Introducción: ¿Qué es un Explain Plan?

Un Explain Plan es el mapa que sigue el motor de la base de datos para ejecutar tu consulta. Te dice:

1. **El orden** de las operaciones.
2. **El método** de acceso a los datos (índices vs. lectura completa).
3. **El costo** estimado (recursos de CPU y E/S).

Paso 1: Preparar el Entorno (Datos de Prueba)

Para entender cómo mejorar una consulta, primero necesitamos una tabla con suficientes datos. En Oracle Live SQL, vamos a crear una tabla desde cero para garantizar que el ejemplo funcione perfectamente.

Copia y ejecuta el siguiente bloque en la consola de Live SQL:

```
-- 1. Crear una tabla simple de 'Pedidos'
CREATE TABLE mis_pedidos (
  pedido_id NUMBER,
  cliente_nombre VARCHAR2(50),
  fecha_pedido DATE,
  monto NUMBER
);

-- 2. Insertar 20,000 filas de datos aleatorios (para generar volumen)
BEGIN
  FOR i IN 1..20000 LOOP
    INSERT INTO mis_pedidos VALUES (
      i,
      'Cliente_' || MOD(i, 100),
      SYSDATE - MOD(i, 365),
      ROUND(DBMS_RANDOM.VALUE(10, 1000), 2)
    );
  END LOOP;
  COMMIT;
```

END;

/

Nota Pro: Después de crear datos masivos, siempre debemos calcular estadísticas para que el optimizador de Oracle sepa la verdad sobre la tabla. Ejecuta esto:

SQL

```
EXEC DBMS_STATS.GATHER_TABLE_STATS(USER, 'MIS_PEDIDOS');
```

Paso 2: Crear el primer Explain Plan (El problema)

Vamos a buscar un pedido específico, el número 15400. Como acabamos de crear la tabla sin índices, Oracle tendrá que leerla toda.

Para generar el plan en Live SQL, usamos dos pasos: EXPLAIN PLAN FOR seguido de la consulta, y luego visualizamos el resultado.

Ejecuta esto:

```
-- Paso A: Solicitar al motor que calcule el plan (sin ejecutar la query)
EXPLAIN PLAN FOR
SELECT * FROM mis_pedidos WHERE pedido_id = 15400;

-- Paso B: Mostrar el plan generado
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

Paso 3: Interpretar el Plan

Al ejecutar el comando anterior, verás una tabla de texto. Aquí te explico cómo leerla:

La salida se verá similar a esto (simplificado):

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	39	**16** (0)	00:00:01
* 1	**TABLE ACCESS FULL**	**MIS_PEDIDOS**	1	39	**16** (0)	00:00:01

Claves para interpretar:

1. **Operation (TABLE ACCESS FULL):** Esta es la alerta roja. Significa que Oracle tuvo que leer las 20,000 filas una por una para encontrar el ID 15400. Es ineficiente para buscar un solo dato.

2. **Rows (1):** El optimizador estima correctamente que encontrará 1 fila.
3. **Cost (16):** Este es un número relativo. Cuanto más bajo, mejor. Aquí el costo es "16" unidades de trabajo.

Paso 4: Mejorar el Plan (La Solución)

Para evitar leer toda la tabla, crearemos un **Índice B-Tree** en la columna pedido_id. Esto funciona como el índice de un libro: le dice a Oracle exactamente en qué página está el dato.

Ejecuta esto:

```
-- Crear el índice
CREATE INDEX idx_pedido_id ON mis_pedidos(pedido_id);

-- Importante: Recalcular estadísticas para que Oracle sepa que el índice existe y es útil
EXEC DBMS_STATS.GATHER_TABLE_STATS(USER, 'MIS_PEDIDOS');
```

Paso 5: Verificar la Mejora

Ahora volvemos a generar el Explain Plan para la misma consulta.

Ejecuta esto nuevamente:

```
EXPLAIN PLAN FOR
SELECT * FROM mis_pedidos WHERE pedido_id = 15400;

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

Nueva Interpretación:

La salida ahora debería verse así:

Id	Operation	Name	Rows	Bytes	Cost (%CPU)
0	SELECT STATEMENT		1	39	**2** (0)
1	TABLE ACCESS BY INDEX ROWID BATCHED	MIS_PEDIDOS	1	39	2 (0)

Id	Operation	Name	Rows	Bytes	Cost (%CPU)
* 2	**INDEX RANGE SCAN**	**IDX_PEDIDO_ID**	1		1 (0)

¿Qué ha cambiado?

1. **Operation:** Ahora ves **INDEX RANGE SCAN** (o UNIQUE SCAN). Significa que Oracle buscó en el índice, no en la tabla completa.
2. **Cost:** Bajó de **16** a **2**. ¡Una mejora masiva en rendimiento!

Paso 6: Un caso más complejo (Filtrado en columnas sin índice)

¿Qué pasa si buscamos por cliente_nombre?

```
EXPLAIN PLAN FOR
SELECT * FROM mis_pedidos WHERE cliente_nombre = 'Cliente_50';

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

Verás que vuelve a aparecer **TABLE ACCESS FULL**.

- **Razón:** Creamos un índice en pedido_id, no en cliente_nombre.
- **Lección:** Los índices deben crearse basándose en las columnas que usas en la cláusula WHERE o JOIN.

Resumen de Operaciones Comunes

Aquí tienes una "tabla de traducción" rápida para cuando leas tus planes:

Operación	Significado	¿Es bueno o malo?
TABLE ACCESS FULL	Lee toda la tabla.	**Malo** si buscas pocas filas. **Normal** si haces reportes de toda la tabla.
INDEX UNIQUE SCAN	Busca un valor único (ej. Primary Key).	**Excelente**. Lo más rápido posible.
INDEX RANGE SCAN	Busca un rango de valores en un índice.	**Muy bueno**.

Operación	Significado	¿Es bueno o malo?
TABLE ACCESS BY INDEX ROWID	Después de mirar el índice, va a la tabla a buscar el resto de datos.	**Normal/Bueno**. Es el paso posterior al índice.

Consejos Finales para Oracle Live SQL

1. **Explain Plan vs Autotrace:** En herramientas de escritorio se usa F10 o Autotrace. En Live SQL, siempre usa el combo EXPLAIN PLAN FOR ... + SELECT ... DBMS_XPLAN.DISPLAY.
2. **Estadísticas:** En la vida real, las estadísticas se calculan automáticamente por la noche. En pruebas de Live SQL, ejecuta DBMS_STATS si creas tablas y los planes no tienen sentido.
3. **Cardinalidad:** Si la columna Rows dice "1" pero tu consulta devuelve 1 millón de filas, tus estadísticas están desactualizadas y el plan será ineficiente.

Ejemplo 2: Análisis de JOINS (Uniones de Tablas)

El rendimiento cambia drásticamente dependiendo de *cómo* Oracle une dos tablas. Vamos a crear una tabla de clientes para unirla con nuestros pedidos.

1. Preparación de datos (Copia y pega):

```
-- Crear tabla de clientes
CREATE TABLE mis_clientes (
  cliente_id NUMBER PRIMARY KEY,
  nombre VARCHAR2(50),
  region VARCHAR2(20)
);

-- Insertar 100 clientes
BEGIN
  FOR i IN 1..100 LOOP
    INSERT INTO mis_clientes VALUES (i, 'Cliente_' || i, 'Region_' || MOD(i, 5));
  END LOOP;
  COMMIT;
END;
```

/

-- IMPORTANTE: Vamos a añadir una columna de enlace a la tabla de pedidos anterior

ALTER TABLE mis_pedidos ADD (cliente_id NUMBER);

UPDATE mis_pedidos SET cliente_id = MOD(pedido_id, 100) + 1;

COMMIT;

-- Recalcular estadísticas

EXEC DBMS_STATS.GATHER_TABLE_STATS(USER, 'MIS_CLIENTES');

EXEC DBMS_STATS.GATHER_TABLE_STATS(USER, 'MIS_PEDIDOS');

2. Escenario A: El "Hash Join" (Consultas Masivas)

Si quieres un reporte de **todos** los pedidos con sus nombres de clientes, Oracle usará un HASH JOIN.

EXPLAIN PLAN FOR

SELECT 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:** Verás la operación **HASH JOIN**.
- **¿Qué significa?** Oracle carga la tabla pequeña (clientes) en memoria (crea un mapa hash) y luego recorre la tabla grande (pedidos) comparando. Es **muy eficiente** para grandes volúmenes de datos sin filtros específicos.

3. Escenario B: El "Nested Loops" (Consultas Específicas)

Ahora busquemos los pedidos de **un solo** cliente específico (ID 50).

EXPLAIN PLAN FOR

SELECT p.pedido_id, c.nombre

FROM mis_pedidos p

JOIN mis_clientes c ON p.cliente_id = c.cliente_id

WHERE c.cliente_id = 50;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

- **Interpretación:** Deberías ver **NESTED LOOPS**.
- **¿Qué significa?** Oracle encuentra el cliente 50 primero, y luego hace un bucle buscando sus coincidencias en la tabla de pedidos.
- **Nota:** Para que esto sea óptimo, deberíamos crear un índice en la columna cliente_id de la tabla mis_pedidos (Foreign Key), de lo contrario, el "Nested Loop" podría ser lento.

Ejemplo 3: Índices Compuestos (El problema del "AND")

A veces filtramos por dos columnas a la vez.

El problema:

Imagina que buscas pedidos del "Cliente_50" hechos en una fecha específica.

```
EXPLAIN PLAN FOR

SELECT * FROM mis_pedidos

WHERE cliente_nombre = 'Cliente_50'

AND fecha_pedido > SYSDATE - 10;
```

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

Interpretación: Probablemente verás un **TABLE ACCESS FULL**. Aunque tengamos índices individuales, Oracle a veces prefiere leer todo si no hay un índice que cubra *ambas* condiciones juntas eficientemente.

La Solución (Índice Compuesto):

```
-- Crear índice que cubre ambas columnas

CREATE INDEX idx_cliente_fecha ON mis_pedidos(cliente_nombre, fecha_pedido);
```

```
-- Ver el nuevo plan

EXPLAIN PLAN FOR

SELECT * FROM mis_pedidos

WHERE cliente_nombre = 'Cliente_50'

AND fecha_pedido > SYSDATE - 10;
```

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

Mejora: Ahora verás **INDEX RANGE SCAN** sobre IDX_CLIENTE_FECHA. Oracle va directo a la sección del "Cliente_50" y dentro de esa sección busca la fecha. El costo bajará drásticamente.

Ejemplo 4: Índices Basados en Funciones (El error invisible)

Este es el error más común de los desarrolladores. Buscar texto ignorando mayúsculas/minúsculas.

El Problema:

-- Buscamos un cliente en mayúsculas

EXPLAIN PLAN FOR

SELECT * FROM mis_clientes WHERE UPPER(nombre) = 'CLIENTE_50';

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

Interpretación:

Verás TABLE ACCESS FULL.

- **¿Por qué?** Aunque tengas un índice en nombre, no tienes un índice en UPPER(nombre). Para la base de datos, la columna transformada es algo totalmente nuevo.

La Solución:

Creamos un índice basado específicamente en la función:

CREATE INDEX idx_nombre_upper ON mis_clientes(UPPER(nombre));

EXPLAIN PLAN FOR

SELECT * FROM mis_clientes WHERE UPPER(nombre) = 'CLIENTE_50';

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

Mejora: Ahora verás **INDEX RANGE SCAN** sobre IDX_NOMBRE_UPPER. Acabas de optimizar una búsqueda "case-insensitive".

Tabla Resumen de Nuevas Operaciones

Agrega esto a tu conocimiento:

Operación	Explicación	Cuándo es buena
HASH JOIN	Une dos tablas en memoria.	Cuando unes muchas filas (ej. reportes completos).
NESTED LOOPS	Bucle: Toma una fila de A y busca en B.	Cuando unes pocas filas (ej. buscar datos de un usuario).
SORT ORDER BY	Ordena los resultados.	Es costosa (usa CPU). Intenta evitarla usando índices ordenados.
FILTER	Aplica condiciones finales.	Normal, pero mejor si el índice ya filtra los datos antes.