¿Qué es la Cardinalidad?

Imagina la Cardinalidad como **la predicción del clima**.

- Oracle no sabe exactamente cuántas filas devolverá tu consulta antes de ejecutarla.
- Así que **adivina** (estima) basándose en estadísticas.
- Esa adivinanza es la columna **Rows** (Filas) en el Explain Plan.

La Regla de Oro:

- Si Oracle predice **pocas filas** (Baja Cardinalidad) Preferirá Índices y Nested Loops.
- Si Oracle predice **muchas filas** (Alta Cardinalidad) Preferirá Full Scans y Hash Joins.

Paso 1: Preparar el Escenario (El problema de los datos sesgados)

Para ver la cardinalidad en acción, necesitamos datos "sesgados" (no uniformes). Vamos a crear una tabla de tareas donde el **99%** están terminadas y solo el **1%** están pendientes.

Copia y ejecuta esto en Oracle Live SQL:

```
-- 1. Crear tabla
CREATE TABLE mis_tareas (
  tarea_id NUMBER,
  descripcion VARCHAR2(50),
  estado VARCHAR2(20) -- 'CERRADO' (Mayoría) vs 'ABIERTO' (Minoría)
);

-- 2. Insertar datos sesgados (Skewed Data)
BEGIN
  -- Insertamos 19,800 tareas CERRADAS
  FOR i IN 1..19800 LOOP
    INSERT INTO mis_tareas VALUES (i, 'Tarea vieja', 'CERRADO');
  END LOOP;

  -- Insertamos solo 200 tareas ABIERTAS
  FOR i IN 19801..20000 LOOP
```

```
    INSERT INTO mis_tareas VALUES (i, 'Tarea urgente', 'ABIERTO');
  END LOOP;

  COMMIT;
END;
/

-- 3. Crear un índice en la columna ESTADO
CREATE INDEX idx_estado_tarea ON mis_tareas(estado);
```

Paso 2: El "Error" (Sin Histrogramas)

Por defecto, si Oracle solo tiene estadísticas básicas, asume que los datos están repartidos equitativamente. Si hay 2 estados ('ABIERTO', 'CERRADO') y 20,000 filas, Oracle pensará inocentemente que hay 10,000 de cada uno.

Vamos a "engañar" a Oracle calculando estadísticas básicas (sin histogramas):

```
-- Forzamos estadísticas básicas (sin histogramas)
EXEC DBMS_STATS.GATHER_TABLE_STATS(USER, 'MIS_TAREAS', METHOD_OPT=>'FOR ALL COLUMNS SIZE 1');
```

Ahora, veamos qué piensa Oracle de la palabra 'ABIERTO':

```
EXPLAIN PLAN FOR
SELECT * FROM mis_tareas WHERE estado = 'ABIERTO';

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

Interpretación del Plan (El fallo):

Id	Operation	Rows (Estimado)	Cost
1	TABLE ACCESS FULL	**10000**	...

- **Rows (10000):** Oracle predice 10,000 filas. ¡Es mentira! Solo hay 200.
- **Operation:** Como cree que son muchas filas, hace un **TABLE ACCESS FULL** (ignora el índice), lo cual es lento para buscar solo 200 datos.

Paso 3: La Solución (Histogramas)

Para corregir la cardinalidad, necesitamos decirle a Oracle: *"Oye, los datos no son uniformes, hay una distribución desigual"*. Esto se hace con **Histogramas**.

Ejecuta esto para calcular estadísticas inteligentes:

```
-- 'SIZE SKEWONLY' le dice a Oracle que busque columnas con datos desbalanceados
EXEC DBMS_STATS.GATHER_TABLE_STATS(USER, 'MIS_TAREAS', METHOD_OPT=>'FOR ALL COLUMNS SIZE SKEWONLY');
```

Paso 4: Verificar la "Mágica" de la Cardinalidad Correcta

Ahora volvemos a ejecutar **exactamente la misma consulta** para 'ABIERTO'.

```
EXPLAIN PLAN FOR
SELECT * FROM mis_tareas WHERE estado = 'ABIERTO';

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

Nueva Interpretación (Caso 'ABIERTO'):

Id	Operation	Rows
1	INDEX RANGE SCAN	**200**

- **Rows:** Ahora dice **200**. ¡Ha acertado!
- **Operation:** Como sabe que son pocas filas, ahora sí usa el **INDEX RANGE SCAN**. Es mucho más rápido.

Paso 5: El cambio de comportamiento (Caso 'CERRADO')

La cardinalidad es dinámica. Si buscamos 'CERRADO', Oracle debería saber que son muchos datos y **no** usar el índice (porque usar el índice para leer el 99% de la tabla es más lento que leer la tabla directamente).

```
EXPLAIN PLAN FOR
SELECT * FROM mis_tareas WHERE estado = 'CERRADO';

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

Interpretación:

- **Rows:** ~19,800 (Alta Cardinalidad).
- **Operation: TABLE ACCESS FULL**.
- **Lección:** Oracle tomó dos decisiones opuestas (Índice vs Full Scan) en la misma tabla, basándose únicamente en la **Cardinalidad**.

Resumen: Cómo detectar problemas de Cardinalidad

Cuando analices problemas de rendimiento, mira la columna Rows del plan y pregúntate: **¿Tiene sentido este número?**

1. **El síntoma:** Tu consulta devuelve 5 filas, pero el Plan dice Rows: 50K.
    - **Consecuencia:** Oracle usará Hash Joins pesados en lugar de índices rápidos.
    - **Solución:** Recalcular estadísticas con histogramas (DBMS_STATS).
2. **El síntoma:** Tu consulta devuelve 1 millón de filas, pero el Plan dice Rows: 1.
    - **Consecuencia:** Oracle intentará usar Nested Loops (bucle a bucle), lo que matará el rendimiento y podría tardar horas.
    - **Solución:** Verificar si las estadísticas están "rancias" (viejas).

Prueba Avanzada

Intenta hacer un JOIN entre esta tabla mis_tareas y la tabla mis_pedidos del ejercicio anterior. Si filtras por estado = 'ABIERTO', verás un **NESTED LOOPS** (porque son pocas). Si filtras por estado = 'CERRADO', verás un **HASH JOIN** (porque son muchas).

¿Qué es un HINT y cómo se escribe?

Un Hint es una instrucción especial que se coloca dentro de un comentario justo después del SELECT.

- Sintaxis: /*+ NOMBRE_HINT(alias) */
- **Regla de Oro:** Si usas un **Alias** en tu tabla (ej: FROM mis_tareas t), el Hint **DEBE** usar el alias (t), no el nombre de la tabla.

Ejemplo 1: Forzar el uso de un Índice (Aunque sea mala idea)

Contexto:

Vimos que buscar tareas 'CERRADO' (que son el 99% de los datos) provoca un TABLE ACCESS FULL. Oracle tiene razón, es lo más rápido. Pero, ¿y si queremos obligarlo a usar el índice idx_estado_tarea?

Código:

Nota el uso del alias t.

```
EXPLAIN PLAN FOR

SELECT /*+ INDEX(t idx_estado_tarea) */ * FROM mis_tareas t

WHERE estado = 'CERRADO';


SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

Interpretación:

1. **Operation:** Verás **INDEX RANGE SCAN** (o similar) forzado.
2. **Cost:** ¡Mira el costo! Probablemente sea **mucho más alto** que el Full Scan original.
    - *Por qué:* Has obligado a Oracle a leer el índice y luego saltar a la tabla millones de veces. Esto demuestra que **usar un índice no siempre es más rápido**.

Ejemplo 2: Forzar un Full Scan (Ignorar el Índice)

Contexto:

Para las tareas 'ABIERTO' (solo 200 filas), Oracle usa el índice automáticamente. Vamos a prohibírselo y forzarlo a leer toda la tabla.

Código:

Usamos el hint FULL.

SQL

```
EXPLAIN PLAN FOR

SELECT /*+ FULL(t) */ * FROM mis_tareas t

WHERE estado = 'ABIERTO';


SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

Interpretación:

1. **Operation:** Aparece **TABLE ACCESS FULL**.
2. Esto es útil cuando crees que el índice está corrupto o fragmentado y sospechas que leer la tabla completa secuencialmente será más rápido.

Ejemplo 3: Controlar los Joins (Hash vs Nested Loops)

Contexto:

Vamos a volver a unir mis_pedidos y mis_clientes.

- Oracle suele elegir **HASH JOIN** para consultas grandes.
- Vamos a forzar un **NESTED LOOPS** (Bucle anidado).

Código:

Usamos el hint USE_NL(tabla_a tabla_b).

```
EXPLAIN PLAN FOR
SELECT /*+ USE_NL(p c) */ p.pedido_id, c.nombre
FROM mis_pedidos p
JOIN mis_clientes c ON p.cliente_id = c.cliente_id;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

Interpretación:

1. **Operation:** Verás **NESTED LOOPS**.
2. **Efecto:** Verás que el costo se dispara y el tiempo estimado aumenta.
3. **¿Cuándo usar esto?** Si solo quieres ver las *primeras 10 filas* en pantalla (FIRST_ROWS) rápidamente, el Nested Loop es mejor porque te da datos al instante. El Hash Join tarda un poco en arrancar (porque primero construye el mapa hash en memoria) pero luego es muy rápido entregando todo.

Resumen de los Hints más poderosos

Guarda esta lista para tus prácticas en Live SQL:

Hint	**Sintaxis**	**Descripción**
Forzar Índice	/*+ INDEX(alias nombre_indice) */	Obliga a usar un índice específico.
Full Scan	/*+ FULL(alias) */	Prohíbe usar índices; lee toda la tabla.
Nested Loops	/*+ USE_NL(a b) */	Fuerza la unión bucle a bucle (bueno para pocas filas).
Hash Join	/*+ USE_HASH(a b) */	Fuerza la unión por hash (bueno para reportes masivos).
Parallel	/*+ PARALLEL(alias 4) */	(Avanzado) Usa 4 hilos de CPU simultáneos.

Advertencia Final (Muy Importante)

Usar Hints en producción es un arma de doble filo.

- **Lo bueno:** Resuelves un problema de rendimiento hoy.
- **Lo malo:** El hint es **estático**. Si hoy fuerzas un índice porque tienes pocos datos, y mañana tu tabla crece a 10 millones de filas, ese hint podría destruir el rendimiento del sistema, y el optimizador no podrá corregirlo porque le has atado las manos.

Recomendación: Usa Hints solo como último recurso cuando las estadísticas (DBMS_STATS) no logren arreglar el plan.