Procesamiento y optimización de consultas.

Corresponde ahora revisar el proceso que sigue el SGBD al ejecutar una consulta, desde que recibe la solicitud en lenguaje SQL, hasta que retorna los resultados al usuario. Es un proceso que el motor lo hace automáticamente, pero que es muy importante comprenderlo porque nos da elementos de juicio para afinar la implementación de nuestra base de datos. También, analizaremos las buenas prácticas que debemos seguir al escribir sentencias SQL.

Con el estudio de este tema, esperamos que usted esté en capacidad de analizar e identificar la mejor estrategia para procesar y ejecutar consultas SQL, y adoptar buenas prácticas en la construcción de consultas SQL.

¡Importante!: para el estudio de esta unidad es imprescindible que usted recuerde lo que aprendió en la asignatura Fundamentos de Bases de Datos, en lo que refiere al Álgebra Relacional. Si es necesario haga un repaso de aquello para volver a familiarizarse con sus operaciones y simbología.

1. Panorámica del procesamiento de consultas

Las consultas SQL, son las operaciones quizá más críticas para el desempeño de una base de datos. Son las que normalmente consumen más recursos y capacidad de procesamiento del servidor. Por ello es muy importante comprender como ocurren.

Son tres los pasos que realiza el SGBD al ejecutar una consulta, que los identificaremos como (1) ANÁLISIS Y TRADUCCIÓN, (2) OPTIMIZACIÓN y (3) EJECUCIÓN. La Figura 1 complementa esta panorámica, ilustrando de forma general las acciones que conlleva cada paso.

Como se ve, el SGBD recibe una consulta expresada en lenguaje de alto nivel (SQL); a esta, en un primer paso, la valida, simplifica y traduce a lenguaje de bajo nivel (algebra relacional); luego en el paso 2, evalúa y selecciona el plan de ejecución más apropiado; el que finalmente, en el paso 3, lo traduce a lenguaje de máquina, que es el que se ejecuta y genera los resultados. Todo este proceso el SGBD lo hace automáticamente.

Pero, el hecho que el motor optimice la consulta automáticamente no significa que debemos despreocuparnos de como la escribimos, al contrario, mucho puede hacer el programador al estructurar la consulta para evitar que el SGBD, realice un trabajo excesivo. Y porque, además, como veremos, el análisis semántico que realiza el motor es limitado.

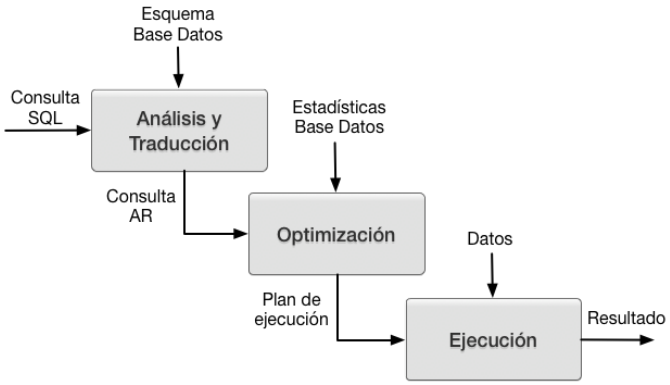


Figura 1. Procesamiento de una consulta SQL
Fuente: Universidad de Sevilla, 2011

2. Fase de análisis y traducción

También conocida como fase de descomposición, su objetivo es validar, simplificar y traducir la consulta a algebra relacional.

RECURSO EVA	En el Entorno Virtual de Aprendizaje revise el recurso "**Análisis y traducción de una consulta SQL** ". El cuál ofrece una explicación detallada del proceso que realiza un SGBD al analizar una consulta SQL y transformarla a algebra relacional.

Como ve, esta fase es muy importante, pues el SGBD puede de hecho optimizar en mucho el enunciado de la consulta SQL, antes de definir un plan de ejecución. Veamos otros ejemplos:

Suponga las siguientes relaciones:

```
medicinas (idmed, nombre, tipo, pres, idlab)
laboratorios (idlab, lab)
```

donde todos los atributos son obligatorios.

Ejemplo 1:

Analice la siguiente consulta:

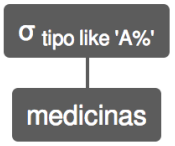
```
SELECT *
FROM medicinas
WHERE tipo = 'ANALGESICO'
   OR tipo like 'A%';
```

Sintácticamente, la consulta está bien. En este caso, el motor buscará normalizar y simplificar la condición de selección. Si analizamos, la expresión condicional *tipo = 'ANALGESICO'* ya está contenida en la condición *tipo like 'A%'*, es decir "ANALGESICO" está incluido en el conjunto de los valores que empiezan con "A", y al ser una disyunción, se reduciría a la segunda expresión, así:

```
SELECT *
FROM medicinas
WHERE tipo like 'A%';
```

Que en algebra relacional equivaldría a:

$$\sigma_{\text{tipo like 'A\%'}}(\text{medicinas})$$



Con ello el motor evitará realizar una doble comparación innecesaria al momento de procesar los datos de la tabla.

Ejemplo 2:

Considere la siguiente consulta, que busca obtener los tipos de medicinas, que no tienen ningún laboratorio que los provea:

```
SELECT tipo, count(distinct idlab)
FROM medicinas
GROUP BY tipo
HAVING count(distinct idlab) = 0;
```

Es una consulta expresada correctamente, que no tiene errores léxico-sintácticos, pero si un error semántico, que el motor lo detectará. Y es que, en este caso, si IDLAB es NOT NULL, es imposible que dentro de la tabla MEDICINAS se encuentre un tipo que no tenga el id de laboratorio informado; por lo que la función COUNT en este caso como mínimo devolverá 1, jamás 0.

El motor al detectar esta particularidad semántica sabe que la consulta retorna un resultado vacío, y no requiere avanzar a las fases de optimización y ejecución. Con ello el SGBD se habrá ahorrado el trabajo, los recursos y el tiempo que implicaría procesar, agrupar y contar las filas en la tabla MEDICINAS.

Ejemplo 3:

La siguiente consulta obtiene los nombres de las medicinas de laboratorio ASOL que vienen en presentación JARABE:

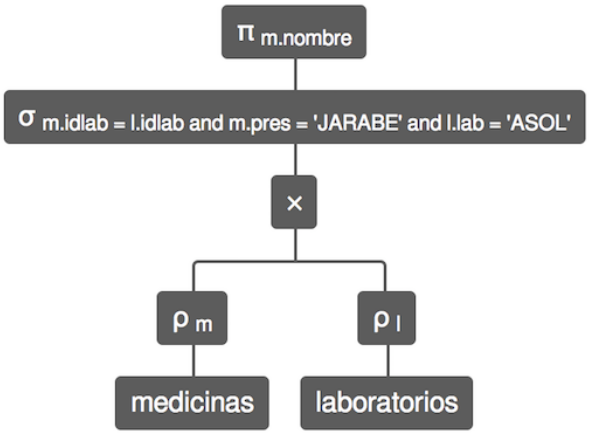
```
SELECT m.nombre
FROM medicinas m, laboratorios l
WHRERE m.idlab = l.idlab
   AND m.pres = 'JARABE'
   AND l.lab = 'ASOL';
```

La consulta ya está estructurada correctamente (no requiere simplificación). Entonces, en este caso el SGBD lo que hará es convertirla a álgebra relacional para que pase a la fase de optimización. Y, como se ha mencionado, puede haber varias representaciones en algebra relacional para la misma consulta, a continuación, algunas de las opciones que el SGBD podría evaluar.

La que sería la traducción literal del enunciado SQL, es la siguiente:

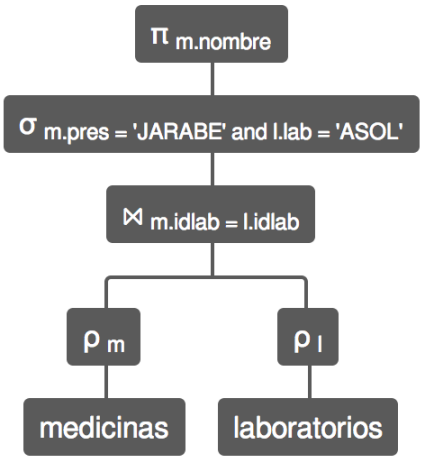
$$\pi_{\text{m.nombre}} (\sigma_{\text{m.idlab = l.idlab and m.pres = 'JARABE' and l.lab = 'ASOL'}} ((\rho_{\text{m}} \text{ medicinas}) \times (\rho_{\text{l}} \text{ laboratorios})))$$

donde, primero realiza un producto cartesiano y luego combina y filtra las filas. El árbol quedaría así.



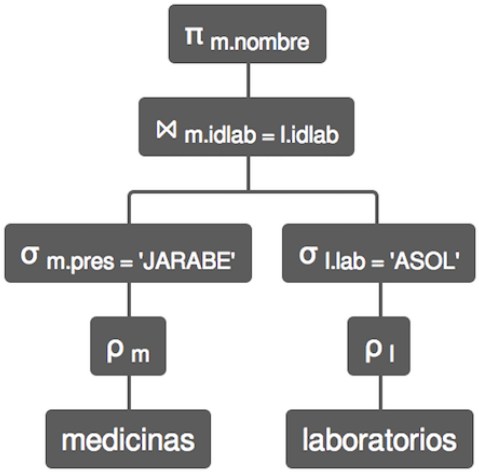
Una representación alternativa sería, que primero combine las tablas, y luego filtre las filas, así:

$$\pi_{m.nombre}(\sigma_{m.pres = 'JARABE' \text{ and } l.lab = 'ASOL'}((\rho_{m} medicinas) \bowtie_{m.idlab = l.idlab} (\rho_{l} laboratorios)))$$



Una tercera opción sería, que primero filtre las filas en cada tabla individualmente, y luego combine los resultados, así:

$$\pi_{m.nombre}((\sigma_{m.pres = 'JARABE'}(\rho_{m} medicinas)) \bowtie_{m.idlab = l.idlab} (\sigma_{l.lab = 'ASOL'}(\rho_{l} laboratorios)))$$



Y así, como estas, usted podrá inferir otras equivalencias en algebra relacional de la misma consulta SELECT. Son las alternativas que el motor deberá evaluar en la fase de optimización y de ellas elegir la mejor (la que represente menor tiempo de obtención de resultados).

Actividades propuestas:

Dada las siguientes relaciones:

```
empleados (cedula, apellidos, nombres, dirección, fecha_ing,
email, idcargo)

cargos (idcargo, cargo)
```

donde EMAIL es llave única.

1. La siguiente consulta que pretende obtener la lista de los empleados de la 'A' a la 'C'. Analícela e identifique si tiene algún error sintáctico o semántico, que el SGBD detectaría.

```
SELECT *
FROM empleados e
WHERE e.apellidos like 'A%'
  AND e.apellidos like 'B%'
  AND e.apellidos like 'C%';
```

2. Analice las siguientes consultas y diga, ¿cómo quedaría su enunciado SQL luego del análisis que realiza el SGBD?

Caso 1

```
SELECT *
FROM empleados
WHERE fecha_ing > '01-ENE-1998'
  AND fecha_ing > '15-MAY-2001'
   OR genero = 'M';
```

Caso 2

```
SELECT email, count(cedula)
FROM empleados
WHERE genero = 'M'
GROUP BY email
HAVING count(cedula) > 1;
```

3. Para la siguiente consulta, ¿cuáles serían las representaciones alternativas en álgebra relacional?

```
SELECT e.*
FROM empleados e, cargos c
WHERE e.idcargo = c.idcargo
  AND c.cargo = 'FISCALIZADOR';
```

Espero, haya quedado más claro lo que implica esta fase, y como este proceso de análisis puede repercutir en el desempeño del motor. Pasaremos ahora a la fase donde el SGBD define el plan de ejecución.

3. Fase de optimización

Si durante la fase de análisis y traducción, la consulta no presentó errores que impidan su ejecución, y si el enunciado de la consulta simplificada no conlleva a un resultado vacío, entonces el SGBD proseguirá con su procesamiento.

La consulta traducida a algebra relacional, pasa a la fase de optimización en la cual, el SGBD arma un plan de ejecución, es decir, define la secuencia de acciones que deberán ejecutar para procesar la consulta y obtener los resultados. Ese plan, debe ser lo óptimo posible, en cuanto al tiempo de ejecución.

Ahora, usted se debe preguntar ¿cómo determinar cuál plan es mejor sin ejecutarlo aun?... Existen dos técnicas de optimización:

1. Optimización basada en costo.
2. Optimización basada en reglas.

Optimización basada en costo

Bajo este modo, el SGBD analiza varias alternativas de ejecución -como vimos el apartado anterior, para un mismo enunciado SQL, hay varias representaciones en algebra relacional - y de ellas, debe escoger una.

¿Cuál escoge?

La que represente un menor costo de consumo de recursos de procesamiento y por ende es la que se ejecutaría en menor tiempo. Y el coste más importante de un plan, normalmente es el número de operaciones I/O (accesos a disco).

Ante todo, usted debe tener claro que en realidad el cálculo que realiza el SGBD al momento de determinar el costo de ejecución de una consulta, es bastante complejo; y está supeditado al análisis de algunas variables, tales como: el tamaño de la memoria intermedia, la velocidad del procesador, la configuración del almacenamiento secundario, la distribución física de los datos, la existencia de índices, la complejidad de la consulta, y otros.

Pero quizá -como ya anticipamos- la variable que más repercute, es el "número de accesos a disco" o "número de operaciones I/O". Puesto que, en comparación al resto del proceso, el trabajo de ubicar los bloques de datos en disco y cargarlos a memoria RAM, es la operación más lenta; el resto del proceso se realiza en memoria principal. Podríamos decir entonces que, el plan de ejecución que requiera menos accesos a disco tiene una alta probabilidad de ser el más eficiente.

Veamos entonces, como sería el cálculo de coste de un plan de ejecución en función del número de operaciones I/O.

En el Entorno Virtual de Aprendizaje revise el recurso "**Cálculo del costo I/O en consultas SQL**". El cuál le explica cómo calcular el número de operaciones de lectura/estructura requeridos para las distintas alternativas de ejecución -expresiones en algebra relacional- de una consulta SQL.

Para determinar el costo, el tema de las estadísticas es muy importante, porque le permiten al optimizador, conocer información acerca de una tabla, sin necesidad de accederla, por ejemplo, saber cuántos registros tiene, el número de bloques que ocupa en disco, etc. Sin las estadísticas, el cálculo del costo de un plan de ejecución sería inviable.

Con base en lo estudiado, le invito a desarrollar la siguiente actividad.

Actividad propuesta:

Dada las siguientes relaciones:

```
empleados (cedula, apellidos, nombres, dirección, fecha_ing,
email, idcargo)

cargos (idcargo, cargo, sueldo)
```

La siguiente consulta le permite obtener la lista de empleados que ganan menos de $500.

```
SELECT e.apellidos, e.nombres
FROM empleados e, cargos c
WHERE e.idcargo = c.idcargo
  AND c.sueldo < 500;
```

	Suponga las siguientes estadísticas: • empleados: 800 • cargos: 20 • cargos con sueldo inferior a $500: solo 2 (mantenimiento y conserje) • empleados de mantenimiento más conserjes: 25 Con base en lo anterior determine los posibles planes de ejecución (representaciones en algebra relacional), para esa consulta, y por cada uno calcule el costo en términos de operaciones I/O. ¿Cuál es el plan óptimo?

Veamos ahora el otro método de optimización.

Optimización basada en reglas

Con este método, el SGBD, lo que hace es aplicar un conjunto de reglas de transformación de expresiones (reglas de equivalencia) sobre la consulta expresada en álgebra relacional, de manera que, con base en un conjunto de equivalencias, restructura el orden de evaluación de los componentes de la consulta a fin de mejorar la eficiencia de su ejecución.

Para propósitos de optimización, cada SGBD establece ciertas heurísticas (o "reglas heurísticas"), que no son otra cosa que estrategias basadas en las reglas de equivalencia del algebra relacional, que "aconsejan" como se debería reestructurar el orden de ejecución de las operaciones de una consulta para hacerla más eficiente. Connolly y Begg (2005), explica algunas de las reglas que podrían aplicarse para optimizar las consultas:

a) Realizar las operaciones de selección lo antes posible.
b) Combinar el producto cartesiano con una operación de selección subsiguiente cuyo predicado represente una condición de combinación, para formar una operación de combinación.
c) Utilizar la asociatividad de las operaciones binarias para reordenar los nodos hoja, de modo que los nodos hoja con las operaciones de selección más restrictivas se ejecuten primero.
d) Realizar las operaciones de proyección lo antes posible.
e) Calcular una única vez las expresiones comunes (p.588).

Como puede ver, estas heurísticas lo que buscan es ordenar las operaciones de una consulta expresada en algebra relacional, para que, al ejecutar ese plan, el SGBD vaya desechando las filas y/o columnas irrelevantes (innecesarias) lo antes posible, y así reducir el número de lecturas y escrituras de bloques de datos al mínimo posible.

Actividad propuesta:

Dada las siguientes relaciones:

```
empleados (cedula, apellidos, nombres, dirección, fecha_ing,
email)

cargos (idcargo, cargo, sueldo)
```

Considere las siguientes consultas, expresadas en álgebra relacional, y para cada una de ellas, determine como se podrían expresar de una forma más eficiente aplicando reglas heurísticas:

1)

	2) $\pi_{e.cedula,\ e.apellidos}\ (\ \sigma_{e.idcargo\ =\ c.idcargo\ and\ c.sueldo\ >\ 800}\ ((\ \rho_{e}\ empleados)\ \times\ (\ \rho_{c}\ cargos)))$ $\pi_{e.cedula,\ e.idcargo}\ (\ \sigma_{e.email\ like\ '\%@gmail.com'\ and\ c.cargo\ =\ 'ASESOR'}\ ((\ \rho_{e}\ empleados)\ \bowtie_{e.idcargo\ =\ c.idcargo}\ (\ \rho_{c}\ cargos)))$

Conclusión

Cada SGBD particulariza la aplicación de estos métodos y maneja sus propias heurísticas y sus propios algoritmos para el cálculo del costo de un plan de ejecución. En algunos casos combinan ambos métodos, es decir, de todos los planes de ejecución posible, primero filtran algunos aplicando optimización basada en reglas, luego con los restantes aplican optimización basada en costos para elegir el mejor. En otros casos, el SGBD permite seleccionar el método a aplicar.

Una vez el SGBD elige el plan a ejecutar, este es compilado, es decir, es traducido a lenguaje de máquina o lenguaje intermedio, según la implementación; y acto seguido se ejecuta.

La ejecución es la materialización del plan; es cuando a través del sistema operativo, se transfieren los bloques de datos desde disco a memoria intermedia, se realizan las operaciones de combinación, selección, proyección, ordenación, etc., según sea el caso, y se generan los resultados, que a su vez son enviados al usuario o sistema, que los solicitó.

Son procesamientos como este, los que marcan diferencia entre los distintos SGBD, haciendo que unos tengan mejor desempeño que otros, sobre todo en bases de datos que manejan grandes volúmenes de información.

4. Importancia de los índices y la memoria intermedia

Hasta ahora hemos estudiado la optimización de consulta asumiendo que el SGBD siempre requiere leer los datos desde almacenamiento secundario (disco) y pasarlos a memoria intermedia; y que para el procesamiento no usa índices.

Pero, como ya se mencionó antes, realmente son muchos más los factores de inciden en el rendimiento de una consulta y que el motor evalúa al momento de obtener el plan de ejecución óptimo. Entre ellos están, la gestión de la memoria intermedia y los índices.

Sin duda, los índices ayudan a acelerar en gran medida obtención de resultados de una consulta SQL, pero no olvide que hay el lado negativo: el exceso de índices en una tabla puede ralentizar de forma importante las operaciones de actualización de datos. Por lo tanto, siempre se debe analizar si un índice es realmente requerido.

En el tema de la memoria intermedia -que diríamos es la porción de RAM asignado para el SGBD- es muy importante no desaprovechar la memoria libre disponible en nuestro servidor. Mientras más RAM le podamos asignar al motor, más caché dispondrá para almacenar y mantener datos y planes de ejecución.

	Recuerde, si para la consulta recibida, ya existe un plan de ejecución en memoria intermedia, el SGBD, no necesitará volver a analizar la consulta, únicamente verificará permisos, y procederá a ejecutar con el plan guardado.

5. Buenas prácticas SQL

¿Si el motor ya se encarga de simplificar y optimizar la consulta, entonces, da igual como la escribamos? ¿Es incensario esforzarnos escribiendo ordenadamente, si el SGBD la reordena?

¡No! Como usuario de la base de datos, usted también puede ayudar a que las consultas se procesen eficazmente.

¿Cómo hacerlo?

Adoptando buenas prácticas en la codificación del lenguaje SQL.

A continuación, algunas recomendaciones importantes, a tomar en cuenta cuando usted formule consultas SQL:

a) Traer únicamente los datos que necesítanos procesar, a veces ese común la tentación del **SELECT ***, sin limitar selección ni proyección.

b) Ordenar solo si es necesario. La operación ORDER BY es una de las operaciones que genera más consumo de recursos para su procesamiento.

c) Cualificar los nombres de columna en consultas multitabla. Con ello cuando aparece el nombre de una columna, el motor no tendrá que ir a buscar en todas las tablas que constan en la cláusula **FROM**

d) En subconsultas, si es posible, usar IN en lugar de NOT IN. Si usa NOT IN el SGBD deberá evaluar la condición con TODOS los resultados de la consulta, en cambio con IN, solo deberá evaluar hasta que encuentre una coincidencia.

e) Usar EXISTS en lugar de IN. Porque IN lo que busca es validar que exista al menos una coincidencia, EXIST es más eficiente ya que simplemente evalúa si el resultado de la subconsulta es vacío o no vacío. Por ejemplo, en lugar de:

```
SELECT *
FROM clientes
WHERE ciudad_res IN (SELECT ciudad
                     FROM sucursales);
```

es mejor:

```
SELECT *
FROM clientes c
WHERE EXISTS (SELECT 1
              FROM sucursales s
              WHERE s.ciudad = c.ciudad_res);
```

En el 2do caso, no es importante lo que devuelve la subconsulta, solo interesa saber si devuelve algo o no. En este caso, solo devolverá 1 cuando la ciudad de la sucursal coincida con la ciudad de residencia del cliente que se esté evaluando.

Igualmente, a nivel de las expresiones condicionales que se usan en las cláusulas WHERE y HAVING hay algunas recomendaciones muy importantes. Se las expone en la Tabla 1.

Tabla 1. Buenas prácticas para expresiones condicionales

Recomendación	Ejemplo

En lo posible evitar usar expresiones o funciones como operando de la condición.	Es más rápido evaluar la condición
	`PRECIO > COSTO`
	que evaluar
	`PRECIO > COSTO * 1.12.`
	En el segundo caso el motor tendrá que evaluar primero la expresión aritmética y luego comparar.

Tomar en cuenta que para un SGBD es más rápido comparar operando numéricos que comparar fechas, cadenas y otros.	Es más rápido comparar
	`CODIGO = 1285`
	que comparar
	`CODIGO='1285'`
	Asumiendo claro que en el primer caso CODIGO es tipo entero y el segundo es tipo cadena.

También tener en cuenta que el SGBD compara más rápido las igualdades que las desigualdades.	Es más rápido
	`GENERO = 'M'`
	que
	`GENERO <> 'F'`

Del punto anterior se puede inferir que, en una condicional conjuntiva múltiple, el orden de las comparaciones puede ser importante, primero deberían ir las igualdades y luego las desigualdades.	En lugar de:
	`TIPO <> 'REPUESTO' AND DESCUENTO = 0.2 AND PRECIO > 200`
	usar
	`DESCUENTO = 0.2 AND PRECIO > 200 AND TIPO <> 'RESPUESTO'`
	Con ello, primero ejecutará la comparación más rápida, y luego con el conjunto de filas ya reducido, realizará las otras operaciones.

Igualmente, tomando como base la lógica proposicional, se puede inferir que, en una condicional conjuntiva múltiple, es mejor que la condición con mayor probabilidad de ser falsa vaya primero.	Suponga, que la siguiente condición se usa para filtrar las viviendas de una ciudad, según la barrio y tipo de vivienda. Y sabemos que los tipos de vivienda son dos, mientras que los barrios son 50. En este caso, en lugar de usar:
	`TIPO = 'UNIFAMILIAR' AND BARRIO = 'SAUCES'`
	es mejor usar
	`BARRIO = 'SAUCES' AND TIPO = 'UNIFAMILIAR'`
	porque la probabilidad de que 1era condición sea falsa es de 49/50 = 98%, mientras que en la 2da condición es 1/2 = 50%. Es decir, es más probable que no coincida el barrio a que no coincida el tipo, porque en barrio la dispersión es más alta. Con ello, al ocurrir que la primera condición es falsa, por ser una conjunción, ya no es necesario evaluar la 2da. Recuerde que con el operador AND si uno de los operando es falso, el resultado es falso.

Aplicando el mismo razonamiento anterior, en una condicional disyuntiva múltiple, es mejor que la condición con mayor probabilidad de ser verdadera, vaya primero.	Para el mismo ejemplo anterior, en lugar de usar:
	`BARRIO = 'SAUCES' OR TIPO = 'UNIFAMILIAR'`
	usar
	`TIPO = 'UNIFAMILIAR' OR BARRIO = 'SAUCES'`
	porque la probabilidad de que coincida el tipo es 1/2=50%, mientras que la probabilidad de que coincida el barrio es 1/50=2%. Y en el caso del OR, si uno de los operando es verdadero, el resultado es verdadero

Aunque el SGBD lo suele corregir en la fase de análisis y traducción, es buena práctica evitar usar el operador lógico NOT.	En lugar de escribir:
	`NOT (PRECIO > 500 AND MARCA = 'LG')`
	es mejor decir (aplicando reglas de equivalencia):
	`PRECIO <= 500 OR MARCA <> 'LG'`
	En el primer caso son 4 operaciones las que tiene que evaluar el motor, mientras que en el segundo son solo 3.

Como usted puede advertir, no solo es cuestión de escribir ordenadamente; hay múltiples aspectos que dependen del programador y que inciden en el rendimiento. Sin mencionar el tema de la semántica, que como sabemos, su validación es limitada en el SGBD; es el programador quien sabe lo que desea obtener y quien debe asegurarse que el enunciado de la consulta se corresponde con lo que necesita.

En sistemas de gestión de bases de datos como ORACLE, es posible incluso que el programador, le envíe al optimizador una "sugerencia" inserta en la consulta SQL. Estas sugerencias (HINTS), le instruyen al optimizador sobre cómo comportarse a la hora de elegir el plan de ejecución.

Actividades propuestas:

1) Evalúe las siguientes expresiones condicionales, y diga cómo se podría mejorar su formulación:

   a) NOT (NUM_PAG < 80 OR EDITORIAL = 'LIMUSA')

   b) NOT (SUELDO < 800) AND COD_POSTAL = '110401'

2) Profundice su estudio sobre este tema, para ello investigue a través de internet u otras fuentes más recomendaciones a tener en cuenta al codificar consultas en SQL, de manera que se le facilite el trabajo al motor.

3) Investigue sobre uso de HINTS en ORACLE, identifique cuáles son los más importantes y su aplicación.

.-