Procesamiento de consultas y vistas

Seguridad en bases de datos

PID_00270597

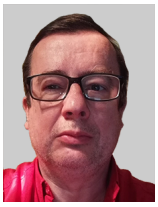
Joan Anton Pérez Braña
Albert Jové Canela
Santiago Ortego Carazo (†2007)
Ivo Plana Vallvé

Tiempo mínimo de dedicación recomendado: 7 horas

Universitat Oberta de Catalunya

Joan Anton Pérez Braña

Ingeniero superior de Informática por la UOC y licenciado en Ciencias Biológicas por la Universidad de Barcelona (UB).



Albert Jové Canela

Licenciado en Biología por la UB, ingeniero informático por la UOC y posgrado en auditoría informática por la UPC. Certificaciones CISA, CISM y CGEIT. Ejerce como profesional *freelance* en proyectos de adecuación de empresas a la normativa de protección de datos y es profesor colaborador en la UOC.

Santiago Ortego Carazo (†2007)

Doctor en Ingeniería Química, profesor de la Escuela Universitaria Politécnica de Mataró en el área de Bases de Datos y Sistemas de Información. Ha sido tutor y consultor de los Estudios de Informática, Multimedia y Telecomunicación, concretamente en el área de bases de datos.

Ivo Plana Vallvé

Ingeniero informático y máster en la Sociedad de la Información y el Conocimiento por la Universidad Oberta de Cataluña (UOC). Profesional de la informática en la empresa semipública desempeñando las funciones de Director de Informática y de Jefe de Transformación Digital. Profesor colaborador de la UOC.

La revisión de este recurso de aprendizaje UOC ha sido coordinada por la profesora: Àngels Rius Gavidia

Sexta edición: septiembre 2022

Av. Tibidabo, 39-43, 08035 Barcelona
Realitzación editorial: FUOC

Índice

Introducción

El lenguaje SQL es un lenguaje declarativo. Como tal, las consultas realizadas con este lenguaje especifican qué se quiere obtener en lugar de indicar cómo se quiere conseguir. En el presente módulo se explica la forma en que los sistemas gestores de bases de datos (SGBD) relacionales evalúan sistemáticamente las posibles estrategias alternativas que se pueden presentar, de acuerdo con las necesidades de los usuarios y las aplicaciones, con el objetivo de escoger la estrategia que se considera óptima.

También veremos que las vistas proporcionan mecanismos de seguridad y permiten al diseñador de la base de datos personalizar el modelo lógico al modelo de los usuarios. Se estudiará en qué casos una vista es actualizable y qué mecanismos incorporan los SGBD para permitir que las vistas sean actualizables.

Por último, estudiaremos las técnicas empleadas para proteger la base de datos contra accesos no autorizados y los mecanismos para otorgar/revocar privilegios a los diferentes usuarios según el modelo de negocio implícito en el sistema de información. También trataremos las obligaciones legales derivadas del hecho de estar en posesión de determinados datos especialmente protegidos.

Objetivos

Los materiales didácticos asociados a este módulo os permitirán lograr los siguientes objetivos:

1. Saber cuáles son los mecanismos de procesamiento y optimización de consultas, y así poderlas plantear de la forma más eficiente posible.

2. Conocer las diferentes estrategias de implementación de las operaciones de álgebra relacional con el fin de evaluar el coste de las consultas.

3. Interpretar y optimizar el plan de ejecución de una consulta de forma adecuada.

4. Presentar las vistas como elementos de diseño externo que permiten la actualización de datos.

5. Conocer nuevas aplicaciones de las vistas para mejorar el diseño de la base de datos.

6. Conocer el alcance de los mecanismos de seguridad de una base de datos.

7. Tomar conciencia de las obligaciones legales derivadas del cumplimiento de la Ley orgánica de protección de datos de carácter personal.

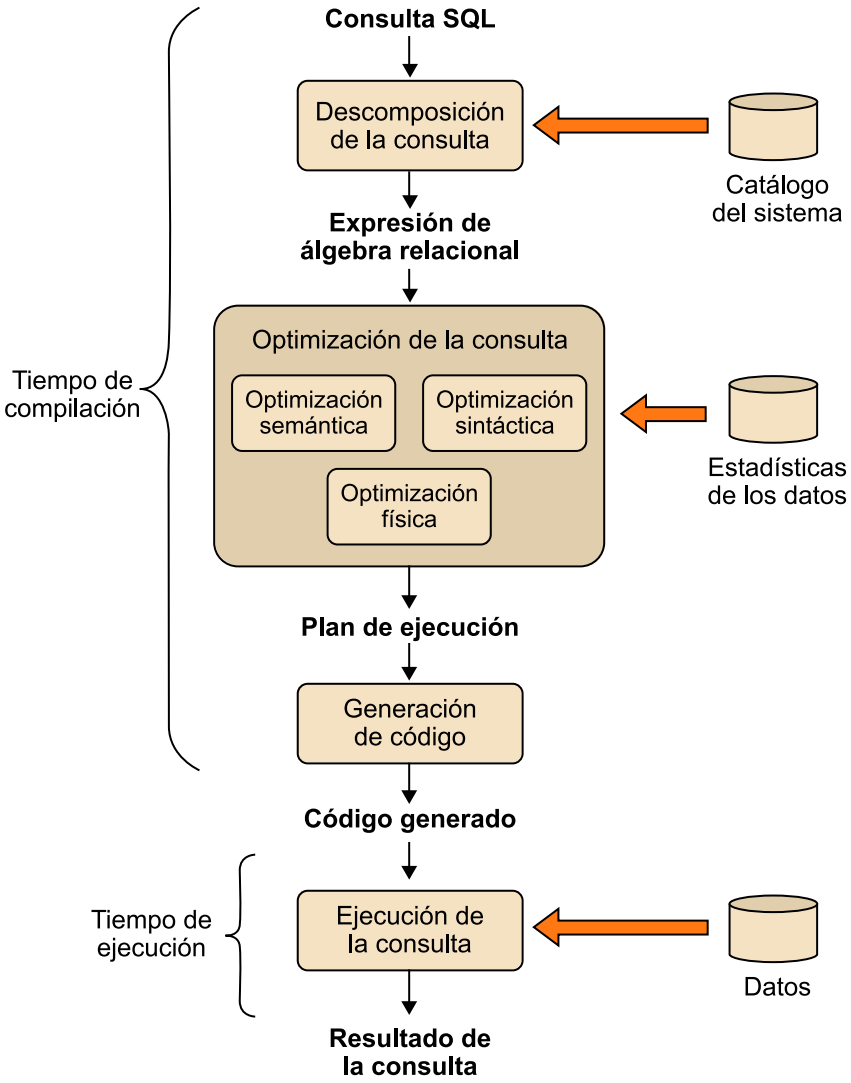
1. Procesamiento de consultas

Una de las principales críticas a los primeros sistemas gestores de bases de datos (SGBD[1]) relacionales fue el bajo rendimiento en el procesamiento de consultas. En los lenguajes no procedimentales, como es el caso del SQL, el usuario especifica los datos que quiere obtener en lugar de indicar cómo quiere conseguirlos.

(1)SGBD es la sigla de *sistema gestor de bases de datos*.

El **procesamiento de consultas** hace referencia al conjunto de actividades que el SGBD lleva a cabo para extraer información de una base datos con el objetivo de lograr la estrategia más eficiente que le permita tener mejor control sobre las prestanciones del sistema.

Figura 1. Etapas del procesamiento de consultas.



En la figura 1 podéis observar gráficamente las cuatro etapas que componen el procesamiento de consultas:

1) descomposición de la consulta,
2) optimización de la consulta,
3) generación de código, y
4) ejecución de la consulta.

La **descomposición de consultas** implica la traducción de consultas expresadas en lenguaje SQL a una representación interna basada en el álgebra relacional que suele ser más útil.

En primer lugar, se comprueba la corrección sintáctica y semántica de la consulta en SQL y después se crea un árbol sobre el cual se realiza el análisis de la consulta, que se transformará en una expresión de álgebra relacional.

Por lo que respecta a las expresiones de álgebra relacional, utilizaremos el siguiente convenio de símbolos:

Concepto	Representación	Observaciones
Relaciones	R, S	La lista de atributos de cada relación se define según el esquema siguiente: $R = (A_1, A_2, ..., A_n)$.
Predicados	p, q	Estos predicados se evaluarán lógicamente en las operaciones de selección y combinación.
Selección	$\sigma_p(R)$	Operación que devuelve las tuplas de la relación R que satisfacen la evaluación del predicado p.
Proyección	$\prod_L(R)$	Donde L corresponde a una lista de atributos que se mostrarán de la relación R.
Unión	$(R) \cup (S)$	Se obtiene una relación nueva que incluye las tuplas de la relación R y S menos las repeticiones.
Intersección	$(R) \cap (S)$	Se obtiene una relación nueva que incluye las tuplas que pertenecen a las dos relaciones, R y S .
Diferencia	$(R) - (S)$	Se obtiene una relación nueva que incluye las tuplas que pertenecen a la relación R pero que no están incluidas en la relación S.
Producto cartesiano	$(R) \times (S)$	Se obtiene una relación nueva formada por todas las tuplas que resultan de concatenar tuplas de la relación R con tuplas de la relación S.
Combinación Theta	$(R) \bowtie_p (S)$	Se obtiene una relación nueva que incluye los atributos de los pares de tuplas correspondientes a R y S que satisfacen el predicado p.

A partir del primer árbol de consulta lógico, que representa la expresión relacional, empieza el proceso de optimización de la consulta.

La **optimización de consultas** consiste en encontrar el plan de ejecución más eficiente de entre varias estrategias disponibles para procesar una consulta dada.

La optimización de consultas se lleva a cabo desde tres vertientes:

1) Optimización semántica, que consiste en reescribir la consulta basándose en las restricciones especificadas en el esquema de la base de datos.

2) Optimización sintáctica, que consiste en transformar heurísticamente la expresión relacional original en otra equivalente, pero que sea mucho más eficiente. Del mismo modo que una consulta se puede expresar de diferentes formas en SQL, una consulta SQL también se puede traducir a diferentes expresiones de álgebra relacional.

Supongamos el esquema de la base de datos *COMPANY* definido a continuación:

```
Jobs(jobId,jobName)

Locs(locId,streetAddress,postalCode,stateProvince,city,countryId)

Emp(empId,firstName,lastName,jobId,deptId,hireDate,
salary,manager)
{jobId} REFERENCES Jobs(jobId),
{manager} REFERENCES Emp(empId),
{deptId} REFERENCES Dept(deptId)

Dept(deptId, deptName, managerId, locId),
{locId} REFERENCES Locs(locId),
{managerId} REFERENCES Emp(empId)
```

Imaginemos que queremos consultar los empleados de la base de datos *COMPANY* que pertenecen al departamento "IT" y que son alta en la empresa a partir del 1 de septiembre del 2007.

```
SELECT *
FROM Emp e, Dept d
WHERE e.dept Id = d.dept Id
AND (e.hireDate > '01-SEP-07'
AND d.deptName LIKE 'IT') ;
```

A partir de esta consulta, podemos obtener tres expresiones de álgebra relacional equivalentes distintas:

Reflexión

Prestad atención al hecho de que la serie de operaciones de álgebra relacional es diferente en cada caso.

a) Se realiza el producto cartesiano entre las relaciones *Emp* y *Dept* para posteriormente realizar las operaciones de selección correspondientes:

$$\sigma_{(deptName='IT')\wedge(hireDate>'01\text{-}SEP\text{-}07')\wedge(Emp:deptId=Dept:deptId)}(Emp \times Dept)$$

b) Se combinan las tablas y después se realizan las operaciones de selección:

$$\sigma_{(deptName='IT')\wedge\ (hireDate>'01\text{-}SEP\text{-}07')}(Emp \bowtie_{(Emp:deptId=Dept:deptId)} Dept)$$

c) Primero se realizan las operaciones de selección correspondientes a cada tabla y después se combinan los resultados:

$$(\sigma_{(deptName='IT')}\ Dept) \bowtie_{(Emp:deptId=Dept:deptId)} (\sigma_{(hireDate>'01\text{-}SEP\text{-}07')}(Emp))$$

3) Optimización física, que permite escoger, de entre los distintos planes de evaluación, cada uno con costes diferentes, el que resulte más eficiente. Para determinar el coste más eficiente de cada consulta, es preciso conocer el coste de la secuencia de operaciones del álgebra que la forman y, por lo tanto, de cada operación, lo cual a menudo depende de diferentes parámetros. Así pues, para especificar completamente cómo se evaluará la consulta, hay que determinar qué algoritmos se utilizarán para cada operación y cuáles serán los índices empleados. Estas especificaciones se denominan **primitivas de evaluación**, y la secuencia de estas primitivas se denomina **plan de ejecución** de una consulta.

Reflexión

Antes del estudio de la optimización física, se ofrecerá una visión de la estimación de costes para las diferentes operaciones del álgebra relacional.

El **plan de ejecución** es la secuencia de pasos que realizará el SGBD para ejecutar una sentencia SQL.

El SGBD construye un plan de ejecución que minimiza el uso de recursos. Así, trata de reducir en la medida de lo posible el tiempo de ejecución de la consulta, ya sea minimizando el tiempo de cada operación o maximizando el número de operaciones paralelas. Por último, el **motor de ejecución** será el que ejecutará el plan trazado y obtendrá el resultado de la consulta a partir de los datos almacenados.

1.1. Descomposición de consultas

Esta es la primera fase del procesamiento de una consulta. Se comprueba que la consulta sea sintáctica y semánticamente correcta y, si es el caso, se transforma la consulta expresada en lenguaje SQL en un conjunto de operaciones de álgebra relacional.

Las etapas típicas de la descomposición de consultas son:

1) validación léxica y sintáctica,
2) normalización de la consulta,
3) análisis semántico, y
4) simplificación.

1.1.1. Validación léxica y sintáctica

La consulta se analiza léxica y sintácticamente empleando las técnicas de compiladores de los lenguajes de programación.

Se accede al diccionario de datos y se comprueba que todas las tablas y atributos que se mencionan en la consulta realmente existen. También se comprueba que el usuario tiene los derechos de acceso correspondientes y que las operaciones realizadas son adecuadas para aquel tipo de objeto.

Ved también

Los derechos de acceso se estudian en el apartado 3 de este módulo didáctico.

Ejemplo de validación léxica y sintáctica

Supongamos el esquema de la base de datos *COMPANY* y la siguiente consulta:

```
SELECT emplId
FROM Emp
WHERE deptId LIKE '10';
```

Esta consulta sería rechazada por dos motivos:

1) El atributo *emplId* no está definido por la relación *Emp*: habría que denominarlo *empId*.

2) La comparación *LIKE '10'* es incompatible con el tipo de datos *deptId*, que es numérico en lugar de ser una cadena de caracteres.

Finalizado este análisis, la consulta se ha transformado en un árbol de consulta construido de la siguiente manera:

1) Por cada relación base de la consulta se crea un nodo hoja.

2) Por cada operación intermedia producida por una operación de álgebra relacional se crea un nodo no hoja.

3) El resultado de la consulta se representa como la raíz del árbol.

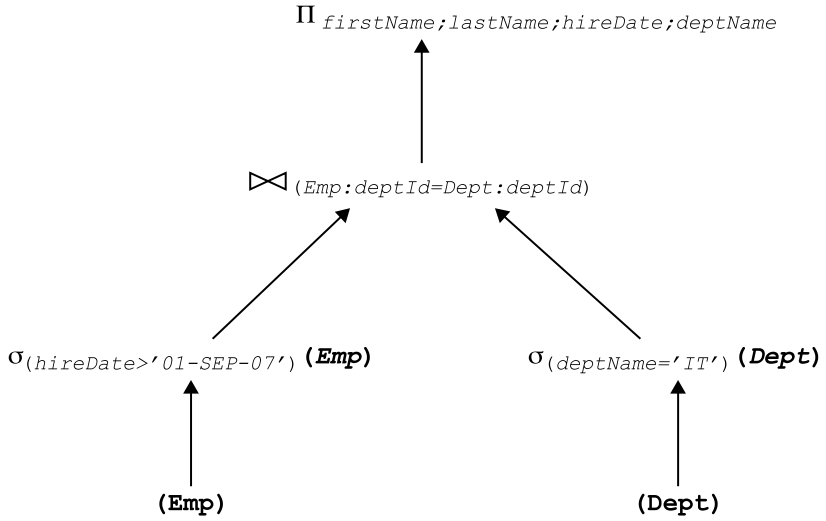
4) La secuencia de operaciones se dirige de los nodos hoja al nodo raíz.

Ejemplo de transformación de una consulta en un árbol de operaciones relacionales

Consideremos la siguiente consulta:

```
SELECT e.firstName, e.lastName, e.hireDate, d.deptName
FROM Emp e, Dept d
WHERE e.dept Id = d.dept Id AND (e.hireDate > '01-SEP-07'
AND d.deptName LIKE 'IT');
```

Esta consulta genera el plan de evaluación siguiente:



Que corresponde a la siguiente expresión relacional:

$\Pi_{firstName;lastName}((\sigma_{(hireDate>'01\text{-}SEP\text{-}07')}(Emp)) \bowtie_{(Emp:deptId=Dept:deptId)} (\sigma_{(deptName='IT')}(Dept)))$

1.1.2. Normalización de la consulta

Se evalúa el predicado de la cláusula WHERE, que a menudo suele ser lo bastante complejo como para convertir la consulta en una de estas formas normalizadas:

- **Forma normal conjuntiva**: genera una secuencia de conjunciones conectadas con el operador $\wedge$ (AND). Cada conjunción puede contener uno o más predicados conectados con el operador $\vee$ (OR).

Ejemplo

$(jobId = 'STClerk' \vee salary > 2000) \wedge manager = 121$

- **Forma normal disyuntiva**: genera una secuencia de disyunciones conectadas con el operador $\vee$ (OR). Cada disyunción puede contener uno o más predicados conectados con el operador $\wedge$ (AND).

Ejemplo

$(jobId = 'STClerk' \wedge salary > 2000) \vee (jobId = 'STClerk' \wedge manager = 121)$

1.1.3. Análisis semántico

En esta etapa se lleva a cabo un análisis semántico con el objetivo de rechazar las consultas normalizadas que sean contradictorias o que no estén bien formuladas.

Una consulta incorrecta es aquella cuyos componentes, una vez formulada, no permiten la generación de resultado, lo que puede suceder si falta alguna especificación de combinación.

Ejemplo de consulta incorrecta

```
SELECT e.firstName, e.lastName
FROM Emp e, Dept d, Locs l
WHERE e.deptId=d.deptId
AND l.city LIKE 'Paris'
AND e.salary > 2000;
```

Esta consulta se considera incorrecta porque las conexiones entre relaciones no están completas, ya que se ha omitido la condición de combinación *d.locId=l.locId*.

Una **consulta contradictoria** es aquella que contiene en su formulación algún predicado que ninguna tupla puede satisfacer.

Ejemplo de consulta contradictoria

```
SELECT firstName, lastName, salary
FROM Emp
WHERE salary < 1200 AND salary > 2000;
```

Esta consulta es contradictoria porque las dos condiciones de la cláusula *WHERE* son incompatibles.

1.1.4. Simplificación

El objetivo de esta etapa es detectar predicados redundantes, eliminar expresiones comunes y transformar una consulta en otra semánticamente equivalente pero que se pueda calcular de una forma más eficiente. Para realizar este proceso, se aplican las reglas de la lógica.

Ejemplo de simplificación

Consideremos la siguiente consulta:

```
SELECT firstName, lastName, salary
FROM Emp
WHERE salary > 1200 AND salary > 2000;
```

Esta consulta se puede simplificar, ya que la segunda condición incluye la primera:

```
SELECT firstName, lastName, salary
FROM Emp
WHERE salary > 2000;
```

Ejemplo de simplificación de predicados redundantes

Consideremos también esta otra consulta:

```
SELECT firstName, lastName, salary, managerId, jobId
FROM Emp
WHERE (jobId LIKE 'STClerk' AND manager = 121)
   AND manager = 121;
```

Se puede transformar en otra consulta con una condición WHERE equivalente:

```
SELECT firstName, lastName, salary, managerId, jobId
FROM Emp
WHERE jobId LIKE 'STClerk' AND manager = 121;
```

1.2. Optimización semántica

La optimización semántica utiliza una técnica que emplea las restricciones especificadas en el catálogo de la base de datos para reducir el espacio de búsqueda.

Ejemplo de optimización semántica

Suponiendo una restricción que indica que el salario debe ser superior a 500 euros, la consulta que presentamos a continuación:

```
SELECT firstName, lastName, salary
FROM Emp
WHERE salary > 500 AND deptId = 3 ;
```

podría simplificarse así:

```
SELECT firstName, lastName, salary
FROM Emp
WHERE deptId = 3 ;
```

Ejemplo de optimización semántica

Del mismo modo, consideremos esta otra consulta:

```
SELECT e.deptId
FROM Emp e, Dept d
WHERE e.deptId = d.deptId;
```

Dado que todos los valores de *e.deptId* deben ser valores de *d.deptId*, la consulta que presentamos a continuación es equivalente a la anterior:

```
SELECT e.deptId
FROM Emp e;
```

La implementación de este tipo de optimizadores en el SGBD permite incrementar el rendimiento, sobre todo en aquellos sistemas que tengan una semántica muy rica y completamente implementada, ya que evita buscar según aquellas condiciones que no se cumplen para ninguna fila de la tabla.

1.3. Optimización sintáctica

Mediante la aplicación de diferentes reglas de transformación, el optimizador puede convertir una expresión de álgebra relacional en otra expresión equivalente pero más eficiente. Para llevar a cabo este proceso, se aplican las reglas de equivalencia entre las operaciones de álgebra relacional, con el objetivo de encontrar una expresión equivalente.

1.3.1. Reglas de equivalencia

Las reglas de equivalencia y las estrategias de procesamiento heurístico que se utilizan para reestructurar el árbol de operaciones de álgebra relacional son las siguientes:

Ved también

Podéis consultar varios ejemplos de reglas de equivalencia en los anexos del apartado 4.

1) Capacidad de transformación de las operaciones conjuntivas de selección en una cascada de operaciones individuales de selección.

2) Conmutatividad en las operaciones de selección.

3) En una secuencia de operaciones de proyección, solo hace falta la última proyección de la secuencia.

4) Conmutatividad de la selección y la proyección.

5) Conmutatividad de la combinación Theta (y del producto cartesiano).

Combinaciones Theta

Combinaciones que utilizan los operadores de comparación como condición de combinación.

6) Conmutatividad de la selección y de la combinación Tetha (y del producto cartesiano).

7) Conmutatividad de la proyección y de la combinación Tetha (y del producto cartesiano)

8) Conmutatividad de la unión y de la intersección, pero no de la diferencia.

9) Conmutatividad de la selección y de las operaciones de conjuntos (unión, intersección y diferencia).

10) Conmutatividad de la proyección y de la unión.

11) Asociatividad de la combinación Tetha (y del producto cartesiano).

12) Asociatividad de la unión y de la intersección (pero no de la diferencia de conjuntos).

1.3.2. Estrategias de procesamiento heurístico

Una serie de reglas heurísticas permiten encontrar una buena expresión equivalente, muchas veces la mejor, a partir de la aplicación de las propiedades mencionadas anteriormente.

Ved también

Observad cómo se han aplicado las estrategias de procesamiento en el árbol representado en el subapartado 1.1.1 de este módulo didáctico.

Los heurísticos normalmente aceptados son los que se presentan a continuación:

1) Realizar las operaciones de selección lo antes posible. Debido a que estas operaciones reducen la cardinalidad de la relación resultante, utilizaremos la regla 1 para "conectar en cascada" todas las operaciones de selección. Después aplicaremos las reglas 2, 4, 6 y 9, definidas anteriormente, referentes a la conmutatividad de la selección con operaciones unarias y binarias. Las operaciones de selección aparecerán cerca de los nodos hoja del árbol indicando que son las primeras en realizarse. Se mantendrán juntos los predicados referentes a la misma relación.

2) Transformar el producto cartesiano de dos relaciones seguido de una operación de selección en una operación de combinación.

3) Emplear la asociatividad de las operaciones binarias para reordenar los nodos hoja. El objetivo es que las operaciones de selección más restrictivas se ejecuten primero.

4) Realizar las operaciones de proyección lo antes posible.

5) El cálculo de las expresiones comunes sólo se realizará una vez.

1.4. Estimación de costes para las operaciones de álgebra relacional

Las implementaciones de operaciones del álgebra relacional son diferentes para cada SGBD, y cada una de ellas tiene un coste asociado. Cuando evaluamos el coste de una consulta, es decir, el tiempo de respuesta para el plan de evaluación de una consulta, hay que tener en cuenta diferentes factores, como el coste de CPU, los accesos a memoria, los accesos a disco e incluso el tiempo empleado por las comunicaciones con sistemas remotos (en el caso de los SGBD distribuidos).

En los grandes SGBD, el coste más importante corresponde al acceso a disco, puesto que es el dispositivo más lento. Así pues, para comparar las diferentes estrategias de manera relativa solo se utilizará el número de páginas de disco a las que se accede (por operaciones de lectura/escritura) en el peor de los casos.

Consideraciones sobre el coste de los accesos a disco

Consideremos que el subsistema de disco tarda t_T segundos en realizar una operación de transferencia de un bloque de datos con un tiempo de acceso a bloque t_B, que corresponde al tiempo de búsqueda en disco más la latencia rotacional. Entonces, una operación que transfiera *nBlocks* y ejecute *N* buscas tendría un coste temporal de $nBlocks \cdot t_T + N \cdot t_B$.

Parámetros de acceso a disco

Para tener una idea de los parámetros asociados a los accesos a disco, podemos suponer que $t_T = 0{,}1$ ms y $t_B = 4$ ms en caso de que la medida de un bloque sea de 4 kilobytes y se disponga de una velocidad de transferencia de 40 megabytes por segundo.

En las siguientes consideraciones, no se tendrán en cuenta las operaciones de escritura a disco que tardan el doble de tiempo ni tampoco si hay alguna información en memoria intermedia. Este valor de estimación de coste se expresará como CE.

1.4.1. Estadísticas de la base de datos

El éxito de la estimación de la medida y del coste de las operaciones intermedias depende de la calidad y del grado de actualización de la información estadística almacenada en un SGBD. Para poder estimar el coste de cada uno de los algoritmos, es preciso disponer de valores estadísticos de las tablas que intervienen en la consulta, la mayoría de los cuales son valores medios que se mantienen calculados (o se recalculan de vez en cuando) en el diccionario de datos. Los estadísticos más importantes son los que se relatan a continuación.

Para cada relación (o tabla) R:

1) $nTuples(R)$: número de tuplas de la relación R; es decir, su cardinalidad.

2) $blockFactor(R)$: factor de bloqueo de R; el número de filas de R que caben en una página.

3) $nBlocks(R)$: número de bloques requeridos para guardar todas las tuplas de R, donde:

$$nBlocks(R) = nTuples(R)/blockFactor(R)$$

4) $t(R)$: tamaño en bytes de una fila de R.

Para cada atributo A de la tabla R:

1) $nDistinct_A(R)$: número de valores diferentes del atributo A que aparecen en R.

2) $min_A(R)$: menor valor del atributo A en la tabla R.

3) $max_A(R)$: mayor valor del atributo A en la tabla R.

4) $CS_A(R)$: cardinalidad de la selección del atributo. Es el número medio de filas que cumplen una condición sobre el atributo A.

Para una condición de igualdad sobre el atributo A, el cálculo es:

a) Si A es una clave candidata:

$$CS_A(R) = 1$$

b) Si los valores están distribuidos de forma uniforme:

$$CS_A(R) = nTuples(R)/nDistinct_A(R)$$

También es posible, suponiendo una distribución uniforme, calcular la cardinalidad de selección para otros casos diferentes de la igualdad:

a) Para predicados de comparación del tipo $A > a$:

$$CS_{A>a}(R) = nTuples(R) \cdot (max_A(R) - a)/(max_A(R) - min_A(R))$$

b) Para predicados de comparación del tipo $A < a$:

$$CS_{A<a}(R) = nTuples(R) \cdot (a - max_A(R))/(max_A(R) - min_A(R))$$

c) Para A $\subset (a_1, \ldots, a_n)$, donde n es el número de elementos del conjunto $(a_1, \ldots, a_n)$:

$$CS_{A\subset(a1,\ ...,\ an)}(R) = n \cdot (nTuples(R)/nDistinct_A(R))$$

d) Para la disyunción de dos predicados, $A \vee B$:

$$CS_{A\vee B}(R) = CS_A(R) \cdot CS_B(R)/nTuples(R).$$

e) Para la conjunción, $A \wedge B$:

$$CS_{A\wedge B}(R) = CS_A(R) + CS_B(R) - (CS_A(R) \cdot CS_B(R)) \ / \ nTuples(R)$$

Para cada índice multinivel *I de un* atributo *A*:

1) *nLevelsA*(*I*): el número de niveles en *I*.

2) *nLfBlocksA*(*I*): el número de páginas de hojas en *I*.

Otro dato importante es el número de páginas de memoria, *M*, que se pueden utilizar como memoria intermedia.

1.4.2. Operación de selección

Hay muchos algoritmos para la selección de valores en una tabla. A continuación, hacemos un repaso de los más importantes e indicamos su coste:

1) Búsqueda lineal sobre ficheros no ordenados: con este algoritmo se exploran todas las páginas que forman la tabla y se comprueba cada una de las filas para ver si contiene el valor adecuado del atributo:

Si se trata de un atributo que no es una clave candidata, el coste será:

$$CE = nBlocks(R)$$

Si el atributo sobre el cual se hace la búsqueda es una clave candidata, de media lo encontraremos hacia la mitad de la tabla, a veces antes y a veces después. En este caso el coste será:

$$CE = nBlocks(R)/2$$

2) Búsqueda binaria sobre ficheros ordenados: consiste en examinar primero la posición central de la tabla de forma que se elimina la mitad que no corresponde y se repite el procedimiento hasta que se encuentra el valor buscado:

- Para encontrar una tupla con un atributo clave, el coste será:

$$CE = \log_2(nBlocks(R))$$

- Si se trata de un atributo no clave y se supone una distribución uniforme de los valores, tendremos:

$$CE = [\log_2(nBlocks(R)] + [CS_A(R)/blockFactor(R)] - 1$$

3) Igualdad con una clave *hash*: si el atributo es una clave *hash*, se aplica el algoritmo *hash* para calcular la dirección correspondiente a una tupla determinada. Si no hay desbordamiento, el coste será 1.

4) Igualdad de clave primaria con un índice B^+ tree: si la búsqueda se hace con respecto a la igualdad de una clave primaria sobre la cual se ha definido un índice con estructura de árbol B^+, el número de páginas leídas será igual al número de niveles del árbol más la página de los datos, es decir:

$$CE = nLevels_A(I) + 1$$

B^+ tree

Un árbol B^+ es un tipo de estructura de datos en forma de árbol donde toda la información se guarda en las hojas, ya que los nodos internos solo contienen claves y punteros. De este modo, se pueden crear índices multinivel y dinámicos, con un límite máximo y mínimo en el número de claves por nodo.

5) Desigualdad con la clave primaria: si la condición de búsqueda es la de desigualdad con la clave principal, se puede emplear el índice para localizar la tupla que satisface el predicado $A = x$ y así, posteriormente, leer las tuplas que se encuentran situadas antes o después de la localizada.

6) Igualdad con un índice secundario: si se trata de un índice que no representa una clave candidata y, por lo tanto, pueden existir diferentes valores que cumplen la igualdad, se deberán leer todas las páginas que los contengan y el coste será:

$$CE = nLevels_A(I) + CS_A(R)/blockFactor(R))$$

Ejemplo de estimación de coste para una operación de selección

Para lograr el propósito de este ejemplo, se dan los siguientes supuestos referentes a la relación *Emp* descrita anteriormente en el esquema de la base de datos *COMPANY*.

- Hay un índice *hash* sobre el atributo de clave principal *empId*.
- Hay un índice de agrupamiento sobre el atributo de clave externa *deptId*.
- Hay un índice árbol B^+ sobre el atributo *salary*.
- La relación *Emp* tiene las siguientes estadísticas almacenadas en el catálogo del sistema:

$nTuples(Emp) = 3000$
$blockFactor(Emp) = 30) \Rightarrow nBlocks(Emp) = 100$
$nDistinct_{jobId}(Emp) = 50) \Rightarrow CS_{jobId}(Emp) = 60$
$nDistinct_{firstName}(Emp) = 3000) \Rightarrow CS_{firstName}(Emp) = 1$

$nDistinct_{salary}(Emp) = 500) \Rightarrow CS_{salary}(Emp) = 6$
$min_{salary}(Emp) = 10.000$
$max_{salary}(Emp) = 50.000$
$nLevels_{jobId}(I) = 2$
$nLevels_{salary}(I) = 2$
$nLfBlocks_{salary}(I) = 50$

El coste estimado de una búsqueda lineal sobre el atributo *deptId* es de 50 bloques y el coste de una búsqueda lineal sobre un atributo no clave es de 100 bloques.

Se calcula el coste estimado para las siguientes operaciones:

1) $\sigma_{(empId='T450')}(Emp)$: la operación de selección contiene una condición de igualdad sobre la clave principal. El atributo *empId* está almacenado mediante un método *hash*, por lo tanto el coste estimado será de un bloque ($CE = 1$) y la cardinalidad estimada de la relación resultante será:

$$CS_{empId}(Emp) = 1$$

2) $\sigma_{(firstName='Smith')}(Emp)$: el atributo es no clave y no indexado, por lo que no se puede mejorar el método de búsqueda lineal. El coste estimado obtenido será de $CE = 100$. La cardinalidad estimada se ha calculado previamente:

$$CS_{firstName}(Emp) = nTuples(Emp) / nDistinct_{firstName}(Emp) = 3.000 / 3.000 = 1.$$

3) $\sigma_{(jobId='prog')}(Emp)$: el atributo del predicado es una clave externa con un índice de *clustering*, por lo tanto el coste será $CE = 2 + [60/30] = 4$ bloques. La cardinalidad estimada será:

$$CS_{deptIp}(Emp) = 60$$

4) $\sigma_{(salary>20.000)}(Emp)$. El predicado implica una búsqueda dentro del rango del atributo *salary* que tiene un índice árbol B^+, por lo que para calcular el coste estimado se realizará el cálculo $CE = 2 + [50/2] + [3.000/2] = 1.527$. La cardinalidad estimada será:

$$CS_{salary>20.000}(Emp) = [3.000 \cdot (50.000 - 20.000)/(50.000 - 10.000)] = 2.250$$

7) Condición de igualdad en un índice secundario sin agrupamiento: si en el predicado aparece una condición de igualdad para el atributo *A*, que no es clave principal, pero es un índice secundario de tipo B+– tree sin agrupamiento, entonces las tuplas se ubican en diferentes bloques. En este caso, el coste estimado sería:

$$CE=nLevels_A(I) + SC_A(R)$$

8) Condición de desigualdad en un índice secundario de tipo B+–tree: si el predicado implica una condición de desigualdad para el atributo *A* y este fuera un índice secundario de tipo B+–tree, entonces, a partir de los nodos hoja del árbol, se puede realizar la exploración del nodo más pequeño hasta el valor *x*, en el caso de que las condiciones de desigualdad fuesen $A<x$ o $A<=x$, o de otro modo, desde *x* hasta el valor máximo, en los casos $A>x$ o $A>=x$.

Considerando una distribución uniforme, se puede estimar un acceso a la mitad de los bloques y a la mitad de las tuplas, el coste estimado sería:

$$CE = nLevels_A(I) + nLfBlocks_A(I)/2 + nTuples(R)/2$$

Con lo que queda explicado el cálculo del punto 4; $\sigma_{(salary>2000)}(Emp)$; CE=2+50/2+3000/2.

1.4.3. Operación de ordenación

Muchas veces resulta necesario ordenar los datos de una tabla, ya sea porque se desea el resultado ordenado o porque se utiliza para implementar una operación de combinación de forma más eficiente.

Quicksort

El ordenamiento rápido, *quicksort* en inglés, es un algoritmo basado en la técnica del "divide y vencerás" que permite, de media, ordenar n elementos en un tiempo proporcional a $n \cdot \log(n)$.

La ordenación se puede conseguir creando un índice sobre el atributo objeto de la ordenación y utilizándolo para realizar una lectura ordenada. No obstante, hay que tener en cuenta que esta ordenación se efectúa en el nivel lógico, pues físicamente se puede traducir en un acceso a disco para cada tupla más la transferencia del bloque correspondiente. Esto puede ser muy costoso, ya que el número de registros suele ser superior al número de bloques, dado que las tuplas no tienen por qué estar almacenadas físicamente de manera consecutiva.

En el caso de que toda la relación se pueda almacenar en la memoria principal, se pueden emplear técnicas de ordenación rápida, como el algoritmo *quicksort*.

Marcos de página

Consideramos que el número de marcos de página, M, es el número de bloques de disco que se pueden almacenar en la memoria intermedia de la memoria principal.

Cuando la relación no cabe en la memoria, se suele utilizar el algoritmo de ordenación-fusión externa. Consideremos que se dispone de M marcos de página en la memoria intermedia de la memoria principal. Esta operación consta de dos fases:

En la primera fase, el algoritmo divide la relación en $N = [nBlocks(R)/M]$ partes, las carga sucesivamente en la memoria para ordenar cada una de ellas y crear un archivo de secuencias S_i.

En la segunda etapa, y en caso de que el número de secuencias N sea inferior a M:

1) En el primer bloque de cada archivo de secuencia, se lee la primera tupla de cada bloque, para proceder a la fusión ordenada de los primeros bloques de cada secuencia.

2) En caso de que el contenido del bloque se haya agotado, se lee el siguiente bloque del mismo archivo de secuencias hasta agotar todos los bloques de cada secuencia, lo cual implicará que se ha obtenido la ordenación.

3) En el caso de que el número de secuencias N todavía sea superior a M, se fusionan las $M - 1$ primeras secuencias para formar otra secuencia ordenada. En este punto, se ha reducido el número de secuencias en $M - 1$.

4) En caso de que el número de secuencias sea menor que M, entonces se puede proceder a la fusión definitiva. En caso contrario, se repite el procedimiento con otras $M - 1$ secuencias hasta obtener un número de secuencias menor que M.

El coste total de este algoritmo es tal como se indica a continuación.

El número inicial de secuencias es $N = [nBlocks(R)/M]$. Debido a que este decrece en un factor $M - 1$ en cada ciclo de fusión, el número total de ciclos requeridos será de $\log_{M-1} \cdot [nBlocks(R)/M]$. En cada ciclo se leen y se escriben los bloques de la relación una sola vez, por lo que el número de transferencias de bloque será $nBlocks(R)[\log_{M-1}[(nBlocks(R)/M] + 1]$. También es necesario añadir los costes de búsqueda en disco. Durante la fase de fusión, si se leen simultáneamente $nBlocks[S]$ de cada secuencia, entonces cada paso implica como mínimo $nBlocks(R) = nBlocks(S)$ búsquedas para la lectura de datos. También hay que tener en cuenta que, aunque la salida se escriba secuencialmente, el cabezal puede haberse desplazado, por lo que habrá que añadir $2[nBlocks(R)/nBlocks(S)]$ búsquedas por cada paso, excepto en el paso final.

Así pues, el coste estimado será:

$$CE = 2[nBlocks(R)/nBlocks(S)] + \\ + [nBlocks(R)/nBlocks(S)](2[log_{M-1}[(nBlocks(R)/M] - 1])$$

1.4.4. Operación de proyección

Una proyección implica dos operaciones: eliminar los atributos no deseados y eliminar, si es preciso, las tuplas repetidas que se han generado con la cláusula *DISTINCT*.

Observación

Si dentro de la lista de atributos de la proyección se incluyen todos los que forman la clave primaria, no se producirán duplicados.

Se puede implementar fácilmente la eliminación de duplicados empleando la ordenación de las tuplas y utilizando como clave de ordenación los atributos resultantes de la proyección. Las tuplas idénticas aparecen contiguas durante la operación de ordenación y, por lo tanto, se pueden eliminar todas menos una. El coste estimado en el peor de los casos es el mismo que el coste estimado en el peor de los casos en la operación de ordenación.

1.4.5. Operación de combinación

Las operaciones de combinación, aparte de las del producto cartesiano, suelen ser las que consumen más tiempo durante el procesamiento de consultas, por lo que marcarán de manera importante la eficiencia global del SGBD.

Los algoritmos más importantes son: la combinación de ciclo anidado, la combinación de ciclo anidado indexado, la combinación por ordenación por fusión, la combinación por función resumen y la combinación por agrupación (clúster).

Combinación de ciclo anidado

La combinación de ciclo anidado[2] consiste en dos ciclos anidados: el bucle externo recorre todas las tuplas de la primera relación *R* y el bucle interno recorre todas las tuplas de la relación *S*. El coste estimado para esta solución es:

(2)En inglés, *nested loop join*.

$$CE = nBlocks(R) + nBlocks(R) \cdot nBlocks(S)$$

Podemos apreciar claramente que, para reducir el coste, es preciso que la primera relación que se utiliza en el bucle externo ocupe el menor número posible de bloques.

Una mejora consiste en leer tantos bloques como sea posible de la relación más pequeña, y reservar un bloque para la relación interna y otro para la relación resultante. Si la memoria intermedia[3] permite almacenar un número de bloques, *nBuffer*, entonces se leen (*nBuffer* – 2) bloques *de R* y un bloque *de S* cada vez. Con esta técnica, la estimación del coste sería:

(3)En inglés, *buffer*.

$$CE = nBlocks(R) + nBlocks(S) \cdot nBlocks(R)/(nBuffer - 2)$$

También deberíamos considerar si la memoria es suficiente para que se puedan leer de la memoria intermedia de la base de datos después de la primera lectura todos los bloques de *R*. En este caso, la estimación del coste sería:

$$CE = nBlocks(R) + nBlocks(S)$$

Combinación de ciclo anidado indexado

El coste de la combinación de ciclo anidado indexado[4] varía según el método de indexación que se utilice. Si existe un índice o función resumen[5] sobre los atributos de combinación de la relación interna, entonces en el bucle interior no se efectúa un recorrido por todas las páginas de la tabla, sino que se va directamente, mediante el índice o la función resumen, a la página adecuada.

(4)En inglés, *index nested loop join*.

(5)En inglés, *hash function*.

Si se tiene un índice árbol B^+ sobre el atributo *A*, para encontrar el coste de la consulta, será necesario conocer: *nLevelsA*(*I*), el número de niveles del árbol B^+ del índice *I* sobre el atributo *A*; y $CS_A(R)$, la cardinalidad de selección del atributo *A* de la tabla *R* y del número de filas que caben en una página *blockFactor*(*R*). Si se trata de un índice sobre una clave candidata de *R*, el coste será:

$$CE = nBlocks(R) + nTuples(R) \cdot (nLevels_A(I) + 1)$$

Si el atributo es un índice de agrupación, entonces la estimación del coste será:

$$CE = nBlocks(R) + nTuples(R) \cdot (nLevels_A(I) + [CS_A(R)/blockFactor(R)])$$

Combinación por ordenación por fusión

En una combinación por ordenación por fusión[6], la combinación más eficiente se logra cuando las relaciones están ordenadas según los atributos de combinación. De no ser así, se requerirá un paso previo para ordenarlas. Una vez ordenadas, solo hay que leer secuencialmente cada una de las tablas y añadir a los resultados aquellos valores que coinciden. Si asumimos que la combinación es de tipo muchos a muchos, es decir, que hay diferentes tuplas de *R* y de *S* con el mismo valor de combinación, y también asumimos que cada conjunto de tuplas con el mismo valor puede caber en la memoria intermedia, entonces solo habrá que leer cada bloque de la relación una única vez. Así, la estimación del coste sería:

(6)En inglés, *sort-merge join*.

$$CE = nBlocks(R) + nBlocks(S)$$

Si es necesario ordenar una de las relaciones, entonces hay que añadir el coste de ordenación:

$$CE = nBlocks(R) + nBlocks(S) \cdot \log_2(nBlocks(S))$$

teniendo en cuenta que se redondea por exceso el cálculo de cada logaritmo.

Combinación por función resumen

El algoritmo de combinación por función resumen[7] se basa en emplear una función resumen (*hash*) que presente características de uniformidad y aleatoriedad para dividir las filas de las dos tablas en conjuntos que tengan el mismo valor de la función resumen. Después se realiza la combinación de las filas de cada partición.

(7)En inglés, *hash join*.

Funciones resumen

Una función resumen es un algoritmo u operación que permite obtener un sumario o resumen a partir de un conjunto de datos. Este sumario o resumen está asociado a los datos originales, y cualquier cambio en los datos originales tiene que repercutir en el sumario o resumen.

El coste de la combinación por función resumen es:

- $CE = 3 \cdot (nBlocks(R) + nBlocks(S))$, si el índice *hash* se almacena en memoria y no se tienen en cuenta los desbordamientos.

- $CE = 2 \cdot (nBlocks(R) + nBlocks(S)) \cdot [\log_{nBuffer-1}(nBlocks(S)) - 1] + nBlocks(R) + nBlocks(S)$, en otro caso.

Ejemplo de estimación de coste para una operación de combinación

Para lograr el propósito de este ejemplo, se dan los siguientes supuestos referentes a las relaciones *Emp* y *Jobs*.

- Existe *hash* sin desbordamiento sobre el atributo de clave principal *depId*.
- Existen unos cien bloques de memoria intermedia de la base de datos.
- Tenemos las siguientes estadísticas almacenadas en el catálogo del sistema:
  `nTuples(Emp) = 3.000`
  `blockFactor(Emp) = 30)` ⇒ `nBlocks(Emp) = 100`

```
nTuples(Jobs) = 50
blockFactor(Jobs) = 10) ⇒ nBlocks(Jobs) = 5
```

Se quiere evaluar el coste de las diferentes estrategias para la combinación siguiente:

$$(Emp \bowtie_{Emp.jobId=Jobs.jobId} Jobs)$$

1) La combinación de ciclo anidado:

- La memoria intermedia sólo contiene un bloque *de Emp* y *Jobs*:
  $CE = nBlocks(Emp) + nBlocks(Emp) \cdot nBlocks(Jobs) = 100 + 100 \cdot 5 = 600$

- El número de bloques que se tratarán por *Emp* es (*nBuffer* – 2):
  $CE = nBlocks(Emp) + nBlocks(Jobs) \cdot nBlocks(Emp)/(nBuffer - 2) = 100 + 100 \cdot 5/98 = 106$

- Todos los bloques de la relación *Emp* caben en la memoria intermedia:
  $CE = nBlocks(Emp) + nBlocks(Jobs) = 100 + 5 = 105$

2) La combinación de ciclo anidado indexado:

$$nTuples(Emp) + nBlocks(Emp) = 3.000 + 100 = 3.100$$

3) La combinación de ordenación por fusión:

- Tuplas desordenadas:
  $CE = nBlocks(Emp) + nBlocks(Jobs) + nBlocks(Emp) \cdot \log_2(nBlocks(Emp)) + nBlocks(Jobs) \cdot \log_2(nBlocks(Jobs)) = 100 + 5 + 100 \cdot 7 + 5 \cdot 3 = 820$

- Tuplas ordenadas:
  $CE = nBlocks(Emp) + nBlocks(Jobs) = 100 + 5 = 105$

4) La combinación por función resumen. Si el índice *hash* cabe en la memoria:

$$CE = 3 \cdot (nBlocks(Emp) + nBlocks(Jobs)) = 3 \cdot (100 + 5) = 315$$

1.4.6. Operaciones de conjuntos de álgebra relacional

Las operaciones de unión $R \cup S$, intersección $R \cap S$ y diferencia $R - S$ se pueden implementar ordenando primero las dos relaciones según un mismo criterio y después produciendo el resultado.

Los resultados para cada operación se calculan de la siguiente manera:

- Cuando se realiza la unión de *R* y *S* ($R \cup S$), se realiza una lectura concurrente de las tuplas de las dos relaciones, y si se detecta la misma tupla en ambas, se elimina el duplicado.

- Cuando se realiza la intersección de *R* y *S* ($R \cap S$), sólo se almacenan las tuplas que aparezcan como duplicados en ambas relaciones.

- Cuando se realiza la diferencia $R - S$, se almacenan las tuplas de *R* que no aparezcan en *S*.

Todo esto tendría un coste $CE = nBlocks(R) + nBlocks(S)$, más el coste de la ordenación de *R* y *S*.

1.4.7. Agregación

Cualquier operación de agrupación se puede implementar de una manera parecida a la eliminación de duplicados, pero en lugar de eliminar las tuplas que tienen el mismo valor, se reúnen por grupos y se aplica la operación correspondiente para obtener el resultado.

Operaciones de agrupación

Son operaciones de agrupación las funciones *min*, *max*, *sum*, *count* y *avg*.

1.5. Optimización física

El optimizador físico de consultas es el componente del SGBD que prepara los planes de ejecución de las consultas para que se lleven a cabo en el mínimo tiempo. Hay varias posibilidades para conseguirlo:

- que el coste de CPU sea mínimo;
- que la necesidad de memoria sea mínima;
- que se minimicen los accesos a disco, y
- otras posibilidades.

Los algoritmos de optimización física se pueden clasificar en dos grandes familias:

1) **Optimización heurística**: También conocida como **optimización basada en reglas**, consiste en escoger un plan físico de ejecución con aquellos algoritmos que normalmente son más eficientes.

2) **Optimización basada en costes**: Consiste en encontrar todas las implementaciones físicas una vez se dispone de todo el espacio de posibles realizaciones de la consulta. Para cada plan se calcula el coste (tiempo, número de accesos a disco, etc.) y, finalmente, se escoge el de coste más bajo.

Los SGBD comerciales acostumbran a permitir escoger un tipo de implementación u otro, aunque la tendencia es ofrecer solo optimización basada en costes, puesto que aunque la heurística es muy rápida de calcular, puede escoger planes de ejecución muy poco eficientes.

Soluciones comerciales optimizadas

Si aumentamos la memoria de nuestro servidor, conseguiremos que muchos de los procesos de combinación se puedan hacer completamente en la memoria. Habrán cambiado todos los tiempos de ejecución, mientras que los planes de ejecución se mantendrán.

1.5.1. Optimización heurística

Para poder realizar una optimización heurística, es preciso que el SGBD tenga definido un orden de eficiencia para las diferentes operaciones lógicas y pueda escoger aquel que suele tener un coste más bajo.

Como la mayoría de las opciones de diseño, la optimización heurística tiene virtudes y defectos: su principal virtud es que no hay que efectuar ningún tipo de cálculo, la obtención de un buen plan de ejecución se realiza de manera inmediata; y el principal defecto es que a veces los planes propuestos pueden diferir mucho de los óptimos.

En una base de datos compleja, una consulta puede involucrar varias tablas, cada una con varios índices y condiciones de selección complejas. Esta complejidad significa que podría haber una gran cantidad de opciones resultantes, y el sencillo conjunto de reglas utilizadas por el optimizador basado en reglas no podría diferenciar lo suficientemente bien las elecciones como para asegurar la mejor elección.

Otra de las debilidades del optimizador basado en reglas es la resolución de las opciones de optimización en caso de obtener dos o más planes de ejecución con el mismo coste. En este caso, el optimizador considera la sintaxis de la sentencia SQL para resolver el empate. La ruta de ejecución ganadora se basa en el orden en que las tablas se presentan en la sentencia SQL.

1.5.2. Optimización basada en costes

Este método selecciona la ruta de ejecución que requiere el menor número de operaciones lógicas de E/S. Con esta finalidad se utilizan estadísticas que se consideran relevantes, recogidas sobre la composición de las estructuras de datos proporcionados. De este modo, el optimizador basado en costes puede conocer qué tabla es la más grande y puede seleccionarla como la tabla de la derecha para iniciar la consulta, independientemente de la sintaxis de la sentencia SQL. Así reduce el número de iteraciones y, si se tercia, el número de accesos a disco.

Ejemplo de reducción del número de iteraciones

Se puede entender el impacto potencial sobre la elección del orden en las tablas observando una situación sencilla en la que se hace una combinación de una tabla pequeña con diez registros, *SmallTable*, con una tabla con diez mil, *LargeTable*.

```
SELECT *
FROM SmallTable s, LargeTable l
WHERE s.id = l.id;
```

La sentencia anterior obtendría un resultado equivalente a esta otra:

```
SELECT *
FROM LargeTable l, SmallTable s
WHERE l.id = s.id;
```

El tamaño de las tablas y el orden en que éstas se leen tiene repercusión en el tiempo de ejecución de la consulta:

- Si el optimizador elige para leer *SmallTable* en primer lugar, el SGBD lee las diez filas y después *LargeTable* diez veces para encontrar las filas coincidentes de cada una de las diez filas.

- Si el optimizador elige *LargeTable* en primer lugar, el SGBD tiene que leer las diez mil filas de *LargeTable* y después las de *SmallTable* diez mil veces para encontrar los registros coincidentes.

Además, en el primer caso, las filas de *SmallTable* probablemente se podrían almacenar en memoria caché, lo que reduciría el impacto de lecturas de disco.

Independientemente del orden en que aparecen las tablas en la sentencia SQL, gracias a la información obtenida a partir de las estadísticas, el optimizador escogerá la tabla más grande como si esta apareciera la última en la sentencia, es decir, más a la derecha.

A partir de las estadísticas, se seleccionan los operadores físicos y se determina la estrategia de ejecución.

Las estadísticas y los histogramas

Las **estadísticas** registran la distribución de los datos y las características de almacenamiento de las tablas, columnas, índices y particiones a partir de los cuales el optimizador estima la cantidad de accesos a disco y memoria necesaria para ejecutar un plano físico particular.

Ved también

Algunos de los parámetros que almacenan los estadísticos se ven en el subapartado 1.4.1. de este módulo didáctico.

Los parámetros estadísticos que los SGBD almacenan más a menudo son los siguientes:

1) Tabla: número de filas, número de páginas, número de páginas vacías, longitud media de una fila.

2) Columna: número de valores diferentes en la columna, número de valores NULL en la columna, histograma de frecuencia de aparición de cada uno de los valores.

3) Índice: número de páginas, niveles, factor de agrupamiento.

En el SGBD comercial Oracle, se puede emplear el paquete PL/SQL DBMS_STATS para generar y gestionar las estadísticas de las tablas, columnas, índices, particiones y otros objetos de la base de datos.

Generar y gestionar estadísticas

Por ejemplo, podemos recopilar estadísticas para el esquema *'COMPANY'* utilizando la siguiente instrucción SQL, donde también se especifica en el segundo parámetro que calcule automáticamente el alcance del tamaño de la muestra.

```
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS('COMPANY',
   DBMS_STATS.AUTO_SAMPLE_SIZE);
```

Disponemos de diferentes opciones a la hora de recopilar estadísticas que permiten especificar el **tipo de muestreo**:

Reflexión

También se puede hacer que Oracle recopile las estadísticas al crear o reconstruir índices especificando la opción *COMPUTE STATISTICS* en las sentencias *CREATE INDEX* o *ALTER INDEX*.

- **muestreo basado en filas**, de forma que se ignora la ubicación física en disco; y

- **muestreo basado en bloques**, de modo que se escoge una muestra aleatoria de bloques y las estadísticas se generan a partir de los datos almacenados en ellos.

Generalmente, un muestreo utiliza menos recursos que si se calcula el valor exacto de la estructura completa.

Las estadísticas se almacenan en el diccionario de datos.

Un **histograma** es una estructura de datos que se puede utilizar para mejorar las estimaciones de las que puede disponer el optimizador.

Un histograma recoge un conjunto de valores junto con sus frecuencias relativas, y proporciona al optimizador las mejores estimaciones en presencia de una distribución uniforme. Existen dos **tipos de histogramas**:

Reflexión

En Oracle, es el usuario quien crea y mantiene los histogramas para las columnas apropiadas empleando el paquete PL/SQL DBMS_STATS.

- **Histograma equilibrado de ancho**, que divide los datos en un número fijo de rangos con el mismo ancho y para cada uno de ellos se realiza el cálculo del número de valores que les pertenece.

- **Histograma equilibrado de altura**, donde se determinan los rangos una vez distribuidos equinuméricamente los valores y se generan rangos de diferente ancho pero con un mismo valor de frecuencia relativa.

Operadores físicos y estrategias de ejecución

El término *operador físico* se utiliza para referenciar a un algoritmo específico que implementa la operación lógica de la base de datos. Por ejemplo, se puede escoger el operador físico de combinación mediante un bucle anidado indexado con encarrilamiento[8].

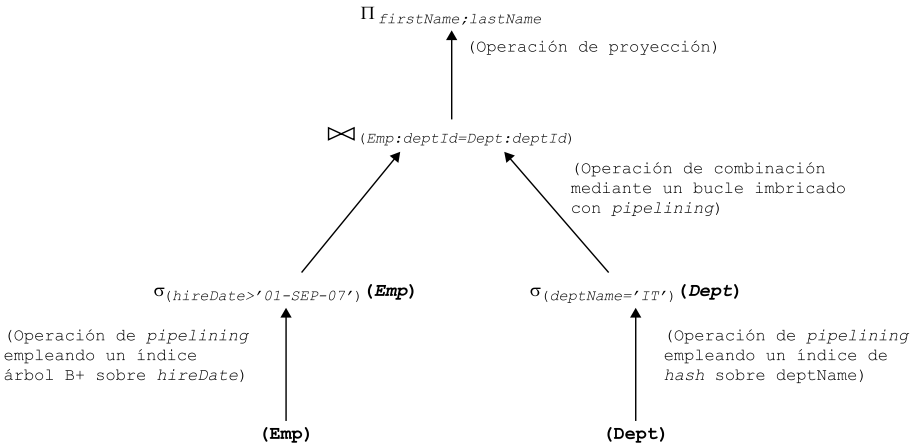
(8)En inglés, *pipelining*.

Encarrilamiento

El encarrilamiento, o *pipelining*, como se ve más adelante, es una técnica que consiste en aprovechar la salida de una operación como entrada de la siguiente con el objetivo de incrementar la eficiencia de las operaciones, ya que ahorra el coste de lectura/escritura de las relaciones intermedias.

Sustituir las operaciones lógicas en un árbol de álgebra relacional por operadores físicos produce una **estrategia de ejecución**, **plan de evaluación de la consulta** o **plan de acceso para la consulta**.

Consideremos la consulta de ejemplo del subapartado 1.1.1.:



Aunque los diferentes SGBD tienen distintas implementaciones para los diferentes operadores físicos, podemos considerar varios operadores abstractos para implementar las funciones especificadas en cada nodo del árbol:

1) ***tableScan*****(*R*)**: exploración de la tabla. Los diferentes bloques de *R* se leen en un orden arbitrario.

2) ***sortScan*****(*R*, *L*)**: exploración ordenada. Las tuplas de *R* se leen por orden de acuerdo con los criterios de ordenación especificados en la lista de atributos *L*.

3) ***indexScan*****(*R*, *P*)**: exploración indexada. El predicado *P* especifica una comparación entre el valor del atributo *A* y un valor *c*. Se accede a las tablas mediante un índice sobre el atributo *A*.

4) ***indexScan*****(*R*, *A*)**: exploración indexada. *A* es un atributo de *R*. Se extrae la relación *R* completa empleando un índice sobre el atributo *A*.

Además, la estrategia de ejecución soporta una interfaz iteradora uniforme que permite ocultar los detalles de implementación de cada operador:

1) ***open***: inicia el estado del iterador antes de extraer la primera tupla y asigna un espacio de memoria intermedia para las entradas y salidas. Se pueden indicar varios parámetros que permitan definir condiciones de selección de forma que modifiquen el comportamiento del operador.

2) ***getNext***: devuelve la siguiente tupla del resultado y la coloca en la memoria intermedia de salida. El estado del iterador se actualiza para reflejar las acciones que ya se han realizado.

3) ***close***: una vez finalizado el proceso de iteración, es decir, una vez generadas todas las salidas solicitadas, el operador realiza las operaciones de limpieza y desasigna espacio de memoria.

Al definir este tipo de operadores, el sistema puede implementar de manera natural el concepto de encarrilamiento o de materializar los valores intermedios.

1) Materialización: con este enfoque se inicia la evaluación para las operaciones de nivel más bajo a partir del árbol de operadores. Se ejecutan las operaciones con los algoritmos estudiados y después se almacenan los resultados en relaciones temporales. Así pues, el coste de toda la expresión es la suma de los costes de cada una de las operaciones más el coste de los resultados intermedios en disco.

2) Encarrilamiento: consiste en aprovechar la salida de una operación como entrada de la siguiente; de este modo nos ahorramos el coste de lectura/escritura de las relaciones intermedias. Para cada operación del encarrilamiento se modeliza un proceso aislado[9] que toma un flujo de tuplas de sus entradas y produce otro para su salida. Para esto es necesario definir en cada operación una memoria intermedia para la entrada y otra para la salida. Con el encarrilamiento también se puede conseguir que varias operaciones se ejecuten en paralelo, de forma que se gana velocidad a cambio de una mayor necesidad de memoria para llevar a cabo los diferentes procesos.

[9]En inglés, *thread*.

Ved también

Podéis consultar la sintaxis y funcionalidad de la sentencia *EXPLAIN PLAN* en la documentación de Oracle.

Visualización del plan de ejecución

Oracle permite visualizar el plan de ejecución que escogerá el optimizador mediante la sentencia *EXPLAIN PLAN*.

Esta sentencia resulta bastante útil cuando se desea analizar por qué la eficiencia de una consulta no es la esperada. La salida de esta sentencia se escribe de manera predeterminada en la tabla *PLAN_TABLE*.

2. Procesamiento de vistas

Desde un punto de vista teórico, una vista se puede definir como el resultado dinámico de una o más operaciones relacionales sobre tablas para producir una relación nueva.

Las vistas son relaciones virtuales que sólo están representadas por su nombre y su definición; no existen físicamente en la base de datos. Sólo sirven a partir del momento en que el usuario las utiliza.

La **sintaxis de creación de vistas** según el estándar SQL:2011 es la siguiente:

```
CREATE VIEW ViewName [(column_name), ... ]
AS query
[WITH [CASCADED|LOCAL]CHECK OPTION ]
```

El estándar SQL

Desde que el lenguaje SQL fue aceptado como un estándar, en primer lugar por el ANSI en 1986 y después por la ISO en 1987, se han ido añadiendo nuevas funcionalidades y características, que se han recogido en las diferentes versiones, documentadas en la referencia ISO/IEC 9075.

La cláusula *WITH CHECK OPTION* asegura que nunca podremos actuar sobre partes de una tabla que no están a la vista. Si se incluye la cláusula *LOCAL*, se valida la integridad de la vista para aquellas operaciones de inserción o actualización que se efectúen, mientras que si se utiliza la cláusula *CASCADED*, también se valida la integridad de todas las otras vistas que dependen de ella.

Ejemplos de creación de vistas

Si se tiene en cuenta la base de datos *COMPANY* definida anteriormente, algunas definiciones de vistas son las siguientes:

```
CREATE VIEW View01 AS
SELECT firstName, lastName, hireDate, salary
FROM Emp e
WHERE salary < 30000;

CREATE VIEW View02 AS
SELECT firstName, lastName, deptName
FROM Emp e, Dept d
WHERE e.deptId = d.deptId;

CREATE VIEW View03 AS
SELECT *
FROM View02
WHERE (firstName, lastName) IN
      (SELECT firstName, lastName
      FROM View01);
```

De manera opcional, se puede asignar un nombre a las diferentes columnas de la vista. En caso contrario, las columnas tomarán el mismo nombre que tienen en la subconsulta.

Si la subconsulta incluye columnas calculadas o funciones, será imprescindible definirles un nombre.

Fijaos en que, tal como se puede ver en el tercer ejemplo, en la subconsulta de una vista puede aparecer otra vista.

Por otro lado, ¿que pasaría si se realizara la siguiente actualización?:

```
UPDATE View01
SET salary = 45000
WHERE salary = 29000;
```

Se daría el caso inverosímil de que se ocultaría una fila a la vista. Una situación parecida se daría si se realizara la siguiente inserción:

```
INSERT INTO View01
VALUES ('John', 'Smith', '12/3/2012', 50000);
```

Si la vista se hubiera creado con la cláusula *WITH CHECK OPTION*, las operaciones se habrían comprobado antes para asegurar que las nuevas filas insertadas o actualizadas cumpliesen la condición de la vista.

Aunque una vista no ocupa mucho lugar en la base de datos, recordad que sólo se trata de una definición y que también se puede destruir.

Según el estándar SQL:2011, la **sintaxis de eliminación de vistas** es la siguiente:

```
DROP VIEW View_Name [CASCADE|RESTRICT]
```

La cláusula opcional *CASCADE* destruirá todos los objetos creados a partir de esta vista. En cambio, si se usa la cláusula *RESTRICT*, se impedirá la eliminación de la vista si existen otros objetos que dependen de ella.

Ejemplo de destrucción de una vista

Con la sentencia siguiente destruiremos la primera de las vistas creada en el ejemplo anterior:

```
DROP VIEW View01 CASCADE;
```

Al ejecutar esta sentencia, además de eliminar la vista *View01*, también se elimina la vista *View03*, ya que en su definición se utiliza *View01*. En caso de haber empleado la cláusula *RESTRICT*, no se habría eliminado ninguna de las dos vistas por esta dependencia.

2.1. Mecanismos de implementación de vistas

Existen dos estrategias fundamentales para implementar las vistas: reescribir la consulta o materializar la vista.

La **estrategia de reescritura** es la más utilizada por el SGBD. Según esta estrategia, la consulta se reescribe de forma que referencie las tablas de la base de datos.

Reescritura o cálculo *inline*

La estrategia de implementación de vistas por reescritura también se conoce con el nombre de *cálculo bajo mandato* o, en inglés, *inline*.

Ejemplo de implementación de vistas por reescritura

Considerad la base de datos definida por el diagrama relacional de la base de datos *COMPANY* y las vistas siguientes:

```
CREATE VIEW DeptSummaryView(department, salary, employees) AS
SELECT d.deptName, AVG(e.salary), COUNT(*)
FROM Emp e, Dept d, Jobs j
WHERE e.deptId = d.deptId
AND e.jobId = j.jobId
AND j.jobName NOT IN ('President', 'Manager')
GROUP BY d.deptName;
```

Una consulta como la que presentamos a continuación:

```
SELECT department, salary
FROM DeptSummaryView
WHERE employees > 4;
```

se convertiría en lo siguiente:

```
SELECT d.deptName, AVG(e.salary)
FROM Emp e, Dept d, Jobs j
WHERE e.deptId = d.deptId
AND e.jobId = j.jobId
AND j.jobName NOT IN ('President', 'Manager')
GROUP BY d.deptName HAVING COUNT(*) > 4;
```

La consulta que realmente se ejecuta con esta estrategia es mucho más compleja que la que se plantea inicialmente, y esta complejidad hará que las consultas requieran mucho tiempo de ejecución. El colmo de la ineficiencia se da cuando hay que hacer varias consultas seguidas sobre una misma vista. También puede ser difícil la conversión en caso de consultas muy complejas.

Con la segunda estrategia, llamada **materialización de la vista**, cuando se hace la consulta se crea una tabla temporal con el contenido físico de la vista.

Duración de las vistas

Si durante un periodo de tiempo la vista no se consulta, el propio sistema la destruye y la vuelve a construir de nuevo, si hace falta, en otro momento.

Ejemplo de implementación de vistas por materialización

Si se aplicara la estrategia de materialización de vistas en el ejemplo anterior, se crearía la tabla temporal *DeptSummaryView* con los datos del resultado de la vista y las consultas se harían físicamente sobre esta tabla.

Según esta estrategia, el caso de mínima eficiencia que presentaba la primera estrategia se transformaría en máxima eficiencia, dado que sólo tendría que calcular la tabla una única vez.

En contraste con esta ventaja, la estrategia de materialización de la vista presenta un inconveniente: si las tablas básicas a partir de las cuales se construye la vista varían de contenido, habrá que modificar la tabla temporal (actualización incremental). Esto puede resultar bastante costoso, sobre todo para algunos casos de funciones agregadas o en las que aparezcan cláusulas *GROUP BY*.

Cabría esperar de un buen SGBD que utilizara la primera estrategia para las consultas sobres vistas simples y la segunda para las complejas.

2.2. Actualización de vistas

Las vistas siempre se pueden consultar, pero no siempre se pueden actualizar. Consideremos, por ejemplo, una tabla con los datos de los empleados y una vista con el nombre del departamento al que están asignados y el salario medio del departamento. ¿Qué significado tendría aumentar un 10% la media de los salarios de un departamento? Este aumento se puede conseguir de muchas formas, subiendo y bajando los salarios de los diferentes miembros del departamento. Por lo tanto, estamos ante una ambigüedad.

De una manera más formal, sea D una base de datos y V, una vista que se construye con la función $V = F(D)$. Si se hace una operación de actualización A sobre la vista $A(V) = A(F(D))$, habrá que encontrar una operación de actualización A' tal que $A(V) = F(A'(V))$. En muchos casos habrá muchas operaciones A' que cumplirán la igualdad anterior, aunque dejarán la base de datos en un estado diferente.

Reflexión

Cualquier operación de actualización sobre una vista deberá respetar las restricciones de integridad definidas en la tabla original.

Solo es posible asegurar que una vista será actualizable si está construida sobre una única tabla y sus atributos contienen una clave candidata de la tabla original. Las vistas definidas sobre más de una tabla acostumbran a no ser actualizables. Las vistas que incluyen cláusulas *GROUP BY* o funciones agregadas nunca son actualizables.

El concepto *actualizable* tiene una base semántica, no sintáctica; se pueden encontrar definiciones de vistas que *a priori* parecen no actualizables, pero que sí lo son.

Ved también

Podéis consultar las consideraciones referentes a las vistas actualizables en el SGBD Oracle 11g en el anexo.

Expresiones actualizables equivalentes

La vista que presentamos a continuación parece no actualizable:

```
CREATE VIEW View04 AS
   SELECT empId
   FROM Emp
   WHERE salary > 30000
      UNION
   SELECT empId
   FROM Emp
   WHERE deptId = 20
```

Pero sí que lo es, dado que es equivalente a esta otra:

```
SELECT empId
FROM Emp
WHERE salary > 30000 OR deptId = 20
```

Y esta expresión sí que es actualizable.

2.2.1. Actualización con disparadores de sustitución

Existen varios mecanismos para romper las posibles ambigüedades en las acciones de actualización sobre vistas, y se basan en el hecho de que se almacena el criterio de actualización adecuado junto con la definición de la vista. El estándar SQL:2008 incorpora el uso de los disparadores de sustitución[10].

(10)En inglés, *triggers instead of*.

Los **disparadores de sustitución** son disparadores[11] que no se ejecutan ni antes ni después, sino "en lugar de" la orden de modificación que se quiere realizar con la vista. Este tipo de disparadores permiten efectuar acciones de inserción, eliminación y actualización sobre vistas que no son actualizables. Solo se pueden definir sobre vistas.

(11)En inglés, *triggers*.

Ejemplo de actualización de vistas con disparadores de sustitución

Consideremos el ejemplo clásico de implementación de vistas. Supongamos que se define la vista siguiente:

```
CREATE VIEW DeptAvgSalaryView(department, salary) AS
   SELECT d.deptName, AVG(e.salary)
   FROM Emp e, Dept d
   WHERE e.deptId = d.deptId
   GROUP BY d.deptName
```

Como ya hemos planteado, esta vista es claramente no actualizable. Ahora bien, si consideramos que la política de la organización es la de realizar incrementos de sueldo proporcionales, un disparador que hiciera actualizable esta vista podría ser el siguiente (la sintaxis utilizada es Oracle 11g):

```
CREATE TRIGGER updatingDeptAvgSalary
INSTEAD OF UPDATE ON DeptAvgSalaryView
DECLARE
   v_increase NUMBER;
   v_old_deptId Dept.deptId%TYPE;
BEGIN
   IF :NEW.salary != 0 and :OLD.salary != 0 THEN
v_increase := :NEW.salary / :OLD.salary;
SELECT deptId INTO v_old_deptId
FROM Dept d
WHERE d.deptName = :OLD.department ;
UPDATE Emp SET salary = v_increase * salary
   WHERE deptId = v_old_deptId
   AND salary IS NOT NULL;
   END IF ;
   IF :NEW.department != :OLD.department THEN
UPDATE Dept SET deptName = :NEW.department
   WHERE Dept.deptName = :OLD.department;
   END IF;
END updatingDeptAvgSalary;
```

En este ejemplo se puede observar que, para actualizar el salario a un valor nuevo, el salario de cada empleado se incrementa de forma proporcional al incremento que tendría la media de salarios de todos los trabajadores. Si la política de actualización de salarios de nuestra organización es proporcional, la actualización funcionará. Del mismo modo, si se cambia el nombre de un departamento, se cambiará en la tabla correspondiente.

Si la política de la organización es que cuando desaparece un departamento todos sus empleados quedan a la espera de destino, dicho de otro modo, con el campo *departamento* en nulo, el correspondiente disparador sería el siguiente:

```
CREATE TRIGGER deletingDeptAvgSalary
INSTEAD OF DELETE ON DeptAvgSalaryView
DECLARE
   v_deptId Emp.deptId%TYPE;
BEGIN
```

```
    SELECT deptId INTO v_deptId
    FROM Dept d
    WHERE d.deptName = :OLD.department;

    UPDATE Emp SET deptId = NULL
    WHERE deptId = v_deptId;

    DELETE FROM Dept
    WHERE deptName = :OLD.department;

END deletingDeptAvgSalary;
```

El uso de este tipo de disparadores facilita el diseño interno y la encapsulación de la información.

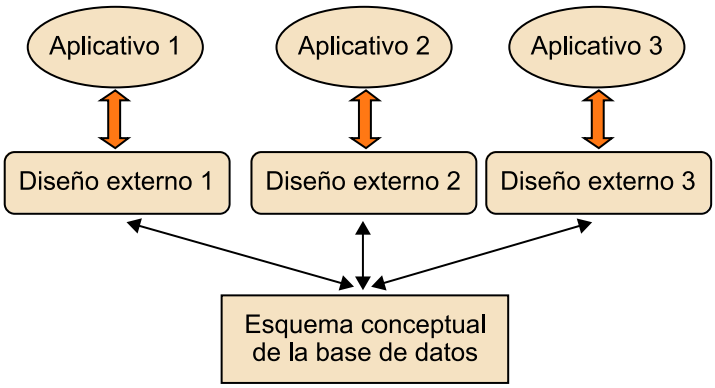
2.3. La vista como elemento de diseño externo

Siempre habría que diseñar las bases de datos teniendo en cuenta cómo son la organización o el sistema, nunca la utilización que se hará de los datos, a menos que sea necesario por razones de eficiencia. Una base de datos corporativa será utilizada por múltiples aplicativos, y a todos les gustaría que la base de datos fuera diseñada según sus necesidades (por lo menos, a sus diseñadores). La manera de conseguir este doble objetivo recibe el nombre de *diseño externo*.

El **diseño externo** consiste en crear una interfaz entre la base de datos y la aplicación.

En el siguiente esquema se puede ver una representación gráfica de esta situación:

Figura 2. Función del diseño externo en una base de datos corporativa



Una manera bastante sencilla y muy segura de implementar el diseño externo es usar una colección de vistas con los respectivos disparadores de sustitución para cada una de las aplicaciones cliente; de este modo, la complejidad del esquema conceptual de la base de datos es transparente a las diferentes aplicaciones y estas se "hacen la ilusión" de que tienen una base de datos diseñada a medida, en la cual toda la información se encuentra tal como les conviene.

Implementación del diseño externo

Para implementar el diseño externo, se podrán usar (según las posibilidades del SGBD) vistas, consultas, vistas objeto, procedimientos, funciones, disparadores de sustitución, métodos, etc.

Ejemplo de implementación de diseño externo con vistas

Podemos considerar una aplicación que podría tener asignada la vista *DeptSummaryView* en su esquema, vista que hemos tratado en ejemplos anteriores, sin ningún tipo de disparador de sustitución porque solo tendrá que hacer consultas.

También podemos tener otra aplicación que "verá" la base de datos como si solo hubiera la tabla *DeptAvgSalaryView*, sobre la cual puede borrar y actualizar gracias a los disparadores de sustitución implementados, pero no podrá realizar ninguna acción de inserción.

Así, se logra la **independencia lógica de los datos**, puesto que en caso de que el esquema conceptual sea modificado, la visión que tienen de él las diferentes aplicaciones no debe verse afectada.

En este apartado se utilizan la notación y las facilidades del SGBD comercial Oracle 11g.

2.4. Las tablas derivadas

La mayor parte de los SGBD incorporan unas estructuras muy parecidas a las vistas, las tablas derivadas, también conocidas como *instantáneas*[12], *agregados* o *vistas materializadas*.

(12)En inglés, *snapshots*.

Ejemplo de tabla derivada

En un supermercado, se podría tener una tabla de ventas, con la suma de las ventas de cada sección. No haría falta insertar ninguna fila, sino que la tabla se actualizaría automáticamente a medida que se fueran insertando filas.

Las vistas materializadas son tablas que, además de almacenar la definición de la vista, también almacenan los datos calculados (o derivados) de la ejecución de la sentencia *SELECT* definida en la vista. Esta tabla se puede definir con los mismos parámetros de almacenamiento que una tabla normal, como por ejemplo el espacio de tablas o la utilización de índices.

Comparación entre vistas y tablas derivadas

Las vistas son simples definiciones que se utilizan cuando hace falta (para materializar o reescribir una vista). En cambio, las tablas derivadas existen físicamente.

Cuando una sentencia SQL o PL/SQL accede a una vista materializada, el SGBD se dirige directamente a los datos de la vista que están almacenados en lugar de utilizar los datos de las diferentes tablas empleados en la definición de la vista.

Es conveniente utilizar vistas materializadas cuando la definición de la vista hace referencia a muchas tablas enlazadas de manera compleja, así como en las vistas que se utilizan con mucha frecuencia, puesto que permiten mejorar el rendimiento al ejecutarse la sentencia SQL solo una vez.

Por otro lado, hay que considerar si la vista materializada se reutilizará en el futuro. Esto implicará actualizar su contenido, puesto que probablemente el contenido de las tablas base será modificado.

A la hora de determinar si hay que definir una vista como materializada o no, es preciso valorar los costes de ejecutar la sentencia SQL que define la vista frente a los costes de almacenamiento y actualización de la vista materializada.

Para **definir una vista materializada**, el patrón de sentencia SQL es el siguiente:

```
CREATE MATERIALIZED VIEW MaterializedViewName
   [TABLESPACE tablespace_name]
   [PARALELL (DEGREE n)]
   [BUILD {IMMEDIATE|DEFERRED}]
   [REFRESH {FAST|COMPLETE|FORCE|NEVER|ON COMMIT|ON DEMAND}]
   [{ENABLE|DISABLE} QUERY REWRITE]
AS SELECT ... FROM ... WHERE ...
```

La cláusula *BUILD* permite elegir si en el momento de crear la vista, aparte de la creación de la tabla que almacenará los resultados, ésta se informa o no. En la primera opción, hay que usar *INMEDIATE*; en la segunda, *DEFERRED*, en cuyo caso los datos no informarán la tabla hasta que se realice la primera actualización de la vista materializada.

La cláusula *REFRESH* permite indicar el mecanismo que utilizará el SGBD para actualizar la vista materializada. El método de actualización que se escoja dependerá de la frecuencia de actualización de las tablas base. Este puede ser:

- *COMPLETE* ('completo'): se recalcula toda la tabla derivada según la consulta que la define.
- *FAST* ('rápido'): este método de actualización especifica un método de refresco incremental; los cambios se efectuarán agregando los nuevos datos que se han añadido a las tablas base.
- *FORCE* ('forzado'): aplica, si es posible, el refresco rápido, y si no es posible, el refresco completo.
- *NEVER* ('nunca'): indica que la vista materializada nunca será refrescada.

Reflexión

Hay que tener en cuenta que las vistas que usan funciones de agregación *SUM*, *AVG*, *MAX*, *MIN* o *COUNT* no admiten la actualización *FAST*.

La actualización se puede producir en dos momentos diferentes:

1) Actualización manual. Hay que definir la vista con la restricción *ON DEMAND*. El refresco se produce cuando el usuario ejecuta manualmente algún proceso de actualización sobre la vista materializada. Las actualizaciones manuales de las vistas materializadas se realizan utilizando el paquete PL/SQL estándar DBMS_MVIEW.

Reflexión

Todos estos procedimientos y funciones admiten parámetros adicionales que se pueden consultar en la documentación de Oracle 11g.

El paquete PL/SQL DBMS_MVIEW

Este paquete incluye un conjunto de funciones y procedimientos que permiten gestionar las vistas materializadas. Entre ellos podemos destacar:

- *DBMS_MVIEW.REFRESH*('MaterializedViewName'): actualiza una vista materializada a partir de su nombre.
- *DBMS_MVIEW.REFRESH_DEPENDENT*('Table1, Table2, ...'): actualiza todas las vistas materializadas que utilicen como tabla base alguna de las tablas o vistas indicadas en la lista.
- *DBMS_MVIEW.REFRESH_ALL_MVIEWS*(*n*): actualiza todas las vistas materializadas devolviendo un número (*n*) que indica la cantidad de registros que se han actualizado.

2) Actualización automática. Esta forma de actualización puede ser:

- *ON COMMIT*: el refresco se produce cuando la transacción que modifica alguna de las tablas base se confirma. Esto significa que la ejecución del *COMMIT* tendrá un mayor coste temporal, lo que puede afectar al rendimiento.
- Actualización programada: la actualización se programa para que suceda en un determinado instante.

Ejemplo de actualización programada

Una vista se podría programar para que se actualice todos los días a una hora determinada mediante el uso de las cláusulas *START WITH* y *NEXT*. La cláusula *START WITH* permite indicar el momento de la primera actualización automática y NEXT indica el intervalo entre actualizaciones automáticas.

```
CREATE MATERIALIZED VIEW NAME_VIEW
...
REFRESH START WITH ROUND(SYSDATE + 1) + 5/24
NEXT NEXT_DAY(TRUNC(SYSDATE), 'SUNDAY') + 15/24
AS SELECT ...;
```

Actualización con tablas derivadas

Consideremos el siguiente modelo relacional de la base de datos operativa de un supermercado:

ProductLine(*idLine*, *description*)

Products(*productCode*, *productLine*, *description*, *priceEach*),
{*productLine*} REFERENCES ProductLine(*idLine*)

Order(*orderNumber*, *orderDate*)

OrderDetail(*orderNumber*, *orderLineNumber*, *productCode*,
quantityOrdered), {*orderNumber*} REFERENCES Order(*orderNumber*)

La tabla derivada correspondiente a los ingresos realizados en los pedidos por línea de producto se podría crear con la sentencia siguiente:

```
CREATE MATERIALIZED VIEW ProductLineIncomeMView
BUILD IMMEDIATE
REFRESH FAST ON DEMAND
ENABLE QUERY REWRITE
AS SELECT pl.description as description,
   SUM(od.quantityOrdered * p.priceEach) as amount
   FROM ProductLine pl, Products p, Order o, OrderDetail od
   WHERE pl.idLine = p.productLine
      AND p.productCode = od.productCode
   GROUP BY pl.description;
```

La cláusula *ENABLE/DISABLE QUERY REWRITE* sirve para determinar si el optimizador de consultas puede reescribir o no las sentencias SQL, de forma que, a ser posible, se utilice la vista materializada en lugar de las tablas base empleando el mecanismo de reescritura. Esta opción sólo está disponible cuando se utiliza el optimizador basado en costes, puesto que el SGBD utiliza las estadísticas para determinar si la ejecución de la sentencia SQL o su reescritura utilizando la vista materializada es la que tiene menor coste.

Mecanismos de reescritura de consultas

Siguiendo con la base de datos del supermercado del ejemplo anterior, imaginemos la consulta siguiente:

```
SELECT pl.description, SUM(od.quantityOrdered * p.priceEach)
FROM ProductLine pl, Product sp, Order o, OrderDetail od
WHERE pl.idLine = p.productLine
AND p.productCode = od.productCode
AND pl.description IN ('vegetables', 'meat', 'drinks')
GROUP BY pl.description
HAVING SUM(od.quantityOrdered * p.priceEach) > 2500000.00;
```

El sistema podría aprovechar la vista materializada y reescribiría la consulta como:

```
SELECT description, amount
FROM ProductLineIncomeMView
WHERE description IN ('vegetables', 'meat', 'drinks')
AND amount > 2500000.00;
```

Las vistas materializadas siempre se calculan a partir de las tablas básicas y la única operación que se puede hacer con ellas es la consulta; no tiene sentido hablar de actualización, borrado o inserción. Dado que se acostumbran a usar para almacenar y precalcular datos agregados, a menudo se denominan *resúmenes*.

Tablas derivadas en entornos distribuidos

En entornos distribuidos, las tablas derivadas se pueden utilizar para replicar datos en las diferentes sedes, lo que permitiría sincronizar las actualizaciones sin crear ningún tipo de conflicto.

2.5. Tablas temporales

Otro tipo de tablas especiales son las tablas temporales, que deberían llamarse, de forma más exacta, *tablas de contenido temporal*.

En las tablas temporales, el contenido tiene vigencia en el transcurso de una sesión (o una transacción) y desaparece en el momento en que ésta finaliza.

La **sentencia SQL para la creación de una tabla temporal** es la siguiente:

```
CREATE {GLOBAL|LOCAL} TEMPORARY TABLE TableName
table_definition
ON COMMIT {PRESERVE ROWS|DELETE ROWS}
```

Estas tablas son muy útiles para almacenar resultados intermedios de una transacción.

Ejemplo de utilización de una tabla temporal

Consideremos la creación de la tabla siguiente, referente a las calificaciones obtenidas por los alumnos en una asignatura:

```
CREATE LOCAL TEMPORARY TABLE SubjectMarks (
   student VARCHAR2(50),
   mark NUMBER(3,1))
ON COMMIT DELETE ROWS;
```

Cuando se realice la primera inserción, la tabla se materializará y será posible trabajar con ella hasta el momento en que se ejecute la cláusula *COMMIT*; en ese momento la tabla se destruirá y solo permanecerá su definición en el diccionario de datos.

2.6. Ventajas e inconvenientes en la utilización de vistas

Como en cualquier decisión de diseño, la utilización de vistas presenta una serie de ventajas e inconvenientes. Las ventajas más importantes son las siguientes:

a) Independencia de los datos y las aplicaciones: Si se usan vistas como interfaz, se puede llegar a una independencia completa entre la estructura real de los datos y la que ven las aplicaciones. En cualquier momento se puede cambiar el nombre de una tabla, añadir nuevas columnas o variar completamente su estructura. Todos estos cambios resultan invisibles para las aplicaciones clientes. Solo habría que redefinir la interfaz, es decir, las vistas.

b) Simplificación del uso para el usuario: La utilización de las vistas permite tener almacenadas consultas bastante complejas con múltiples combinaciones de tablas, condiciones y funciones agregadas, y utilizarlas por medio de una consulta muy simple.

c) Mejora de la seguridad: El usuario no conoce las tablas y las columnas que forman realmente la base de datos, de modo que ni siquiera puede intentar acceder a ellas; sólo tiene noticia de aquellas información y estructura que se le dan a partir de las vistas.

d) Integridad de los datos: Con la cláusula *WITH CHECK OPTION* el usuario no puede llevar datos fuera de los límites que tiene marcados; todos los cambios que haga sobre la vista (si es actualizable) tendrán que dejar la fila consistente con las condiciones de la vista.

e) Rendimiento: Si es posible determinar el tipo de consultas que se llevarán a cabo a partir de la vista, también es posible determinar caminos de acceso que mejoren la eficiencia de la consulta.

Pero no todo son ventajas; los inconvenientes más importantes son los siguientes:

a) Restricciones de actualización: No todas las vistas son actualizables (en realidad, la mayoría no lo son); no es posible hacer un diseño externo sólo con vistas (salvo que se usen disparadores de sustitución).

b) Restricciones de estructura: Algunos SGBD, siguiendo una norma ISO, no permiten construir una vista a partir de cualquier consulta: una vista creada a partir de una sentencia *SELECT* no tendrá en cuenta las columnas nuevas que se hayan podido añadir a la tabla original.

3. La seguridad

En un sistema de información, generalmente, las diferentes aplicaciones y usuarios de la organización utilizan un único conjunto de datos (base de datos corporativa) con el SGBD. Por un lado, esto resuelve problemas de redundancia, inconsistencia e independencia entre los datos y los programas, y por otro, hace que la seguridad se convierta en uno de los problemas más importantes en estos entornos.

La palabra **seguridad** engloba diferentes conceptos. Los más importantes son:

a) Confidencialidad: hay que proteger el uso de la información por parte de personas no autorizadas. Esto implica que un usuario sólo tiene que poder leer la información para la cual tiene autorización y que no podrá inferir información secreta a partir de la información a la que tiene acceso.

b) Integridad: la información se tiene que proteger de modificaciones no autorizadas; esto incluye tanto insertar datos falsos como destruirlos.

c) Disponibilidad: la información debe estar disponible en el momento en que le haga falta al usuario.

Problemas de disponibilidad

Los problemas de disponibilidad incluyen la seguridad física (por problemas de hardware), la saturación del sistema, la denegación de servicio (DoS) por parte de la red, etc.

Las **violaciones de la base de datos** consisten en lecturas o modificaciones incorrectas de los datos. Por *modificación* entendemos las altas y las bajas de información y las modificaciones de información existente. Las consecuencias de estas violaciones se pueden agrupar en tres categorías:

a) Liberación incorrecta de la información. Está causada por la lectura de datos por parte de usuarios impropios mediante un acceso intencionado o accidental. En esta categoría se incluyen las violaciones del secreto derivadas de deducir información confidencial a partir de lecturas de información autorizada.

b) Modificación impropia de los datos. Corresponde a todas las violaciones de la integridad de los datos por tratamientos o modificaciones fraudulentas de éstos. Las modificaciones impropias no se deben necesariamente a lecturas no autorizadas, puesto que los datos pueden ser falsificados sin ser leídos.

c) Denegación de servicios. Corresponde a acciones que puedan impedir que los usuarios accedan a los datos o utilicen determinados recursos.

Las **amenazas a la seguridad** se clasifican según la forma en que pueden producirse:

1) Amenazas no fraudulentas. Son accidentes casuales, entre los cuales se pueden distinguir los siguientes:

a) Desastres accidentales o naturales: terremotos, inundaciones o incendios, que originan accidentes que dañan el hardware del sistema.

b) Errores en el hardware o en el software: pueden conducir a accesos no autorizados.

c) Errores humanos: causados de forma no intencionada al introducir datos o utilizar aplicaciones.

2) Violaciones intencionadas o violaciones fraudulentas. Son causadas por dos tipos de usuarios diferentes:

a) Usuarios autorizados que abusan de sus privilegios.

b) Agentes hostiles, usuarios impropios (internos o externos) que ejecutan acciones de vandalismo sobre el software y/o el hardware del sistema, o lecturas o escrituras de datos; en ambos casos, los usos legales de los datos y las aplicaciones pueden enmascarar el propósito fraudulento real.

A continuación se detallan los principales **mecanismos de seguridad**:

1) Identificación y autenticación. Se trata de mecanismos que identifican al usuario y se aseguran de que es quien dice ser.

2) Control de acceso. Son mecanismos que se cercioran de que los usuarios accedan sólo a los lugares a los que están autorizados para ejecutar las acciones para las cuales han sido autorizados.

Ved también

El control de acceso al SGBD Oracle 11g se puede consultar en el subapartado 4.3. de este módulo didáctico.

3) Integridad y consistencia. Son mecanismos para que la base de datos permanezca siempre en un estado que cumpla todas las reglas del negocio del modelo de datos, aunque se produzcan cambios.

4) Auditoría. Consiste en mecanismos para saber quién ha realizado qué, es decir, para llevar un registro de quién lleva a cabo todos los cambios y consultas en la base de datos. Más que un mecanismo para proporcionar seguridad, se trata de un mecanismo que permite monitorizar a los usuarios del sistema.

3.1. Identificación y autenticación

Los sistemas de información y los datos que almacenan y procesan son recursos muy valiosos que hay que proteger. Uno de los primeros pasos hacia la seguridad en un sistema de información es la capacidad de verificar la identidad del usuarios. Este proceso está formado por dos partes: identificación (ver quién es) y autenticación (comprobar que es quien dice ser).

La identificación implica la forma en que un usuario proporciona su identidad al sistema.

La identidad tiene que ser única para que el sistema pueda diferenciarla entre los distintos usuarios. Según los requisitos operacionales, una identidad puede describir a un individuo, a más de un individuo o a uno o más individuos sólo una parte del tiempo.

Ámbito de la identidad

Una diferencia importante entre la identificación y la autenticación es que las identidades son públicas, mientras que la información de autenticación se mantiene en secreto. Esto proporciona el recurso gracias al cual una persona prueba que es realmente quien dice ser.

La **autenticación** es el proceso de asociar a un individuo con su identidad única, es decir, es el proceso que verifica que un usuario es quien dice que es.

Hay tres **recursos de identificación básicos** para poder demostrar quién es realmente un individuo:

1) Algo que conoce una persona: una contraseña, un número de identificación personal, etc.

2) Algo que posee una persona: una tarjeta, una clave, etc.

3) Algo que caracteriza a una persona: la huella dactilar, la voz, etc.

Estos métodos básicos se pueden emplear individualmente o se pueden combinar para obtener un nivel de seguridad más alto.

Las **contraseñas** son el mecanismo clásico de autenticación. Las contraseñas son palabras (o mejor dicho, combinaciones de caracteres) que solo conoce un usuario. La seguridad de un esquema de contraseñas depende de la capacidad de mantenerlas en secreto. Una contraseña se tiene que elegir de forma que sea fácil de recordar y difícil de adivinar.

Criterios para elegir contraseñas

A continuación presentamos algunos criterios que conviene tener en cuenta a la hora de elegir una contraseña:

1) Seleccionar contraseñas largas: ocho caracteres es una medida adecuada.

2) Combinar diferentes tipos de caracteres: mayúsculas, minúsculas, números, espacios en blanco y signos de puntuación.

3) No usar palabras con significado.

4) Utilizar contraseñas diferentes para acceder a sistemas diferentes.

5) Cambiar la contraseña de manera periódica.

6) No escribir la contraseña en lugares a los que otra persona pueda acceder.

7) Que no sea la misma que el identificador de usuario.

8) Al cambiarla, que difiera de la anterior en al menos tres caracteres.

9) Cambiar la contraseña la primera vez que se inicie una sesión.

Las **tarjetas** dan mayor seguridad. Pueden ser simples trozos de plástico con una banda magnética o incluso pueden incorporar chips, como hacen las tarjetas inteligentes. En ambos casos, la contraseña personal tiene que coincidir con la que hay escrita en la tarjeta, o bien la contraseña y cierta información de la tarjeta deben coincidir con la que hay en el ordenador.

La tendencia actual es ir hacia **sistemas biométricos**, que son métodos automatizados de reconocimiento de una persona que se basan en una característica fisiológica o de comportamiento. Los sistemas biométricos se pueden utilizar como método de identificación, en el que se identifica a una persona dentro de una población buscando la coincidencia de una característica suya registrada en una base de datos. También se pueden utilizar en modo de verificación: el sistema autentica la identidad reclamada de una persona con su patrón previamente grabado.

Para la **validación de un usuario** de un sistema de gestión de bases de datos, se pueden usar cuatro métodos:

1) Autenticación por el mismo SGBD. Es la más utilizada, puesto que las cuentas son más fáciles de controlar y gestionar y el mismo SGBD tiene recursos que permiten la administración de pequeñas comunidades de usuarios.

2) Autenticación por el sistema operativo Es una forma de validación externa. Sólo es posible en aquellos sistemas que permitan la validación de usuarios (UNIX, Windows NT, etc.).

3) Autenticación por el servicio de red. Utiliza productos especializados de red, como Kerberos, CyberSafe, Identix, Radius u otros.

4) Autenticación por una capa intermedia. En un sistema cliente/servidor de tres niveles, la capa intermedia podría ser el software intermediario[13] o la aplicación misma.

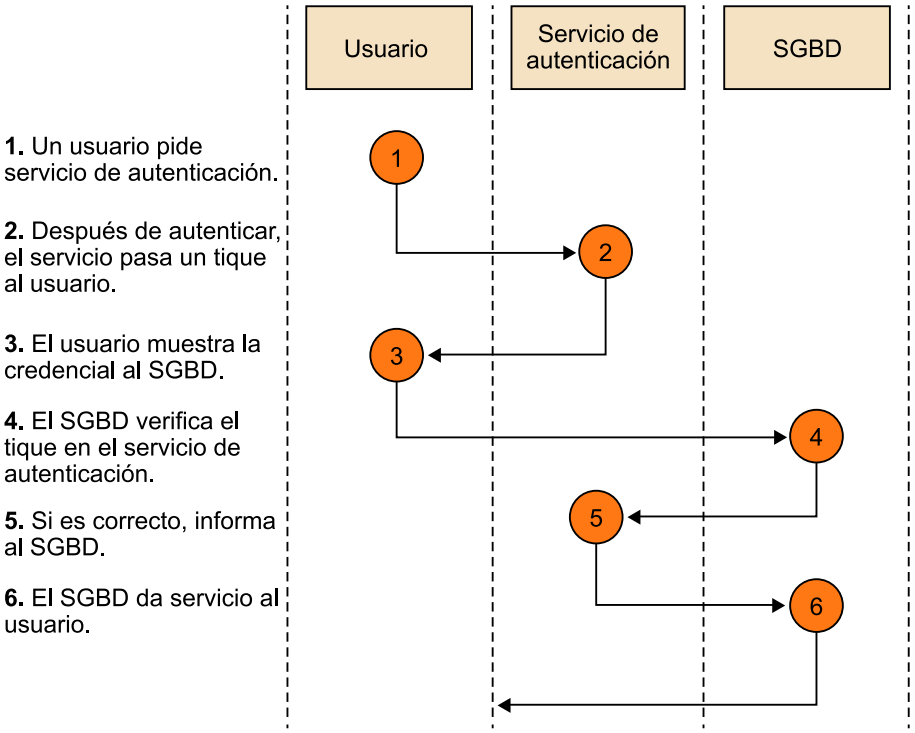
En todos los casos, la comunicación puede ser cifrada.

Sistema de autenticación de la red Kerberos

En sistemas en los que se requiere un cierto nivel de seguridad, es habitual la utilización de sistemas externos (máquinas diferentes de aquella a la que queremos conectarnos) como servicios de autenticación. El más conocido es Kerberos, el cual, utilizando diferentes técnicas de cifrado y emisión de tiques (*tickets*), proporciona un buen nivel de seguridad para la mayoría de los sistemas.

(13)En inglés, *middleware*.

Figura 3. El sistema de autenticación de la red Kerberos



La figura muestra el sistema de autenticación de la red Kerberos, que usa una máquina específica para llevar a cabo la autenticación y, de este modo, libera al SGBD de esta tarea.

3.2. Control de acceso

Se denomina **control de acceso** aquella parte del SGBD que tiene la función de asegurar que los accesos al sistema se llevan a cabo de acuerdo con los modelos y reglas fijadas por la política de protección.

El control de acceso controla la interacción (lectura, escritura...) entre los sujetos (usuarios, aplicativos, procesos...) y los objetos a los que acceden.

El control de acceso está formado por dos componentes:

1) **Políticas de acceso**: definen los principios según los cuales se autoriza a un usuario, se deniega o se filtra un acceso específico a un objeto.

Filtrado

El filtrado devuelve sólo una parte de la información requerida.

2) **Mecanismos de seguridad**: procedimientos que se aplican a las consultas de los usuarios para que cumplan las normas anteriores.

Las diferentes implementaciones de las políticas de acceso se pueden clasificar en control de acceso discrecional y control de acceso obligatorio.

3.2.1. Control de acceso discrecional

El control de acceso discrecional (DAC[14]) se basa en la identidad de los usuarios o grupos de usuarios para restringir el acceso a objetos. El control discrecional es el mecanismo de control más común que se implementa en los sistemas de información actuales.

Las **políticas discrecionales** se fundamentan en el conocimiento de los derechos que cada usuario tiene sobre cada objeto. Estas políticas podrán ser definidas por el administrador de la base de datos o por el propietario del objeto.

[14]DAC es la sigla de *discretionary access controls*.

Uso del DAC

El control de acceso discrecional (DAC) es el más utilizado en los SGBD comerciales.

Los derechos del usuario

Los derechos del usuario pueden incluir el de administrar la seguridad del objeto.

La mejor manera de representar la estructura del control de acceso discrecional es la matricial:

	Objeto 1	**Objeto 2**	**...**	**Objeto *n***
Usuario 1			...	
Usuario 2			...	Derechos del usuario 2 sobre el objeto *n*
...	...	...	...	
Usuario *n*			...	

La intersección entre filas y columnas indica los derechos de cada usuario sobre cada objeto.

Una ampliación habitual de la tabla anterior consiste en incorporar los grupos de usuarios como si se trataran de otro usuario y los privilegios sobre el sistema como si se trataran de un objeto. La descentralización de las autorizaciones provoca que surjan problemas de propagación tanto de autorizaciones como de revocaciones. Si un usuario *M* ha recibido una autorización de un usuario *N* que tenía el derecho de administrar la seguridad sobre un objeto *P*, ¿qué derechos tendrá el usuario *M* si a *N* se le revocan todos los permisos? El sistema deberá prever políticas de autorización y revocación en cadena.

3.2.2. Control de acceso obligatorio

El control de acceso obligatorio (MAC[15]) se suele usar en aquellas bases de datos en las que los datos tienen una estructura de clasificación muy rígida y estática, como por ejemplo las bases de datos militares y gubernamentales.

(15)MAC es la sigla de *mandatory access controls*.

Uso del MAC

El control de acceso obligatorio se suele utilizar en los SGBD que trabajan con información sensible (de alta seguridad).

Las **políticas de control de acceso obligatorio** se basan en la idea de que cada dato tiene un nivel de clasificación (por ejemplo, muy secreto, secreto, confidencial...) y cada usuario tiene un nivel de acreditación (con las mismas posibilidades que el nivel de clasificación). Los diferentes niveles están ordenados de forma estricta e incremental: muy secreto > secreto > confidencial...

Las normas de funcionamiento del control de acceso obligatorio son las siguientes:

- Un usuario puede ver un objeto sólo si su nivel de acreditación es mayor o igual que el nivel de clasificación del objeto.

- Un usuario puede modificar un objeto sólo si su nivel de acreditación es igual al nivel de clasificación del objeto.

Los usuarios deberán tener, primero, el acceso discrecional autorizado y los privilegios adecuados sobre el objeto de la base de datos antes de que se compruebe el sistema de acceso obligatorio. El sistema de seguridad MAC se basa en el concepto de *etiqueta*.

Una **etiqueta** indica el nivel del usuario (acreditación) o del objeto (clasificación). Los valores bajos denotan información no clasificada o menos sensible; los valores altos denotan información más restrictiva o accesible a menos usuarios. Las etiquetas se pueden refinar añadiendo otros subniveles.

Ejemplo de utilización de etiquetas

En el ejército se podría tener una clasificación como la que se presenta en la tabla siguiente, donde a cada etiqueta le corresponde un valor:

El sistema de gestión de bases de datos comprueba primero todas las autorizaciones de acceso discrecional; en caso de que el usuario cumpla todos los niveles de seguridad, el sistema de acceso obligatorio compara las etiquetas del usuario y del objeto para decidir quien tiene acceso.

Nombre	Valor	Descripción
General	80	Alto secreto, personal.
Oficial	60	Alta seguridad, no distribuir.
Suboficial	40	Moderadamente seguro.

Nombre	Valor	Descripción
Militar	20	Nivel básico, no demasiado sensible.
Civil	1	Público conocimiento, libre distribución.

3.2.3. Clasificación de los sistemas de seguridad

Los sistemas de seguridad se clasifican en cuatro niveles de seguridad, que son, de menos a más protección:

Los sistemas de seguridad habituales

Aunque algunos productos comerciales proporcionan un nivel de seguridad B1, lo más normal es que solo lleguen al nivel C2.

1) Protección mínima: la clase D no incorpora ningún mecanismo de seguridad.

2) Protección discrecional: la clase C soporta el control de acceso discrecional y tiene las subclases C1 y C2. La clase C1 requiere separación entre los datos y los usuarios. La clase C2 requiere además procesos de registro, auditoría y aislamiento de recursos.

Ejemplo de cobertura

Un ejemplo de cobertura podría ser inferir la respuesta de una consulta ilegal a partir de una consulta legal.

3) Protección estructurada: la clase B soporta el control de acceso obligatorio y tiene las subclases B1, B2 y B3. La clase B1 requiere protección de seguridad etiquetada: todos los objetos están etiquetados con un nivel de seguridad. La clase B2 requiere además una sentencia formal para cada cosa, así como que los canales de cobertura sean identificados y eliminados. La clase B3 requiere también apoyos de recuperación y auditoría.

4) Protección verificada: la clase A, la más segura, requiere una demostración matemática de que los mecanismos de seguridad son consistentes y adecuados para soportar la política de seguridad especificada.

La **clasificación de los sistemas de seguridad** que se ha presentado aquí fue creada por el Pentágono como sistema de clasificación estándar, y se encuentra definida en el Orange Book, en el que se definen los requisitos de seguridad para cualquier TCB[16], y en el Lavender Book, en el que se define la interpretación de los requisitos para un sistema gestor de bases de datos específico.

[16]TCB es la sigla de la expresión inglesa *trusted computing base*.

3.3. Implementación del control de acceso discrecional a SQL:2011

El estándar actual define la siguiente sintaxis de autorizaciones:

```
<sentencia de autorización> ::=
   <sentencia autorización privilegios>
 | <sentencia autorización roles>
<sentencia autorización privilegios> ::=

   GRANT <privilegios> TO <autorizado> [{ , <autorizado>}]
```

```
    [WITH HIERARCHY OPTION]
    [WITH GRANT OPTION]
    [GRANTED BY <autorizador>]
<privilegios> ::=
    <privilegios de objeto> ON <nombre objeto>

<privilegios de objeto> ::=
    ALL PRIVILEGES|<acción>[{ , <acción>}]
<acción> ::=
    DELETE |
    SELECT [(<nombre columna> [ , <nombre columna>] ... ])]|
    SELECT [(<nombre rutina> [ , <nombre rutina>] ... ])]|
    INSERT [(<nombre columna> [ , <nombre columna>] ... ])]|
    UPDATE [(<nombre columna> [ , <nombre columna>] ... ])]|
    REFERENCES [(<nombre columna> [ , <nombre columna> ... ])]|
    USAGE |
    UNDER |
    TRIGGER |
    EXECUTE

<nombre objeto> ::=
    [ TABLE ] <nombre tabla>
    |DOMAIN <nombre dominio>
    | COLLATION <nombre compaginador de caracteres>
    | CHARACTER SET <nombre juego de caracteres>
    | TRANSLATION <nombre transcripción>
    | TYPE <nombre tipo>
    | SEQUENCE <nombre generador de secuencias>
    | <designador específico de rutina>
    | MODULE <nombre módulo>

<autorizado> ::=
    PUBLIC | <nombre autorizado>

<autorizador> ::=
    CURRENT_USER | CURRENT_ROLE
```

El significado de los privilegios que se pueden dar a un usuario sobre un objeto son los siguientes:

Cláusulas de autorización de privilegios

La opción *WITH GRANT OPTION* permite transmitir los privilegios que se tienen a otro. La cláusula *WITH HIERARCHY OPTION* permite trasladar los permisos a todas las subtablas.

Ejemplos de autorizaciones

A continuación presentamos un ejemplo de cómo se implementan las autorizaciones tratadas en este subapartado:

```
GRANT SELECT ON TABLE Empleado TO Auditor2;
GRANT USAGE ON DOMAIN d_Telefono TO PUBLIC;
```

Privilegios	Definición
INSERT [(lista-nombres-columnas)]	Insertar valores en las columnas relacionadas de una fila.
UPDATE [(lista-nombres-columnas)]	Actualizar valores en las columnas relacionadas de una fila.
DELETE	Borrar filas.
SELECT [(lista-nombres-columnas)]	Consultar valores en las columnas relacionadas.
REFERENCES [(lista-nombres-columnas)]	Referenciar a valores de las columnas relacionadas.
TRIGGER	Crear un disparador sobre una tabla.
EXECUTE	Ejecutar un procedimiento incorporado.

```
GRANT ALL PRIVILEGES ON VIEW Vista1 TO Joan, Pere, Maria
```

Es relativamente frecuente asociar los mismos privilegios a diferentes usuarios, por ejemplo a todo el departamento de contabilidad. Para facilitar este tipo de autorización, se usa el rol.

Un **rol** se puede definir como un conjunto de uno o más privilegios. Si se autoriza un rol a un usuario, automáticamente se le autorizan todos los privilegios incorporados en el rol.

La **sintaxis del rol** es la siguiente:

```
<definición rol> ::=
   CREATE ROLE <nombre rol>
   [WITH ADMIN <autorizador>]

<sentencia autorización rol> ::=
   GRANT <nombre rol> [ { , <nombre rol>}]
   TO <autorizado> [ { , <autorizado>}]
   [WITH GRANT OPTION]
   [GRANTED BY <autorizador>]
```

La cláusula *WITH ADMIN* permite indicar quién administrará el rol.

Ejemplos de creación y utilización de roles

Considerad los ejemplos siguientes de creación y utilización de roles:

```
CREATE ROLE Auditores WITH ADMIN CURRENT_USER;
GRANT SELECT, UPDATE (Salario, Comision) TO Auditores;
GRANT Auditores TO Pere, Joan, Maria;
```

Del mismo modo que se dan autorizaciones, se pueden revocar.

La **sintaxis para revocar roles** es la siguiente:

```
<sentencia revocar privilegios> ::=
REVOKE [{GRANT OPTION FOR|HIERARCHY OPTION FOR}]
   <privilegios> FROM <autorizado> [{ , <autorizado>} ... ]
[ FROM {CURRENT_USER|CURRENT_ROLE}]
[ GRANTED BY <autorizador>]
{ RESTRICT|CASCADE}

<sentencia revocar rol> ::=
REVOKE [ADMIN OPTION FOR] <nombre rol> [{ , <nombre rol> } ... ]
FROM <autorizado> [ { , <autorizado >} ... ]
[FROM {CURRENT_USER|CURRENT_ROLE}]
[GRANTED BY <autorizador>]
{RESTRICT|CASCADE}
```

Problemas de propagación de privilegios

Las opciones *RESTRICT* y *CASCADE* tienen una importancia especial para resolver los problemas de propagación. Cuando A autoriza un privilegio a B (con la opción de administrarlo) y B lo autoriza a C, si a A se le quitan los privilegios que tenía, B y C se quedarán con los privilegios "abandonados". Si la opción elegida cuando se revocan los privilegios a A es *CASCADE*, los permisos de B y C también serán revocados. Si la opción es *RESTRICT*, sólo se podrá revocar los permisos de A si previamente se han eliminado los permisos candidatos a ser abandonados. Si no se utiliza ninguna de las opciones, B y C se quedarán con los permisos, y se correrá el riesgo de que B vuelva a dar privilegios a A.

3.4. Auditoría

La auditoría es el registro y monitorización de algunas acciones específicas de usuarios sobre la base de datos.

La carga del trabajo de auditoría

El trabajo de auditoría puede representar una sobrecarga superior al nivel del trabajo normal.

La auditoría se utiliza normalmente para los casos siguientes:

1) La investigación de una actividad sospechosa.

Ejemplo de investigación de una actividad sospechosa

Si un usuario no autorizado intenta borrar datos de las tablas, el administrador de seguridad podrá decidir si audita todas las conexiones de la base de datos y todos los intentos (con éxito o no) de borrar datos.

2) La monitorización y la recogida de actividades específicas de la base de datos.

Ejemplo de monitorización de la base de datos

El administrador de la base de datos puede recoger datos sobre qué tablas se actualizan o cuántos usuarios están conectados en momentos punta.

El sistema de auditoría debe permitir diferentes formas de utilización:

- Auditar sentencias. La auditoría[17] indicará cuándo y quién ha utilizado un tipo de sentencia concreta; por ejemplo, auditar todas las inserciones o los borrados.

(17) *Auditar* significa averiguar quién es el autor de cualquier cambio en la base de datos.

- Auditar objetos. El sistema registrará cada vez que se realice alguna operación sobre un objeto determinado.

- Auditar sentencias sobre objetos, una versión combinada de las dos anteriores.

- Auditar a usuarios o grupos.

También se puede decidir si se desea un único registro por sesión, por sentencia o por acceso; y si sólo se quiere registrar cuándo tiene éxito, cuándo fracasa o en ambos casos. La información que suele registrar la auditoría es el nombre del usuario, el identificador de sesión, el identificador de terminal, el nombre del objeto al que se ha accedido, la operación ejecutada o que se ha intentado, el código completo de la operación, la fecha y la hora.

La creación de auditorías

Algunos SGBD incorporan sentencias SQL que permiten generar una auditoría de manera declarativa; en otros casos habrá que crear disparadores que vayan rellenando las tablas de auditoría conforme se producen acontecimientos.

3.5. El Reglamento General de Protección de Datos Personales (RGPD)

Las siglas RGPD corresponden al reglamento europeo relativo a la protección de datos de las personas físicas, que regula tanto el tratamiento de datos personales como su libre circulación. El RGPD establece obligaciones que deben ser analizadas y aplicadas por cada organización teniendo en cuenta sus propias circunstancias.

El RGPD se aplica a organizaciones establecidas en la Unión Europea (UE) que traten datos personales o cuyo tratamiento de datos estén destinados a ciudadanos de la UE.

Su entrada en vigor el 25 de mayo de 2018 ha dejado obsoleta la Ley Orgánica de Protección de Datos de Carácter Personal (LOPD) de 1999, siendo sustituida el 6 de diciembre de 2018 por la Ley Orgánica de Protección de Datos Personales y Garantía de los Derechos Digitales (LOPDGDD) que incluye algunas precisiones en materias en las que el RGPD lo permite.

3.5.1. Principios generales de protección de datos

1) Definición de dato personal: es un dato personal toda información sobre una persona física identificada o identificable («el interesado/a»); se considerará persona física identificable toda persona cuya identidad pueda ser determinada directamente o indirectamente, en particular mediante un identificador, como un nombre, un número de identificación, datos de localización, un identificador en línea, o uno o varios elementos propios de la identidad física, fisiológica, genética, psíquica, económica, cultural o social de esta persona.

Finalidad del tratamiento de los datos

Los datos de carácter personal objeto del tratamiento no se podrán utilizar para finalidades incompatibles con aquellas para las que se han recogido.

2) Tipos de datos: los datos personales se agrupan en dos categorías: no sensibles y especialmente protegidos. Los datos especialmente protegidos son los datos de salud, étnicos, religiosos o filosóficos, los relacionados con la vida u orientación sexual, filiación sindical o ideología política, los datos genéticos y los biométricos (aquellos obtenidos a partir del tratamiento con medios técnicos de características físicas o conductuales que permitan la identificación de la persona, como la imagen facial o la huella dactilar).

3) Un continente, una norma: se establece una legislación unitaria para todos los países de la Unión Europea.

4) Evaluación específica: no existen medidas generales de seguridad, estas deben establecerse de modo individualizado a partir del estudio de cada caso.

5) Privacidad por defecto: la privacidad de la persona siempre debe prevalecer (*privacy by default*), solo deben tratarse los datos personales necesarios para cada uno de los fines específicos del tratamiento. Por ejemplo, en las redes sociales, las medidas de privacidad se cumplirían si se aplica al usuario una configuración de seguridad restrictiva al registrarse.

6) Privacidad desde el diseño y responsabilidad activa: la protección de datos debe tenerse en cuenta desde las primeras etapas del diseño (*data protection by design*). El/la responsable o encargado/a del tratamiento tiene la responsabilidad (*accountability*) de aplicar las medidas técnicas y organizativas necesarias para garantizar el cumplimiento del RGPD.

Comunicación de los datos personales

Los datos personales solo podrán ser comunicados a terceros cuando exista una legitimación que lo permita, como el consentimiento del/de la interesado/a, obligación legal o que sea necesaria para ejecutar un contrato.

La privacidad comprende todo el ciclo de vida de los datos.

7) Licitud del tratamiento: Los datos serán tratados de forma lícita, leal y transparente en relación con el/la interesado/a y serán recogidos con fines determinados, explícitos y legítimos, y no se tratarán posteriormente de modo incompatible con dichos fines.

8) Si el tratamiento está legitimado en el **consentimiento**, debe ser «libre, específico, informado e inequívoco», no se acepta el consentimiento tácito. Un consentimiento no se puede utilizar para más de un tratamiento. Cada tratamiento debe tener su base legal, ya sea el consentimiento del interesado/a, el interés legítimo del responsable del tratamiento o cualquier otro de los previstos en el RGPD.

Consentimiento

No están permitidas las casillas premarcadas ni las peticiones en negativo.

Se considera que un consentimiento no es libre cuando no permite autorizar por separado las diferentes operaciones o cuando se obliga a aceptar la prestación de un servicio pese a que este no sea necesario para realizar el tratamiento inicial.

9) Responsable de tratamiento: la persona física o jurídica que determina la finalidad y los medios relacionados con el tratamiento de los datos personales.

10) Encargado/a de tratamiento: la persona física o jurídica que lleva a cabo el tratamiento de datos por encargo del/de la responsable del tratamiento, siguiendo sus indicaciones (p.e. una gestoría que prepara las nóminas para una empresa).

11) Corresponsables del tratamiento: cuando dos o más responsables determinen conjuntamente los objetivos y los medios del tratamiento.

12) Tratamiento de datos de menores: un/a menor puede dar su consentimiento a partir de los 16 años, pero en España este límite es de 14 años. Por debajo de esta edad, se exige el consentimiento de padres o tutores.

Las personas tienen los derechos siguientes:

1) Derecho de acceso: el/la interesado/a tiene derecho a recibir confirmación si se están tratando sus datos y, si fuera el caso, a conocer los fines del tratamiento, qué datos se tratan, los destinatarios, el plazo de conservación y si se aplican decisiones automatizadas (como la elaboración de perfiles).

2) Derecho de rectificación: el/la interesado/a deberá obtener, sin dilación indebida, la rectificación de sus datos personales que sean inexactos.

3) Derecho de supresión: el/la interesado/a tiene derecho a solicitar la supresión de sus datos (derecho al olvido) si revoca el consentimiento, se opone al tratamiento, los datos se han tratado ilícitamente o en aplicación de una obligación legal.

4) Derecho a la limitación del tratamiento: a petición del/la interesado/a, sus datos personales no seran tratados mientras tenga pendiente alguna reclamación sobre el tratamiento de dichos datos, incluída la petición de borrado.

Tutela de los derechos

Las actuaciones contrarias al RGPD pueden ser objeto de reclamación por pare de los/las interesados/as ante la Agencia Española de Protección de Datos (AEPD).

5) Derecho de portabilidad: el/la interesado/a tiene derecho a recibir los datos personales que haya facilitado en un formato de uso común y de lectura mecánica, y que se pueda transmitir a otro responsable.

6) Derecho de oposición: aunque un tratamiento se haya iniciado de modo lícito, el/la interesado/a puede oponerse. Por ejemplo, el màrqueting directo, para el cual no es necesario el consentimiento previo del/de la interesado/a.

7) Derecho a no ser sujeto de decisiones automatizadas: todo/a interesado/a tiene derecho a no ser objeto de decisiones automatizadas, como la elaboración de perfiles, que puedan producir efectos jurídicos en él o le afecten a su persona.

3.5.2. Obligaciones de las organizaciones

Uno de los aspectos esenciales es el que se conoce como **responsabilidad activa** (*accountability*). Las organizaciones deben adoptar medidas que aseguren razonablemente que están en condiciones de cumplir con los principios, los derechos y las garantías que el reglamento establece.

Actuar solo cuando ya se ha producido una incidencia es insuficiente como estrategia, dado que se pueden haber causado daños a los/las interesados/as que pueden ser muy difíciles de compensar o reparar.

En este sentido, el RGPD prevé una serie de actuaciones que se deben acreditar:

1) Asegurarse de que el tratamiento sea **lícito** y que los datos se hayan obtenido de manera legítima, determinada, explícita y que no serán tratados posteriormente de modo incompatible con la finalidad para la que fueron facilitados.

2) Supervisar los avisos de privacidad: hay que identificar al/a la responsable, informar sobre la base legal del tratamiento, su finalidad, los plazos de retención de los datos, si serán cedidos o no a terceros, si se usarán para elaborar perfiles, si habrá transferencias internacionales, dónde puede el/la interesado/a ejercer los derechos y notificarle que puede presentar sus reclamaciones ante la autoridad de control. Todo en un lenguaje claro, sencillo y conciso.

Relación responsable-encargado

Las relaciones entre el/la responsable y el/la encargado/da deben formalizarse en un contrato o acto jurídico que los vincule.

3) El responsable deberá verificar que los/las encargados/das del tratamiento ofrecen garantías técnicas y organizativas suficientes para cumplir con el RGPD.

4) Realizar un **análisis de riesgos** en materia de privacidad con carácter previo a la introducción de un producto o servicio en el mercado. El tipo de análisis variará en función de:

Análisis de riesgos

Todos los/las responsables deberán hacer una valoración de los riesgos de los tratamientos que llevan a cabo para poder determinar qué medidas aplicar.

- Los tipos de tratamiento.
- La naturaleza de los datos.
- El número de interesados/as afectados/as.
- La cantidad y variedad de tratamientos que una misma organización lleve a cabo.

El análisis de riesgos deberá realizarse mediante alguna de las metodologías existentes. Pero en organizaciones pequeñas y con tratamientos de poca complejidad, el análisis será el resultado de una reflexión, mínimamente documentada, sobre cómo pueden afectar los tratamientos a los derechos y las libertades de los/las interesados/as.

5) Donde sea probable que haya un riesgo para los derechos y las libertades de los interesados/as, hacer una **evaluación de impacto sobre la protección de datos (EIPD)** con carácter previo a la puesta en marcha del tratamiento. Se considera que un tratamiento comporta un alto riesgo cuando hay:

Evaluación de impacto (EIPD)

Adicionalmente a lo que dice el RGPD, la AEPD ha elaborado una lista de tratamientos en los que se debe realizar una EIPD.

- Evaluación sistemática y exhaustiva de aspectos personales que se basen en un tratamiento automatizado, como la elaboración de perfiles con los que se tomen decisiones que tengan efectos jurídicos o le afecten a nivel personal.
- Tratamiento a gran escala de datos de categoría especial.
- Observación sistemática y a gran escala de una zona de acceso público.

De modo complementario a los criterios anteriores, la Agencia Española de Protección de Datos (AEPD) ha preparado también una lista orientativa de tipos de tratamientos que también requieren una EIPD.

La EIPD facilitará una descripción sistemática de las operaciones de tratamiento, de sus riesgos y de las medidas previstas para afrontar estos riesgos. En los casos en los que se haya identificado un alto riego que, a juicio del/de la responsable, no se pueda mitigar por medios razonables en términos de tecnología disponible y costes de aplicación, se deberá consultar a la autoridad de control, la cual puede emitir recomendaciones o ejercer cualquier otro de los poderes que el RGPD le confiere, como prohibir el tratamiento.

6) Mantenimiento de un **registro de actividades** donde se describa el tratamiento de datos y las medidas técnicas y organizativas aplicadas para garantizar el cumplimiento legal. Deberá detallar:

- Nombre y datos de contacto del/de la responsable y del/de la delegado/a de protección de datos (si lo hay).
- Finalidades del tratamiento.

- Descripción de categorías de interesados/as y categorías de datos personales tratados.
- Cesiones y transferencias de datos.

7) Notificación de violaciones de la seguridad de los datos: excepto en los casos en los que sea improbable que la violación suponga un riesgo para los derechos y las libertades de los/las afectados/as, se deberá informar antes de 72 horas a la autoridad de control, y a los/las mismos/as afectados/as en los casos en los que sea probable que la violación comporte un alto riesgo para sus derechos o sus libertades. La notificación debe incluir, como mínimo:

- La naturaleza de la violación.
- Categorías de datos y de interesados/as afectados/das.
- Medidas adoptadas por el/la responsable para solucionar la incidencia.
- Si fuera el caso, las medidas aplicadas para paliar los posibles efectos negativos sobre los/las interesados/as.

8) Promoción de **códigos de conducta y esquemas de certificación**: la adhesión de un/a encargado/a de tratamiento a un código de conducta o certificarse es un modo de demostrar que se ofrecen las garantías exigidas por el RGPD.

3.5.3. Consentimiento, recogida y tratamiento de datos

En la recogida y el tratamiento de datos es necesario garantizar una serie de puntos que condicionarán significativamente el diseño de las bases de datos, y siempre teniendo en cuenta los criterios de privacidad desde el diseño y la responsabilidad activa:

1) Se tratarán los datos de modo lícito, leal y transparente. Se considerará un tratamiento lícito cuando se cumpla uno, o mas, de los siguientes puntos:

- El interesado/a ha dado su consentimiento.
- Sea necesario para la ejecución de un contrato, cumplimiento de una obligación legal o por interés público.
- Sea necesario para proteger intereses vitales del/de la interesado/a.
- Exista un interés legítimo por parte del/de la responsable que no comprometa los derechos y las libertades del interesado/a.

Consentimiento tácito

Con la entrada en vigor del RGPD, el consentimiento tácito ya no se considera válido. En los casos en los que se aplicaba se debe encontrar otra base de legitimación o volver a pedirlo.

2) El consentimiento será legítimo cuando sea libre, específico, informado e inequívoco. También se contemplan situaciones en las que, además, debe ser explícito:

- Tratamiento de datos sensibles.
- Adopción de decisiones automatizadas.
- Transferencias internacionales.

El consentimiento puede concederse implícitamente cuando se deduzca de una acción del/de la interesado/a (por ejemplo, cuando se sigue navegando por un sitio web aceptando el uso de cookies para monitorizar la navegación).

3) Se recopilarán para fines determinados, explícitos y legítimos y no pueden ser tratados posteriormente de modo incompatible con dichos fines.

4) Serán adecuados, pertinentes y limitados a la finalidad que debe llevarse a cabo.

5) Serán exactos y, si fuese necesario, actualizados. Se deberán adoptar medidas razonables para garantizar su exactitud.

6) Serán mantenidos el tiempo necesario para llevar a cabo su finalidad y sus obligaciones legales que se deriven, pasado este período se tendrán que bloquear.

El **bloqueo de los datos** consiste en la identificación y reserva de los mismos, adoptando medidas para impedir su tratamiento, excepto para ponerlos a disposición de la Administración de Justicia, las administraciones públicas y las autoridades de protección de datos mientras existan responsabilidades derivadas del tratamiento y hasta el plazo de prescripción de los mismos. Transcurrido este plazo se deben destruir.

Se podrán conservar los datos personales durante plazos más largos con fines de archivo, interés público o investigación científica o histórica, siempre que se establezcan medidas técnicas y organizativas apropiadas.

7) Serán tratados de modo que se garantice un nivel adecuado de seguridad, incluyendo la protección contra el tratamiento no autorizado o ilícito y contra su pérdida o destrucción.

3.5.4. Delegado/a de protección de datos

El RGPD define el/la **delegado/a de protección de datos (DPD)** como el/la encargado/a de supervisar internamente el cumplimiento legislativo. Puede ser un/a empleado/a de la organización o externo, pero debe tener garantizada la independencia en el ejercicio de sus funciones, que son:

- Informar y asesorar al/la responsable o al/la encargado/a del tratamiento de las obligaciones que se deben cumplir.
- Supervisar el cumplimiento de lo dispuesto en el RGPD.
- Ofrecer el asesoramiento que se le solicite sobre las EIPD.
- Cooperar con la autoridad de control.
- Actuar como punto de contacto de la autoridad de control en cuestiones relativas al tratamiento.

Habrá que nombrar a un/a DPD cuando nos encontremos con:

- Organizaciones e instituciones públicas.
- Responsables o encargados/as que tengan entre sus actividades principales las operaciones de tratamiento que requieran una observación habitual y sistemática de interesados/as a gran escala.
- Responsables o encargados/as que tengan entre sus actividades principales el tratamiento a gran escala de datos sensibles.

También habrá que nombrar a un/a DPD en los casos que se detallan en la LOPDGDD. En el resto de organizaciones, disponer de un/a DPD es opcional. El/la DPD debe ser designado/a en función de sus cualificaciones profesionales, y de su conocimiento de la legislación y la práctica de la protección de datos. Existe la posibilidad de certificarse como DPD, aunque no es un requisito para serlo.

DPD

Los datos de contacto del/de la DPD deben hacerse públicos y comunicarlos a la autoridad de control.

La posición del/de la DPD en las organizaciones debe cumplir unos requisitos, básicamente:

- Debe de disponer de total autonomía en el ejercicio de sus funciones.
- Debe de relacionarse con el nivel superior de la dirección.
- Debe de disponer de los recursos necesarios para desarrollar su actividad.

3.5.5. Transferencias internacionales de datos

Los datos se podrán comunicar fuera del Espacio Económico Europeo en determinados casos:

- A países, territorios o sectores específicos (se incluyen también organizaciones internacionales) sobre los que se haya adoptado una decisión reconociendo que ofrecen un nivel de protección adecuado.
- Cuando se hayan ofrecido garantías adecuadas sobre el nivel de protección que los datos recibirán en su destino.
- Cuando se aplique alguna de las excepciones que permiten transferir los datos sin garantías de protección adecuada por razones de necesidad vinculadas al propio interés del titular de los datos o a intereses generales.

3.5.6. La Agencia Española de Protección de Datos

La Agencia es un ente de derecho público, con personalidad jurídica propia y plena capacidad pública y privada. Actúa con total independencia de las administraciones públicas en el ejercicio de sus funciones. Su finalidad principal consiste en velar por el cumplimiento de la legislación sobre protección de datos personales y controlar su aplicación, asumiendo entonces el papel de **Autoridad de Control** que el RGPD establece.

La Agencia de Protección de Datos en Internet

Podéis encontrar las últimas actualizaciones de todas las normativas, resoluciones, informes jurídicos, guías y herramientas en su web: www.aepd.es

Esta web permite también realizar consultas, presentar reclamaciones y hacer comunicaciones (ya sea de fugas de información o de nombramiento de DPD).

4. Anexos

4.1. Reglas de equivalencia de operaciones de álgebra relacional

Las reglas de equivalencia se utilizan para reestructurar el árbol de operaciones de álgebra relacional que se ha obtenido en el proceso de descomposición de la consulta.

A continuación, vamos a definir algebraicamente cada una de ellas y a mostrar algunos ejemplos aclaratorios:

1) Las operaciones conjuntivas de selección pueden transformarse en una cascada de operaciones individuales de selección.

$$\sigma_{(p)\wedge(q)\wedge(r)}(R) = \sigma_p(\sigma_q(\sigma_r(R))), \text{ donde } p \in \{A_1, \ldots, A_n\}$$

Ejemplo

$$\sigma_{(salary>2.500)\wedge(hireDate>'01\text{-}SEP\text{-}97')}(Emp) = \sigma_{(salary>2.500)}(\sigma_{(hireDate>'01\text{-}SEP\text{-}97')}(Emp))$$

2) Las operaciones de selección son conmutativas.

$$\sigma_q(\sigma_p(R)) = \sigma_p(\sigma_q(R)), \text{ donde } p, q \in \{A_1, \ldots, A_n\}$$

Ejemplo

$$\sigma_{(salary>2.500)}(\sigma_{(hireDate>'01\text{-}SEP\text{-}97')}(Emp)) = \sigma_{(hireDate>'01\text{-}SEP\text{-}97')}(\sigma_{(salary>2.500)}(Emp))$$

3) En una secuencia de operaciones de proyección, sólo es necesaria la última proyección de la secuencia.

$$\Pi_L\Pi_M\Pi_N(R) = \Pi_L(R)$$

Ejemplo

$$\Pi_{empId}\Pi_{empId;firstName;lastName}(Emp) = \Pi_{empId}(Emp)$$

4) Conmutatividad de la selección y la proyección.

$$\Pi_L(\sigma_p(R)) = \sigma_p(\Pi_L(R)), \text{ donde } p \in \{A_1, \ldots, A_n\}$$

Ejemplo

$$\Pi_{empId;firstName;lastName}(\sigma_{(salary>2.500)}(Emp)) = \\ = \sigma_{(salary>2.500)}(\Pi_{empId;firstName;lastName}(Emp))$$

5) Conmutatividad de la combinación Theta (y del producto cartesiano).

Combinaciones Theta

Combinaciones que utilizan los operadores de comparación como condición de combinación.

$$R \bowtie_p S = S \bowtie_p R$$
$$R \times S = S \times R$$

Ejemplo

$$Emp \bowtie_{(Emp:deptId=Dept:deptId)} Dept = Dept \bowtie_{(Dept:deptId=Emp:deptId)} Emp$$

6) Conmutatividad de la selección y de la combinación Tetha (y del producto cartesiano).

$$\sigma_p(R \bowtie_t S) = (\sigma_p(R)) \bowtie_t S, \text{ donde } p \in \{A_1, \ldots, A_n\}$$
$$\sigma_p(R \times S) = \sigma_p(R) \times S$$

En caso de que el predicado sea de la forma $(p \wedge q)$, donde p corresponde a los atributos de R y q a los de S, entonces las operaciones son conmutativas de la forma siguiente:

$$\sigma_{(p \wedge q)}(R \bowtie_t S) = (\sigma_p(R)) \bowtie_t (s_q(S))$$
$$\sigma_{(p \wedge q)}(R \times S) = (\sigma_p(R)) \times (\sigma_q(S))$$

Ejemplo

$$\sigma_{(deptName='IT') \wedge (hireDate>'01\text{-}SEP\text{-}97')}(Emp \bowtie_{(Emp:deptId=Dept:deptId)} Dept) =$$
$$= \sigma_{(deptName='IT')}(Emp \bowtie_{(Emp:deptId=Dept:deptId)} (\sigma_{(hireDate>'01\text{-}SEP\text{-}97')} Dept)$$

7) Conmutatividad de la proyección y de la combinación Tetha (y del producto cartesiano).

$$\Pi_{L1 \cup L2}(R \bowtie_t S) = (\Pi_{L1(R)}) \bowtie_t (\Pi_{L2(S)})$$

Ejemplo

$$\Pi_{firstName;lastName;deptId;deptName}((Emp) \bowtie_{Emp:deptId=Dept:deptId} (Dept)) =$$
$$= (\Pi_{firstName;lastName;deptId}(Emp)) \bowtie_{Emp:deptId=Dept:deptId} (\Pi_{deptId;deptName}(Dept))$$

8) Conmutatividad de la unión y de la intersección, pero no de la diferencia.

$$R \cup S = S \cup R$$
$$R \cap S = S \cap R$$

9) Conmutatividad de la selección y de las operaciones de conjuntos (unión, intersección y diferencia).

$$\sigma_p(R \cup S) = \sigma_p(R) \cup \sigma_p(S)$$
$$\sigma_p(R \cap S) = \sigma_p(R) \cap \sigma_p(S)$$
$$\sigma_p(R - S) = \sigma_p(R) - \sigma_p(S)$$

10) Conmutatividad de la proyección y de la unión.

$$\prod_L(R \cup S) = \prod_L(R) \cup \prod_L(S)$$

11) Asociatividad de la combinación Tetha (y del producto cartesiano).

$$((R \bowtie S) \bowtie T) = (R \bowtie (S \bowtie T))$$
$$((R \times S) \times T) = (R \times (S \times T))$$

Ejemplo

$$((Emp \bowtie_{Emp:deptId=Dept:deptId} Dept) \bowtie_{Dept:locid=Locs:locid} Locs) =$$
$$= (Emp \bowtie_{Emp:deptId=Dept:deptId} (Dept \bowtie_{Dept:locid=Locs:locid} Locs))$$

12) Asociatividad de la unión y de la intersección (pero no de la diferencia de conjuntos).

$$(R \cup S) \cup T = S \cup (R \cup T)$$
$$(R \cap S) \cap T = S \cap (R \cap T)$$

4.2. Consideraciones sobre vistas en el SGBD Oracle 11g

En este apartado del anexo nos detendremos a considerar diferentes aspectos referentes al uso de las vistas en el SGBD Oracle 11g:

1) las vistas del diccionario de datos, y

2) las operaciones de actualización sobre vistas.

4.2.1. Vistas del diccionario de datos

El diccionario de datos está formado por un conjunto de tablas y vistas que proporcionan información sobre el contenido de una base de datos:

- estructuras de almacenamiento;
- usuarios y sus derechos, y
- los objetos: tablas, vistas, índices, procedimientos y funciones.

El diccionario de datos se carga en la memoria y se utiliza internamente para el tratamiento de las consultas. Existen dos grupos de tablas/vistas en el diccionario de datos:

1) Vistas estáticas. Se basan en tablas almacenadas en el espacio de tablas[18] *SYSTEM*. Hay tres grandes grupos de vistas estáticas:

[18] En inglés, *tablespace*.

a) *USER_%*. Contienen información sobre los objetos que pertenecen al usuario.

b) *ALL_%*. Contienen información sobre los objetos a los que el usuario tiene acceso.

c) *DBA_%*. Contienen información sobre todos los objetos de la base de datos.

Después del prefijo, el nombre describe la información a la que se accede cuando se hace una consulta sobre la vista.

2) Vistas dinámicas de rendimiento. Aunque contienen información que no se basa en tablas ni en otras vistas, sino que se captura de la memoria o de partes del fichero de control, estas vistas permiten efectuar consultas sobre el rendimiento de la base de datos o del SGBD. Se caracterizan porque tienen el prefijo *V$* y a continuación su nombre describe la información que representan. Estas vistas son accesibles sólo si el usuario tiene el privilegio DBA.

4.2.2. Operaciones de actualización sobre vistas

Las operaciones de actualización (*INSERT*, *DELETE* y *UPDATE*) son un tema conflictivo para los varios SGBD, ya que las vistas se basan en sentencias *SELECT*, en las que pueden intervenir muchas o pocas tablas e incluso otras vistas, y por lo tanto hay que decidir a cuál de estas tablas y/o vistas corresponde la operación de actualización solicitada.

Reflexión

Habrá que conocer muy bien las operaciones de actualización sobre las vistas que permite cada SGBD.

En el SGBD Oracle 11g, el concepto de *tabla key-preserved* es fundamental para entender las restricciones en las actualizaciones sobre vistas basadas en operaciones de combinación (*JOIN*).

Una tabla es *key-preserved* en una operación de combinación (*JOIN*) si cada valor clave de la tabla también puede ser valor clave en el resultado del *JOIN*.

La propiedad *key-preserved* de una tabla en un JOIN no depende de los datos de las tablas, sino que es una propiedad deducida a partir de su definición.

Ejemplo

En la vista *EmpDeptView*, a la que nos hemos referido ya en este módulo, la tabla *Emp* es *key-preserved* en *un join* con la tabla *Dept*, dado que la columna *empId* (clave primaria de la tabla *Emp*) continúa siendo única en el resultado de la combinación. En cambio, la tabla *Dept* no es *key-preserved*, puesto que la columna *deptId* (clave primaria de la tabla *Dept*) no es única en el resultado de la combinación.

El SGBD Oracle 11g proporciona la vista *USER_UPDATABLE_COLUMNS*, que permite conocer todas las columnas que podemos actualizar en las tablas y vistas a las que tenemos acceso.

Analizando el descriptor de la vista *USER_UPDATABLE_COLUMNS*, podemos observar que ésta nos muestra todas las columnas de todas las tablas y vistas a las que el usuario tiene acceso junto con las operaciones que puede ejecutar.

Descriptor de una tabla o vista

Para visualizar la descripción de una tabla o vista, se utiliza la sentencia SQL:

```
DESC [table_name|
table_view]
```

- La columna *owner* muestra el esquema que contiene la tabla o vista.
- La columna *table_name* contiene los nombres de las tablas y de las vistas a las que el usuario tiene acceso.
- La columna *column_name* nos indica las diversas columnas.

Ejemplos de columnas actualizables en una vista

Recordemos las dos vistas creadas sobre el esquema *COMPANY*. En caso de que queramos saber qué operaciones podemos realizar sobre las columnas de estas dos vistas, podemos consultar la vista *USER_UPDATABLE_COLUMNS*:

```
SELECT table_name, column_name, updatable, insertable, deletable
FROM user_updatable_columns
WHERE owner = 'COMPANY'
AND (table_name in ('EmpDeptView', 'EvenDeptView'));
```

Fijaos en que la columna *deptName* de la vista *EmpDeptView* no es actualizable, y que tampoco se pueden efectuar inserciones ni eliminaciones, es decir:

- Si ejecutamos una sentencia *DELETE* sobre la vista *EmpDeptView*, eliminaremos filas de la tabla *Emp* (a la que pertenecen las columnas que tienen *YES* como *deletable*), pero no eliminaremos ninguna fila de la tabla *Dept* (a la que pertenece la columna *deptName*).
- Podemos ejecutar *INSERT* sobre la vista *EmpDeptView* para rellenar filas de la tabla *Emp*, pero no para rellenar ninguna fila en la tabla *Dept*.
- Podemos ejecutar *UPDATE* sobre la vista *EmpDeptView* para modificar el contenido de las columnas provenientes de la tabla *Emp*, pero no podremos hacerlo para la columna *deptName* proveniente de la tabla *Dept*.

Condiciones para que una vista sea actualizable

La vista *USER_UPDATABLE_COLUMNS* sólo muestra las vistas que el sistema considera que pueden ser actualizables. Para que una vista tenga este reconocimiento, deben darse las condiciones siguientes:

a) Cada columna de la vista tiene que corresponder con una columna de una tabla simple.

b) La vista no puede contener ninguno de los componentes siguientes:

- operador *DISTINCT*;
- funciones de agrupamiento;
- cláusulas *GROUP BY*, *ORDER BY*, *CONNECT BY* o *START WITH*;

- subconsultas en la cláusula *SELECT*;
- vista creada con la opción *WITH READ ONLY*, y
- operaciones de combinación, con algunas excepciones, que están recogidas en la documentación de Oracle.

c) Además, si una vista actualizable contiene pseudocolumnas o expresiones, la operación de actualización no puede hacer referencia a ninguna de ellas.

d) Para que una vista basada en un *JOIN* sea actualizable, deben cumplirse todas las condiciones siguientes:

- La sentencia de actualización sólo puede afectar a una única tabla de las que forman parte de la operación de combinación.
- Para una sentencia *INSERT*, la vista no puede haber sido creada con *WITH CHECK OPTION*, y todas las columnas para las que se insertarán valores deben de pertenecer a una tabla *key-preserved*.
- Para una sentencia *UPDATE*, la vista no puede haber sido creada con *WITH CHECK OPTION*, y todas las columnas modificadas deben de pertenecer a una tabla *key-preserved*.
- Para las sentencias *DELETE* sobre vistas basadas en *JOIN*, si el *JOIN* está formado por más de una tabla *key-preserved*, se efectúa la eliminación sobre la primera tabla indicada en la cláusula *FROM*.

Ejemplo de operaciones de actualización sobre vistas

Supongamos que efectuamos algunas actualizaciones de departamentos pares por medio de la vista *EvenDeptView*.

```
INSERT INTO EvenDeptView VALUES (50,'R & D', 'Barcelona');
```

Esta instrucción provoca la inserción sin ningún problema de una fila en la tabla *Dept*.

Ahora bien, si la vista *EvenDeptView* hubiera sido creada con la opción *WITH CHECK OPTION*, al ejecutar cualquiera de las sentencias siguientes:

```
INSERT INTO EvenDeptView VALUES (55, 'Design', 'Girona');
UPDATE EvenDeptView SET deptId = deptId + 1 WHERE deptId = 50;
```

no se podrían realizar las acciones solicitadas, ya que no verifican la cláusula *WHERE* de la definición de la vista.

4.3. Aspectos referentes a la seguridad en el SGBD Oracle 11g

4.3.1. Gestión de usuarios

a) Creación de usuarios

Rol DBA

El rol DBA es aquel papel, rol, que permite a un usuario realizar tareas que corresponden al administrador de la base de datos.

La sentencia *SQL CREATE USER* permite crear un nuevo usuario.

La sintaxis es:

```
CREATE USER user_name
IDENTIFIED {BY password|EXTERNALLY}
[DEFAULT TABLESPACE tablespace_name]
[TEMPORARY TABLESPACE tablespace_name]
[QUOTA {value [K|M]|UNLIMITED} ON tablespace_name [ , ... ]]
[PROFILE profile_name]
[PASSWORD EXPIRE]
[ACCOUNT {LOCK|UNLOCK}];
```

Espacios de tablas *SYSTEM* y *SYSAUX*

Cuando se instala el SGBD Oracle, automáticamente se crean los espacios de tablas *SYSTEM*, en los que se almacenan los objetos del sistema, y *SYSAUX*, que se utiliza para hacer operaciones auxiliares del sistema.

Para que un usuario pueda conectarse, hay que darle los derechos sobre lo que puede hacer, en este caso atribuyéndole los privilegios de sistema *CREATE SESSION*.

Descripción de los parámetros:

- *IDENTIFIED*: esta cláusula indica si el usuario es identificado por el sistema operativo (*EXTERNALLY*) o mediante contraseña (*BY* contraseña). Desde la versión 11, las contraseñas son por defecto sensibles a mayúsculas/minúsculas (parámetro *SEC_CASE_SENSITIVE_LOGON = TRUE* por defecto). Esto significa que podemos tener creadas cuentas de usuario sin permiso de conexión. Aunque puede ser útil en la preparación de las cuentas de usuario sin activarlas de inmediato y deshabilitar así el acceso a un usuario de forma temporal, hoy en día se bloquea o desbloquea explícitamente una cuenta mediante *ACCOUNT LOCK|UNLOCK*.

- *DEFAULT NAMESPACE*: esta cláusula indica en qué espacio de tablas se crearán por defecto los segmentos del usuario en caso de que no se explicite ninguna cláusula *TABLESPACE* en el momento de la creación del segmento. Si esta cláusula se omite, permanece el espacio de tablas definido por defecto en la base de datos.

- *TEMPORARY NAMESPACE*: esta cláusula indica en qué espacio de tablas se crearán por defecto los segmentos de temporales del usuario, por ejemplo, creados en el momento de ejecución de una operación de ordenación. Si esta cláusula se omite, permanece el espacio de tablas temporal por defecto que haya definido en la base de datos.

- *QUOTA*: este concepto permite limitar el espacio que un usuario puede emplear en un espacio de tablas. Esta funcionalidad sólo afecta a los usuarios que pueden crear segmentos, y en ningún caso a los usuarios finales de una aplicación, dado que éstos se limitan a manipular datos. Por defecto, los usuarios no tienen ninguna cuota en ningún espacio de tablas, y en cambio los DBA tienen una cuota ilimitada en todos los espacio de tablas. En todo caso, hay que evitar dar cuotas a los usuarios en el *TABLESPACE SYSTEM* y en el *SYSAUX*.

- *PROFILE*: esta cláusula indica el perfil asignado al usuario.

- *PASSWORD EXPIRE*: esta cláusula permite forzar la modificación de la contraseña en el momento de la primera conexión. Carece de sentido si el usuario se identifica mediante el sistema operativo.

- *ACCOUNT*: esta cláusula admite uno de los dos parámetros siguientes: *LOCK*, si la cuenta existe pero evitamos que el usuario pueda conectarse a ella, o *UNLOCK*, donde la conexión está autorizada.

Ejemplo

```
CREATE USER joan IDENTIFIED BY passtemp
   DEFAULT TABLESPACE tablespace1
   QUOTA UNLIMITED ON tablespace1
   PASSWORD EXPIRE;
```

b) Modificación de un usuario

La sentencia SQL *ALTER USER* permite modificar un usuario. Las cláusulas son las mismas que para la creación.

Ejemplo

```
ALTER USER joan
   IDENTIFIED BY otherpasstemp
   PASSWORD EXPIRE;

ALTER USER joan
   DEFAULT TABLESPACE tablespace2
   QUOTA UNLIMITED ON tablespace2;

ALTER USER joan ACCOUNT LOCK;
```

El primer ejemplo cambia la contraseña de un usuario y le obliga a volver a cambiarla en la primera conexión. El segundo ejemplo modifica el espacio de tablas y asigna una cuota sin límite. El tercer ejemplo prohíbe temporalmente la conexión al usuario especificado.

C) Eliminación de un usuario

La sentencia SQL *DROP USER* permite eliminar a un usuario.

```
DROP USER user_name [CASCADE];
```

Si un usuario es propietario de un conjunto de objetos, la opción *CASCADE* es necesaria para eliminar tales objetos. En caso contrario, nos devolverá el código de error ORA-01922.

Reflexión

No se puede eliminar a un usuario que está conectado. En este caso nos devuelve el error ORA-01940.

También hay que recordar que con *DROP USER* no hay posibilidad de *ROLLBACK* porque es una sentencia DDL.

4.3.2. Definición de perfiles

Un perfil es un conjunto de limitaciones de recursos identificadas por un nombre que pueden asignarse a un usuario.

a) Creación de un perfil

La sentencia SQL *CREATE PROFILE* permite crear un nuevo perfil.

```
CREATE PROFILE profile_name LIMIT constraint_resources
```

Las limitaciones pueden ser de restricciones de recursos o para la gestión de contraseñas:

Parámetros para definir restricciones:

1) Resticciones de recursos:

- *SESSION_PER_USER*: número de sesiones simultáneas.
- *CPU_PER_SESSION*: asignación de CPU total por sesión.
- *CPU_PER_CALL*: asignación de CPU total por llamada.
- *CONNECT_TIME*: duración total de la conexión, en minutos.
- *IDLE_TIME*: tiempo de inactividad.
- *LOGICAL_READS_PER_SESSION*: número de lecturas lógicas por sesión.
- *LOGICAL_READS_PER_CALL*: número de lecturas lógicas por llamada.

- *COMPOSITE_LIMIT*: suma ponderada de *CPU_PER_SESSION*, *CONNECT_TIME*, *LOGICAL_READS_PER_SESSION* y *PRIVATE_SGA*. La vista *RESOURCE_COST* permite consultar las ponderaciones empleadas y la sentencia *ALTER RESOURCE COST* permite modificar las ponderaciones.

2) Restricciones sobre contraseñas:

- *FAILED_LOGIN_ATTEMPTS*: número de intentos de conexión fallidos como paso previo al bloqueo de cuenta.
- *PASSWORD_LOCK_TIME*: duración del bloqueo.
- *PASSWORD_LIFE_TIME*: duración de vida de la contraseña.
- *PASSWORD_GRACE_TIME*: periodo de gracia después de la caducidad de la contraseña.
- *PASSWORD_REUSE_TIME*: número de cambios de contraseña antes de que una contraseña pueda reutilizarse.
- *PASSWORD_VERIFY_FUCTION*: función de verificación de la complejidad de la contraseña. El *script utlpwdmg:sql* del repositorio $ORACLE_HOME/rdms/admin contiene un ejemplo de función de verificación.

En la mayoría de las restricciones mencionadas, se pueden emplear también las palabras clave *UNLIMITED* y *DEFAULT*.

b) Modificación de un perfil

La sentencia *ALTER PROFILE* permite modificar un perfil. Por ejemplo:

```
ALTER PROFILE default LIMIT
   SESSION_PER_USER 2
   IDLE_TIME 15
   FAILED_LOGIN_ATTEMPTS 3;
```

Los valores de los otros parámetros conservan su valor por defecto (*UNLIMITED*).

c) Asignación de un perfil a un usuario

Puede asignarse un perfil a un usuario en las situaciones siguientes:

1) En el momento de la creación de un usuario. Por ejemplo:

```
CREATE USER jordi IDENTIFIED BY password
   PROFILE default
   PASSWORD EXPIRE;
```

2) Cuando se modifica un usuario. Por ejemplo:

```
ALTER USER joan PROFILE default;
```

d) Información referente a usuarios y sus perfiles

Para obtener información sobre usuarios y perfiles, se pueden consultar las siguientes vistas del diccionario de datos:

- *DBA_USERS*: información sobre los usuarios.
- *DBA_TS_QUOTAS*: información sobre cuotas de usuarios.
- *DBA_PROFILES*: información sobre los perfiles.

4.3.3. Identificación de usuarios

Un usuario puede identificarse desde el SGBD Oracle o bien desde el propio sistema operativo.

1) Identificación por Oracle: el usuario se conecta a la base de datos utilizando un nombre y una contraseña. Por ejemplo:

```
SQL > CONNECT joan/GNB9175$
Conectado.
```

2) Identificación por el sistema operativo: Oracle no verifica la contraseña. Debe configurarse el parámetro *OS_AUTHEN_PREFIX = OPS* si se quiere tener esta funcionalidad habilitada.

4.3.4. Gestión de los privilegios

En una base de datos Oracle, los derechos de los usuarios se gestionan a partir del concepto de *privilegio*. Un **privilegio** es:

- el derecho a ejecutar una sentencia SQL general, por ejemplo de crear una tabla (este concepto se denomina **privilegio de sistema**);
- el derecho a acceder a un objeto de otro usuario (este concepto se denomina **privilegio de objeto**).

1) Privilegios de sistema

Un privilegio de sistema es el derecho a ejecutar una sentencia SQL en general. Cada sentencia SQL presenta como mínimo un privilegio de sistema asociado que tiene el mismo nombre que la sentencia SQL. Así pues, la sentencia SQL *CREATE TABLE* tiene un privilegio de sistema asociado llamado *CREATE TABLE*, que otorga el derecho de crear una tabla en su propio esquema. También hay que tener en cuenta que el privilegio *CREATE ANY TABLE* proporciona el derecho de crear tablas en cualquier esquema de la base de datos.

Los privilegios de sistema son una fuente de riesgo, sobre todo aquellos relacionados con la gestión de usuarios y sus derechos (*CREATE USER*, *ALTERUSER*, *DROP USER*, *GRANT ANY PRIVILEGE*, *GRANT ANY ROLE*) y todos aquellos que permiten eliminar objetos (*DROP ANY TABLE*, *DROP TABLESPACE*, etc.).

Algunos privilegios

- *CREATE SESSION*: otorga a un usuario el derecho a conectarse. Si un usuario carece de este privilegio, se devuelve el error ORA-01045.
- *SELECT ANY DICTIONARY*: permite consultar cualquier objeto que pertenezca al diccionario de datos del esquema SYS.

Los privilegios de sistema se utilizan principalmente para controlar el uso de las sentencias DDL; por lo general, están destinados a administradores y/o desarrolladores y a la cuenta propietaria de la aplicación, rara vez a usuarios finales.

La **sentencia para otorgar privilegios** es:

```
GRANT privilege_name [ , ... ] TO (authorized|PUBLIC ) [ , ... ]
[WITH ADMIN OPTION];
```

Un privilegio puede asignarse a un usuario, a un grupo de usuarios o a todo el mundo (*PUBLIC*). El privilegio asignado está activo inmediatamente. La cláusula *WITH ADMIN OPTION* proporciona al beneficiario el derecho a transmitir este privilegio de sistema.

Es necesario que un usuario que quiera asignar o revocar un privilegio de sistema haya recibido previamente:

- el mismo privilegio con la cláusula *WITH ADMIN OPTION*, y
- el privilegio de sistema *GRANT ANY PRIVILEGE*.

Para revocar un privilegio a un usuario, se utiliza la sentencia *REVOKE*.

La **sentencia de revocación de privilegios** es la siguiente:

```
REVOKE privilege_name [ , ... ] FROM (authorized|PUBLIC) [ ,... ]
```

No hay cascada en la revocación de un privilegio de sistema que haya sido transmitido gracias a la cláusula *WITH ADMIN OPTION*.

Ejemplo

Si se asigna a Juan un privilegio con la opción *WITH ADMIN OPTION* y este lo ha transmitido a Jorge, revocar este privilegio a Juan no tiene ningún efecto sobre el privilegio transmitido por él a Jorge.

Si se ha asignado un privilegio a un usuario con la opción *WITH ADMIN OPTION* y se desea eliminar esta opción, hay que revocar el privilegio y asignarlo de nuevo sin la opción *WITH ADMIN OPTION*.

También hay que tener en cuenta que la sentencia *REVOKE* permite revocar privilegios que un usuario haya recibido directamente, no los que el usuario recibe vía *PUBLIC*. Se pueden revocar todos los privilegios de sistema mediante la sentencia:

```
REVOKE ALL PRIVILEGES FROM autorizado;
```

2) Privilegios de objeto

Un privilegio sobre un objeto es el derecho a acceder a un objeto de otro usuario. Por defecto, el propietario es el único que tiene derecho a acceder al objeto.

Los privilegios de objeto se utilizan fundamentalmente para permitir a los usuarios finales de una aplicación acceder a los objetos de la aplicación creados en una cuenta propietaria de la aplicación. Para que otro usuario pueda acceder al objeto, es preciso que el propietario le asigne un privilegio objeto:

Tabla 6. Ejemplos de privilegios.

Privilegio	Definición	Tabla	Vista	Secuencia	Programa
`SELECT [(columnas)]`	Derecho de lectura de datos.	√	√	√	√
`INSERT [(columnas)]`	Derecho de creación de datos.	√	√		
`UPDATE [(columnas)]`	Derecho de actualización de datos*.	√	√		
`DELETE`	Derecho de eliminación de datos*.	√	√		
`EXECUTE`	Derecho de ejecución de un programa.				√

* Para lograr los privilegios *UPDATE* y *DELETE*, hay que conceder también el privilegio *SELECT*.

La sentencia SQL *GRANT* permite asignar un privilegio objeto.

La **sentencia de otorgamiento de privilegios** es la siguiente:

```
GRANT {privilege_name [(column_list)][ , ... ]|ALL [PRIVILEGES]}
ON [schema_name.] object_name
TO {authorized|PUBLIC} [ , ... ]
[WITH GRANT OPTION];
```

Para los privilegios *INSERT* y *UPDATE*, se pueden especificar las columnas para indicar a cuáles de ellas se limita el privilegio.

Un privilegio puede asignarse a un usuario, a un grupo de usuarios o a todo el mundo (*PUBLIC*).

La cláusula *WITH GRANT OPTION* otorga al beneficiario el derecho a transmitir este privilegio objeto.

Para asignar o revocar un privilegio objeto, es necesario:

- ser el propietario del objeto;
- haber recibido el mismo privilegio con la cláusula *WITH ADMIN OPTION*, y
- haber recibido el privilegio de sistema *ANY OBJECT PRIVILEGE*.

Cuando queremos acceder a un objeto del que se han recibido privilegios con la opción *WITH GRANT OPTION*, es preciso calificarlo con el nombre del propietario, porque el SGBD asume por defecto que este objeto se encuentra en el propio esquema.

Para facilitar la escritura de sentencias y que el esquema propietario de los objetos sea transparente, es necesario emplear los sinónimos. La existencia de un sinónimo, y aunque éste sea público, no da ningún derecho sobre el objeto subyacente.

La sentencia SQL *REVOKE* permite revocar un privilegio objeto.

La **sentencia de revocación de privilegios** es la siguiente:

```
REVOKE {privilege_name [ , ... ]|ALL [PRIVILEGES]}
ON [schema_name.] object_name
FROM {authorized|PUBLIC}[ , ... ];
```

También hay que tener en cuenta que la sentencia *REVOKE* permite revocar privilegios que un usuario haya recibido directamente, no los que el usuario recibe vía *PUBLIC*.

Se pueden revocar todos los privilegios de sistema mediante la sentencia:

```
REVOKE ALL PRIVILEGES ON...FROM... autorizado;
```

Se produce cascada en la revocación de un privilegio de objeto que haya sido transmitido gracias a la cláusula *WITH GRANT OPTION*.

Ejemplo

Si se asigna a Juan un privilegio con la opción *WITH GRANT OPTION* y éste lo ha transmitido a Jorge, el hecho de revocar después este mismo privilegio a Juan comporta también la revocación inmediata sobre Jorge.

Si se ha asignado un privilegio a un usuario con la opción *WITH GRANT OPTION* y se desea eliminar esta opción, es necesario revocar el privilegio y asignarlo de nuevo sin la opción *WITH GRANT OPTION*.

Algunas consideraciones sobre privilegios en vistas y programas almacenados

El hecho de que un usuario disponga de un derecho sobre una vista no implica que tenga ningún derecho sobre los objetos subyacentes a la vista. En cambio, por defecto, un programa almacenado se ejecuta con los derechos del propietario. El comportamiento deseado se define en el momento de la creación del programa almacenado gracias a la cláusula *AUTHID*.

La sintaxis de la cláusula *AUTHID* es:

```
AUTHID {CURRENT_USER|DEFINER}
```

Consulta de información al catálogo sobre privilegios

Existen diferentes vistas en el catálogo sobre los privilegios de sistema:

- *DBA_SYS_PRIVS*: muestra los privilegios de sistema asignados a los usuarios o a los roles.
- *SESION_PRIVS*: muestra los privilegios de sistema actualmente activos en la sesión, ya sean obtenidos directamente o mediante un rol.
- *SYSTEM_PRIVILEGE_MAP*: lista todos los privilegios de sistema.

También podemos obtener del diccionario de datos información sobre los privilegios objeto a través de las vistas siguientes:

- *DBA_TAB_PRIVS*: muestra los privilegios objeto asignados a los usuarios o a los roles sobre la totalidad del objeto.
- *DBA_COL_PRIVS*: muestra los privilegios objeto asignados únicamente sobre ciertas columnas del objeto.
- *TABLE_PRIVILEGE_MAP*: muestra la lista de todos los privilegios de objeto.

4.3.5. Roles

Dentro de una organización, los roles se crean para modelizar funciones de trabajo diferentes. Los permisos para realizar determinadas operaciones están asignados a roles específicos. A los usuarios del sistema se les asigna determinados roles, a través de los cuales adquieren los diferentes permisos que les

permiten ejercer funciones específicas en el sistema informático. Dado que los usuarios no tienen permisos de forma directa, sino que los adquieren a través de su rol, la gestión de los derechos de cada usuario se convierte simplemente en una cuestión de asignar los roles apropiados a la cuenta del usuario, cosa que simplifica las operaciones más comunes, como por ejemplo añadir un usuario o cambiar las funciones de un usuario.

El modelo basado en roles RBAC define las siguientes tres normas primarias:

1) Asignación de roles: un sujeto puede ejercer un permiso sólo si el sujeto ha sido seleccionado o se le ha asignado un rol.

2) Autorización de rol: es necesario que el rol activo de un sujeto esté autorizado por el sujeto. Junto con el artículo 1, esta norma garantiza que los usuarios puedan asumir roles sólo para los cuales están autorizados.

3) Autorización de permiso: un usuario puede ejercer un permiso sólo si el permiso es autorizado por el rol activo del usuario. Junto con las normas 1 y 2, esta norma garantiza que los usuarios ejerzan sólo los permisos para los cuales están autorizados.

Como restricciones adicionales, los roles se pueden combinar en una jerarquía de niveles, donde los roles de nivel superior asumen los permisos de propiedad de sus subroles.

Figura 4. Modelo RBAC

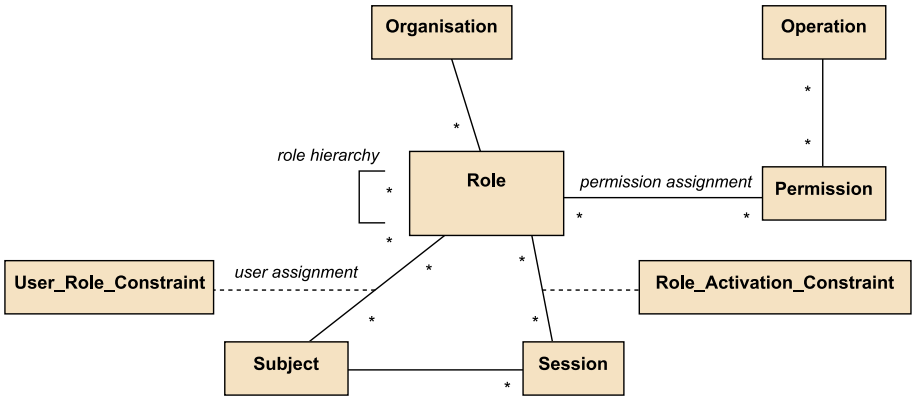


Figura 4

En la figura podemos observar las reglas siguientes:

- Un sujeto puede tener varios roles.
- Un rol puede tener varios sujetos.
- Un rol puede tener muchos permisos.
- Un permiso puede ser asignado a múltiples roles.
- Una operación puede tener asignados muchos permisos.
- Un permiso puede estar asignado a muchas operaciones.

Los privilegios se pueden asignar directamente a los usuarios o mediante roles. El uso de RBAC para administrar los privilegios de usuario está ampliamente aceptado como una buena práctica. Oracle permite modelizar RBAC.

Un **rol** es un agrupamiento de privilegios definido por un nombre que puede atribuirse a un usuario, de modo que el usuario recibe automáticamente los privilegios contenidos en el rol. Los roles se utilizan para la gestión de los derechos.

Reflexión

Para crear un rol un usuario debe tener el privilegio de sistema *CREATE ROLE*.

La sentencia SQL *CREATE ROLE* permite crear un rol.

La **sentencia de creación de un rol** presenta la sintaxis siguiente:

```
CREATE ROLE role_name
[IDENTIFIED {BY password|EXTERNALLY|USING package}
|NOT IDENTIFIED];
```

En el momento de creación del rol es posible precisar con qué mecanismo se podrá activar, ya sea mediante una contraseña, con autenticación externa (por ejemplo, mediante el sistema operativo) o mediante un paquete.

Los requisitos y la sintaxis referente a la asignación y/o revocación de privilegios de sistema y de objeto por roles son los mismos que para los usuarios. Los privilegios son inmediatamente asignados o revocados según cuál sea la instrucción, y su efecto es inmediato sobre los usuarios conectados que tienen el rol activo.

Reflexión

Un usuario puede tener varios roles, en cuyo caso los privilegios se acumulan (no hay efecto "negativo").

La **sentencia de asignación de roles** a usuarios presenta la sintaxis siguiente:

```
GRANT role_name [ , ... ]
TO {user_name|role_name|PUBLIC} [ , ... ]
[WITH ADMIN OPTION];
```

Los requisitos y la sintaxis referentes a la revocación de un rol a un usuario o a otro rol y sobre la eliminación de un rol son parecidos a los explicados anteriormente.

Un rol asignado a un usuario se activa por defecto de forma automática en el momento de la conexión del usuario. Si el usuario ya estaba conectado en el momento de la asignación, la activación inmediata del rol no es automática, sino que hay que activar el rol con la sentencia *SET ROLE*.

Disponer de la posibilidad de emplear diferentes roles sin que estén activos al mismo tiempo es interesante porque:

- el parámetro *MAX_ENABLED_ROLES* (por defecto, treinta) limita el número de roles activos simultáneos para un usuario;
- los roles protegidos con contraseña se pueden asignar a los usuarios aunque permanezcan inactivos y, sin dar la contraseña al usuario, encargar a las aplicaciones la activación de los roles, proporcionando las contraseñas cuando sea necesario.

La sentencia *ALTER USER* permite definir los roles por defecto de un usuario.

La **sintaxis de la sentencia *ALTER USER*** es la siguiente:

```
ALTER USER user_name
DEFAULT ROLE {role_name [ , ... ]|ALL {EXCEPT role_name [ , ... ]|NONE};
```

La sentencia *SET ROLE* permite activar o desactivar un rol.

La **sintaxis de la sentencia *SET ROLE*** es la siguiente:

```
SET ROLE {role_name [IDENTIFIED BY password][ ,... ]|ALL {EXCEPT role_name[ , ... ]|NONE};
```

Consulta al catálogo para obtener información sobre roles

Además de las ya consideradas (*DBA_SYS_PRIVS*, *DBA_TAB_PRIVS* y *DBA_COL_PRIVS*), existen varias vistas del diccionario de datos que permiten obtener información sobre los roles:

- *DBA_ROLES*: listado de los roles existentes en la base de datos.
- *DBA_APLICATION_ROLES*: descripción de los roles que tienen los sistema de activación por medio de un paquete.
- *DBA_ROLE_PRIVS*: roles asignados a usuarios o a otros roles.
- *ROLE_SYS_PRIVS*: privilegios de sistema asignados a roles.
- *ROLE_TAB_PRIVS*: privilegios de objeto asignados a roles.
- *ROLE_ROLE_PRIVS*: roles asignados a otros roles.
- *SESION_ROLES*: roles actualmente activos en la sesión.

4.4. Estadísticas de la base de datos y planes de ejecución en Oracle

Como hemos visto en el apartado «Estadísticas de bases de datos» de este módulo, el SGBD tiene en cuenta las estadísticas que le constan de los datos obtenidos en las tablas de la base de datos para optimizar las consultas. Por este hecho, resulta importante refrescar a menudo los datos de estas estadísticas y muy especialmente después de un poblado masivo de datos.

La actualización de las estadísticas de la base de datos con Oracle se realiza ejecutando este procedimiento:

```
BEGIN
Dbms_Stats.Gather_Schema_Stats (
 ownname => 'User_Name',
 estimate_percent => 100);
END;
```

Donde *User_Name* es el usuario del propietario de las tablas.

Además, se debe de tener en cuenta que no conviene forzar los planes de ejecución de las consultas usando los HINT o equivalentes. Esto es porque una variación en la proporción de los datos de las columnas implicadas en la consulta u otras casuísticas puede provocar que los planes de ejecución cambien. Esto haría que un plan que inicialmente pueda ser considerado óptimo deje de serlo.

HINT en el Oracle

Los HINT son unas indicaciones que se pueden añadir a las consultas SQL (no forman parte del estándar), en el caso concreto del SGBD Oracle, para forzar que el planificador de ejecución de las consultas utilice un plan de acceso concreto: utilizar o evitar un índice, forzar un recorrido de toda la tabla...

Supongamos que creamos una tabla para almacenar los datos de los productos existentes en un comercio:

```
CREATE TABLE Product (
idProduct CHAR(6 CHAR) CONSTRAINT PK_Product PRIMARY KEY,
type VARCHAR2(20 CHAR) CONSTRAINT NN_type NOT NULL,
description VARCHAR2(20 CHAR),
price NUMBER(4,2) CONSTRAINT NN_price NOT NULL
);
```

Y además, como se prevé que a menudo se hagan consultas de productos por tipo de producto, crearemos un índice sobre el campo *type*.

```
CREATE INDEX IDX_type ON Product(type);
```

Imaginemos que consultamos los productos que tienen el valor *Cleaning* en este campo.

```
SELECT type FROM Product WHERE type = 'Cleaning';
```

Dependiendo de la cantidad de productos distintos que pueda tener la columna *type*, y la proporción que represente sobre el total de productos, se usará el índice creado o no. En el caso de que aproximadamente el número de filas con este valor represente menos del 15% del total de filas de la tabla *Product*, se usará el índice.

¿Qué sucedería si aumentara la proporción de filas con el valor *Cleaning* en el campo *type* y no se actualizasen las estadísticas de la base de datos? Pues que el optimizador seguiría eligiendo el mismo plan de ejecución (seguiría usando el índice), cuando con toda seguridad sería mejor realizar un recorrido completo de la tabla para recuperar los registros que cumplieran la condición, ya que leyendo bloques enteros serían necesarios menos accesos a disco que realizando múltiples accesos a los datos vía el índice.

Para verlo más claro, supongamos ahora dos consultas que fuerzan al optimizador. Una mediante un HINT para usar un índice concreto, y la otra para que haga un recorrido completo de la tabla:

```
-- Se fuerza a utilizar el índice denominado IDX_type
```

```
SELECT /*+ INDEX(Product IDX_type)*/ *
FROM Product
WHERE type = 'Cleaning';



-- Se fuerza a realizar un recorrido completo de la tabla

SELECT /*+ FULL(Product)*/ *
FROM Product
WHERE type = 'Cleaning';
```

Independientemente de que recalculemos las estadísticas, siempre nos encontraríamos con que uno de los dos casos no sería óptimo al forzar el tipo de ejecución, prescindiendo del que el optimizador considerase como mejor plan de ejecución.

La mejor opción siempre será dejar que el optimizador elija el plan de ejecución que considera óptimo, y actualizar las estadísticas de la base de datos siempre que se pueda. Por eso, en muchos sistemas en explotación existe una tarea nocturna programada que se ejecuta diariamente refrescando estadísticas.

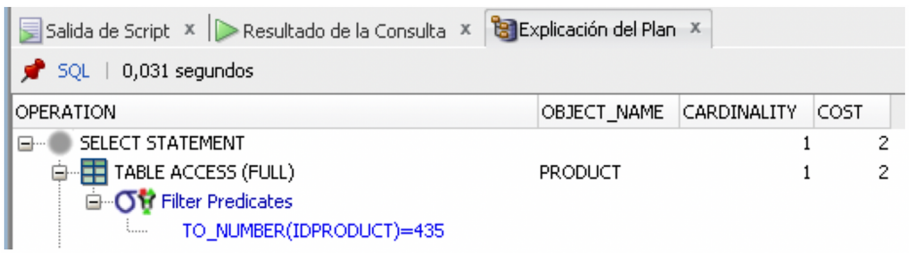
Otra consideración interesante es que se debe hacer una revisión del primer plan de ejecución de las primeras consultas previstas, con el objetivo de asegurarnos de que su ejecución será la esperada. Así evitaremos cualquier problema que pueda surgir si falta algún índice, si hay algún error en la consulta o incluso algún defecto del programario del SGBD.

Si usáramos el Oracle (Express Edition Release 11.2.0.2.0) y quisiéramos localizar los datos del producto con el valor 435 como identificador de producto, ejecutaríamos una sentencia como esta:

```
SELECT * FROM Product WHERE idProduct = 345;
```

Suponiendo que su plan de ejecución sea este:

Figura 5. Plan de ejecución de la consulta `(SELECT * FROM Product WHERE idProduct = 435)` amb SQL Developer



Seguramente nos sorprendería que, aunque se cumplan los requisitos para que el optimizador elija el índice creado automáticamente al definir la clave primaria (*PK_Product*), el optimizador no lo haga. La explicación de este hecho es que la versión de Oracle que utilizamos busca si hay algún tipo de índice compatible con la columna para la que se filtra antes de realizar la conversión de tipo de la condición de filtrado. En este caso, la columna *idProduct* está definida como *CHAR(6 CHAR)* mientras que la condición de filtrado es por tipo entero. Dado que el optimizador detecta que no existe un índice compatible con el tipo del valor dado, no lo utiliza.

Resumen

En este módulo hemos estudiado el procesamiento de consultas y vistas, así como la seguridad en las bases de datos.

Hemos visto que transformar una consulta en otra, para que sea más eficiente, es responsabilidad del SGBD. Este proceso de búsqueda de una buena estrategia para realizar el procesamiento de una consulta recibe el nombre de *optimización de consultas*. En primer lugar se comprueba la correctitud semántica y léxica de la consulta SQL, y después se transforma en un árbol que permite su análisis y optimización. Después se decide cuál es la mejor estrategia de implementación física.

Una vista es una relación virtual que no existe físicamente en la base de datos, sino que se genera cada vez que un usuario efectúa una solicitud. El mecanismo de las vistas contribuye a la seguridad al permitir ocultar los detalles de la base de datos a ciertos usuarios. Con el uso de disparadores de sustitución se puede conseguir que las vistas sean actualizables.

Hemos expuesto el concepto de seguridad de una base de datos como el conjunto de mecanismos que protegen a la base de datos frente a amenazas intencionadas o accidentales. La mayoría de SGBD proporcionan un mecanismo denominado *control de acceso discrecional* (DAC) que gestiona los privilegios empleando el lenguaje SQL. Aunque algunos SGBD proporcionan técnicas de control de acceso obligatorio (MAC) basadas en políticas de nivel de sistema que no pueden ser alteradas por los usuarios, el estándar SQL no incluye soporte para MAC. Finalmente, hemos introducido brevemente la legislación vigente de protección de datos, así como la necesidad de velar por cualquier tipo de datos de carácter personal que deben estar protegidos por ley.

Glosario

amenaza *f* Cualquier situación o suceso intencionado o accidental que pueda afectar de manera adversa al sistema y, en consecuencia, a la organización.

autenticación *f* Mecanismo por el cual se determina si un usuario es quien dice ser.

autorización *f* Concesión de un derecho o privilegio que permite a un sujeto acceder legítimamente al sistema o a un objeto del sistema.

cifrado *m* Codificación de datos mediante un algoritmo especial que provoca que estos datos no sean legibles para ningún programa que no disponga de la clave de descifrado.

heurístico -a *adj.* Cualidad de los métodos que utilizan el razonamiento y las experiencias pasadas para encontrar la mejor solución a un problema.

mecanismo de copia de seguridad *m* Proceso de realizar de forma periódica una copia de la base de datos, del archivo de registro y, posiblemente, de algún programa, y almacenarla en un dispositivo de almacenamiento fuera de línea.

optimización *f* Proceso por el cual se transforma una consulta en otra equivalente pero más eficiente. La optimización se puede realizar en el ámbito semántico, sintáctico y físico.

plan de ejecución *m* Conjunto de operaciones (lógicas o físicas) necesarias para obtener el resultado de una consulta.

registro *m* Proceso de mantener un diario donde se almacenen los cambios efectuados en la base de datos con el objetivo de realizar la recuperación de manera efectiva en caso de quiebra del sistema.

vista *f* Resultado dinámico de una o más operaciones relacionales sobre una base de datos con el objetivo de producir otra relación.

Bibliografía

AEPD. Directrices para la elaboración de contratos entre responsables y encargados del tratamiento. Recuperado de https://www.aepd.es/media/guias/guia-directrices-contratos.pdf

AEPD. Guía del Reglamento General de Protección de Datos para responsables de tratamiento. Recuperado de https://www.aepd.es/media/guias/guia-rgpd-para-responsables-de-tratamiento.pdf

AEPD. Guía para el cumplimiento del deber de informar. Recuperado de https://www.aepd.es/media/guias/guia-modelo-clausula-informativa.pdf

Connolly, T.; Begg, C. (2005). *Sistemas de bases de datos* (4.ª ed.). Madrid: Pearson.

Huey, P. (2011). *Oracle database security guide 11g release 1*. Oracle.

Lorentz, D.; Roeser, M. B. (2011). *Oracle database SQL language reference, 11g Release 2*. Oracle.

Silberschatz, A.; Korth, H.; Sudarshan, S. (2006). *Fundamentos de bases de datos* (5.ª ed.). Madrid: McGraw-Hill. Edición de 2011 en eBook.

Weinberg, P.; Groff, J.; Oppel, A. (2009). *SQL. The complete reference* (3.ª ed.). McGraw-Hill.