




 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

Índice

1. Objetivos y alcance.....	3
2. Selección de campos	4
2.1. Nombres de columnas cualificados.....	4
2.2. DISTINCT, GROUP BY y UNION	6
2.3. ¿Qué campos se seleccionan con GROUP BY?	7
2.4. Uso de COUNT()	8
2.5. DECODE y CASE.....	9
2.6. NULLIF y COALESCE.....	11
2.7. Evitar el uso de Select *	12
2.8. Subconsultas Correlacionadas	13
2.9. No utilizar el usuario propietario para referenciar un objeto	14
3. Condiciones de Búsqueda	15
3.1. LIKE sin comodines.....	15
3.2. Rangos ILIMITADOS ('<','<','=','>','>=')	16
3.3. Operador BETWEEN	17
3.4. NULL	18
3.5. NOT: Uso inadecuado.....	19
3.6. Pseudocolumna ROWNUM	20
3.7. No usar funciones sobre columnas en la cláusula WHERE.....	22
3.8. Evitar conversiones implícitas de datos.....	25
3.9. Dejar RowId en manos del gestor de base de datos.....	26
4. Consultas Ordenadas	28
4.1. Order by	28
4.2. Evitar el uso de ORDER BY de manera indiscriminada.	29
4.3. Evitar el uso de Order By 1,2,	30
4.4. Ordenación de datos con Group by.....	31
5. Consultas sobre conjunto de tablas.....	32
5.1. Cláusula WHERE con predicados simples	32
5.2. Orden de las tablas	33
5.3. Evitar el uso de OUTER JOINS (+)	34
5.4. Evitar la composición de OUTER JOINS (+) incompletos	36
5.5. Consultas recursivas.....	37
5.6. Cláusulas WHERE y HAVING	38
5.7. Subconsultas anidadas. ¿EXISTS o IN?	39
5.8. No usar "EXISTS" en lugar de "JOIN"	40
5.9. EXISTS frente a DISTINCT	42
5.10. NOT, '!=' y NOT IN	43
5.11. Operadores de conjuntos (UNION, MINUS, INTERSEC)	45
5.12. Uso de RETURNING	46
6. Otras recomendaciones para el acceso a datos.....	47
6.1. Uso de vistas	47
6.2. Tablas temporales.....	48
6.3. Sentencias SQL sobrecargadas	49

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

6.4.	%TYPE y %ROWTYPE	50
6.5.	Uso de SQL dinámico nativo	51
6.6.	Uso del carácter comilla dentro de cadenas	52
6.7.	Comprobaciones de existencia de filas	54
7.	Transacciones y Actualizaciones	56
7.1.	¿Actualizaciones masivas en una única sentencia?	56
7.2.	COMMIT dentro de un cursor (Fetch across commit)	57
7.3.	NOWAIT	58
7.4.	CURSOR FOR UPDATE y CURRENT OF.....	59
7.5.	Ejecutar COMMIT O ROLLBACK al terminar la transacción	60
7.6.	No insertar valores sin especificar los campos	61
7.7.	Encapsulamiento de INSERT, UPDATE y DELETE.....	62
A I.	Anexo: Análisis de los Planes de Ejecución	63
A I.1.	Jerarquía de un plan de ejecución	64
A I.2.	Métodos de acceso.....	65
A I.3.	Métodos de Joins	66
A I.4.	Operaciones	69
A I.5.	¿qué hacer en caso de Bind Variables?	69
A I.6.	Estadísticas de AUTOTRACE.....	70
A II.	Anexo: Consultas SIRhUS	72
A II.1.	Vistas de acceso a filas basadas en otras tablas de acceso a filas	72
A II.2.	Uso de IH_FU_VALOR.....	72
A II.3.	Estructura de dependencias	73



	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

1. Objetivos y alcance

El objetivo de este apartado es la identificación de aquellas prácticas aconsejables en el desarrollo de Sistemas de Información que contienen sentencias de acceso y manipulación de datos basados en el Gestor de Bases de Datos *Oracle*, principalmente en los aspectos de optimización de tiempos de respuesta.

Este documento se orienta principalmente a la versión *Oracle9i*, normalmente basadas en el **Optimizador Basado en Coste (Cbo)**.

La relación de las prácticas aquí descritas son de especial interés para aquellos accesos que conllevan consultas o modificaciones con rendimientos deficientes.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

2. Selección de campos

2.1. Nombres de columnas cualificados



Siempre que en una consulta de selección intervenga más de una tabla, será necesario cualificar todas las columnas presentes, con el alias o el nombre de la tabla correspondiente.

Esta norma será de aplicación independiente a cada una de las consultas de selección que compongan sentencias de tipo UNION, INTERSECTION y MINUS.

Esta norma se aplicará igualmente en las actualizaciones (UPDATE), o borrados (DELETE), que incluyan una o más subconsultas en su predicado.

Es sabido que cuando se consultan campos de igual nombre pertenecientes a tablas distintas es necesario anteponer el nombre de la tabla a las columnas. Esta forma de identificar a las columnas está aconsejada siempre que se seleccionen campos de varias tablas, aun no habiendo coincidencia de nombres.

El uso de nombres de columnas totalmente cualificados facilita la etapa de **parsing** de *Oracle*. Cuando no se indica el nombre de la tabla, *Oracle* tiene que identificar la tabla a la que pertenece cada columna como primera operación.

Además, esto evita que posibles incorporaciones de columnas con nombre igual a columnas de otras tablas provoquen errores de ambigüedad al identificar las columnas seleccionadas.

• Ejemplos



Selección simple de columnas


```
SELECT Nombre, Apellido, cod_muni,
       FROM Empleados, Municipio
       WHERE MunicipioId = Municipio.Id;
```



Nombres de columnas totalmente cualificados

```
SELECT Empleados.Nombre, Empleados.Apellido, Municipio.cod_muni
       FROM Empleados, Municipio
       WHERE Empleados.MunicipioId = Municipio.Id;
```

La ampliación de una tabla puede ocasionar que los componentes software desarrollado provoquen errores:

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	





Sentencia incluida en un componente software:

```
SELECT Nombre, Apellido, CodigoPostal, cod_muni
FROM Empleados, Municipio
WHERE MunicipioId = Municipio.Id;
```

Incluimos para los municipios que tienen sólo un Código Postal un campo: *CodigoPostal*

```
ALTER TABLE Municipio ADD (CodigoPostal number(5));
```

El componente software provocaría error, en la selección ambigua de *CodigoPostal*

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

2.2. DISTINCT, GROUP BY y UNION



No se deben escribir sentencias del tipo:



- `SELECT DISTINCT GROUP BY`
- `SELECT DISTINCT UNION SELECT DISTINCT`

Cuando se trate de obtener distintas filas mediante una consulta, no siempre es necesario utilizar **DISTINCT**. Hay operaciones que por sí solas filtran las filas coincidentes. Es el caso de operaciones de grupo mediante el uso de **GROUP BY**, y operaciones de conjunto de consultas a través de **UNION**. Por tanto, el uso de **DISTINCT** en estos casos es redundante y tiene peor rendimiento.

• Alternativas

Utilizar **DISTINCT** (o su sinónimo **UNIQUE**) sólo en el caso en el que realmente se obtengan filas duplicadas.

Se puede usar **DISTINCT** junto a **UNION ALL**, ya que éste no elimina filas coincidentes.



	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

2.3. ¿Qué campos se seleccionan con GROUP BY?



Cuando se utilice **GROUP BY**, los campos que formen parte de éste deben así mismo aparecer como campos de selección.

Si no se seleccionan todas las columnas que forman la agrupación, se seleccionan filas con información común duplicada y difícil de interpretar.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

2.4. Uso de COUNT()



Oracle proporciona una serie de funciones de agregación de datos en *SQL*. Mención especial merece la función **COUNT()**. Se debe tener claro qué resultados ofrece la función **COUNT()**.

Comúnmente **COUNT** se suele aplicar al operador asterisco (*) para conocer el número de filas de una consulta cumpliendo una condición, incluyendo duplicados y nulos. Aplicado sobre una *expresión* devuelve el número de filas donde la *expresión* no es nula.

Con los modificadores **DISTINCT** y **ALL** se puede forzar a que se devuelvan solamente los valores distintos de la *expresión* o que devuelvan todos los valores, la opción por defecto.

• Alternativas

Se deben emplear expresiones que aseguren la obtención del resultado buscado. Se puede utilizar *COUNT(constante)* en vez de *COUNT(*)* si se quiere usar siempre expresiones por homogeneidad y evitar usar *COUNT(*)* por error. En *PL/SQL* *COUNT(contante)* es más óptimo que *COUNT(*)*.

• Ejemplos



Número de filas de la tabla Empleados

```
SELECT COUNT(*)
FROM Empleados;
```



Número de filas de la tabla Empleados

```
SELECT COUNT(1)
FROM Empleados;
```





Número de empleados de los que se conoce su código postal

```
SELECT COUNT(CodigoPostal)
FROM Empleados;
```



Número de códigos postales distintos conocidos

```
SELECT COUNT(DISTINCT CodigoPostal)
FROM Empleados;
```


 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

2.5. DECODE y CASE



DECODE y **CASE** son dos operadores útiles en la simplificación y optimización de consultas.

DECODE y **CASE** son dos operadores para *SQL* que proporciona *Oracle* similares a una sentencia *IF...THEN...ELSE* de los lenguajes tradicionales. No solamente permiten elegir el valor que se devuelve, sino que se puede conseguir que en función de los valores de una columna, se muestren valores distintos de otras, o incluso combinaciones de valores de columnas.

Para casos en los que se quiera hacer operaciones matemáticas en función de los valores de otra columna (como por ejemplo una fecha) se puede optar por el uso de **DECODE** para conseguir los valores positivos y negativos que posteriormente se vayan a sumar.

DECODE se puede usar para “descodificar” enumerados o booleanos y mostrar un valor intuitivo para la persona que vaya a leer los resultados de las consultas, en vez de presentar valores numéricos para los que hay que recordar su significado.

A veces es necesario calcular varios agregados para diferentes rangos de datos de una misma tabla, se pueden hacer todos a la vez con **CASE**. Si se recorren los datos una sola vez, se consigue una mejora considerable. Se puede combinar los diferentes rangos de las agrupaciones, poniendo las condiciones que delimitan los rangos en una sentencia **CASE**.





DECODE sólo debe usarse cuando la operación dependa de los valores recuperados. Se producirá una mejora de rendimiento si el comportamiento de **DECODE** puede emularse mediante instrucciones **IF**

• Ejemplos



```
Ajuste de cuentas, restando lo gastado en Junio, y sumando lo gastado en
Julio y Agosto. Agosto con un recargo del 20%. Se diferencian los pedidos
Extraordinarios del resto de tipos de pedidos
SELECT TipoPedido, SUM(AjusteCuentas)
FROM
  (SELECT
    DECODE(TO_CHAR(Fecha, 'MM'),
      '06', -Importe,
      '07', Importe,
      '08', Importe*1.20) AjusteCuentas,
    DECODE(Tipo, 0, 'EXTRAORDINARIO', 'ORDINARIO') TipoPedido
  FROM
    Pedidos);
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	



Solución típica con diferentes sentencias SELECT

```
SELECT COUNT(*)
FROM Empleados
WHERE sueldo < 2000;

SELECT COUNT(*)
FROM Empleados
WHERE sueldo BETWEEN 2000 AND 4000;

SELECT COUNT(*)
FROM Empleados
WHERE sueldo > 4000;
```





Solución ÓPTIMA con una única sentencia SELECT

```
SELECT COUNT(CASE WHEN sueldo < 2000
                  THEN 1 ELSE null END) count1,
       COUNT(CASE WHEN sueldo BETWEEN 2000 AND 4000
                  THEN 1 ELSE null END) count2,
       COUNT(CASE WHEN sueldo > 4000
                  THEN 1 ELSE null END) count3
FROM Empleados;
```



Otro uso de CASE

```
SELECT x_empleado,
       (CASE
        WHEN EDAD<20 THEN 'Joven'
        WHEN EDAD>=60 THEN 'Jubilable'
        ELSE 'Empleado Tipo')
FROM EMPLEADOS;
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

2.6. NULLIF y COALESCE



El uso de **NULLIF** y **COALESCE** puede simplificar determinadas consultas.

El desconocimiento de determinadas funciones de Oracle, como el caso de **NULLIF** y **COALESCE**, puede conllevar la realización de innecesarias y complicadas consultas.

Con **NULLIF** podemos conocer si dos argumentos son iguales, y con **COALESCE** argumentos no nulos.

• Ejemplos





Uso de Nullif

```
SELECT x_employado, NULLIF(x_depto, 0) DPTO
FROM EMPLEADOS
```



Uso de Coalesce

```
SELECT x_employado, COALESCE(opcion1, opcion2, opcion3, opcion4, 0) SEL
FROM EMPLEADOS
```

	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
06. Manual de Instrucciones Técnicas		
06.01. Normas y Recomendaciones de Accesos a Bases de Datos		


2.7. Evitar el uso de Select *



Sólo se permitirá el uso de `SELECT *` cuando la información recuperada vaya a usarse únicamente para informar un parámetro de entrada de alguna unidad de programa en base de datos o librería, definido como `%ROWTYPE` de una única tabla.

Al usar `SELECT *` ... se seleccionan todos los campos de las tablas, consumiendo más recursos de los necesarios en el caso de que no sea necesario elegir todos los campos.

En herramientas en las que el componente desarrollado es posteriormente interpretado (**Developer**) durante su ejecución, el asterisco (*) es sustituido por la relación de las columnas existentes en el momento de compilación, de manera que se pueden producir errores (**"variable not in select list"**) al ejecutarse en otro entorno donde no estén definidas las tablas exactamente igual a las utilizadas en su compilación, o donde se realicen futuras modificaciones sobre estas tablas.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

2.8. Subconsultas Correlacionadas



Se debe limitar el uso de subconsultas correlacionadas ya que suelen penalizar el rendimiento de las consultas.

Oracle realiza subconsultas correlacionadas cuando en la subconsulta se hace referencia a columnas de la sentencia padre. Una subconsulta correlacionada se evalúa una vez para cada fila procesada por la sentencia padre. La sentencia padre puede ser una sentencia *SELECT*, *UPDATE* o *DELETE*.

• Alternativas

Las subconsultas no correlacionadas son optimizadas fácilmente por **CBO**.

Las subconsultas correlacionadas se deben limitar a casos en los que no sean viables otras alternativas.

• Ejemplos



Ejemplos genéricos de consultas correlacionadas

```
SELECT lista_select
  FROM tabla1 t_alias1
 WHERE expr operador
       (SELECT lista_columnas
        FROM tabla2 t_alias2
        WHERE t_alias1.columna
              operador t_alias2.columna);



UPDATE tabla1 t_alias1
  SET columna =
    (SELECT expr
     FROM tabla2 t_alias2
     WHERE t_alias1.columna = t_alias2.columna);

DELETE FROM tabla1 t_alias1
  WHERE columna operador
        (SELECT expr
         FROM tabla2 t_alias2
         WHERE t_alias1.columna = t_alias2.columna);
```



Empleados que ganan más que la media de los salarios de los departamentos

```
SELECT DepartamentoId, Apellidos, sueldo
  FROM Empleados emp
 WHERE sueldo > (SELECT AVG(sueldo)
                 FROM Empleados
                 WHERE Emp. DepartamentoId = DepartamentoId)
 ORDER BY department_id;
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

2.9. No utilizar el usuario propietario para referenciar un objeto



No se permite anteponer a un objeto el nombre del usuario propietario.



Hay que tener en consideración los siguientes puntos:

- No hay dos objetos con el mismo nombre en los esquemas propietarios
- La separación en distintos esquemas propietarios debe ser transparente al programador. Sólo sirve para la organización de los objetos de los distintos subsistemas.
- Para desactivar el acceso a filas hay que hacer uso del paquete `IH_PQ_ACCFIL`, en lugar de poner el usuario propietario.

-
-



Por razones de diseño, el acceso a la tabla `IHCABACTADM` tiene una vista en `NCOWNER` del mismo nombre para actuar sobre la recuperación de los valores sobre `L_STATUS`. Sólo en el caso que se quiera el valor real de `L_STATUS` se debe de poner `IHOWNER.IHCABACTADM`. Siempre y cuando no se pueda codificar de otra forma (p.ej. poner el código en `IHOWNER`).

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

3. Condiciones de Búsqueda

3.1. LIKE sin comodines



Nunca se debe usar **LIKE** en lugar de la igualdad.

El operador **LIKE** se debe usar sólo cuando se seleccione con comodines. El **Optimizador** simplifica las condiciones que usan **LIKE** para comparar expresiones sin comodines en una condición equivalente que use el operador de igualdad (=).

El **Optimizador** solamente puede modificar este tipo de comparaciones cuando son tipos de datos de longitud variable (por ejemplo `VARCHAR(10)`). En el caso de tipos de datos de tamaño fijo (por ejemplo un `CHAR(10)`) el **Optimizador** no puede transformar las comparaciones, debido a que con el operador de igualdad se entiende que los caracteres que siguen hasta rellenar todo el tipo de dato son blancos, mientras que con **Like** no se comporta así.

- **Alternativas**

Si se usa el **Optimizador Basado En Coste** y tipos de datos de longitud variable se puede confiar en que el **Optimizador** corrija el error, pero por coherencia es más prudente usar siempre el operador de igualdad si no se va a usar con comodines.

- **Ejemplos**





Uso incorrecto de like

```
SELECT
    Nombre, Apellidos
FROM
    Empleados
WHERE
    Apellidos LIKE 'PEREZ';
```



Versión Correcta

```
SELECT
    Nombre, Apellidos
FROM
    Empleados
WHERE
    Apellidos = 'PEREZ';
```

	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

3.2. Rangos ILIMITADOS ('<','<=','>','>=')





Para obtener resultados de forma más eficiente es preferible establecer rangos limitados que ilimitados, si se sabe el rango de valores que se está buscando.

La búsqueda de filas son siempre más eficientes si se puede usar un índice y un índice es más eficiente cuanto más delimitado es el rango de valores posibles. Las columnas para las que se usen rangos de búsqueda deberían estar indexadas.

- **Alternativas**

Siempre que sea posible se deben delimitar los rangos con dos valores usando el operador *BETWEEN* o con los operadores de comparación \geq y \leq .

En el caso de los índices múltiples hay que tener en cuenta que hay que acceder por la columna más significativa del mismo.

	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

3.3. Operador BETWEEN



Para delimitar los rangos de valores se puede usar indistintamente el operador *BETWEEN* o una condición equivalente con \geq y \leq .

El **optimizador** siempre reemplaza la condición que usa el operador de comparación *BETWEEN* por una condición equivalente que utiliza los operadores de comparación \geq y \leq .



- **Alternativas**

Se debe codificar de manera que el código quede lo más claro posible para facilitar le mantenimiento respetando las indicaciones del manual de estilo.

- **Ejemplo**



Transformación del operador BETWEEN
<p>sueldo BETWEEN 2000 AND 3000</p> <p>Se transforma en:</p> <p>sueldo \geq 2000 AND sueldo \leq 3000</p>

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

3.4. NULL



Evitar las búsquedas por **NULL** o **NOT NULL**.

Las búsquedas por **NULL** o **NOT NULL** son potencialmente más costosas al despreciarse el uso de índices de los campos que se busquen por **NULL** en el caso de estar indexados.



- **Alternativas**

En el caso de una búsqueda por **NOT NULL**, cuando sea posible es preferible usar un rango ilimitado antes que un **NOT NULL**.

- Cuando se está trabajando con tipos de datos numéricos, se puede optar por una expresión del tipo: *WHERE x >= 0 OR x < 0*. Da un mejor rendimiento que: *WHERE x IS NOT NULL*.
- Cuando se necesita consultar los valores no nulos de una columna alfanumérica, se puede usar una expresión del tipo: *WHERE Cadena >= 'A%' OR Cadena < 'A%'*.



Debe ponderarse el uso de esta recomendación. Su uso influye negativamente en la claridad del código generado.

	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Manual de Instrucciones Técnicas 06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

3.5. NOT: Uso inadecuado.





Nunca se debe utilizar el operador **NOT** para obtener el resultado contrario a un operador de comparación.

NOT en la mayoría de sus usos inutiliza los índices.

- **Alternativas**

El comportamiento del operador **NOT** puede emularse con el uso del resto de operadores lógicos ('<','<=','>','>=').

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

3.6. Pseudocolumna ROWNUM



Cuando se necesita comprobar la existencia filas en una tabla, y obtener algunos valores de muestra, **ROWNUM** puede resultar muy útil.

ROWNUM es una pseudocolumna de *Oracle* que indica en una consulta el número de fila que se está mostrando. En determinados casos, en los que se necesita saber si una consulta obtiene datos, o en los que se sabe que hay una cantidad enorme de datos y solamente se quiere una muestra pequeña, se puede evitar sobrecargar la base de datos limitando el número de filas que se quieren obtener.

Hay que tener en cuenta que **ROWNUM** es un valor calculado, no identifica a una fila, como es el caso de **ROWID**. Se obtiene en la fase anterior a las ordenaciones, por lo que una consulta ordenada por alguna columna, variará el orden de **ROWNUM**, pero no quiere decir que **ROWNUM** sea un valor fijo. Siempre es un valor calculado para las filas devueltas, pudiendo cambiar en cada sentencia su relación con la fila.

- Ejemplos**



Selección de usuarios al azar donde se ve como RowNum puede aparecer ordenado según otra columna

```
SELECT RowNum, SUBSTR(NomClien,12,10) as Orden , NomClien
FROM Clientes
WHERE rownum < 5 and nomclien > 'APELLIDO1_50%'
ORDER BY Orden DESC;
```

ROWNUM	ORDEN	NOMCLIEN
4	03	APELLID APELLIDO1_503 APELLIDO2_503 NOMBRE503
3	02	APELLID APELLIDO1_502 APELLIDO2_502 NOMBRE502
2	01	APELLID APELLIDO1_501 APELLIDO2_501 NOMBRE501
1	00	APELLID APELLIDO1_500 APELLIDO2_500 NOMBRE500



Selección de usuarios al azar donde se ve como RowNum parece que identifica siempre a las mismas filas al coincidir con la consulta anterior

```
SELECT RowNum, SUBSTR(NomClien,12,10) as Orden , NomClien
FROM Client
WHERE rownum < 5 and nomclien > 'APELLIDO1_50%'
ORDER BY Orden;
```

ROWNUM	ORDEN	NOMCLIEN
1	00	APELLID APELLIDO1_500 APELLIDO2_500 NOMBRE500
2	01	APELLID APELLIDO1_501 APELLIDO2_501 NOMBRE501
3	02	APELLID APELLIDO1_502 APELLIDO2_502 NOMBRE502
4	03	APELLID APELLIDO1_503 APELLIDO2_503 NOMBRE503 *



Selección de usuarios al azar donde se ve que RowNum no identifica siempre las mismas filas al cambiar las filas solicitadas

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	



```
SELECT RowNum, SUBSTR(NomClien,12,10) as Orden , NomClien
FROM Client
WHERE rownum < 5 and nomclien > 'APELLIDO1_502%'
ORDER BY Orden;
```

ROWNUM	ORDEN	NOMCLIEN
1	03 APELLID	APELLIDO1_503 APELLIDO2_503 NOMBRE503 *
2	04 APELLID	APELLIDO1_504 APELLIDO2_504 NOMBRE504
3	05 APELLID	APELLIDO1_505 APELLIDO2_505 NOMBRE505
4	06 APELLID	APELLIDO1_506 APELLIDO2_506 NOMBRE506

 JUNTA DE ANDALUCÍA CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas 06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

3.7. No usar funciones sobre columnas en la cláusula WHERE



En la medida de lo posible, es conveniente utilizar las columnas sin aplicar ninguna función sobre ellas.

El uso de funciones en la cláusula **WHERE**, con columnas como parámetros, puede provocar que se ignore el uso del índice (incluso los índices únicos) a no ser que haya expresamente creado un índice basado en función.

Expresiones complejas sobre columnas en la cláusula **WHERE** pueden provocar que el **Optimizador** calcule mal el número de filas que se van a devolver o cuáles son, y puede afectar al plan completo y el método de **JOIN** que elija.

- **Alternativas**

- Es mejor usar:

```
WHERE Pedidos1.NumeroPedido = Pedidos2.NumeroPedido
```

En vez de:

```
WHERE TO_NUMBER(
SUBSTR(Pedidos1.NumeroPedido, INSTR(Pedidos2.NumeroPedido, '.')-1))
= TO_NUMBER(
SUBSTR(Pedidos1.NumeroPedido, INSTR(Pedidos2.NumeroPedido, '.')-1)).
```

- Se deben evitar expresiones complejas del tipo:

```
- col1 = NVL (:b1,col1)
- NVL(col1,-999) = ...
- TO_DATE(), TO_NUMBER(), etc
```

Siempre que haya que usar estas funciones, debe ser sobre columnas sin índices o cuyo índice no resulte significativo.

- Es preferible una cláusula **WHERE**:



```
WHERE ( col1 = :b1 OR :b1 IS NULL )
```

- En vez de usar la técnica de **NVL**:

```
WHERE col1 = NVL (:b1,col1)
```

- **Ejemplo**

Un ejemplo claro lo tenemos en la aplicación de la función `IH_FU_VALOR(cod_constante)`, que devuelve el valor de la constante de código `cod_constante`. Es más eficiente calcular previamente a la definición del cursor el valor de la constante que la aplicación directa de la función en la cláusula **WHERE**. Otro ejemplo es la aplicación de **ROWIDTOCHAR**. En este caso se pierden las ventajas en el acceso a filas mediante el **ROWID**, es más recomendable utilizar la conversión contraria, **CHARTOROWID**

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	



Forma correcta de uso de la función IH_FU_VALOR.

```

DECLARE
  -- ACT. ADM. NOMBRAMIENTO. (01)
  CTE_ARRA CONSTANT VARCHAR2(2):=IH_FU_VALOR('ARRA');

  CURSOR CU_NOMPER( P_X_PERSONA NUMBER) IS
    SELECT X_CABACTADM,F_INIEFE
    FROM IHCABACTADM
    WHERE PER_X_PERSONA = P_X_PERSONA AND
          APC_ACA_C_ACTADM = CTE_ARRA
    ORDER BY F_INIEFE;

BEGIN

  FOR R_NOMPER IN CU_NOMPER LOOP
    NULL;
  END LOOP;

EXCEPTION
  WHEN OTHERS THEN
    RAISE_APPLICATION_ERROR(-20000,'APL-10646#1ERROR EN X');
END;
```



Uso recomendado de la función IH_FU_VALOR. La función se ejecuta innecesariamente una vez por cada fila de la tabla

```

DECLARE
  V NUMBER(8);
BEGIN
  SELECT COUNT(1)
  INTO V
  FROM IHCABACTADM
  WHERE APC_ACA_C_ACTADM = IH_FU_VALOR('ARRA');
END;
```



Equivalente correcto de la función anterior. Almacenamos el valor en una constante

```

DECLARE
  V NUMBER(8);
  V_CTE_ARRA CONSTANT VARCHAR(2):=IH_FU_VALOR('ARRA');
BEGIN
  SELECT COUNT(1)
  INTO V
  FROM IHCABACTADM
  WHERE APC_ACA_C_ACTADM = V_CTE_ARRA;
END;
```



Uso incorrecto de ROWIDTOCHAR

```



SELECT *
FROM IHCABACTADM
WHERE ROWIDTOCHAR(ROWID) = P_ROWID_ORIG
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	



Uso correcto de CHARTOROWID

```
SELECT *
FROM IHCABACTADM
WHERE ROWID = CHARTOROWID(P_ROWID_ORIG)
```


 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

3.8. Evitar conversiones implícitas de datos



No es recomendable obligar al motor de Oracle realizar conversiones automáticas de tipos de datos

Oracle permite la comparación de dos variables de tipos compatibles, así como la asignación de valores a variables de distinto tipo. En estos casos Oracle se encarga de transformar adecuadamente el tipo de dato.

La realización de conversiones implícitas de datos presenta una serie de inconvenientes:

- Se obtienen peores rendimientos
- La conversión realizada por Oracle puede no ser la que esperamos
- Puede cambiar de comportamiento al actualizar la versión de la base de datos o al modificar los parámetros de Oracle, como por ejemplo NLS_DATE_FORMAT

• Alternativas

En el supuesto de dos columnas de tipos de datos diferentes, se hará la conversión explícitamente en aquella que no sea componente de un índice o sea menos significativo.

No se debe mezclar tipos de datos. *Oracle* transforma predicados del tipo:



```
AND ColumnaDeTexto = ExpresionNumerica
```

en:

```
AND TO_NUMBER(ColumnaDeTexto) = ExpresionNumerica.
```

En estos casos es mejor hacer la conversión explícitamente:

```
AND ColumnaDeTexto = TO_CHAR(ExpresionNumerica).
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

3.9. Dejar RowId en manos del gestor de base de datos



No se debe usar **Rowid** para acceder y/o modificar los datos en transacciones diferentes.

Aunque **RowId** identifica unívocamente las filas de una tabla de la manera más eficiente posible, no hay garantía de que los **RowId** no se modifiquen y reutilicen cuando se producen movimientos de datos entre **tablespaces**, restauraciones de backups o incluso borrados y reinserciones. El **RowId** permanece constante sólo durante la vida de la fila a nivel físico en la tabla, no a nivel lógico.

Además la estructura interna del **RowId** puede variar entre las diferentes versiones de *Oracle* por lo que el código desarrollado basado en **RowId** puede no ser compatible entre las diferentes versiones.

• Alternativas

Para identificar de manera unívoca una fila y poder actualizarla eficientemente se deben usar las claves primarias y/o los índices únicos, que permiten localizar de manera única a un dato y *Oracle* usará los accesos por **RowId** internamente a partir del índice si lo considera necesario **CBO**.

Si se diese la situación en la que no hubiese más remedio que utilizar **RowId** habría que asegurarse que las filas a modificar están bloqueadas durante toda la transacción.

No se deben usar los accesos directos por **RowId** en procesos en los que no se controle lo que sucede en la base de datos entre la lectura del **RowId** y la escritura por **RowId**.

En caso de usarse, hay que extremar el cuidado con los posibles problemas de concurrencia. El borrado de una fila que se esté tratando con **RowId** por otro usuario, y una posterior inserción (por otro usuario) puede provocar que el **RowId** que apuntaba a la fila borrada apunte ahora a otra fila.

Una forma correcta de controlar el uso de **RowId** es haciendo uso del paquete **DBMS_ROWID**

• Ejemplos



Uso erróneo del rowid. Rowid correspondiente a otra tabla

```
SELECT *
  FROM IHCABACTADM CAA
 WHERE CHARTOROWID('AAAGUWAAJAAABWJAAA') = CAA.ROWID;
```

1 FILA ENCONTRADA

```
SELECT *
  FROM IHPT PT
 WHERE CHARTOROWID('AAAGUWAAJAAABWJAAA') = PT.ROWID;
```



ORA-01410: invalid ROWID

	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	



Uso del paquete DBMS_ROWID

```
SELECT *
  FROM IHPT PT
 WHERE CHARTOROWID( 'AAAGUWAAJAAABWJAAA' ) =
        DECODE( DBMS_ROWID.ROWID_OBJECT( PT.ROWID ) ,
        DBMS_ROWID.ROWID_OBJECT( 'AAAGUWAAJAAABWJAAA' ) , PT.ROWID )
```

	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	



4. Consultas Ordenadas

4.1. Order by



Si se necesita procesar una información de manera ordenada, se puede usar la cláusula **ORDER BY** de las consultas. No se debe implementar algoritmos de ordenación.

Cuando se requiere tratar la información de manera ordenada, se pueden usar los **cursores** con la cláusula **ORDER BY** ordenando de manera adecuada. No se debe intentar implementar un algoritmo de ordenación para datos que haya en la base de datos, ya que *Oracle* tiene optimizadas las estructuras y los algoritmos para poder ordenar más rápidamente.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	



4.2. Evitar el uso de **ORDER BY** de manera indiscriminada.



No se debe hacer uso de **ORDER BY** por sistema. Si no se requieren datos ordenados, no se debe utilizar.

Al ordenar los resultados de una consulta, la base de datos dedica importantes recursos a ello. Cuando no se necesita un orden concreto de los datos, se sobrecarga la base de datos innecesariamente si se fuerza su ordenación.

Los procesos que no requieren un tratamiento ordenado de los datos obtenidos, deben ser implementados sin criterios de ordenación.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

4.3. Evitar el uso de Order By 1,2, ...



No se debe usar la posibilidad de ordenar los datos obtenidos de una consulta en función de la posición que ocupa la columna.

En la cláusula ORDER BY se deben usar los nombres de las columnas o sus alias.

Debido a limitaciones tecnológicas, esta norma puede ser inaplicable en el caso de formularios ("FORMS") y consultas de unión (UNION). Únicamente en estos casos, y en lo que la tecnología utilizada limite efectivamente la aplicación de la presente norma, se permitirá el uso de números indicadores de posición de las columnas en las cláusulas "ORDER BY".

En la cláusula **ORDER BY** se puede usar el nombre de la columna, el alias de dicha columna o la posición de la columna para indicar el orden en el que aparecerán los resultados.

La opción de especificar los criterios de ordenamiento en función de la posición de la columna se desaconseja por ser una costumbre propensa a errores.

- **Alternativas**

Usar el nombre de la columna o su alias.

- **Ejemplos**





Ejemplo de uso prohibido del Order by

```
SELECT EMP.NOMBRE AS NOMB, EMP.APELLIDOS AS APELL
FROM EMPLEADOS EMP,
DEPARTAMENTO DEP
WHERE DEP.DEPARTAMENTOID = EMP.DEPARTAMENTOID
ORDER BY 2,1;
```



Ejemplo de uso apropiado del Order by

```
SELECT EMP.NOMBRE AS NOMB, EMP.APELLIDOS AS APELL
FROM EMPLEADOS EMP,
DEPARTAMENTO DEP
WHERE DEP.DEPARTAMENTOID = EMP.DEPARTAMENTOID
ORDER BY Apell, Nombre;
```

 JUNTA DE ANDALUCÍA CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

4.4. Ordenación de datos con Group by





Se debe utilizar el orden de datos que proporciona **GROUP BY** y no utilizar **ORDER BY**.

Cuando se agrupan conjuntos de datos utilizando **GROUP BY**, se impone una ordenación de los datos mostrados.

- **Alternativas**

Cuando se requiera un orden en los datos la agrupación debe hacerse por las columnas por las que se quiera obtener los datos ordenados. Solamente cuando se necesite ordenar por alguno de los campos agregados (por ejemplo `sum(...)`), habría que utilizar el **ORDER BY** para reordenar la consulta.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

5. Consultas sobre conjunto de tablas

5.1. Cláusula WHERE con predicados simples



En lo posible, los predicados en las cláusulas **WHERE** deben construirse exclusivamente en base a combinaciones de los operadores **AND** y **'='**.

Para mejorar la eficiencia de *SQL* lo más adecuada es utilizar predicados formados por **EQUIJOINS**.



Un **EQUIJOIN** es un **JOIN** que se forma mediante el operador de igualdad. Un **EQUIJOIN** combina las filas que tienen valores equivalentes para las columnas especificadas.

Las consultas que se componen con otro tipo de operadores (**'OR', 'IN', 'EXISTS', ...**) suelen tener un coste mayor.

- **Alternativas**

Es preferible el uso de **EQUIJOINS** en vez de otro tipo de **JOINS** (**OUTERJOINS** con el operador **(+)**), o **PRODUCTOS CARTESIANOS** (tablas sin relacionar en la cláusula **WHERE**).

Cuando se tiene que consultar los datos usando un predicado con **OR**, se puede intentar dividir las condiciones de **OR** en varias consultas con **UNION ALL**, manteniendo la semántica, pero facilitando la ejecución por el **Optimizador**.

 JUNTA DE ANDALUCÍA CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	



5.2. Orden de las tablas



En **join indexados** es conveniente situar aquella tabla con mayor número de registros en el último lugar de la cláusula **from**

Oracle ordena todos los caminos de accesos a las tablas implicadas y elige como tabla conductora aquella con el camino más lento. Compara cada fila de la tabla conductora con las restantes tablas.

En el caso de que Oracle no encuentre un camino claro para seleccionar la tabla conductora, elige como tal la situada en último lugar de la cláusula **from**.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

5.3. Evitar el uso de OUTER JOINS (+)



Limitar el uso de los **OUTER JOINS** en la medida de lo posible.

Un **OUTER JOIN** extiende el resultado de un **JOIN** normal. Devuelve todas las filas que satisfacen la condición de **JOIN** y también algunas o todas las filas de una tabla para las cuales no hay filas en la otra tabla que satisfagan la condición de **JOIN**.

Para escribir una consulta que haga un **OUTER JOIN** de las tablas A y B y devuelva todas las filas de A (**LEFT OUTER JOIN**), se aplica el operador (+) a todas las columnas de B en la condición de **JOIN** en la cláusula **WHERE**. Para todas las filas en A que no tengan la correspondiente fila en B, *Oracle* devuelve **NULL** para cualquier expresión del **SELECT** que contenga columnas de B.

Análogamente, si lo que se quiere hacer es una **OUTER JOIN** sobre las tablas A y B y que devuelva todas las filas de B (**RIGHT OUTER JOIN**), se aplica el operador (+) en todas las columnas de A en la condición de **JOIN** de la cláusula **WHERE**.

- Alternativas**

Se puede optar por componer el mismo resultado con los operadores de conjunto cuando no se vayan a usar adecuadamente los índices debido a que la cantidad de datos que se obtengan sea grande en comparación con el tamaño total de las tablas.

Se puede obtener las diferentes partes de una agrupación usando operadores como **DECODE** o **NVL**.

Se debe intentar que las consultas en las que se use un **OUTER JOIN** sean consultas sencillas.

NO se debe usar **OUTER JOIN** entre vistas o dentro de vistas. El rendimiento del **OUTER JOIN** puede quedar oculto para el **Optimizador**.

- Ejemplos**



Uso del operador de OUTER JOIN

```
SELECT Dep.DepartamentoId,
       Emp.Nombre,
       Emp.Apellidos
FROM Departamento Dep,
     Empleados Emp
WHERE Dep.DepartamentoId = Emp.DepartamentoId (+)
ORDER BY Dep.DepartamentoId;
```

Uso del operador OUTER JOIN para consultar los elementos no coincidentes. Nombres de empleados no asignados a un departamento





```
SELECT Emp.Nombre,
       Emp.Apellidos
FROM Empleados Emp,
     Departamento Dep
WHERE Dep.DepartamentoId (+) = Emp.DepartamentoId
AND Dep.DepartamentoId IS NULL;
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	



Consulta de los elementos no coincidentes con MINUS. Nombres de empleados no asignados a un departamento
<pre> SELECT Emp.Nombre. Emp.Apellidos FROM Empleados Emp MINUS SELECT Emp.Nombre. Emp.Apellidos FROM Empleados Emp, Departamento Dep WHERE Dep.DepartamentoId = Emp.DepartamentoId; </pre>



	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

5.4. Evitar la composición de OUTER JOINS (+) incompletos



Se debe tener la precaución de utilizar el operador (+) en todas las columnas del **JOIN** entre las tablas para componer un **OUTER JOIN**.

Si falta alguna columna por marcar para **OUTER JOIN** se producirá un **JOIN** normal sin que *Oracle* 'avise' que no se está realizando el **OUTER JOIN**, obteniendo así resultados inesperados.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

5.5. Consultas recursivas.



La combinación de las cláusulas **START** y **CONNECT BY** es la forma más adecuada de abordar consultas a tablas de forma recursiva.

El uso de las cláusulas **START** y **CONNECT BY** evita complicadas funciones recursivas. Es trasladable a otros casos de auto joins.

• Ejemplos



Uso de start y connect by

```
SELECT  x_estorg,
        d_estorg,
        LEVEL nivel
FROM    ihestor
START WITH x_estorg = 18610
CONNECT BY PRIOR eo_x_estorg = x_estorg;
```



Forma errónea de acceder a una fila y a sus dependencias



```
SELECT X_BOLTRA
FROM (SELECT X_BOLTRA
      FROM IHBOLTRA
      WHERE BOTR_X_BOLTRA IN (SELECT X_BOLTRA
                             FROM IHBOLTRA
                             WHERE C_BOLTRA LIKE UPPER (P_C_BOLTRA)))

UNION ALL
(SELECT X_BOLTRA
 FROM IHBOLTRA
 WHERE X_BOLTRA IN (SELECT X_BOLTRA
                    FROM IHBOLTRA
                    WHERE C_BOLTRA LIKE UPPER (P_C_BOLTRA))
 AND BOTR_X_BOLTRA IS NULL);
```



Alternativa correcta al ejemplo anterior

```
SELECT X_BOLTRA
FROM IHBOLTRA
START WITH (C_BOLTRA LIKE UPPER (P_C_BOLTRA)
 AND BOTR_X_BOLTRA IS NULL)
CONNECT BY PRIOR X_BOLTRA = BOTR_X_BOLTRA
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

5.6. Cláusulas WHERE y HAVING



Es preferible el uso de **WHERE** al de **HAVING** cuando hay posibilidad de elección.

Únicamente se debe utilizar **HAVING** cuando se requiera un filtrado de las filas obtenidas. En aquellos casos en los cuales podemos especificar una condición bien mediante el uso de **WHERE** o bien mediante el uso de **HAVING**, es preferible optar por **WHERE**. Esto es debido a que es menos costoso en términos de rendimiento restringir el número de filas a tratar (**WHERE**) que filtrar sobre las filas obtenidas una vez realizada la operación de grupo (**HAVING**).

• Ejemplos





Consulta empleando la cláusula HAVING

```
SELECT edad, COUNT(*)
FROM empleados
GROUP BY edad
HAVING edad = 50;
```



Consulta empleando la cláusula WHERE

```
SELECT edad, COUNT(*)
FROM empleados
WHERE edad = 50
GROUP BY edad;
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

5.7. Subconsultas anidadas. ¿EXISTS o IN?



En general, es preferible el uso de **EXISTS** al de **IN**. Sin embargo, en determinadas circunstancias es mejor usar **IN**: si el predicado por el que se filtra (predicado selectivo) está en la subconsulta.

En algunas ocasiones *Oracle* puede reescribir una subconsulta cuando se usa una cláusula **IN** para aprovechar el subconjunto de filas seleccionado en la subconsulta. Esto es más beneficioso cuando más selectiva sea la subconsulta y más índices se usen en las columnas de los **JOINS**.

El uso de **EXISTS** es beneficioso cuando se filtran más filas a través de la consulta padre. Esto permite aplicar antes los predicados más selectivos de la consulta padre.

Cuando una subconsulta referencia a una columna de una tabla que está en la consulta padre, *Oracle* realiza una subconsulta correlacionada (**CORRELATED SUBQUERY**). Una consulta correlacionada es evaluada para cada fila procesada por la sentencia.

• Ejemplos



Ejemplo con EXISTS con predicado selectivo (PS) en la subconsulta

```
SELECT Emp.EmpleadoId, Emp.Nombre, Emp.Apellido, Emp.Sueldo
FROM Empleados Emp
WHERE EXISTS (SELECT 1
              FROM Pedidos Ped
              WHERE Emp.EmpleadoId = Ped.ComercialId
              AND Ped.ClienteId = 144);          /* PS */
```



Ejemplo con IN con predicado selectivo (PS) en la subconsulta

```
SELECT Emp.EmpleadoId, Emp.Nombre, Emp.Apellido, Emp.Sueldo
FROM Empleados Emp
WHERE Emp.EmpleadoId IN (SELECT Ped.ComercialId
                        FROM Pedidos Ped
                        WHERE Ped.ClienteId = 144);          /* PS */
```





Ejemplo con IN con predicado selectivo (PS) en la sentencia padre

```
SELECT Emp.EmpleadoId, Emp.Nombre, Emp.Apellido, Emp.Sueldo
FROM Empleados Emp
WHERE Emp.DependienteId = 80 AND /* PS */
      Emp.TipoTrabajo = 'COMERCIAL' AND /* PS */
      Emp.EmpleadoId IN (SELECT Ped.ComercialId
                        FROM Pedidos Ped);
```



Ejemplo con EXISTS con predicado selectivo (PS) en la sentencia padre

```
SELECT Emp.EmpleadoId, Emp.Nombre, Emp.Apellido, Emp.Sueldo
FROM Empleados Emp
WHERE Emp.DependienteId = 80 AND /* PS */
      Emp.TipoTrabajo = 'COMERCIAL' AND /* PS */
      EXISTS (SELECT 1
              FROM Pedidos Ped
              WHERE Emp.EmpleadoId = Ped.ComercialId);
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

5.8. No usar "EXISTS" en lugar de "JOIN"



No se debe usar **EXISTS** en sustitución de condiciones triviales de **Join**.

No se debe utilizar *EXISTS* cuando la solución lógica se obtiene mediante un simple **JOIN** entre tablas. Las consultas sobre varias tablas que están relacionadas mediante columnas se resuelven con **JOINS** entre las tablas, es decir mediante operadores de igualdad en las condiciones de búsqueda de la cláusula **WHERE**. Sustituir esta técnica por el uso de *EXISTS* solamente redundará en un peor rendimiento de la base de datos salvo que la solución adecuada para la consulta sea mediante *EXISTS*.

- **Alternativas**

Se puede usar siempre *EXISTS* en el caso de optimizar los accesos usando la técnica de filtrar por una consulta padre que se relaciona con *EXISTS* con una subconsulta.

- **Ejemplos**



Ejemplo TRIVIAL con solución INADECUADA

```
SELECT Empleados.Nombre, Empleados.Apellidos
FROM Empleados
WHERE EXISTS (SELECT '1'
              FROM Municipio
              WHERE EmpleadoIdMunicipio=IdMunicipio
              AND Municipio.Nombre = 'SEVILLA');
```



Ejemplo TRIVIAL con solución RECOMENDADA

```
SELECT Empleados.Nombre, Empleados.Apellidos
FROM Empleados ,Municipio
WHERE Empleados.IdMunicipio = IdMunicipio
AND Municipio.Nombre = 'SEVILLA';
```


 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	





Ejemplo COMPLEJO con solución INEFICIENTE

```
SELECT COUNT(*)
  FROM NifExp,Expedi
 WHERE NifExp.SujCir_CodCircu = 'ITPAJD'
    AND NifExp.SujCir_CodTipSu = 'DE'
    AND Expedi.Circui_CodCircu = NifExp.Expedi_CodCircu
    AND Expedi.CodTer_CodTerri = NifExp.Expedi_CodTerri
    AND Expedi.EjeExped = NifExp.Expedi_EjeExped
    AND Expedi.NumExped = NifExp.Expedi_NumExped
    AND Expedi.FecApeEx BETWEEN TO_DATE('01/01/1998','DD/MM/YYYY') AND
                                TO_DATE('27/04/2000','DD/MM/YYYY')
```



Ejemplo COMPLEJO con solución EFICIENTE

```
SELECT COUNT(*)
  FROM NifExp
 WHERE NifExp.SujCir_CodCircu = 'ITPAJD'
    AND NifExp.SujCir_CodTipSu = 'DE'
    AND EXISTS (SELECT '1'
                  FROM Expedi
                 WHERE Circui_CodCircu = NifExp.Expedi_CodCircu
                   AND CodTer_CodTerri = NifExp.Expedi_CodTerri
                   AND EjeExped = NifExp.Expedi_EjeExped
                   AND NumExped = NifExp.Expedi_NumExped
                   AND Expedi.FecApeEx BETWEEN
                     TO_DATE('01/01/1998','DD/MM/YYYY') AND
                     TO_DATE('27/04/2000','DD/MM/YYYY')
                )
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

5.9. EXISTS frente a DISTINCT



El uso de **EXISTS** es preferible al de **DISTINCT**

Una de las acciones más costosas para un motor de bases de datos es la realización de ordenaciones de los resultados obtenidos, sobre todo en los casos en los que hay más de una tabla implicada. Cuando requerimos un conjunto de filas no repetidas y aplicamos para ello la palabra clave **DISTINCT**, obligamos a realizar una ordenación previa de los resultados

- **Alternativas**

Cuando la consulta a realizar es sobre una única tabla se puede utilizar la palabra clave **DISTINCT**.

- **Ejemplos**





Empleados con puesto asignado. Uso de **DISTINCT**

```
SELECT DISTINCT Num_emple, nombre, apellidol
FROM Empleados e, Puestos p
WHERE e.Num_emple = p.Num_emple ;
```



Empleados con puesto asignado. Uso óptimo de **EXISTS**

```
SELECT Num_emple, nombre, apellidol
FROM Empleados e
WHERE EXISTS ( SELECT 1
FROM Puestos p
WHERE e.Num_emple = p.Num_emple ) ;
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

5.10. NOT, '!=' y NOT IN



Se debe evitar siempre que sea posible el uso de los operadores **NOT** y **'!='** sobre columnas o subconsultas.

Estos operadores en la mayoría de los casos inutilizan los índices. Un caso extremo se produce cuando se utiliza **NOT** en combinación con **IN**. **NOT IN** es muy ineficiente cuando se usa para obtener los valores de una serie de tablas que no están contenidos en el resultado de otras subconsultas.

- Alternativas**

Normalmente se puede transformar un **NOT IN** de una consulta que maneje una gran cantidad de datos por unas consultas que obtengan el mismo conjunto de filas resultado, mediante el uso de operadores de conjunto como **MINUS**.

- Ejemplos**



Consulta con NOT IN

```
SELECT Nombre, Apellidos
FROM Empleados
WHERE EmpleadosIdMunicipio NOT IN
(SELECT IdMunicipio
FROM Municipio, Provincia
WHERE MunicipioIdProvincia=IdProvincia
AND Provincia.Nombre='GRANADA')
```

Si el número de filas de Municipio es menor que el número de filas de Empleados y se van a obtener pocas filas de Empleados pero de la mayoría de los municipios, se puede optar por hacer un recorrido completo de la tabla Municipio ignorando índices:



Consulta con MINUS (N° Municipios < N° Empleados)

```
SELECT Nombre, Apellidos
FROM Empleados,
(SELECT IdMunicipio
FROM Municipio
MINUS
SELECT IdMunicipio
FROM Municipio, Provincia
WHERE Municipio.IdProvincia = IdProvincia
AND Provincia.Nombre='GRANADA') muni
WHERE Empleados.IdMunicipio = muni.IdMunicipio
```

Si por el contrario, es menor el número de filas de la tabla Empleados que de la tabla Municipio, o se va a obtener un porcentaje considerable de las filas de la tabla Empleados, una transformación más óptima sería:

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	





Consulta con MINUS (Nº Empleados < Nº Municipios)

```

SELECT Nombre, Apellidos
  FROM Empleados
 MINUS
SELECT Nombre, Apellidos
  FROM Empleados, Municipio, Provincia
 WHERE EmpleadosIdMunicipio = IdMunicipio
    AND MunicipioIdProvincia=IdProvincia
    AND Provincia.Nombre='GRANADA'

```

Para poder decidir cual es la solución mejor, se debe revisar el **Explain Plan**.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

5.11. Operadores de conjuntos (UNION, MINUS, INTERSEC)



El uso de los operadores de conjuntos (UNION, MINUS, INTERSEC) se recomienda en cierto tipo de consultas frente a otras alternativas.



UNION ALL, *MINUS* e *INTERSEC* son operadores que dan muy buenos resultados en cuanto a eficiencia.

El objetivo que se suele buscar usando los operadores de conjunto consiste en utilizar consultas sencillas y óptimas de subconjuntos de datos, que formen el conjunto completo de los datos con los operadores de conjunto de manera que si se resolviese con una única sentencia sin operadores de conjunto, resultaría más costoso.

No se debe usar *UNION ALL* en el caso de consultas no excluyentes, ya que no elimina las filas duplicadas (a no ser que se tenga en cuenta y se quieran obtener esos duplicados).

El operador *UNION ALL* simplemente concatena los resultados de las consultas que une. Cuando se simplifica una consulta dividiéndose ésta en varias subconsultas unidas por el operador de conjunto *UNION ALL*, se obtiene una consulta más óptima.

El uso del operador de conjunto *UNION* es menos óptimo que la utilización del operador *UNION ALL* debido a que se eliminan las filas duplicadas resultantes de la unión de las consultas. Para ello requiere un paso adicional respecto a *UNION ALL*, la ordenación de las filas obtenidas.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

5.12. Uso de RETURNING



RETURNING es la mejor opción para recuperar información sobre filas modificadas.

RETURNING permite recuperar información sobre las filas que se acaban de modificar mediante un INSERT, UPDATE o DELETE. Permite realizar en un solo paso lo que antes precisaba de dos (UPDATE y SELECT por ejemplo).

- Ejemplo





Recuperación de información en dos pasos sucesivos

```
INSERT INTO PERSONAS (PERSONA_ID, NOMBRE, APELLIDO)
VALUES (PERSONAS_SEQ.NEXTVAL, 'ALBERTO', 'GARCÍA');
-- Recuperamos el Id obtenido
SELECT PERSONA_ID INTO V_PERSONA_ID
FROM PERSONAS
WHERE NOMBRE = 'ALBERTO'
AND APELLIDO = 'GARCÍA'
```



Recuperación de información en un solo paso

```
INSERT INTO PERSONAS (PERSONA_ID, NOMBRE, APELLIDO)
VALUES (PERSONAS_SEQ.NEXTVAL, 'ALBERTO', 'GARCÍA')
RETURNING PERSONA_ID INTO V_PERSONA_ID
```

	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

6. Otras recomendaciones para el acceso a datos

6.1. Uso de vistas





Las vistas pueden llegar a optimizar mucho una aplicación, pero un uso inadecuado puede provocar unos costes excesivos. Hay que tener mucho cuidado cuando se reutiliza una vista existente para un nuevo objetivo.

Se debe intentar que las vistas sean sencillas y específicas para la tarea que se diseña.

- **Alternativas**

Por norma general se debe evitar hacer **JOINS** de vistas complejas.

No es una buena práctica reciclar una vista para un propósito diferente al original por el que fue diseñada. Consultar una vista implica la consulta de todas sus tablas y el acceso a sus datos para devolver el resultado. Hay que constatar que no se usa en esa vista más tablas de las necesarias, ni se recorre un subconjunto de datos mucho más amplio del necesario.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

6.2. Tablas temporales



Considerar la conveniencia de utilizar tablas temporales

Las tablas temporales tienen dos características que hacen de ellas una opción interesante:

1. Sus datos son accesibles por una única sesión, aquella que los ha insertado.
2. Las filas de datos insertadas se borran al cerrar la sesión (siempre) o al efectuar un *commit*, dependiendo de cómo se haya definido en la creación de la tabla

Por tanto evitan la concurrencia de varias sesiones a un mismo conjunto de datos. Conservan las ventajas de las tablas estándar al admitir índices y restricciones.

• Ejemplos



Tabla temporal que se borra al hacer commit

```
SQL> CREATE GLOBAL TEMPORARY TABLE TT ( COLUMN VARCHA2(200) )
      ON COMMIT DELETE ROWS;
SQL> INSERT INTO TT VALUES ( 'SESSION 1' );
SQL> SELECT * FROM TT;

COLUMNA
-----
SESSION 1

SQL> COMMIT;

Validación terminada.



SQL> SELECT * FROM TT;

ninguna fila seleccionada
```



EJEMPLO: Tabla temporal de tipo on commit preserve rows atacada por dos sesiones distintas

SESSION 1	SESSION 2
SQL> INSERT INTO VALUES('SESSION 1');	
SQL> COMMIT;	<- Después del commit de la sesión 1.
SQL> SELECT * FROM TT;	SQL> SELECT * FROM TT;
COLUMNA ----- SESSION 1	ninguna fila seleccionada

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

6.3. Sentencias *SQL* sobrecargadas



Se debe evitar agrupar demasiada funcionalidad en una única sentencia *SQL*.

SQL no es un lenguaje procedimental. Usando una única sentencia que haga diferentes tareas en función de parámetros puede resultar menos óptimo que tener una sentencia dedicada a cada tarea específica.



- **Alternativas**

- Uso de paquetes *PL/SQL*

PL/SQL sí es un lenguaje procedimental. El uso de paquetes *PL/SQL* está totalmente recomendado con sentencias *SQL* específicas, ya que se ejecutan siempre del lado del servidor. Además se procesan en el servidor de una sola vez, devolviendo el servidor asimismo el resultado del proceso de una sola vez.

- La alternativa del **UNION ALL**

Cuando no queda más remedio que hacer una única sentencia *SQL* que haga muchas tareas, hay que considerar la alternativa de utilizar **UNION ALL** entre varias sentencias pequeñas y específicas que sean excluyentes.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

6.4. %TYPE y %ROWTYPE



Cuando se declaran variables sobre objetos de la base de datos se aconseja definir su tipo de datos mediante el atributo `%TYPE` o `%ROWTYPE`. También es recomendable en la definición de los parámetros de los procedimientos y funciones.

El atributo `%TYPE` proporciona el tipo de dato de una variable o de una columna de una tabla de forma dinámica.

Para facilitar el mantenimiento del código es beneficiosa la utilización de `%TYPE` ya que si cambia el tipo de datos de las columnas de las tablas, el código no necesita ser revisado.

El atributo `%ROWTYPE` representa a una fila de una tabla o una vista en un registro. Las columnas de las filas y los respectivos campos del registro tienen el mismo nombre y tipo de dato de su origen. Muy adecuado en la definición de cursores.

Con el uso de `%TYPE` no se heredan las restricciones `NOT NULL` de las columnas.



La utilización de `%ROWTYPE` en módulos ejecutables (Developer) no está aconsejada, puede provocar el error ORA-01007 en tiempo de ejecución al añadir o eliminar una columna en la tabla de referencia.

• Ejemplos



Ejemplo en el que se ve la declaración de una variable y como no le afecta que la columna de la tabla tenga restricciones sobre valores nulos o no.

```
DECLARE
    my_sueldo Empleados.sueldo%TYPE;
...
BEGIN
    my_sueldo := NULL; -- esto funciona
...

```



Ejemplo de declaración usando `%ROWTYPE`



```
DECLARE
    RegistroEmpleado Empleados%ROWTYPE;
    CURSOR c1 IS
        SELECT DepartamentoId, Nombre FROM Departamentos;
    RegistroDepartamento c1%ROWTYPE;
...

```



Uso de `ROWTYPE` en Forms

```
declare
    vr_registro ihtabla%rowtype;
    cursor cul is
        select * from tabla;
begin
    open cul;
    fetch cul into vr_registro;
    -- hagamos algo con la variable vr_registro.
    ...
end;
```

 JUNTA DE ANDALUCÍA CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas 06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

6.5. Uso de SQL dinámico nativo



Para los programas que elaboran las sentencias *SQL* en tiempo de ejecución (**SQL Dinámico**) es conveniente la utilización de **SQL Dinámico Nativo**.



En versiones anteriores a *Oracle8i* para ejecutar **SQL Dinámico** había que utilizar el paquete **DBMS_SQL**. Ahora se puede ejecutar una gran variedad de sentencias *SQL* dinámicas con un nuevo interfaz llamado **SQL Dinámico Nativo**. Es más fácil de usar y más rápido que el paquete.

- **Ejemplos**



```

DECLARE
TYPE EmpCurTyp IS REF CURSOR;
emp_ct EmpCurTyp;
mi_nombre VARCHAR2(15);
mi_sueldo NUMBER := 1000;
BEGIN
    OPEN emp_ct FOR 'SELECT nombre, sueldo
                    FROM Empleados WHERE sueldo > :s'
                    USING mi_sueldo;
    ...
END;
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

6.6. Uso del carácter comilla dentro de cadenas



Cuando se elaboran sentencias *SQL* con el carácter comilla simple (') dentro de una cadena es conveniente la especificación este carácter mediante su código *ASCII* (*CHAR(39)*) en lugar de anteponer otra comilla simple para que el *parser* de *Oracle* haga la conversión automática.

Normalmente cuando se tienen que introducir comillas dentro de cadenas se tiende a usar dos comillas simples para evitar que se interprete la comilla interior como la comilla de fin de cadena. Dos son los motivos por los que no se recomienda esta técnica:

- En primer lugar el código generado puede resultar del todo poco legible, sobre todo en ciertos editores de código.
- En segundo lugar, esta técnica puede provocar problemas en el servidor cuando la sentencia es compleja.

• Alternativas


Especificar el código ASCII del carácter comilla.

• Ejemplos

Es muy común la obtención de cadenas especialmente complejas y difíciles de tratar cuando se generan sentencias *SQL* en modo texto dinámicamente basándose en valores de variables.



```
INSERT INTO sentencia
SELECT 'INSERT INTO PARAM_T
      (CODPARAM, NOMPARAM, DESPARAM, FORPARAM,
       LONPARAM, INDINTER, LONPARAE, INDfesti) ' ||
      'VALUES ('' || codparam || ''','' ||
nomparam || ''','' || desparam ||
''','' || forparam || ''','' ||
REPLACE(TO_CHAR(lonparam,'9999d99'),' ','.') || ','' ||
indinter || ''','' ||
REPLACE(TO_CHAR(lonparae,'9999d99'),' ','.') || ','' ||
indfesti || ''')';
s_traza.NEXTVAL ,
SYSDATE
FROM paramet p
WHERE codparam = p_codparam ;
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	



```

INSERT INTO sentencia
SELECT 'INSERT INTO PARAM_T
      (CODPARAM, NOMPARAM, DESPARAM, FORPARAM,
       LONPARAM, INDINTER, LONPARAE, INDFESTI) ' ||
      'VALUES (' || CHR(39) || codparam || CHR(39) || ',' ||
CHR(39) || nomparam || CHR(39) || ',' || CHR(39) || desparam ||
CHR(39) || ',' || CHR(39) || forparam || CHR(39) || ',' ||
REPLACE(TO_CHAR(lonparam,'9999d99'),' ','.') || ',' ||
CHR(39) || indinter || CHR(39) || ',' ||
REPLACE(TO_CHAR(lonparae,'9999d99'),' ','.') || ',' ||
CHR(39) || indfesti || CHR(39) || ');',
      s_traza.NEXTVAL ,
      SYSDATE
FROM   param p
WHERE  codparam = p_codparam ;

```

A continuación se usa un ejemplo más sencillo para mostrar la técnica de sustitución aplicada con mayor claridad.



```

SELECT '''Texto entrecomillado'''
FROM   DUAL;



```



```

SELECT CHAR(39) || 'Texto entrecomillado' || CHAR(39)
FROM   DUAL;

```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

6.7. Comprobaciones de existencia de filas



Para comprobar si una consulta devuelve o no filas se recomienda utilizar un cursor cuya lectura se realice sin definir un bucle

Es una consulta habitual detectar la existencia de registros en una o varias tablas cumpliendo ciertas condiciones. De todas las formas posibles de implementar este comportamiento, la óptima es mediante la lectura de la primera fila de un cursor.

• Ejemplos



No es necesario recorrer la tabla completa

```
DECLARE
  CURSOR CU_TEST( P_CODIGO ) IS
    SELECT COUNT(*)
    FROM IHDETALLE
    WHERE CDG_C_CODIGO = P_CODIGO;
  V_AUX NUMBER;
BEGIN
  OPEN CU_TEST( IH_FU_VALOR('CTAR') );
  FETCH CU_TEST INTO V_AUX;
  CLOSE CU_TEST;
  IF V_AUX>0 THEN
    IH_PR_PROCESO;
  END IF;
  ...
END;
```



Como en el caso anterior, no es necesario recorrer la tabla completa. Además excluye el caso de valores nulos

```
DECLARE
  CURSOR CU_TEST( P_CODIGO ) IS
    SELECT SUM(N_IMPORTE)
    FROM IHDETALLE
    WHERE CDG_C_CODIGO = P_CODIGO;
  V_AUX NUMBER;
BEGIN
  OPEN CU_TEST( IH_FU_VALOR('CTAR') );
  FETCH CU_TEST INTO V_AUX;
  CLOSE CU_TEST;
  IF V_AUX>0 THEN
    IH_PR_PROCESO;
  END IF;
  ...
END;
```

Evita recorrer la tabla en su totalidad usando **Fetch** sin bucle. Además no realiza operaciones ni aplica funciones. Tampoco realiza operaciones de grupo

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	



```



DECLARE
  CURSOR CU_TEST( P_CODIGO ) IS
    SELECT 1
      FROM IHDETALLE
     WHERE CDG_C_CODIGO = P_CODIGO;
  V_AUX NUMBER :=NULL;
BEGIN
  OPEN CU_TEST( IH_FU_VALOR('CTAR') );
  FETCH CU_TEST INTO V_AUX;
  CLOSE CU_TEST;
  IF V_AUX IS NOT NULL THEN
    IH_PR_PROCESO;
  END IF;
  ...
END;
```



Alternativa más eficaz respecto al optimizador de Oracle

```

DECLARE
  CURSOR CU_TEST( P_CODIGO ) IS
    SELECT 1
      FROM DUAL
     WHERE EXISTS (
        SELECT 1
          FROM IHDETALLE
         WHERE CDG_C_CODIGO = P_CODIGO);
  V_AUX NUMBER :=NULL;
BEGIN
  OPEN CU_TEST( IH_FU_VALOR('CTAR') );
  FETCH CU_TEST INTO V_AUX;
  CLOSE CU_TEST;
  IF V_AUX IS NOT NULL THEN
    IH_PR_PROCESO;
  END IF;
  ...
END;
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
06. Manual de Instrucciones Técnicas		
06.01. Normas y Recomendaciones de Accesos a Bases de Datos		

7. Transacciones y Actualizaciones

7.1. ¿Actualizaciones masivas en una única sentencia?



Cuando la base de datos está bien dimensionada y con suficientes recursos, una actualización de un número de filas considerable en una única sentencia es aceptable dado que el servidor podrá resolver la sentencia eficientemente.

Cuando una máquina está bien dimensionada para consultas y actualizaciones concurrentes de un tamaño considerable, no hay problema en lanzar actualizaciones de grupos de filas de un tamaño aceptable dado que el servidor dispone de mecanismos para facilitar la ejecución de esas consultas y las actualizaciones.



La ejecución de una actualización de datos masiva en una única sentencia limita los recursos de **CPU** y espacio de **Rollback** disponibles para otros usuarios de manera considerable. Sin embargo tiene la ventaja de llevar al servidor la mayor carga de trabajo, reduciendo el tráfico en la red.

Cuando se realiza una actualización uno de los recursos principales que se consume es el espacio de **Rollback**. Este espacio lo utiliza el gestor de base de datos para guardar información de las transacciones que realiza cada usuario de manera que si el usuario ejecuta el comando `ROLLBACK` se deshacen los cambios ejecutados durante la transacción. En cambio si se ejecuta el comando `COMMIT` la actualización será definitiva liberándose el espacio de **Rollback** utilizado durante la transacción. Además mientras que un usuario está modificando los datos de una serie de tablas en una transacción, el resto de usuarios pueden obtener lecturas de datos consistentes gracias a que el gestor de base de datos utiliza la información del espacio de **Rollback** para devolver los datos tal como estarían sin la modificación que está en curso. Todo este proceso implica tiempo de **CPU** y mucha entrada y salida de disco.

- **Alternativas**

Si al ejecutar una actualización excesivamente grande para el servidor de base de datos se produjera algún tipo de error referente al espacio de **Rollback** o se resintiera la aplicación se deberían barajar otras alternativas no triviales de codificar.

En principio habría que estudiar el rendimiento de cada opción ya que aquí no hay dos consultas para ver cual es más óptima con el **Explain Plan** si no que son métodos totalmente diferentes. Hay que sopesar siempre la dificultad de desarrollo con la necesidad de optimización.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

7.2. COMMIT dentro de un cursor (Fetch across commit)



Se recomienda no ejecutar la sentencia `COMMIT` dentro de ningún tipo de cursor cuando se están actualizando los datos de las tablas implicadas en el cursor. Se debe esperar a terminar de ejecutar `FETCH` sobre él.



Oracle permite ejecutar la sentencia `COMMIT` dentro de un **Cursor** aún no siendo del tipo `CURSOR FOR UPDATE`. Si las tablas que se actualizan son las mismas sobre las que se consulta, se corre el riesgo de que se produzca el error: `ORA-01555 Snapshot too old` porque los segmentos de rollback son cíclicos y una vez que se ejecuta la sentencia `COMMIT` se pueden sobrescribir cuando lo necesite el sistema.

• Alternativas

Se debe optar por soluciones que ejecuten **COMMIT** dentro de un cursor que lee los datos que se actualizan cuando tenga importancia el rendimiento y se controlen los posibles errores.

Cuando surgen problemas de *"Snapshot too old"* se puede optar por una de las siguientes soluciones:

- Reescribir el programa para impedir que se ejecuten `COMMIT` dentro de un cursor.
- Ejecutar con menor frecuencia los `COMMIT`.

	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Manual de Instrucciones Técnicas 06.01. Normas y Recomendaciones de Accesos a Bases de Datos	



7.3. NOWAIT



Si se tiene que evitar que un proceso quede bloqueado a la espera de que libere algún recurso bloqueado por *Oracle* se puede usar la palabra clave *NOWAIT* para que no se espere en algunas situaciones.

La sentencia *SELECT FOR UPDATE* bloquea recursos pero tiene que esperar a que los recursos queden libres si previamente estaban bloqueados.

Cuando no interesa que esta espera se produzca y es preferible abandonar el intento de actualización, se pueden usar estas sentencias con la palabra clave *NOWAIT*, forzando un error (*EXCEPTION*) en caso de que los recursos estén bloqueados, devolviendo el control al procedimiento para que se puedan efectuar las acciones alternativas.



 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

7.4. CURSOR FOR UPDATE y CURRENT OF



Se debe limitar el uso de *CURSOR FOR UPDATE* a los casos en los que sea imprescindible bloquear las filas del cursor y no se debe ejecutar la sentencia *COMMIT* o la sentencia *ROLLBACK* en medio del bucle del **Cursor**.

Cuando se crea un **cursor** del tipo *CURSOR FOR UPDATE* se bloquean todas las filas de las tablas implicadas que va a devolver el **cursor** en el momento de la creación, no cuando se lee la fila con *FETCH*. Este bloqueo impedirá a otros usuarios acceder al mismo recurso y sobrecarga el sistema al tener que usarse el espacio de **Rollback** para que el resto de usuarios obtengan lecturas consistentes. No se debe usar este tipo de **cursores** cuando se trate de actualizaciones masivas.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

7.5. Ejecutar COMMIT O ROLLBACK al terminar la transacción



Es indispensable la gestión explícita de las transacciones mediante el uso de las sentencias *COMMIT* o *ROLLBACK*.

La no utilización de manera adecuada de las sentencias *COMMIT* o *ROLLBACK* entre transacciones pueden generar datos inconsistentes en la base de datos, provocando bloqueos por transacciones pendientes en otras sesiones. Además, las transacciones pendientes estarán supeditadas a la ejecución de *COMMIT* o *ROLLBACK* en otros procesos que se ejecuten con posterioridad.

En caso de no ejecutar las sentencias *COMMIT* o *ROLLBACK* en el momento de cerrar la sesión, se deja en manos de la configuración de la herramienta utilizada la forma de finalizar la transacción.

- **Alternativas**

Se pueden utilizar transacciones autónomas cuando se necesite tener una transacción independiente dentro de una transacción general del proceso.



Se pueden deshacer parcialmente los cambios hasta un punto *SAVEPOINT* que sirve para nombrar y marcar un punto concreto dentro de una transacción.

- **Ejemplos**



Ejemplo de uso de SAVEPOINT

```
DECLARE
Emp_Id Empleados.EmpleadoId%TYPE;
BEGIN
UPDATE Empleados SET ... WHERE EmpleadoId = Emp_Id;
DELETE FROM Empleados WHERE ...
...
SAVEPOINT Do_Insert;
INSERT INTO Empleados VALUES (Emp_Id, ...);
EXCEPTION
WHEN DUP_VAL_ON_INDEX THEN
ROLLBACK TO Do_Insert;
END;
```

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas 06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

7.6. No insertar valores sin especificar los campos



Cuando se insertan datos en una tabla, es obligatorio especificar los nombres de los campos en los que se insertan los datos.

Si se insertan datos en una tabla sin especificar los campos, se dificulta la lectura del código para su mantenimiento y se deja un código que provocará errores con cualquier modificación sobre las columnas de las tablas.

• Ejemplo



Insertión sin especificar los campos que se insertan

```
INSERT INTO IHERRORES
VALUES
('APL','10234','Error al consultar','W','N','S',NULL);
```



Insertión especificando los campos que se insertan

```
INSERT INTO IHERRORES(C_ORIERR,C_ERROR,D_ERROR, T_ERROR
,L_ALERTA,L_SUPWRN,T_AMPERR)
VALUES
('APL','10234','Error al consultar','W','N','S',NULL);
```



En el caso de que el origen de los datos sea una variable registro, se permite no especificar los campos. Esta variable deberá estar definida como %rowtype de la tabla.

• Ejemplo



```
P_R_DATRECPROIT IN IHDATRECPROIT%ROWTYPE;
```

```
INSERT INTO IHDATRECPROIT
VALUES P_R_DATRECPROIT;
```



 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

7.7. Encapsulamiento de INSERT, UPDATE y DELETE



Este tipo de sentencias no es conveniente implementarlas directamente en el código

Si en su lugar situamos en procedimientos o paquetes estas sentencias conseguimos que las aplicaciones vayan más rápido, al reducir el tiempo de parsing y la demanda de memoria SGA

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
06. Manual de Instrucciones Técnicas		
06.01. Normas y Recomendaciones de Accesos a Bases de Datos		



A I. Anexo: Análisis de los Planes de Ejecución

El objetivo de este apartado es la iniciación en el análisis de los planes de ejecución de las sentencias **SQL** para ayudar a mejorar el rendimiento de sentencias costosas y conseguir sentencias **SQL** más eficientes en el futuro. El plan de ejecución es la representación de los caminos que toma **Oracle** cuando ejecuta una consulta. Al calcularlo, **Oracle** reescribe la sentencia para obtener la solución más óptima posible a partir de ella, pero un mal punto de partida implica una mala solución. Hay que estudiar bien la solución que se va a adoptar y los planes de ejecución ayudan a comparar el coste de las diferentes soluciones.

Este documento se orienta a versiones de **Oracle** basadas en el **Optimizador Basado En Coste (CBO)**. Cuando **Oracle** no dispone de estadísticas (las tablas no están analizadas) se usa el **Optimizador Basado En Reglas (RBO)** que usa un conjunto de técnicas heurísticas para determinar el mejor camino para resolver la consulta. Este documento trata los aspectos relacionados con la optimización de sistemas **OLTP** (transaccionales), no entra en analizar aspectos de sistemas **Data Warehouse** o de sistemas de ayuda a la decisión (**DSS**).

En este apartado se va a proceder a dar una explicación que ayude a interpretar los resultados del **Explain Plan**. Un ejemplo de salida del **Explain Plan** podría ser:

ID	OPERATION	OPTIONS	OBJECT_NAME	OPTIMIZER	COST
0	SELECT STATEMENT			CHOOSE	756
1	SORT	AGREGATE			
2	INDEX	FAST FULL SCAN	CASVAL_PK	ANALYZED	756

 JUNTA DE ANDALUCÍA CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
06. Manual de Instrucciones Técnicas 06.01. Normas y Recomendaciones de Accesos a Bases de Datos		

A I.1. Jerarquía de un plan de ejecución

La operación que está más a la derecha y más arriba del plan de ejecución es la primera que se ejecutará. En los ejemplos que se han mostrado sería: INDEX FAST FULL SCAN CASVAL_PK. Esto significa que se está recorriendo las columnas de la clave primaria y se está devolviendo los datos sin ordenar. Cuando esta operación termina el conjunto de filas se pasa al siguiente nivel: SORT AGGREGATE. Esta operación no implica necesariamente una ordenación. Se usa cuando se calculan funciones de agregación como en la consulta del ejemplo:

```
SELECT COUNT(*) FROM CASVAL;
```

Después los datos resultantes de esta operación pasan al siguiente nivel, en este caso el nivel superior de la consulta: SELECT STATEMENT.

CHOOSE es una indicación de que el optimizador puede que esté configurado como **CBO**. Esto se confirma si la consulta tiene un valor no nulo para el coste. Si el coste es nulo, se habrá calculado el plan de ejecución mediante el **RBO**.

ANALYZED indica que el objeto está analizado y se dispone de estadísticas para que las utilice el **CBO**.

El valor del coste de la consulta es un valor estimado basándose en las estadísticas del diccionario sobre la distribución de los datos y las características de las tablas, índices y particiones que se usan en la sentencia.



En realidad las herramientas obtienen una estimación del **coste**, la **cardinalidad** y los **bytes** a los que accederá la sentencia:

El **coste** es una **estimación** hecha por el optimizador en función del coste de CPU y el coste de operaciones de Entrada/Salida.

La **cardinalidad** es una **estimación** del número de filas que devuelve o a los que se accede con la sentencia.

Los **bytes** son una **estimación** del número de bytes recorridos para obtener el resultado de la ejecución de la sentencia.

Esta información se usa internamente para comparar los costes de las diferentes alternativas y elegir el mejor plan de ejecución. El coste es proporcional a los recursos estimados que se necesitarán para ejecutar la sentencia con el plan de ejecución calculado. El plan de ejecución y el coste puede variar cuando se usa **CBO** entre los diferentes entornos de ejecución. El coste de dos sentencias diferentes no es comparable directamente en principio, aunque en la mayoría de los casos da una idea muy aproximada del comportamiento de las sentencias. De todas formas hay que tener en cuenta siempre las recomendaciones para la realización de consultas óptimas con **CBO** porque hay casos en los que el valor del coste puede llevar a confusión.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

A I.2. Métodos de acceso

AI I.2.1. Full Table Scan (FTS)

Cuando se produce un **FTS**, se lee la tabla completa hasta la **high water mark (HWM)**. La **HWM** marca el último bloque de la tabla que se ha escrito alguna vez. Si se borran todas las filas de la tabla, la **HWM** no varía y se sigue leyendo hasta ella. Si se trunca la tabla (**TRUNCATE TABLE**) se vuelve a colocar la **HWM** al principio de la tabla.

Los buffers de **FTS** se colocan al final del **buffer cache**, que se rige por el algoritmo **LRU (Least Recently Used)** de manera que rápidamente salen del caché.

FTS utiliza lecturas multibloque.

FTS no se recomienda para tablas grandes salvo que se estén leyendo mas del 5% o 10% de los datos.

Ejemplo de FTS con autotrace	
SQL> select * from DUAL;	
D	
-	
X	
Execution Plan	

0	SELECT STATEMENT Optimizer=CHOOSE
1 0	TABLE ACCESS (FULL) OF 'DUAL'

AI I.2.2. Index Scans

Se accede a los datos buscando las claves en un índice y devolviendo **rowids**. Un **rowid** identifica unívocamente una fila. Si toda la información está en el índice puede que no haga falta obtener datos de la tabla y no se produce accesos a la tabla. Las búsquedas por índices usan lecturas individuales de bloques de datos. En caso contrario se accederá a la tabla normalmente por el **rowid** devuelto. Hay cuatro métodos basados en índices para acceder:



- Index Unique Scan

Método por el que se busca un valor clave a través de un índice único. Siempre devuelve un único valor. Cuando se trata de un índice múltiple, se tiene que proporcionar las columnas más significativas del índice para obtener el dato por este método y puede que se devuelva más de una fila al no tener garantizada la unicidad.

- Index Range Scan

Con este método se acceden a varios valores de una columna. Como siempre si el índice es múltiple se tiene que proporcionar las columnas más significativas para que se use. Este método se suele usar en las operaciones de rangos (>, <, <>, >=, <=, between).

Si el índice no es único se devolverán varios valores a través del índice.

 JUNTA DE ANDALUCÍA CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

- **Index Full Scan**

Este método escanea el índice completo. Hay veces que es más óptimo el escaneo del índice completo que el escaneo de un rango. Por ejemplo cuando no se restringen los valores que se seleccionan de una tabla es mejor el uso de este método que el uso de **Index Range Scan**.

Cuando se utiliza **CBO** y se tiene estadísticas que indican que es más óptimo el uso de **Index Full Scan**, elige este método en vez de utilizar un **Full Table Scan** y un **Sort**. Hay ocasiones que un **FTS** y una ordenación posterior puede ser más óptimo, bien porque el orden buscado no sea el del índice o porque sea más ineficiente la utilización del índice debido a que usa lecturas de bloques individuales en vez de lecturas multibloque como **FTS**.

- **Index Fast Full Scan**

Este método es similar al anterior, se escanean todos los bloques del índice pero no se devuelven ordenados. Este tipo de búsqueda por índice usa lecturas multibloque en vez de lecturas de bloques únicos.

Se puede usar como método de acceso para acceder a la segunda columna de un índice concatenado.

AI I.2.3. Rowid

Éste es el método más rápido para acceder a los datos. **Oracle** obtiene el bloque de datos especificado y extrae las filas necesarias.

Rowid es una representación interna de **Oracle** para la localización exacta de los datos. Suele aparecer en el plan de ejecución como *TABLE ACCESS BY ROWID*.



Normalmente se accede por **rowid** tras obtener el **rowid** a partir de un índice.

A I.3. Métodos de Joins

Un **Join** es un predicado que intenta combinar dos conjuntos de filas. Los pasos de un **Join** se ejecutan en serie, por lo que el orden en que se ejecutan los **Joins** es muy significativo. Puede que los predicados de una consulta se satisfagan solamente al producirse los **Joins** en un determinado orden, y esto puede condicionar al método de acceso elegido por el **CBO**.

El **optimizador** estima el coste de cada método de **Join** y elige el de menor coste. Cuando el **JOIN** devuelve muchas filas el optimizador considera:

- Un **Nested Loop Join** es ineficiente cuando devuelve un número grande de filas (más de 10000).
- Con **CBO** el método más eficiente para hacer **Join** de un número grande de filas es el **Hash Join**.
- Con **RBO** el método más eficiente para los **Joins** de cantidades grandes de filas es el **MERGE JOIN**.

 JUNTA DE ANDALUCÍA CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

AI I.3.1. Nested Loop

El método de **Join Nested Loop** es útil cuando se trata con pequeños subconjuntos de datos que se van a relacionar y si la condición de **Join** es una buena condición directora de búsqueda entre las dos tablas implicadas.

Con **Nested Loop** primero el **optimizador** determina la tabla directora de la consulta y la selecciona como tabla exterior. La otra tabla se designa como tabla interior. Para cada fila de la tabla exterior **Oracle** accede a todas las filas de la tabla interior. El bucle exterior recorre cada fila de la tabla exterior, mientras que el bucle interior recorre las filas de la tabla interior.

El bucle exterior aparece primero en el plan de ejecución.
<pre> NESTED LOOPS Bucle_Exterior Bucle_Interior </pre>

Cuando la tabla interior es independiente de la exterior, es decir no está dirigida la búsqueda por la tabla exterior, se recuperan las filas de ambas tablas en el bucle exterior, degradando el rendimiento hasta llegar a ser tan ineficiente como un **Producto Cartesiano**. Para estos casos es más recomendable el método **hash join**.

AI I.3.2. Hash Joins

El método **Hash Join** se introduce en la versión 7.3 y es más eficiente en teoría que el método **Nested Loop** o el **Sort Merge Join**. Solamente está disponible con **CBO**.

Hash Joins se usan para relacionar grandes conjuntos de datos. El **optimizador** usa el conjunto de datos más pequeño para construir una tabla hash en memoria. Después se explora el conjunto de datos mirando en la tabla hash para encontrar las filas que se relacionan.

Este método funciona mejor cuando el conjunto de datos pequeño cabe en la memoria disponible. El coste se limita a una única pasada de lectura por los datos de las dos tablas.



El **optimizador** elige este método cuando se relacionan las tablas mediante **equijoins** (con el operador de igualdad) y si se cumple que se van a unir una gran cantidad de datos o que una parte grande de una de las tablas relacionadas se va a unir con los datos de la otra por la condición de búsqueda.

AI I.3.3. Sort Merge Join

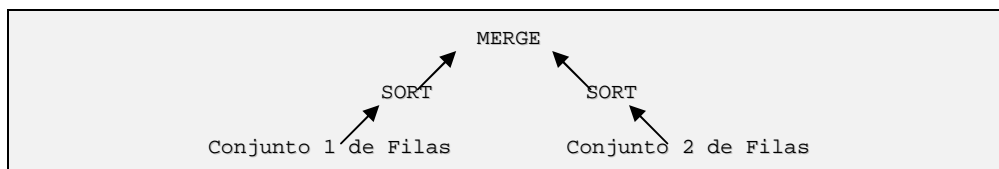
Sort Merge Join se usa para unir las filas de dos fuentes de datos independientes. Normalmente **Hash Join** tiene un mejor rendimiento que **Sort Merge Join** aunque **Sort Merge Join** se puede comportar mejor si se cumple que las filas ya se encuentran ordenadas y que no se tiene que ejecutar la operación de ordenación.

Si el uso del **Sort Merge Join** implica la utilización de un método más lento para acceder a los datos, se pierden las ventajas de este método de **Join**.

Este método de **Join** es útil cuando se usan condiciones de desigualdades (exceptuando el operador distinto ' \neq '). Se comporta mejor que **Nested Loop Join** con grandes cantidades de datos. El **Hash Join** no se puede usar si no hay un operador de igualdad.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

Se obtiene primero el conjunto de filas de la primera tabla. Ordena estas filas las filas de este primer conjunto. Después obtiene el segundo conjunto de filas de la siguiente tabla. A continuación ordena este segundo conjunto en el mismo orden que el primer conjunto. Finalmente se unen los dos conjuntos (**Merge Join**). No se acceden a los dos conjuntos de filas concurrentemente.





Si los conjuntos de filas están ya ordenados, no se realiza la ordenación de los mismos. Los conjuntos de filas preordenados son columnas indexadas y conjuntos de filas que se han ordenado en pasos previos de la consulta.

Las ordenaciones son operaciones costosas, especialmente con tablas grandes. Por este motivo **Sort Merge Join** no suele ser eficiente.

AI I.3.4. Producto Cartesiano

El **Producto Cartesiano** no es realmente un tipo de **Join** ya que no se unen los datos de dos tablas mediante ningún predicado. El optimizador une cada fila de una tabla con cada fila de la otra tabla, creado en **Producto Cartesiano** de los dos conjuntos de datos. El rendimiento de un **Producto Cartesiano** es muy pobre.

Se suele dar por errores de código en las consultas. Hay ocasiones en las que se escapa una tabla de más en la cláusula **FROM** y no se relaciona con ninguna tabla en la cláusula **WHERE**. Esto suele dar filas repetidas que se anulan usando el operador **DISTINCT**. Esto degrada aún más el rendimiento de la consulta. Se debe tener mucho cuidado con este tipo de errores, porque a veces al optimizador le basta una condición común de filtrado entre las dos tablas como condición de búsqueda (o **Join**) siendo este caso aún más peligroso porque no se marca el **Producto Cartesiano (CARTESIAN)** en el plan de ejecución y puede pasar desapercibido.

 JUNTA DE ANDALUCÍA CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

A I.4. Operaciones

En los planes de ejecución se pueden mostrar diferentes tipos de operaciones.

AI I.4.1. Sort

Hay muchas operaciones que provocan ordenaciones:

- Cláusulas *Order By*
- Cláusulas *Group By*
- **Sort Merge Join**

Si los datos ya están ordenados adecuadamente, no se produce la ordenación. Las ordenaciones son operaciones muy costosas, especialmente en tablas grandes que no caben en memoria y terminan ordenándose en disco. Por defecto, los bloques ordenados se colocan en el **buffer cache**, provocando que bloques de otros procesos que se encuentren en la caché salgan de ella.

AI I.4.2. Filter



Filter puede tener muchos significados. En los planes de ejecución se puede usar para indicar que no se usa una partición. También aparece en los planes de ejecución para indicar procesos de filtrado reales donde se filtra un conjunto de datos basándose en funciones de otro conjunto de datos.

AI I.4.3. Views

Normalmente el **optimizador** intenta fusionar las vistas con la consulta principal convirtiendo la consulta de sobre la vista en una consulta sobre las tablas de la vista. En ocasiones no es posible fusionar una vista en la consulta principal. En estos casos se seleccionan los datos directamente de la vista en vez de hacer **Join** entre las tablas. En el plan de ejecución aparecerá esta operación como **VIEW**. Las subconsultas en la cláusula **FROM** no se suelen poder fusionar.

A I.5. ¿qué hacer en caso de Bind Variables?

Cuando la consulta se va a ejecutar en un procedimiento con **bind variables** se debe calcular el plan de ejecución con **bind variables**. No se debe calcular el plan de ejecución con valores fijos, ya que puede generar un plan de ejecución adaptado a esos valores concretos. Para obtener el plan correcto que se calculará en tiempo de ejecución hay que probarlo con las variables.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

A I.6. Estadísticas de AUTOTRACE

Para ver como se pueden obtener estadísticas con el modo *AUTOTRACE*:

```
SQL> set autot on explain statistics
SQL> SELECT COUNT(*) FROM CasVal;

COUNT(*)
-----
510335

Execution Plan
-----
0      SELECT STATEMENT Optimizer=CHOOSE (Cost=756 Card=1)
1      0      SORT (AGGREGATE)
2      1      INDEX (FAST FULL SCAN) OF 'CASVAL_PK' (UNIQUE) (Cost=756
              Card=503866)

Statistics
-----
0      recursive calls
4      db block gets
5019   consistent gets
5020   physical reads
0      redo size
413    bytes sent via SQL*Net to client
470    bytes received via SQL*Net from client
4      SQL*Net roundtrips to/from client
1      sorts (memory)
0      sorts (disk)
1      rows processed
```

En este apartado se va a dar una breve explicación del significado de los principales valores que afectan al rendimiento.

- **Recursive Calls**



Hay veces que para resolver una sentencia **SQL** ejecutada por el usuario, **Oracle** tiene que ejecutar sentencias adicionales. Estas sentencias se llaman **Recursive Calls** o sentencias **SQL** recursivas. Por ejemplo cuando se inserta una fila y no hay espacio para albergar la fila, **Oracle** hace llamadas recursivas para reservar el espacio necesario para colocar la fila en la tabla. Cuando se accede a la información del **diccionario** y no está disponible en la caché, se tiene que buscar esta información en el disco, produciéndose **Recursive Calls**.

Afectan negativamente al rendimiento.

- **Consistent Gets**

Número de lecturas consistentes que se requieren para un bloque. Se pueden dar cuando se leen datos del segmento de **rollback** por modificaciones en los datos por otro usuario.

Si el valor es alto frente a la cantidad de datos obtenidos, es señal de que la consulta es costosa y se debe revisar para optimizarla.

	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

- **Physical Reads**

Número total de bloques leídos del disco. Este número es igual al valor de las lecturas físicas directas del disco más las lecturas del **buffer cache**.

Si el valor es alto frente a la cantidad de datos obtenidos, es señal de que la consulta es costosa y se debe revisar para optimizarla.



- **Sorts (memory)**

Número de operaciones de ordenación que se realizan completamente en memoria sin escribir en disco.

- **Sorts (disk)**

Número de operaciones que necesitan al menos una escritura de disco.

Si la consulta necesita las ordenaciones que se producen, indica un tamaño incorrecto de **SORT_AREA_SIZE** para poder soportar estas ordenaciones (tarea del administrador).

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

A II. Anexo: Consultas SIRhUS

En este anexo se especifican consultas recomendadas específicamente en SIRhUS.

A II.1. Vistas de acceso a filas basadas en otras tablas de acceso a filas



El acceso a filas se realiza a través de vistas que contienen la condición. Un sinónimo público con el mismo nombre de la tabla en realidad hace referencia a la vista.

Cuando se quiere crear una vista de acceso a filas basada en el acceso a filas de la tabla `IHESTORG` o en la `IHPT` la vista de acceso a filas de la tabla ha de hacer uso de la vista `IHVESTORG_ACC` e `IHVPT_ACC` y no de la tabla `IHESTORG`, `IHPT`.

Cuando un usuario hace uso del objeto `IHINCIDENCIAS` esté se resuelve a través del sinónimo público del mismo nombre pero que apunta a una vista `IHVINCIDENCIAS_ACC`. Si en esta vista se hace uso de la tabla `IHESTORG` está se resuelve directamente a la tabla sin pasar de nuevo por el sinónimo público `IHESTORG`.



EJEMPLO: Vista incorrecta. No realiza el filtro deseado

```
CREATE VIEW IHVINCIDENCIAS_ACC AS
SELECT *
FROM IHINCIDENCIAS, IHESTORG
WHERE EO_X_ESTORG = X_ESTORG
```





EJEMPLO: Uso de la vista de acceso a filas

```
CREATE VIEW IHVINCIDENCIAS_ACC AS
SELECT * FROM IHINCIDENCIAS, IHVESTORG_ACC
WHERE EO_X_ESTORG = X_ESTORG
```

Recordar que la implementación actual del acceso a filas se realiza a través de vistas que contienen la condición. Un sinónimo público con el mismo nombre de la tabla en realidad hace referencia a la vista.

A II.2. Uso de `IH_FU_VALOR`

Para encontrar el valor correspondiente a un código es obligatorio el uso de la función `IH_FU_VALOR`.

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

No se debe incluir directamente en la cláusula where (al igual que cualquier otra función), ya que provoca lentitud en su ejecución. Además el valor retornado por esta función permanece invariable por lo que carece de sentido su reevaluaci.



EJEMPLO: Forma correcta de uso de la función IH_FU_VALOR. Almacenamos el valor en una constante

```

DECLARE
  V NUMBER(8);
  V_CTE_ARRA CONSTANT VARCHAR(2):=IH_FU_VALOR('ARRA');
BEGIN
  SELECT COUNT(1)
    INTO V
    FROM IHCABACTADM
    WHERE APC_ACA_C_ACTADM = V_CTE_ARRA;
END;
```



EJEMPLO: uso correcto pero no recomendado de la función IH_FU_VALOR. La función se ejecuta innecesariamente una vez por cada fila de la tabla

```

DECLARE
  V NUMBER(8);
BEGIN
  SELECT COUNT(1)
    INTO V
    FROM IHCABACTADM
    WHERE APC_ACA_C_ACTADM = IH_FU_VALOR('ARRA');
END;
```

A II.3. Estructura de dependencias

Mediante la siguiente consulta obtenemos la estructura de dependencias de la unidad:



Descendientes a una unidad

```

SELECT      x_estorg,
            d_estorg,
            LEVEL nivel
  FROM      ihestor
 START WITH x_estorg = 18610
CONNECT BY PRIOR eo_x_estorg = x_estorg;
```

Estructura de dependencias de la unidad:

Descendientes a una unidad		
X_ESTORG	D_ESTORG	NIVEL
18610	D.P. IGUALDAD Y B. SOCIAL DE SEVILLA	1
3305310	IGUALDAD Y BIENESTAR SOCIAL	2
110	JUNTA DE ANDALUCÍA	3

 JUNTA DE ANDALUCÍA <small>CONSEJERÍA DE JUSTICIA Y ADMINISTRACIÓN PÚBLICA</small>	SISTEMA DE ASEGURAMIENTO DE LA CALIDAD	 SIRhUS
	06. Manual de Instrucciones Técnicas	
	06.01. Normas y Recomendaciones de Accesos a Bases de Datos	

Análogamente para ver los descendientes a una unidad:



Cálculo de los descendientes a una unidad

```

SELECT x_estorg,
       d_estorg,
       LEVEL nivel
FROM ihestor
START WITH eo_x_estorg = 18610
CONNECT BY PRIOR x_estorg = eo_x_estorg

```