



## 5. OPTIMIZACIÓN DE CONSULTAS DISTRIBUIDAS - PARTE 2

<b>5. OPTIMIZACIÓN DE CONSULTAS DISTRIBUIDAS - PARTE 2</b>	<b>1</b>
5.3.5. Joins	1
5.3.5.1. Nested Loops Joins.	2
Ejemplo 1	3
Solución	4
Ejemplo 2	8
Ejemplo 3	11
5.3.5.2. Hash Joins.	14
Algoritmo:	14
Ejemplo	15
5.3.5.3. Sort Merge Join.	18
Algoritmo de un Sort Merge Join:	19
Ejemplo:	19
Ejemplo:	20
5.3.5.4. Cartesian Join.	21
Algoritmo:	21
Ejemplo:	21
5.3.5.5. Métodos para procesar un outer join.	22
5.3.5.5. Métodos adicionales para optimizar joins.	22

### 5.3.5. Joins

Un Join combina la salida de 2 fuentes de datos llamadas “Row Sources”. Obtiene como salida otro “Row Source” llamado “Data Set”.

La condición del Join permite realizar la comparación de las 2 fuentes de datos. De no especificar dicha condición se genera producto cartesiano estableciendo una correspondencia de cada uno de los registros de un *rowsource* con cada uno de los registros del otro.

Cuando existen múltiples tablas en una consulta, el optimizador debe tomar en consideración diversos aspectos para determinar las operaciones más eficientes para cada pareja de “Row sources”:

- Access Paths (métodos de acceso).

- Join Methods: Para realizar la comparación o ejecución del join el optimizador aplica alguno de los métodos disponibles. Es decir, en esta etapa se define el método a emplear para obtener el resultado de un join entre 2 "Row sources":
  - Nested loop
  - Sort Merge
  - Hash Join
  - Cartesian Join.
- Join Types
  - Outer
  - Inner
- Join order: Empleado especialmente con sentencias que contienen más de 2 tablas. El orden en el que se realizan las operaciones de Joins es importante.

Con todas estas variantes el optimizador genera diversos planes de ejecución seleccionando aquel que tenga el menor costo. A nivel general, el optimizador considera los siguientes aspectos:

- Revisa si el resultado de un Join contiene a lo más 1 registro.
  - Para ello revisa la existencia de constraints **unique** Y **primary key**.
  - Si esto ocurre, el join se ejecuta al principio.

Antes de iniciar con la revisión de los distintos métodos que existen para implementar una operación join es importante considerar los siguientes 2 conceptos:

Cada método define 2 conceptos principales:

- Una de las fuentes de datos se selecciona como **driving table** o **outer table**.
- La otra fuente de datos se le conoce como **driven table** o **inner table**.

Lo anterior es análogo a un ciclo for anidado:

```
for (int i =0 :i<n; i++) {    => driving table
    for(int j =0;j<k;j++) { => driven table
        . . . .
    }
}
```

#### 5.3.5.1. Nested Loops Joins.

- En este tipo de Join, para cada registro de la tabla Outer se buscan correspondencias con N registros de la tabla Inner.
- Para cada registro de la tabla Outer que cumpla con el predicado de la operación join, la BD obtiene todos los registros que satisfacen el predicado.
- Si existe un índice que permita recuperar los datos de la tabla Inner, el manejador puede emplearlo a través de rowids (Aplica solo para los datos de la tabla Inner).

- En un plan de ejecución un **NESTED LOOP** aparece de la siguiente manera:

```
NESTED LOOP
  Outer_table
  Inner_table
```

¿En qué situaciones es conveniente el uso de un Nested Join?

- Las fuentes de datos tienen pocos registros.
- La condición del Join permite acceder a los datos en la Inner table de forma eficiente. Por ejemplo: Suponer una operación join entre **diagnostico** y **cita**, su predicado es **d.diagnostico\_id = c.diagnostico\_id**.
- **diagnostico** será seleccionada como outer table al detectar una menor cantidad de registros con respecto a cita.
- **cita** podría ser seleccionada como la tabla inner. Sin embargo, ¿Qué sucedería si la llave foránea en **cita c.diagnostico\_id** está indexada?
- Si esto ocurre, el optimizador puede seleccionar al índice como tabla inner en lugar de usar a **cita**.
- Esto es factible ya que para verificar correspondencias entre las tablas basta con tener los valores de **diagnostico\_id** de cada conjunto de datos con sus respectivos rowids.
- De lo anterior, es importante mencionar que la inner table no siempre corresponde con una tabla existente en la BD. Tanto la outer como la inner table pueden ser índices.
- En general este método es adecuado para Joins con pocos registros en las tablas fuente en el que las columnas que participan están indexadas.
- Existe un umbral interno que puede influenciar al optimizador el uso de un Nested Loop. Si el número de registros en la tabla Outer no excede el umbral, el optimizador puede considerar Nested Loop. De lo contrario, Hash Join podría ser la alternativa.

#### Ejemplo 1

Obtener la fecha de las citas cuya clave de diagnóstico inicie con **A85**. Suponer las siguientes condiciones:

- Existencia del siguiente índice:

```
create index cita_diagnostico_id_ix on cita(diagnostico_id);
```

- El DBMS decide usar la técnica de Nested Loop haciendo uso de los índices a medida de lo posible.
- Para efectos del ejemplo, considerar las siguientes muestras de datos:

cita				
c.row_id	c.fecha_cita	cita_id	medico_id	diagnostico_id
0001	10/05/2013	1	45	411
0004	23/04/2015	2	32	412
0002	14/08/2003	3	23	412
0003	20/02/1988	4	125	411
0005	30/09/2001	5	4	413
0008	22/07/2006	6	9	304
0009	10/11/2004	7	13	396
0006	14/14/2001	8	23	987

diagnostico			
Row_id	d.diagnostico_id	d.clave	Nombre
01	409	A90	Diarrea
02	410	A91	Migraña
03	411	A851	Obesidad
04	412	A857	Leucemia
05	413	A854	Depresión
06	414	A852	Estrés
07	415	A853	Estreñimiento

- Generar la sentencia SQL encargada de construir el plan de ejecución
- Determine el método de acceso y la fuente de datos que producirá a la outer table
- Considerando la muestra de datos anterior, generar la tabla de datos que representa a la tabla Outer.
- Determinar la fuente de datos que producirá a la tabla inner.
- Considerando la muestra de datos anterior, generar la tabla de datos que representa a la tabla inner.
- Determinar la expresión booleana que se aplicará a cada registro de la tabla inner, así como el método de acceso que se empleará sobre la tabla inner para recuperar los datos requeridos.
- Determinar el resultado del Nested loop.
- Determinar la siguiente operación del plan de ejecución para mostrar el resultado final de la consulta.

#### Solución

- Sentencia SQL que genera el plan de ejecución

```
explain plan for
select d.diagnostico_id, c.fecha_cita
from cita c
join diagnostico d on c.diagnostico_id = d.diagnostico_id
where d.clave like 'A85%';
```

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

#### B. Tabla outer.

- El optimizador trata de aplicar posibles operaciones de selección ( $\sigma_p$ ) para reducir las cardinalidades de las tablas:
  - Para la tabla **diagnostico** existe el predicado **clave like 'A85%'**. Debido a que el campo **clave** no está indexado, la única opción es realizar un table access full. De la muestra de datos se obtendrán 5 registros que serán considerados para realizar el join.
  - Para la tabla **cita** no existe algún predicado por lo que los 8 registros participarán para realizar el join.
- Para ejecutar el nested loop se requieren los siguientes datos:
  - d.diagnostico\_id** el cual ya se tiene disponible ya que se hizo un table Access full en el punto anterior.
  - c.diagnostico\_id** La forma más rápida de acceder a este campo es haciendo uso del índice **cita\_diagnostico\_id\_ix**
- De lo anterior, el optimizador elige a la tabla **diagnostico** como tabla outer ya que tiene solo 5 registros.

#### C. Tabla outer con datos.

De la tabla **diagnostico** solo se selecciona la columna **diagnostico\_id** con los 5 identificadores que se requieren para ejecutar el join. Notar que no se necesita ninguna otra columna.

d.diagnostico_id
411
412
413
414
415

#### D. Tabla inner.

- La tabla inner estará representada por el índice **cita\_diagnostico\_id\_ix**, ya que representa la forma más eficiente de acceder a los valores de **c.diagnostico\_id**
- Notar que tanto la tabla inner como la outer puede estar representada por índices, no solo por tablas.

#### E. Tabla inner con datos:

cita_diagnostico_ix	
c.diagnostico_id	row_id
411	0001
412	0004
412	0002
411	0003
413	0005
304	0008
396	0009
987	0006

#### F. Expresión booleana

```
d.diagnostico_id = c.diagnostico_id
```

Por cada registro de la tabla outer (**d.diagnostico\_id**) se hará un `index range scan` al índice **cita\_diagnostico\_ix** para obtener una lista de rowids, es decir:

Iteración 1: `index range scan` para obtener todos los rowids donde **d.diagnostico\_id = 411**

- **L = {0001, 0003}**

Iteración 2: `index range scan` para obtener todos los rowids donde **d.diagnostico\_id = 412**

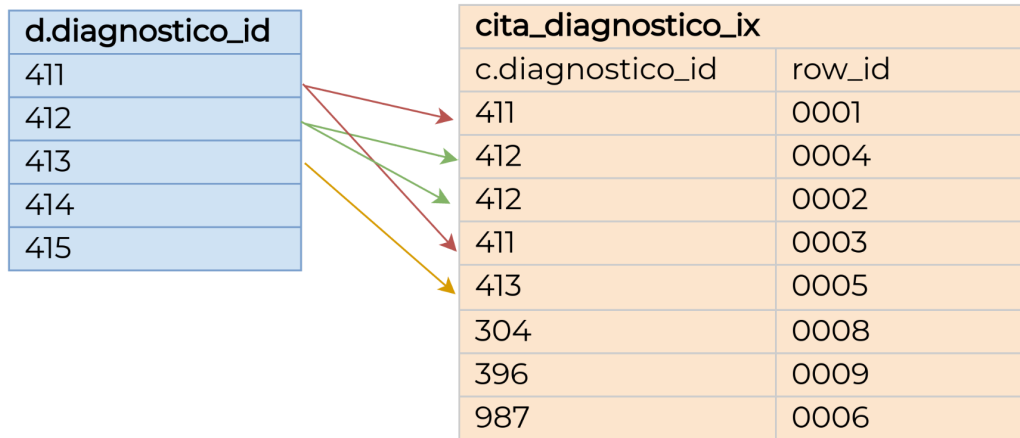
- **L = {0002, 0004}**

Iteración 3: `index range scan` para obtener todos los rowids donde **d.diagnostico\_id = 413**

- **L = {0005}**

En las iteraciones restantes no se obtienen correspondencias.

Se requiere un *index range scan* ya que por cada valor de **d.diagnostico\_id** se pueden obtener varias correspondencias en el índice



G. Resultado del nested loop.

diagnostico	cita_diagnostico_ix
d.diagnostico_id	row_id
411	0001
411	0003
412	0002
412	0004
413	0005

H. Finalmente, en esta operación, empleando los ROWIDs de la tabla anterior, se realizará un *table Access by index rowid* para recuperar el campo **fecha\_cita** en la tabla **cita**.

diagnostico	cita_diagnostico_ix
d.diagnostico_id	row_id
411	0001
411	0003
412	0002
412	0004
413	0005

cita				
c.row_id	c.fecha_cita	cita_id	medico_id	diagnostico_id
0001	10/05/2013	1	45	411
0004	23/04/2015	2	32	412
0002	14/08/2003	3	23	412
0003	20/02/1988	4	125	411
0005	30/09/2001	5	4	413
0008	22/07/2006	6	9	304
0009	10/11/2004	7	13	396
0006	14/14/2001	8	23	987

El resultado final será:

diagnostico	cita
d.diagnostico_id	c.fecha_cita
411	10/05/2013
411	23/04/2015
412	14/08/2003
412	20/02/1988
413	30/09/2001

En Oracle, este último paso se representa como un segundo nested loop:

Plan hash value: 3773197897

Id	Operation	Name	Rows	Bytes	Cost	(%CPU)	Time
0	SELECT STATEMENT		3	282	37	(0)	00:00:01
1	NESTED LOOPS		3	282	37	(0)	00:00:01
2	NESTED LOOPS		3	282	37	(0)	00:00:01
* 3	TABLE ACCESS FULL	DIAGNOSTICO	1	25	34	(0)	00:00:01
* 4	INDEX RANGE SCAN	CITA_DIAGNOSTICO_ID_IX	2		1	(0)	00:00:01
5	TABLE ACCESS BY INDEX ROWID	CITA	2	138	3	(0)	00:00:01

Predicate Information (identified by operation id):

```

3 - filter("D"."CLAVE" LIKE 'A85%')
4 - access("C"."DIAGNOSTICO_ID"="D"."DIAGNOSTICO_ID")

```

- Observar que el segundo nested loop con id = 1, la tabla outer corresponde con el resultado del primer nested loop.
- Por cada rowid contenido en la tabla outer, se realizará la operación table access by index rowid y se recupera el valor de la columna **fecha\_cita**, empleando a la tabla **cita** como tabla inner (paso 5).

## Ejemplo 2

- Considerar nuevamente la consulta anterior.
- ¿Qué efectos tendrá el plan de ejecución si se elimina el índice **cita\_diagnostico\_id\_ix** y se agrega la condición **c.medico\_id = 2286**?

```

drop index cita_diagnostico_id_ix;

explain plan for
select d.diagnostico_id, c.fecha_cita
from cita c
join diagnostico d on c.diagnostico_id = d.diagnostico_id
where d.clave like 'A85%'

```



```
and c.medico_id = 2286;

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

Plan hash value: 4020497077

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	64	37 (0)	00:00:01
1	NESTED LOOPS		1	64	37 (0)	00:00:01
2	NESTED LOOPS		7	64	37 (0)	00:00:01
* 3	TABLE ACCESS FULL	CITA	7	273	30 (0)	00:00:01
* 4	INDEX UNIQUE SCAN	DIAGNOSTICO_PK	1		0 (0)	00:00:01
* 5	TABLE ACCESS BY INDEX ROWID	DIAGNOSTICO	1	25	1 (0)	00:00:01

Predicate Information (identified by operation id):

```
3 - filter("C"."MEDICO_ID"=2286)
4 - access("C"."DIAGNOSTICO_ID"="D"."DIAGNOSTICO_ID")
5 - filter("D"."CLAVE" LIKE 'A85%')
```

- En este ejemplo solo se cuenta con los índices de las PKs de ambas tablas. El manejador hará lo posible por emplearlos.
- Para ejecutar el predicado `c.medico_id = 2286` la única opción es Table Access Full. Se estiman 7 registros.
- Debido a que se realizó un table access full, se cuenta con todos los valores de las columnas de la tabla cita. Uno de estos atributos es `c.diagnostico_id` cuyos valores corresponden con la PK de diagnóstico y ahí si existe un índice.
- Para cada uno de estos 7 valores, el manejador podría acceder al índice de la PK en la tabla `diagnostico` evitando así otro table access full.
- Para poder hacer esto, el manejador deberá elegir como tabla outer a cita ya que, para cada uno de los 7 valores, se hará un unique index scan al índice `diagnostico_pk`.
- Observar que en este caso no se respetó la regla de seleccionar a la tabla con el menor número de registros como tabla outer.
- Este proceso se muestra a detalle con la siguiente muestra de datos:

A. **id 3:** El optimizador elige a cita como la tabla outer. La tabla outer contendrá `fecha_cita` y `diagnostico_id` para poder ejecutar el Nested Loop. Solo se muestran 4 registros por simplicidad. Se agrega el campo `fecha_cita` ya que fue solicitado en la consulta y se encuentra disponible al haber realizado el table access full.

cita	
c.fecha_cita	c.diagnostico_id
10/05/2013	3
23/04/2015	4
14/08/2003	5
20/02/1988	6

B. **Id 4:** La tabla Inner está representada por el índice **diagnostico\_pk**.

- La tabla **cita** fue seleccionada como tabla outer.
- Al ejecutar el Nested Loop, para cada valor de **cita.diagnostico\_id** se buscarán correspondencias con la tabla **diagnostico**. El campo **diagnostico\_id** representa la PK de **diagnostico** y tiene un índice único asociado. El manejador detecta esta condición y decide emplear el índice **diagnostico\_pk** empleando un *unique index scan*.

cita		diagnostico_pk	
c.Fecha_cita	c.diagnostico_id	diagnostico_id	Row_id
10/05/2013	3	3	0003
23/04/2015	4	4	0004
14/08/2003	5	5	0005
20/02/1988	6	6	0006

C. **Id 2:** El resultado del Nested Loop es:

cita		diagnostico_pk
c.Fecha_cita	c.diagnostico_id	d.Row_id
10/05/2013	3	0003
23/04/2015	4	0004
14/08/2003	5	0005
20/02/1988	6	0006

- D. **Id 1:** En el siguiente Nested Loop, la tabla outer es representada por la tabla anterior.
- E. **Id 5:** La tabla inner es representada por **diagnostico**. Por cada registro de la tabla outer, se emplea el rowid para localizar a los registros de **diagnostico** empleando la operación *table access by index rowid*. En este mismo paso, al recuperar a cada registro se le aplica el filtro **d.clave like 'A85%'**

cita		diagnostico_pk	diagnostico	
c.Fecha_cita	c.diagnostico_id	d.Row_id	d.Row_id	d.Clave
10/05/2013	3	0003	0003	A857
23/04/2015	4	0004	0004	A868
14/08/2003	5	0005	0005	A859
20/02/1988	6	0006	0006	A867

F. Finalmente, el resultado de la consulta considerando esta muestra ficticia será:

cita	diagnostico
c.Fecha_cita	d.diagnostico_id
10/05/2013	3
20/02/1988	6

Haber quitado el índice `cita_diagnostico_id_idx` provocó el cambio de elección de la tabla outer.

### Ejemplo 3

Obtener el nombre de los médicos cuyo nombre inicie con A y la fecha de sus citas a realizarse en el consultorio A03. La consulta hace un uso de un **hint** indicado entre `/*+ */` para forzar la selección de la tabla inner. Observar el plan de ejecución generado y explicar cada actividad.

```
create index cita_medico_idx on cita(medico_id);

explain plan for
select /*+ ORDERED USE_NL(c) */ m.nombre,c.fecha_cita
from medico m, cita c
where c.medico_id = m.medico_id
and m.nombre like 'A%'
and c.consultorio = 'A03'
```

## PLAN\_TABLE\_OUTPUT

Plan hash value: 4174180984

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		20	640	933 (0)	00:00:01
1	NESTED LOOPS		20	640	933 (0)	00:00:01
2	NESTED LOOPS		564	640	933 (0)	00:00:01
* 3	TABLE ACCESS FULL	MEDICO	141	1551	238 (0)	00:00:01
* 4	INDEX RANGE SCAN	CITA_MEDICO_IDX	4		1 (0)	00:00:01
* 5	TABLE ACCESS BY INDEX ROWID	CITA	1	21	5 (0)	00:00:01

## PLAN\_TABLE\_OUTPUT

Predicate Information (identified by operation id):

- 3 - filter("M"."NOMBRE" LIKE 'A%')
- 4 - access("C"."MEDICO\_ID"="M"."MEDICO\_ID")
- 5 - filter("C"."CONSULTORIO"='A03')

- A. Se ejecuta el paso 3 para recuperar a todos los médicos con nombre iniciando en A (Tabla Outer). Notar que se hace uso de un **Hint** para forzar a que se realice un Nested Loop con la tabla Cita c como tabla inner. Al existir un índice en **cita.diagnostico\_id**, el manejador decide usarlo. El efecto de hint es que se selecciona al índice como inner en lugar de haber seleccionado a **cita** como outer. De lo anterior, la tabla Outer podría contener los siguientes datos. Por simplicidad se muestran 5 registros

## MEDICO

m.medico_id	Nombre
1	Ale
2	Amalia
3	Aurora
4	Aldo

- B. Con base al plan, se harán 141 iteraciones, en cada una de ellas se ejecuta el paso 4 empleando como tabla inner al índice **cita\_medico\_idx**. Por cada iteración se aplicará un index range scan, se obtienen los ROWIDs que cumplan la condición **c.medico\_id(inner) = m.medico\_id (outer)**. El optimizador estima que por cada iteración se obtendrán 4 registros, en total se tendrán  $141 \times 4 = 564$  registros al terminar de procesar el NESTED JOIN del paso 2.

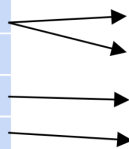
Suponer los siguientes datos:

MEDICO

m.medico_id	Nombre
1	Ale
2	Amalia
3	Aurora
4	Aldo

CITA\_MEDICO\_IX

c.medico_id	c.Row_id
1	001
1	002
3	003
4	004



El resultado del primer NESTED LOOP será

NL

m.medico_id	m.nombre	c.rowid
1	Ale	001
1	Ale	002
3	Aurora	003
4	Aldo	004

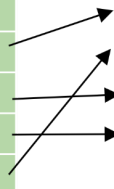
- C. El resultado del paso anterior se convierte en la tabla outer del segundo nested join (paso 1). La tabla Inner ahora estará representada por la tabla cita. Por cada registro de la tabla outer se realizará un acceso a la tabla empleando table access by index rowid haciendo uso del campo **c.rowid** de la tabla outer:

NL

m.medico_id	m.nombre	c.row_id
1	Ale	001
1	Ale	002
3	Aurora	003
4	Aldo	004

CITA

c.row_id	c.fecha_cita
0001	10/05/2013
0004	23/04/2015
0002	14/08/2003
0003	20/02/1988
0005	30/09/2001



- D. Del segundo nested join, se realizaron 564 iteraciones. Observar que el optimizador solo espera 20 registros como resultado de esta operación (paso 1). Lo anterior significa que solo se obtuvieron 20 registros de la tabla **cita** (tabla Inner), es decir, solo 20 citas se realizaron en el consultorio **A03**. El resultado se verá así:

RESULTADO

m.medico_id	m.nombre	c.fecha_cita
1	Ale	10/05/2013
1	Ale	14/08/2003
3	Aurora	20/02/1988
4	Aldo	23/04/2015

Como conclusión, al forzar esta estrategia con un Hint, el ejemplo resultó ser no tan óptimo, generó un costo total de 933. Una mejor solución sería dejar que el manejador elija el mejor plan:

Observar que el costo disminuye a 640. La principal diferencia aquí, es que el filtro consultorio='a03' se hace desde el primer paso (paso 3), lo que genera una tabla outer pequeña de 20 registros, en este caso no se emplea el índice de la FK.

#### PLAN\_TABLE\_OUTPUT

Plan hash value: 2487834737

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		20	640	54 (0)	00:00:01
1	NESTED LOOPS		20	640	54 (0)	00:00:01
2	NESTED LOOPS		20	640	54 (0)	00:00:01
* 3	TABLE ACCESS FULL	CITA	20	420	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	11	1 (0)	00:00:01

#### PLAN\_TABLE\_OUTPUT

Predicate Information (identified by operation id):

```

3 - filter("C"."CONSULTORIO"='A03')
4 - access("C"."MEDICO_ID"="M"."MEDICO_ID")
5 - filter("M"."NOMBRE" LIKE 'A%')

```

#### 5.3.5.2. Hash Joins.

- Se emplean para realizar Joins con conjuntos grandes de datos.
- La condición del JOIN es la igualdad (equi-join).
- El optimizador selecciona al Source Row de menor tamaño para construir un hash table.
- La BD hace uso del HashTable para localizar las correspondencias con la otra tabla.
- Esta técnica resulta eficiente siempre y cuando el row source menor pueda ser cargado completamente en memoria.
- Si los datos no caben en memoria, se aplica un “particionamiento” del row source, incrementan las operaciones I/O especialmente en el tablespace Temporal. Parte del hashTable se almacena en memoria y la otra parte en disco.

Algoritmo:

1. Típicamente la BD hace un full scan del row source menor llamada “Build Table”, aplica una función hash a cada uno de los valores de la columna empleada como condición del join. Se construye el Hash Table y se guarda en la PGA (se guardan todas las columnas de la tabla en el hashTable).

```
for small_table_row in (select * from small_table)
loop
  slot_number := hash(small_table_row.join_key);
  insert_hash_table(slot_number,small_table_row);
end loop;
```

2. La tabla que no fue seleccionada para crear el hash table se le conoce como “probe table”. Básicamente, para cada valor de la columna que participa en la condición del Join se le aplica la función Hash. El resultado es empleado para localizar las correspondencias en el HashTable.

### Ejemplo

Obtener el nombre del médico y el nombre de su especialidad para todos los registros existentes en la base de datos.

- A. Generar la sentencia SQL que genera el plan de ejecución
- B. Describir el algoritmo considerando los siguientes datos asumiendo que el manejador selecciona la técnica de la tabla hash.

### ESPECIALIDAD

Especialidad_id	nombre	anios	requisito
1	Anatomía Patológica	4	Medicina general
2	Anestesiología	3	Medicina general
3	Cardiología	1	Medicina general

### MEDICO

medico_id	nombre	especialidad_id
1	Ale	3
2	Amalia	3
3	Aurora	2
4	Aldo	2
5	Felipe	3
6	Laura	2

- A. Sentencia SQL que genera el plan de ejecución.

```

explain plan for
select m.nombre,e.nombre
from especialidad e
join medico m
on m.especialidad_id = e.especialidad_id;

select plan_table_output
from table(dbms_xplan.display);

```

```

PLAN_TABLE_OUTPUT
-----
Plan hash value: 2505893939

-----
| Id | Operation                | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-----
|  0 | SELECT STATEMENT         |                | 5000  | 166K  | 241  (0)   | 00:00:01 |
|*  1 |   HASH JOIN              |                | 5000  | 166K  | 241  (0)   | 00:00:01 |
|  2 |     TABLE ACCESS FULL   | ESPECIALIDAD   | 53    | 1272  | 3    (0)   | 00:00:01 |
|  3 |     TABLE ACCESS FULL   | MEDICO         | 5000  | 50000 | 238  (0)   | 00:00:01 |
-----

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

   1 - access("M"."ESPECIALIDAD_ID"="E"."ESPECIALIDAD_ID")

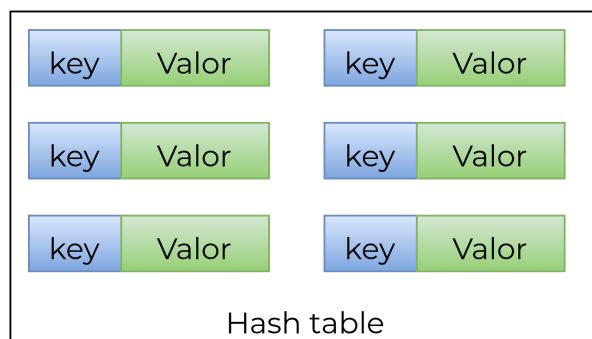
```

## B. Algoritmo.

- De la imagen se puede observar que efectivamente se selecciona a la tabla **especialidad** como **build table**. Con ella se construirá la HashTable. Su construcción se puede representar de la siguiente manera. Notar que el “valor” de cada pareja contiene al registro completo.

key	value			
ora_hash(especialidad_id)	especialidad_id	nombre	anios	requisito
2342552567	1	Anatomía Patológica	4	Medicina general
2064090006	2	Anestesiología	3	Medicina general
2706503459	3	Cardiología	1	Medicina general





Para generar hash codes se puede emplear la función `ora_hash`. Notar que el parámetro que se le pasa a la función `ora_hash` corresponde con los valores de la columna que se usa en el predicado de la operación join, `especialidad_id`.

```
col nombre format A30
col requisito format A30

select  especialidad_id,nombre,anios,requisito,ora_hash(especialidad_id)
from    especialidad
where   rownum <5;
```

Al ejecutar la sentencia se obtiene lo siguiente:

ESPECIALIDAD_ID	NOMBRE	ANIOS	REQUISITO	ORA_HASH(ESPECIALIDAD_ID)
1	Anatomia Patologica	4	Medicina General	2342552567
2	Anestesiologia y Recuperacion	3	Medicina General	2064090006
3	Anestesiologia Pediatrica	1	Especialista en Anestesiologia	2706503459
4	Cardiologia	4	Medicina General	3217185531

- Una vez que el hash table se ha construido, se comienza a iterar a la **probe table** que en este caso es representada por la tabla `medico`.
- Por cada registro de la tabla `medico`, se realizará una búsqueda en el HashTable empleando el valor del atributo `m.especialidad_id`

En SQL sería algo similar a la siguiente instrucción:

```
select hash_table.search(ora_hash(m.especialidad_id))
from medico;
```

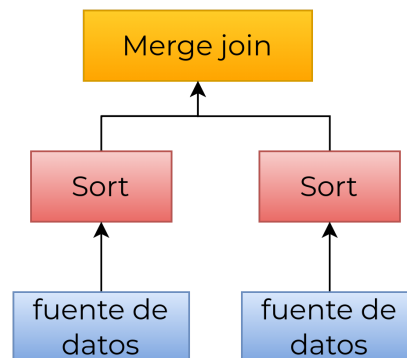
Observar que al campo `m.especialidad_id` se le aplica la función `ora_hash`. El código hash generado se emplea como llave para localizar correspondencias en el hash table.

Finalmente, se unen las columnas con base a las correspondencias anteriores y se obtiene el resultado.

Medico	Especialidad
m.nombre	e.nombre
Ale	Cardiología
Amalia	Cardiología
Aurora	Anestesiología
Aldo	Anestesiología
Felipe	Cardiología
Laura	Anestesiología

### 5.3.5.3. Sort Merge Join.

- Representa una variación de un Nested loop Join.
- La BD realiza un ordenamiento de las 2 fuentes de datos en caso de que estas no estén ya ordenadas (SORT JOIN).
- Por cada registro en la primera fuente de datos, se verifica si existe correspondencia con la segunda fuente de datos (MERGE JOIN).
- Esta operación se realiza de manera secuencial debido a que ambas fuentes de datos están ordenadas.



Esta estrategia puede emplearse bajo las siguientes condiciones:

- Se emplea en lugar de un Hash Join para procesar grandes cantidades de datos.
- La condición del join no es una igualdad, por ejemplo: `>=`, `>`, `<=`, `<`
- En un Hash Join siempre se requiere la igualdad “=”. Por lo tanto, esta técnica no se podría emplear con un “non-equi-join”.
- Si existe un índice, el manejador puede omitir el ordenamiento del primer conjunto de datos. El segundo conjunto de datos siempre se ordena.
- Esta técnica puede ser mejor que un Hash Join una vez que se cuenta con los datos ordenados, de no existir, el ordenamiento es más costoso que la construcción de una tabla hash.
- Si los datos no caben en memoria para construir un hash table, Sort Merge Join es la mejor opción.

En caso de que la memoria no sea suficiente para almacenar a los 2 conjuntos de datos, también se emplea disco, sin embargo, el número de lecturas es menor a las requeridas por un Hash Join.

#### Algoritmo de un Sort Merge Join:

```
read data_set_1 sort by join key to temp_ds1
read data_set_2 sort by join key to temp_ds2

read ds1_row from temp_ds1
read ds2_row from temp_ds2
while not eof on temp_ds1,temp_ds2
loop
  if ( temp_ds1.key = temp_ds2.key ) output join ds1_row,ds2_row
  elsif ( temp_ds1.key <= temp_ds2.key ) read ds1_row from temp_ds1
  elsif ( temp_ds1.key => temp_ds2.key ) read ds2_row from temp_ds2
end loop
```

#### Ejemplo:

Suponer los siguientes conjuntos de datos ordenados:

##### Data Set 1

10,20,30,40,50,60,70

##### Data set 2

20,20,40,40,40,40,40,60,70,70

##### Comparación 1

Ds1 Ds2

10 < 20 => **no match**, continua con el siguiente valor de Ds1, no hay match con 10 **next(Ds1)**  
 20 = 20 => **match**, continua revisando con Ds2 **next(Ds2)**  
 20 = 20 => **match**, (segundo valor de Set 2), continúa con Ds2 **next(Ds2)**  
 20 < 40 => **no match** continua con el siguiente valor de Ds1, no hay match con 20 **next(Ds1)**  
 30 > 20 => **no match** compara con el último match de Ds2, en este caso con el segundo 20 **next(Ds2)** ya que puede haber match al ser mayor  
 30 < 40 => **no match**, **next(Ds1)** ya no puede existir match  
 40 > 20 => **no match**, **next(Ds2)**  
 40 = 40 => **match**, **next(Ds2)**  
 40 = 40 => **match**, **next(Ds2)**  
 40 = 40 => **match**, **next(Ds2)**  
 40 = 40 => **match**, **next(Ds2)**  
 40 = 40 => **match**, **next(Ds2)**  
 40 < 60 => **next(Ds1)** ya no puede existir match

```

50 > 40 => next(Ds2)
50 < 60 => next(Ds1)
70 > 40 => next(Ds2)
70 > 60 => next(Ds2)
70 = 70 => match, next(Ds2)
70 = 70 => match , next(Ds2)

```

- Observar que no se necesita leer todos los valores de Ds2 , esto representa una ventaja sobre Nested Loop.

### Ejemplo:

Obtener los nombres de los médicos y los nombres de sus especialidades. Considerar únicamente nombres de médicos que inician con A.

```

explain plan for
select m.nombre,e.nombre
from especialidad e, medico m
where e.especialidad_id = m.especialidad_id
and m.nombre like 'A%' ;

select plan_table_output from table(dbms_xplan.display);

```

#### PLAN\_TABLE\_OUTPUT

Plan hash value: 3154104233

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		141	4794	241 (1)	00:00:01
1	MERGE JOIN		141	4794	241 (1)	00:00:01
2	TABLE ACCESS BY INDEX ROWID	ESPECIALIDAD	53	1272	2 (0)	00:00:01
3	INDEX FULL SCAN	ESPECIALIDAD_PK	53		1 (0)	00:00:01
* 4	SORT JOIN		141	1410	239 (1)	00:00:01
* 5	TABLE ACCESS FULL	MEDICO	141	1410	238 (0)	00:00:01

Predicate Information (identified by operation id):

```

4 - access("E"."ESPECIALIDAD_ID"="M"."ESPECIALIDAD_ID")
    filter("E"."ESPECIALIDAD_ID"="M"."ESPECIALIDAD_ID")
5 - filter("M"."NOMBRE" LIKE 'A%')

```

- Id 3.** Se realiza un index full scan del índice **especialidad\_pk**.
- Id 2.** Se obtienen todos datos de especialidad empleando un table acces by index row id (53 registros). Esto se realiza de esta manera para tener los datos ordenados.

- C. **Id 5.** Se realiza un table access full a medico aplicando el predicado nombre like 'A%', se estiman 141 registros.
- D. **Id 4.** Al no contar con un índice aquí, se tiene que lanzar una operación de ordenamiento sobre los 141 registros: SORT JOIN en el paso 4.
- E. **Id 1.** Una vez que se tienen los 2 conjuntos de datos ordenados, se aplica el algoritmo MERGE JOIN en el paso 1.

#### 5.3.5.4. Cartesian Join.

- Se produce al omitir alguna condición de JOIN.
- En algunos casos el optimizador puede seleccionar este método, por ejemplo, hacer un producto cartesiano entre 2 tablas muy pequeñas que en conjunto hacen JOIN con una tabla grande.

Algoritmo:

```
for ds1_row in ds1 loop
  for ds2_row in ds2 loop
    output ds1_row and ds2_row
  end loop
end loop
```

ds1 corresponde con el conjunto de datos de menor tamaño.

Ejemplo:

```
SQL>
explain plan for
select m.nombre, e.nombre
from medico m, especialidad e;

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

#### PLAN\_TABLE\_OUTPUT

Plan hash value: 631757493

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		265K	7246K	7016 (1)	00:00:01
1	MERGE JOIN CARTESIAN		265K	7246K	7016 (1)	00:00:01
2	TABLE ACCESS FULL	MEDICO	5000	35000	238 (0)	00:00:01
3	BUFFER SORT		53	1113	6778 (1)	00:00:01
4	TABLE ACCESS FULL	ESPECIALIDAD	53	1113	1 (0)	00:00:01

- A. El primer paso a ejecutar es el número 4. Se obtiene el contenido completo de especialidad.
- B. El siguiente paso es el 3. BUFFER SORT significa, copiar los datos de la SGA a la PGA. Esto se debe a que el producto requiere múltiples lecturas hacia los datos de especialidad. Para evitar la contención con otras consultas, se aíslan estas lecturas, pasando los datos a la PGA.
- C. En el paso 1, se aplica el algoritmo para obtener el producto.

#### *5.3.5.5. Métodos para procesar un outer join.*

- Nested Loop outer Join
- Hash Join Outer Join
- Sort Merge Outer Join

#### *5.3.5.5. Métodos adicionales para optimizar joins.*

- Bloom filters
- Partition Wise Joins

Para más detalles consultar el siguiente [enlace](#).