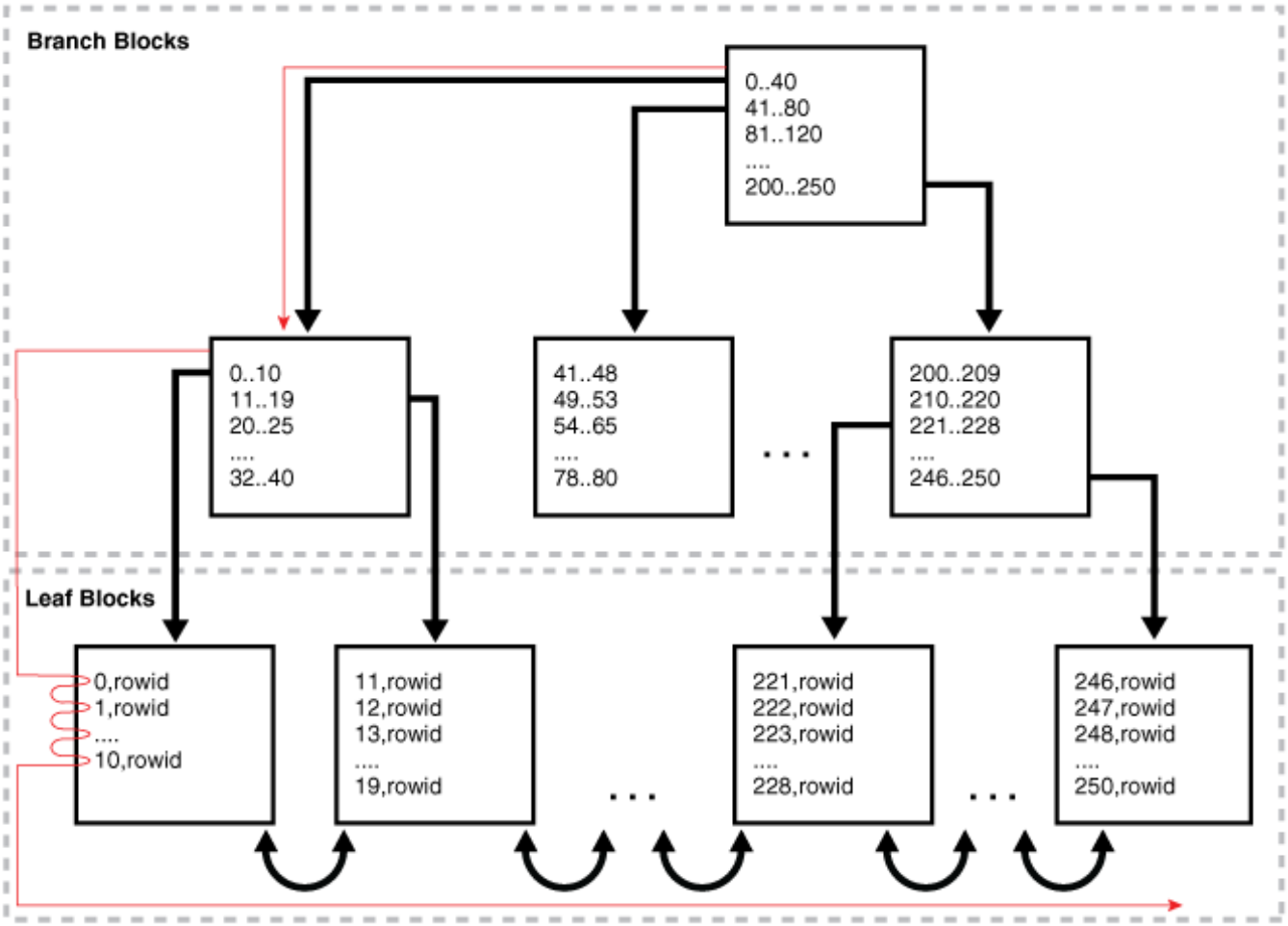
Respuesta:
La condición de ordenamiento involucra a un campo diferente al de la condición.

5.3.4.3. Index Full scan

- Se realiza la lectura completa del índice de forma ordenada.
- Empleado en casos donde el predicado hace referencia a alguna de las columnas de un índice compuesto, dicha columna no debe ser la primera columna especificada en el índice.
- No existe predicado, pero existen las siguientes condiciones:

- Todas las columnas solicitadas por la consulta están en el índice
- Al menos una de las columnas indexadas está declarada como NOT NULL

- La sentencia incluye ORDER BY sobre una columna indexada.



Ejemplo:

```
explain plan for
select nombre
from paciente
order by nombre;

select plan_table_output
from table(dbms_xplan.display);
```

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------------------------
Plan hash value: 898347700

-------------------------------------------------------------------------------
| Id  | Operation        | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT |                     | 16584 |   356K|    45   (0)| 00:00:01 |
|   1 |  INDEX FULL SCAN | PACIENTE_NOMBRE_IDX | 16584 |   356K|    45   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

5.3.4.4. Index fast full scan

- En este caso se leen todos los bloques del índice de forma desordenada, tal cual se almacenaron en disco.
- Este tipo de índice se emplea en consultas donde se seleccionan únicamente columnas contenidas en el índice.

Físicamente, los bloques que integran al índice no necesariamente están ordenados dentro del segmento asociado al índice

Ejemplo:

```
explain plan for
select email
from paciente
where email is not null;

select plan_table_output
from table(dbms_xplan.display);
```

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------
--------------------------------------------------
Plan hash value: 3687434030

-------------------------------------------------------------------------------------------------
| Id  | Operation            | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |                    | 14438 |  1438K|    20   (0)| 00:00:01 |
|*  1 |  INDEX FAST FULL SCAN| PACIENTE_EMAIL_IDX | 14438 |  1438K|    20   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------
```

Ejemplo:

```
explain plan for
select count(*)
from paciente;

select plan_table_output
from table(dbms_xplan.display);
```

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
----------------------------------------------
Plan hash value: 2895057998

--------------------------------------------------------------------------------
| Id  | Operation             | Name        | Rows  | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |             |     1 |     9   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE       |             |     1 |            |          |
|   2 |   INDEX FAST FULL SCAN| PACIENTE_PK | 15000 |     9   (0)| 00:00:01 |
--------------------------------------------------------------------------------
```

5.3.4.5. Index skip Scan

- Empleado en casos con índices compuestos.

- La primera columna tiene muy pocos valores diferentes.
- La segunda columna tiene una gran cantidad de valores distintos.
- La consulta no incluye en el predicado a la primera columna.

Ejemplo:

Suponer un índice compuesto con las columnas de la tabla paciente genero e email:

```
. . .
F,Wolf@company.example.com,rowid
F,Wolsey@company.example.com,rowid
F,Wood@company.example.com,rowid
F,Woodman@company.example.com,rowid
F,Yang@company.example.com,rowid
F,Zimmerman@company.example.com,rowid
M,Abbassi@company.example.com,rowid
M,Abbey@company.example.com,rowid
. . .
```

El manejador procesa la consulta creando 2 sub-índices de manera lógica. Un índice para procesar los valores ‘F’ y otro para los valores ‘M’

```
( select *
  from   paciente
  where  genero = 'F'
  and    email = 'abbey@company.com' )
union all
( select *
  from   paciente
  where  genero = 'M'
  and    email = 'abbey@company.com' )
```

Plan de ejecución:

```
create index paciente_email_genero_idx
on paciente(genero,email);

explain plan for
select *
from paciente
where email='smith@mail.com';

select plan_table_output
from table(dbms_xplan.display);
```

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------
Plan hash value: 3560863078

------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name                       | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                            |     1 |   773 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PACIENTE                   |     1 |   773 |     4   (0)| 00:00:01 |
|*  2 |   INDEX SKIP SCAN                   | PACIENTE_EMAIL_GENERO_IDX  |     1 |       |     3   (0)| 00:00:01 |
------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------
------------------------------------------------

   2 - access("EMAIL"='smith@mail.com')
       filter("EMAIL"='smith@mail.com')
```

5.3.4.6. Index Join Scan

- Significa realizar un Hash Join entre los resultados obtenidos al leer N índices.
- Empleado generalmente cuando las columnas solicitadas por las columnas están contenidas en los índices, no se requiere hacer acceso a los registros de las tablas. Es decir, se hace join entre índices.

Ejemplo:

```
create index paciente_nombre_idx
on paciente(nombre);

create index paciente_ap_paterno_idx
on paciente(ap_paterno);

explain plan for
select nombre,ap_paterno
from paciente
where nombre like 'A%';

select plan_table_output
from table(dbms_xplan.display);
```

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------
----------------------------------------------------
Plan hash value: 3337046678

-------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |                         |   422 |  6330 |    42   (0)| 00:00:01 |
|*  1 |  VIEW                  | index$_join$_001        |   422 |  6330 |    42   (0)| 00:00:01 |
|*  2 |   HASH JOIN            |                         |       |       |            |          |
|*  3 |    INDEX RANGE SCAN    | PACIENTE_NOMBRE_IDX     |   422 |  6330 |     3   (0)| 00:00:01 |
|   4 |    INDEX FAST FULL SCAN| PACIENTE_AP_PATERNO_IDX |   422 |  6330 |    49   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------
----------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("NOMBRE" LIKE 'A%')
   2 - access(ROWID=ROWID)
   3 - access("NOMBRE" LIKE 'A%')
```

Resolver Serie del tema parte 1.