Processament de consultes i vistes

Seguretat en bases de dades

Joan Anton Pérez Braña
Santiago Ortego Carazo (†2007)

PID_00187553

UOC
Universitat Oberta de Catalunya

Índex

Introducció..... 5

Objectius..... 6

1. Processament de consultes..... 7
1.1. Descomposició de consultes 11
1.1.1. Validació lèxica i sintàctica 11
1.1.2. Normalització de la consulta 12
1.1.3. Anàlisi semàntica 13
1.1.4. Simplificació 13
1.2. Optimització semàntica 14
1.3. Optimització sintàctica 15
1.3.1. Regles d'equivalència 15
1.3.2. Estratègies de processament heurístic 16
1.4. Estimació de costos per a les operacions d'àlgebra relacional 16
1.4.1. Estadístiques de la base de dades 17
1.4.2. Operació de selecció 19
1.4.3. Operació d'ordenació 21
1.4.4. Operació de projecció 22
1.4.5. Operació de combinació 22
1.4.6. Operacions de conjunts d'àlgebra relacional 25
1.4.7. Agregació 25
1.5. Optimització física 26
1.5.1. Optimització heurística 26
1.5.2. Optimització basada en costos 27

2. Processament de vistes..... 32
2.1. Mecanismes d'implementació de vistes 34
2.2. Actualització de vistes 35
2.2.1. Actualització amb disparadors de substitució 36
2.3. La vista com a element de disseny extern 37
2.4. Les taules derivades 38
2.5. Taules temporals 42
2.6. Avantatges i inconvenients en la utilització de vistes 42

3. La seguretat..... 44
3.1. Identificació i autenticació 46
3.2. Control d'accés 48
3.2.1. Control d'accés discrecional 49
3.2.2. Control d'accés obligatori 50
3.2.3. Classificació dels sistemes de seguretat 51

3.3. Implementació del control d'accés discrecional a SQL:2011 51
3.4. Auditoria 54
3.5. La Llei de protecció de dades de caràcter personal 55
3.5.1. Principis generals de protecció de dades 55
3.5.2. Els nivells de seguretat 57
3.5.3. L'Agència de Protecció de Dades 59

4. Annexos..... 60
4.1. Regles d'equivalència d'operacions d'àlgebra relacional 60
4.2. Consideracions sobre vistes en l'SGBD Oracle 11g 62
4.2.1. Vistes del diccionari de dades 62
4.2.2. Operacions d'actualització sobre vistes 63
4.3. Aspectes referents a la seguretat en l'SGBD Oracle 11g 65
4.3.1. Gestió d'usuaris 65
4.3.2. Definició de perfils 67
4.3.3. Identificació d'usuaris 69
4.3.4. Gestió dels privilegis 70
4.3.5. Rols 74

Resum..... 78

Glossari..... 79

Bibliografia..... 80

Introducció

El llenguatge SQL és un llenguatge. Com a tal, les consultes realitzades amb aquest llenguatge especifiquen què es vol obtenir en lloc de com es vol aconseguir. En el present mòdul s'explica com els sistemes gestors de bases de dades (SGBD) relacionals avaluen sistemàticament les estratègies alternatives que es poden presentar, d'acord amb les necessitats dels usuaris i les aplicacions, amb l'objectiu d'escollir l'estratègia que es considera òptima.

També veurem que les vistes proporcionen mecanismes de seguretat i permeten al dissenyador de la base de dades personalitzar el model lògic al model dels usuaris. Analitzarem els casos en què una vista és actualitzable i els mecanismes que incorporen els SGBD per fer actualitzable una vista.

Finalment estudiarem les tècniques que s'utilitzen per a protegir la base de dades contra accessos no autoritzats i els mecanismes que hi ha per a atorgar o revocar privilegis als diferents usuaris segons el model de negoci implícit en el sistema d'informació. També veurem les obligacions legals derivades del fet de tenir determinades dades especialment protegides.

Objectius

Els materials didàctics associats a aquest mòdul us permetran assolir els objectius següents:

1. Saber els mecanismes de processament i optimització de consultes que hi ha, per tal de poder plantejar les consultes de la manera més eficient possible.

2. Conèixer les diverses estratègies d'implementació de les operacions d'àlgebra relacional per a poder avaluar el cost de les consultes.

3. Interpretar i optimitzar adequadament el pla d'execució d'una consulta.

4. Presentar les vistes com a elements de disseny extern que permeten l'actualització de dades.

5. Conèixer noves aplicacions de les vistes per tal de poder millorar el disseny de la base de dades.

6. Conèixer l'abast dels mecanismes de seguretat d'una base de dades.

7. Prendre consciència de les obligacions legals derivades del compliment de la Llei orgànica de protecció de dades de caràcter personal.

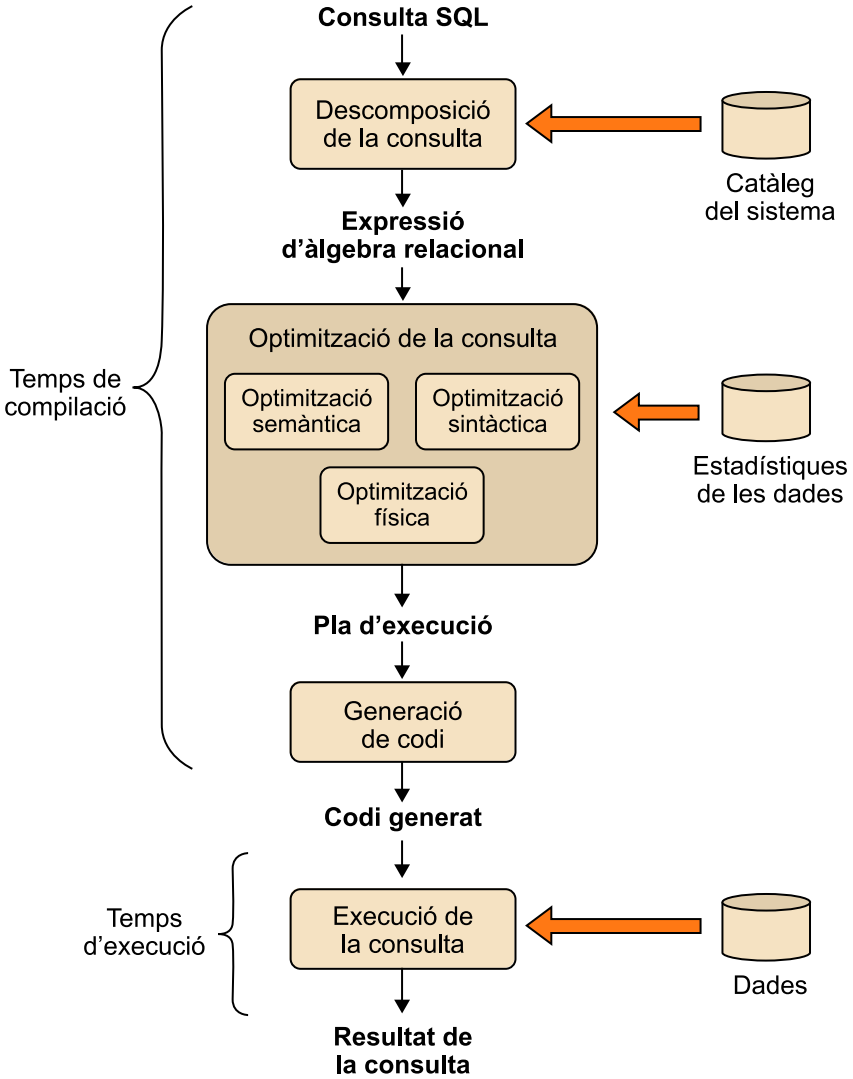
1. Processament de consultes

Una de les crítiques principals als primers sistemes gestors de bases de dades (SGBD[1]) relacionals va ser el baix rendiment en el processament de consultes. En els llenguatges no procedimentals, com és el cas de l'SQL, l'usuari especifica quines dades vol obtenir en lloc de com vol aconseguir-les.

(1) *SGBD* és la sigla de *sistema gestor de bases de dades.*

El **processament de consultes** fa referència al conjunt d'activitats que l'SGBD duu a terme per extreure informació d'una base dades amb l'objectiu d'assolir l'estratègia més eficient i que li permeti tenir un major control sobre les prestacions del sistema.

Figura 1. Etapes del processament de consultes



En la figura 1 es poden veure gràficament les quatre etapes que componen el processament de consultes:

1) Descomposició de la consulta

2) Optimització de la consulta

3) Generació de codi

4) Execució de la consulta

La **descomposició de consultes** implica la traducció de consultes expressades en llenguatge SQL a una representació interna basada en l'àlgebra relacional, que sol ser més útil.

Primerament es comprova la correctesa sintàctica i semàntica de la consulta en SQL i posteriorment es crea un arbre sobre el qual es fa l'anàlisi de la consulta, que es transformarà en una expressió d'àlgebra relacional.

Pel que fa a les expressions d'àlgebra relacional, utilitzarem el conveni de símbols següent:

Concepte	Representació	Observacions
Relacions	R, S	La llista d'atributs de cada relació es defineix segons l'esquema següent: $R = (A_1, A_2, ..., A_n)$.
Predicats	p, q	Aquests predicats s'avaluaran lògicament en les operacions de selecció i combinació.
Selecció	$\sigma_p(R)$	Aquesta operació retorna els tuples de la relació R que satisfan l'avaluació del predicat p.
Projecció	$\Pi L(R)$	En què L correspon a una llista d'atributs que es mostraran de la relació R.
Unió	$(R) \cup (S)$	S'obté una relació nova que inclou els tuples de la relació R i S menys les repeticions.
Intersecció	$(R) \cap (S)$	S'obté una relació nova que inclou els tuples que pertanyen a totes dues relacions, R i S.
Diferència	$(R) - (S)$	S'obté una relació nova que inclou els tuples que pertanyen a la relació R, però que no estan inclosos en la relació S.
Producte cartesià	$(R) \times (S)$	S'obté una relació nova formada per tots els tuples que resulten de concatenar tuples de la relació R amb tuples de la relació S.
Combinació theta	$(R) \bowtie_p (S)$	S'obté una relació nova que inclou els atributs dels parells de tuples corresponents a R i S que satisfan el predicat p.

A partir del primer arbre de consulta lògic, que representa l'expressió relacional, comença el procés d'optimització de la consulta.

L'**optimització de consultes** consisteix a trobar el pla d'execució més eficient entre diverses estratègies disponibles, per al processament d'una consulta donada.

L'optimització de consultes es duu a terme des de tres vessants:

1) Optimització semàntica. Consisteix a reescriure la consulta basant-se en les restriccions especificades en l'esquema de la base de dades.

2) Optimització sintàctica. Consisteix a transformar heurísticament l'expressió relacional original en una altra d'equivalent, però que sigui molt més eficient. De la mateixa manera que una consulta es pot expressar en SQL de formes diverses, una consulta SQL també es pot traduir a diferents expressions d'àlgebra relacional.

Considerem l'esquema de la base de dades *COMPANY* que es defineix a continuació. Les claus primàries es mostren subratllades.

```
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)
```

Imaginem que volem consultar els empleats de la base de dades *COMPANY* que pertanyen al departament 'IT' i que són alta a l'empresa a partir de l'1 de setembre de 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 d'aquesta consulta podem obtenir tres expressions d'àlgebra relacional equivalents:

Reflexió

Fixeu-vos que la sèrie d'operacions d'àlgebra relacional és diferent en cada cas.

a) Es fa el producte cartesià entre les relacions *Emp* i *Dept* i, posteriorment, s'efectuen les operacions de selecció corresponents.

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

b) Es combinen les taules i aleshores es duen a terme les operacions de selecció.

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

c) Primerament s'efectuen les operacions de selecció corresponents a cada taula i posteriorment es combinen els resultats.

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

3) Optimització física. Permet escollir entre els diversos plans d'avaluació, amb costos diferents, el que és més eficient. Per a determinar el cost més eficient de cada consulta cal conèixer el cost de la seqüència d'operacions de l'àlgebra que la formen, i per tant, de cada operació, que sovint depèn de diferents paràmetres. Així doncs, per a especificar completament com s'avaluarà la consulta cal determinar els algorismes que s'utilitzaran en cada operació i els índexs que s'hi faran servir. Aquestes especificacions s'anomenen **primitives d'avaluació** i la seqüència d'aquestes primitives s'anomena **pla d'execució d'una consulta**.

Reflexió

Prèviament a l'estudi de l'optimització física s'oferirà una visió de l'estimació de costos per a les diferents operacions de l'àlgebra relacional.

El **pla d'execució** és la seqüència de passos que farà l'SGBD per executar una sentència SQL.

L'SGBD construeix un pla d'execució que minimitza l'ús de recursos de manera que intenta reduir al mínim el temps d'execució de la consulta, ja sigui minimitzant el temps de cada operació o maximitzant el nombre d'operacions paral·leles. Finalment, el **motor d'execució** executarà el pla i obtindrà el resultat de la consulta a partir de les dades emmagatzemades.

Funcions resum

Una funció resum és un algorisme o operació que permet obtenir un sumari o resum a partir d'un conjunt de dades. Aquest sumari o resum està associat a les dades originals, i qualsevol canvi en les dades originals ha de repercutir en el sumari o resum.

Tot seguit veurem amb més detall les diferents etapes que componen el processament de consultes en SQL.

1.1. Descomposició de consultes

Es tracta de la primera fase del processament d'una consulta. Es comprova que la consulta és correcta des dels punts de vista sintàctic i semàntic i, si és així, es transforma la consulta expressada en llenguatge SQL en un conjunt d'operacions d'àlgebra relacional.

Les etapes típiques de la descomposició de consultes són les següents:

1) Validació lèxica i sintàctica

2) Normalització de la consulta

3) Anàlisi semàntica

4) Simplificació

1.1.1. Validació lèxica i sintàctica

La consulta s'analitza lèxicament i sintàcticament emprant les tècniques de compiladors dels llenguatges de programació.

S'accedeix al diccionari de dades i es comprova que totes les taules i els atributs que s'esmenten en la consulta realment existeixen. També es comprova que l'usuari té els drets d'accés adequats i que les operacions fetes són pròpies per a aquest tipus d'objecte.

Vegeu també

Els drets d'accés s'estudien en l'apartat 3 d'aquest mòdul didàctic.

Exemple de validació lèxica i sintàctica

Considerem l'esquema de la base de dades *COMPANY* i la consulta següent.

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

Aquesta consulta seria rebutjada per dos motius:

1) L'atribut *emplId* no està definit per la relació *Emp*; caldria anomenar-lo *empId*.

2) La comparació *LIKE '10'* és incompatible amb el tipus de dades *deptId* que és numèric, en lloc d'una cadena de caràcters.

Un cop acabada aquesta anàlisi, la consulta s'ha transformat en un arbre de consulta construït de la manera següent:

1) Per a cada relació base de la consulta es crea un node fulla.

2) Per a cada operació intermèdia produïda per una operació d'àlgebra relacional es crea un node no fulla.

3) El resultat de la consulta es representa com l'arrel de l'arbre.

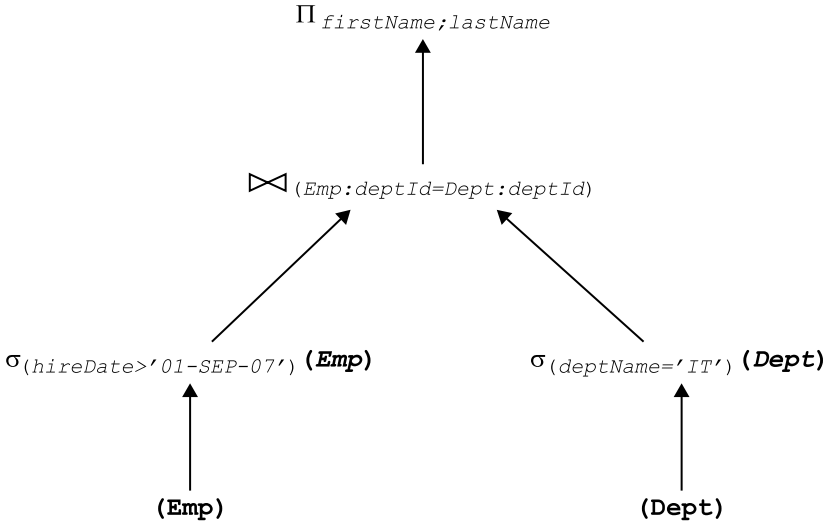
4) La seqüència d'operacions es dirigeix dels nodes fulla al node arrel.

Exemple de transformació d'una consulta en un arbre d'operacions relacionals

Donat l'esquema de la base de dades *COMPANY* i la consulta següent:

```
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');
```

L'arbre que mostra el pla d'avaluació es presenta tot seguit.



Que correspon a l'expressió relacional següent:

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

1.1.2. Normalització de la consulta

S'avalua el predicat de la clàusula *WHERE*, que sovint sol ser prou complex, per a convertir la consulta en una d'aquestes formes normalitzades:

- **Forma normal conjuntiva**. Genera una seqüència de conjuncions connectades amb l'operador $\wedge$ (AND). Cada conjunció pot contenir un o més predicats connectats amb l'operador $\vee$ (OR).

  Exemple

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

- **Forma normal disjuntiva**. Genera una seqüència de disjuncions connectades amb l'operador $\vee$ (OR). Cada disjunció pot contenir un o més predicats connectats amb l'operador $\wedge$ (AND).

  Exemple

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

1.1.3. Anàlisi semàntica

En aquesta etapa es duu a terme una anàlisi semàntica amb l'objectiu de rebutjar les consultes normalitzades que són contradictòries o no estan ben formulades.

Una consulta és incorrecta si, després d'haver-la formulat, els components que la integren no permeten la generació de resultat, cosa que pot passar si falta alguna especificació de combinació.

Exemple 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;
```

Aquesta consulta es considera incorrecta atès que les connexions entre relacions no són completes, ja que s'ha omès la condició de combinació *d.locId=l.locId*.

Una **consulta contradictòria** és la que en la formulació conté algun predicat que no pot satisfer cap tuple.

Exemple de consulta contradictòria

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

Aquesta consulta és contradictòria perquè les dues condicions de la clàusula *WHERE* són incompatibles.

1.1.4. Simplificació

L'objectiu d'aquesta etapa és detectar predicats redundants, eliminar expressions comunes i transformar una consulta en una altra que sigui semànticament equivalent, però que es pugui calcular d'una manera més eficient. Per a realitzar aquest procés s'apliquen les regles de la lògica.

Exemple de simplificació

Considerem la consulta següent.

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

Aquesta consulta es pot simplificar, ja que la segona condició inclou la primera:

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

Exemple de simplificació de predicats redundants

Considerem també aquesta altra consulta.

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

Es pot transformar en una altra consulta amb una condició *WHERE* equivalent:

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

1.2. Optimització semàntica

L'optimització semàntica fa servir una tècnica que utilitza les restriccions especificades en el catàleg de la base de dades per a reduir l'espai de cerca.

Exemple d'optimització semàntica

Si hi ha una restricció que indica que el salari ha de ser superior a 500 euros, la consulta que presentem a continuació:

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

Es podria simplificar així:

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

Exemple d'optimització semàntica

Considerem tanmateix aquesta altra consulta.

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

Atès que tots els valors d'*e.deptId* han d'aparèixer a *d.deptId*, la consulta que presentem a continuació és equivalent a l'anterior:

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

La implementació d'aquest tipus d'optimitzador en l'SGBD permet incrementar el rendiment, sobretot en els sistemes que tenen una semàntica molt rica i completament implementada, ja que evita haver de cercar segons les condicions que no es compleixen per a cap fila de la taula.

1.3. Optimització sintàctica

Mitjançant l'aplicació de diferents regles de transformació, l'optimitzador pot transformar una expressió d'àlgebra relacional en una altra expressió equivalent, i que sigui més eficient. Per a realitzar aquest procés s'apliquen les regles d'equivalència entre les operacions d'àlgebra relacional amb l'objectiu de trobar una expressió equivalent.

1.3.1. Regles d'equivalència

Les regles d'equivalència i les estratègies de processament heurístic que s'utilitzen per a reestructurar l'arbre d'operacions d'àlgebra relacional són les següents:

Vegeu també

Podeu consultar als annexos de l'apartat 4 diversos exemples de regles d'equivalència.

1) Les operacions conjuntives de selecció es poden transformar en una cascada d'operacions individuals de selecció.

2) Les operacions de selecció són commutatives.

3) En una seqüència d'operacions de projecció tan sols és necessària l'última projecció de la seqüència.

4) Commutativitat de la selecció i la projecció.

5) Commutativitat de la combinació theta (i del producte cartesià).

Combinacions theta

Són combinacions que utilitzen els operadors de comparació com a condició de combinació.

6) Commutativitat de la selecció i de la combinació tetha (i del producte cartesià).

7) Commutativitat de la projecció i de la combinació tetha (i del producte cartesià).

8) Commutativitat de la unió i de la intersecció, però no de la diferència.

9) Commutativitat de la selecció i de les operacions de conjunts (unió, intersecció i diferència).

10) Commutativitat de la projecció i de la unió.

11) Associativitat de la combinació tetha (i del producte cartesià).

12) Associativitat de la unió i de la intersecció (però no de la diferència de conjunts).

1.3.2. Estratègies de processament heurístic

Hi ha una sèrie de regles heurístiques que permeten trobar una bona expressió equivalent, moltes vegades la millor, a partir de l'aplicació de les propietats esmentades anteriorment.

Vegeu també

Observeu que en l'arbre que es representa en el subapartat 1.1.1 s'han aplicat aquestes estratègies de processament.

Els passos heurístics normalment acceptats són els que es presenten a continuació.

1) Realitzar les operacions de selecció tan aviat com sigui possible. Atès que les operacions de selecció redueixen la cardinalitat de la relació resultant, utilitzarem la regla 1 amb l'objectiu de "connectar en cascada" totes les operacions de selecció. Posteriorment s'aplicaran les regles 2, 4, 6 i 9 definides anteriorment referents a la commutativitat de la selecció amb operacions unàries i binàries. Les operacions de selecció apareixeran a prop dels nodes fulla de l'arbre, cosa que indicarà que són les primeres a efectuar-se. Es mantindran junts els predicats referents a la mateixa relació.

2) Transformar el producte cartesià de dues relacions seguit d'una operació de selecció, en una operació de combinació.

3) Emprar l'associativitat de les operacions binàries per a reordenar els nodes fulla. L'objectiu és que les operacions de selecció més restrictives s'executin en primer lloc.

4) Dur a terme les operacions de projecció tan aviat com sigui possible.

5) El càlcul de les expressions comunes tan sols es farà una vegada.

1.4. Estimació de costos per a les operacions d'àlgebra relacional

Les implementacions d'operacions de l'àlgebra relacional són diferents per a cada SGBD i cadascuna té un cost associat. Quan avaluem el cost d'una consulta, és a dir, el temps de resposta per al pla d'avaluació d'una consulta, s'han de tenir en compte diversos factors, com ara el cost de CPU, els accessos a memòria, els accessos a disc i fins i tot el temps emprat per les comunicacions amb sistemes remots (en el cas dels SGBD distribuïts).

En els grans SGBD el cost més important correspon a l'accés a disc, ja que és el dispositiu més lent. Així doncs, per a comparar les diverses estratègies de manera relativa tan sols s'utilitzarà el nombre de pàgines de disc a les quals s'ha accedit (per a operacions de lectura o escriptura) en el pitjor dels casos.

Consideracions respecte al cost dels accessos a disc

Considerem que el subsistema de disc triga t_T segons a efectuar una operació de transferència d'un bloc de dades amb un temps d'accés a bloc t_B, que correspon al temps de cerca a disc més la latència rotacional. Llavors una operació que transfereixi *nBlocks* i executi N cerques tindria un cost temporal de $nBlocks \cdot t_T + N \cdot t_B$.

Paràmetres d'accés a disc

Per a tenir una idea dels paràmetres associats als accessos a disc podem suposar que t_T = 0,1 ms i t_B = 4 ms, en el cas que la mida d'un bloc sigui de 4 kilobytes i es disposi d'una velocitat de transferència de 40 megabytes per segon.

En les consideracions següents no es tindran en compte les operacions d'escriptura a disc que triguen el doble de temps, ni tampoc si hi ha cap informació en memòria intermèdia. Aquest valor d'estimació de cost s'expressarà com a *CE*.

1.4.1. Estadístiques de la base de dades

L'èxit de l'estimació de la mida i del cost de les operacions intermèdies depèn de la qualitat i del grau d'actualització de la informació estadística emmagatzemada en un SGBD. Per a poder estimar el cost de cadascun dels algorismes, cal disposar de valors estadístics de les taules que intervenen en la consulta, la majoria dels quals són valors mitjans que es mantenen calculats (o es recalculen de tant en tant) en el diccionari de dades. Els estadístics més importants són els següents.

Per a cada relació (o taula) *R:*

1) *nTuples*(*R*): nombre de tuples de la relació *R*. És a dir, la seva cardinalitat.

2) *blockFactor*(*R*): factor de blocatge de *R*. El nombre de files de *R* que caben en una pàgina.

3) *nBlocks*(*R*): nombre de blocs requerits per a guardar tots els tuples de *R*, en què:

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

4) *t*(*R*): mida en bytes d'una fila de *R*.

Per a cada atribut *A* de la taula *R*:

1) $nDistinct_A(R)$: nombre de valors diferents de l'atribut *A* que apareixen a *R*.

2) $min_A(R)$: valor més petit de l'atribut *A* en la taula *R*.

3) $max_A(R)$: valor més gran de l'atribut *A* en la taula *R*.

4) $CS_A(R)$: cardinalitat de la selecció de l'atribut. És el nombre mitjà de files que compleixen una condició sobre l'atribut *A*.

Per a una condició d'igualtat sobre l'atribut A, el càlcul és el següent.

a) Si A és una clau candidata:

$$CS_A(R) = 1$$

b) Si els valors estan distribuïts uniformement:

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

També és possible, considerant una distribució uniforme, calcular la cardinalitat de selecció per a altres casos diferents de la igualtat.

a) Per a predicats de comparació del tipus $A > a$:

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

b) Per a predicats de comparació del tipus $A < a$:

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

c) Per a $A \subset (a_1, \ldots , a_n)$, en què n és el nombre d'elements del conjunt $(a_1, \ldots , a_n)$:

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

d) Per a la disjunció de dos predicats, $A \wedge B$:

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

e) Per a la conjunció, $A \vee B$:

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

Per a cada índex multinivell I d'un atribut A:

1) $nLevelsA(I)$: el nombre de nivells en I.

2) $nLfBlocksA(I)$: el nombre de pàgines de fulles en I.

Una altra dada important és el nombre de pàgines de memòria, M, que es poden utilitzar com a memòria intermèdia.

1.4.2. Operació de selecció

Hi ha molts algorismes per a la selecció de valors en una taula. A continuació fem un repàs dels més importants i n'indiquem el cost.

1) Cerca lineal sobre fitxers no ordenats. Amb aquest algorisme s'exploren totes les pàgines que formen la taula i es comprova cadascuna de les files per a veure si conté el valor adequat de l'atribut.

Si es tracta d'un atribut que no és una clau candidata, el cost serà:

$$CE = nBlocks(R)$$

Si l'atribut sobre el qual es fa la cerca és una clau candidata, de mitjana el trobarem per la meitat de la taula, de vegades abans i de vegades després. En aquest cas el cost serà:

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

2) Cerca binària sobre fitxers ordenats. Consisteix a examinar en primer lloc la posició central de la taula de manera que s'elimina la meitat que no pertoca i es repeteix el procediment fins que es troba el valor cercat.

- Per a trobar un tuple amb un atribut que sigui clau, el cost serà:

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

- Si es tracta d'un atribut no clau i es considera una distribució uniforme dels valors, tindrem:

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

3) Igualtat amb una clau *hash*. Si l'atribut és una clau *hash*, s'aplica l'algorisme *hash* per a calcular l'adreça corresponent a un tuple determinat. Si no hi ha desbordament, el cost serà 1.

4) Igualtat de clau primària amb un índex B$^+$*tree*. Si la cerca es fa respecte a la igualtat d'una clau primària sobre la qual s'ha definit un índex amb estructura d'arbre B$^+$, el nombre de pàgines llegides serà igual al nombre de nivells de l'arbre més la pàgina de les dades; és a dir:

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

B$^+$*tree*

Un arbre B$^+$ és un tipus d'estructura de dades en forma d'arbre en què tota la informació es guarda en les fulles, ja que els nodes interns només contenen claus i punters. D'aquesta manera es poden crear índexs multinivell dinàmics amb un límit màxim i mínim en el nombre de claus per node.

5) Desigualtat amb la clau primària. Si la condició de cerca és la de desigualtat amb la clau principal, es pot emprar l'índex per a localitzar el tuple que satisfà el predicat $A = x$ i així, posteriorment, llegir els tuples que es troben situats abans o després de la localitzada.

6) Igualtat amb un índex secundari. Si es tracta d'un índex que no representa una clau candidata i, per tant, hi ha diferents valors que poden complir la igualtat, s'hauran de llegir totes les pàgines que els continguin i el cost serà:

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

Exemple d'estimació de cost per a una operació de selecció

Per a assolir el propòsit d'aquest exemple, es duen a terme els supòsits següents referents a la relació *Emp* descrita anteriorment en l'esquema de la base de dades *COMPANY*.

- Hi ha un índex *hash* sobre l'atribut de clau principal *empId*.
- Hi ha un índex d'agrupament sobre l'atribut de clau externa *deptId*.
- Hi ha un índex arbre B^+ sobre l'atribut *salary*.
- La relació *Emp* té les estadístiques següents emmagatzemades al catàleg del sistema:

$nTuples(Emp) = 3000$
$blockFactor(Emp) = 30) \Rightarrow nBlocks(Emp) = 100$
$nDistinct_{jobId}(Emp) = 50) \Rightarrow CS_{jobId}(Emp) = 60$
$nDistinct_{firstName}(Emp) = 3000) \Rightarrow CS_{firstName}(Emp) = 1$
$nDistinct_{salary}(Emp) = 500) \Rightarrow CS_{salary}(Emp) = 6$
$min_{salary}(Emp) = 10.000$
$max_{salary}(Emp) = 50.000$
$nLevels_{jobId}(I) = 2$
$nLevels_{salary}(I) = 2$
$nLfBlocks_{salary}(I) = 2$

El cost estimat d'una cerca lineal sobre l'atribut *deptId* és de 50 blocs i el cost d'una cerca lineal sobre un atribut no clau és de 100 blocs.

Es calcula el cost estimat per a les operacions següents:

1) $\sigma_{(empId='T450')}(Emp)$. L'operació de selecció conté una condició d'igualtat sobre la clau principal. L'atribut *empId* està emmagatzemat mitjançant un mètode *hash*; per tant, el cost estimat serà d'un bloc ($CE = 1$) i la cardinalitat estimada de la relació resultant és:

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

2) $\sigma_{(firstName='Smith')}(Emp)$. L'atribut és no clau i no indexat; per tant, no es pot millorar el mètode de cerca lineal. El cost estimat obtingut serà $CE = 100$. La cardinalitat estimada s'ha calculat prèviament:

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

3) $\sigma_{(jobId='prog')}(Emp)$. L'atribut del predicat és una clau externa amb un índex de clusterització; per tant, el cost serà $CE = 2 + [60/30] = 4$ blocs. La cardinalitat estimada és:

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

4) $\sigma_{(salary>20.000)}(Emp)$. El predicat implica una cerca dins del rang de l'atribut *salary* que té un índex arbre B+, per la qual cosa per a calcular el cost estimat s'efectuarà el càlcul $CE = 2 + [50/2] + [3.000/2] = 1.527$. La cardinalitat estimada és:

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

1.4.3. Operació d'ordenació

Sovint cal ordenar les dades d'una taula, tant perquè es vol el resultat ordenat com perquè s'utilitza per a implementar més eficientment una operació de combinació.

Ordenació ràpida

L'ordenació ràpida (*quick sort*, en anglès) és un algorisme basat en la tècnica de "divideix i venceràs", que permet, de mitjana, ordenar n elements en un temps proporcional a $n \cdot \log(n)$.

L'ordenació es pot aconseguir amb la creació d'un índex sobre l'atribut objecte de l'ordenació i llavors aquest índex es pot emprar per a fer una lectura ordenada. No obstant això, cal tenir en compte que aquesta ordenació és des del punt de vista lògic, ja que físicament això es pot traduir en un accés a disc per a cada tuple, més la transferència del bloc corresponent. Això pot ser molt costós atès que el nombre de registres sol ser superior al nombre de blocs, ja que els tuples no han d'estar necessàriament emmagatzemats físicament d'una manera consecutiva.

Marcs de pàgina

Considerem que el nombre de marcs de pàgina M és el nombre de blocs de disc que es poden emmagatzemar a la memòria intermèdia de la memòria principal.

En cas que tota la relació es pugui emmagatzemar a la memòria principal, es poden emprar tècniques d'ordenació ràpida, com ara l'anomenat *algorisme Quicksort*.

Quan la relació no cap a la memòria, s'acostuma a fer servir l'algorisme d'ordenació-fusió externa. Considerem que es disposa de M marcs de pàgina a la memòria intermèdia de la memòria principal. Aquesta operació consta de dues fases:

L'algorisme en la primera fase divideix la relació en $N = [nBlocks(R)/M]$ parts i les carrega successivament a la memòria per ordenar cada part i crear un arxiu de seqüències S_i.

En la segona etapa, i en cas que el nombre de seqüències N sigui inferior a M:

1) Es llegeix en el primer bloc de cada arxiu de seqüència el primer tuple de cada bloc, per a procedir a la fusió ordenada dels primers blocs de cada seqüència.

2) En cas que el contingut del bloc s'hagi exhaurit, es llegeix el bloc següent del mateix arxiu de seqüències fins a esgotar tots els blocs de cada seqüència, la qual cosa implicarà que s'haurà obtingut l'ordenació.

3) En cas que el nombre de seqüències N encara sigui superior a M, es fusionen les $M - 1$ primeres seqüències per a formar una altra seqüència ordenada. En aquest punt s'ha reduït el nombre de seqüències en $M - 1$.

4) En cas que el nombre de seqüències sigui inferior a M, aleshores es pot procedir a la fusió definitiva. En cas contrari, es repeteix el procediment amb altres $M - 1$ seqüències fins a assolir un nombre de seqüències més petit que M.

El cost total d'aquest algorisme és tal com s'indica a continuació.

El nombre inicial de seqüències és $N = [nBlocks(R)/M]$. Com que aquest nombre decreix en un factor $M - 1$ en cada cicle de fusió, el nombre total de cicles requerits serà $\log_{M-1} \cdot [nBlocks(R)/M]$. En cada cicle es llegeixen i s'escriuen els blocs de la relació un sol cop; per tant, el nombre de transferències de bloc serà $nBlocks(R)[\log_{M-1}[(nBlocks(R)/M] + 1]$. També cal afegir els costos de cerca a disc. En la fase de fusió, si es llegeixen simultàniament $nBlocks[S]$ de cada seqüència, aleshores cada pas implica com a mínim $nBlocks(R) = nBlocks(S)$ cerques per a la lectura de dades. També cal tenir en compte que, encara que la sortida s'escrigui seqüencialment, el capçal es pot haver desplaçat; per tant, caldrà afegir $2[nBlocks(R)/nBlocks(S)]$ cerques per a cada pas, excepte en el pas final.

Així doncs, el cost estimat serà:

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

1.4.4. Operació de projecció

Una projecció implica dues operacions: eliminar els atributs no volguts i eliminar, si escau, els tuples repetits que s'han generat amb la clàusula *DISTINCT*.

Observació

Si dins de la llista d'atributs de la projecció s'inclouen tots els que formen la clau primària, no es produiran duplicats.

Es pot implementar fàcilment l'eliminació de duplicats emprant l'ordenació dels tuples i utilitzant com a clau d'ordenació els atributs resultants de la projecció. Els tuples idèntics apareixen contigus durant l'operació d'ordenació, per tant, es poden eliminar tots menys un. El cost estimat en el pitjor dels casos és el mateix que el cost estimat en el pitjor dels casos en l'operació d'ordenació.

1.4.5. Operació de combinació

Les operacions de combinació, a part de la del producte cartesià, acostumen a ser les que consumeixen més temps durant el processament de consultes i, per tant, marcaran d'una manera important l'eficiència global de l'SGBD.

Els algorismes més importants són els següents: la combinació de cicle imbricat, la combinació de cicle imbricat indexat, la combinació per ordenació per fusió, la combinació per funció resum i la combinació per agrupació (clúster).

Combinació de cicle imbricat

La combinació de cicle imbricat[2] consisteix en dos cicles imbricats: el bucle extern recorre tots els tuples de la primera relació R i un bucle intern recorre tots els tuples de la relació S. El cost estimat per a aquesta solució és:

(2)En anglès, *nested loop join*.

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

Podem veure clarament que, per a reduir el cost, la primera relació que s'utilitza en el bucle extern ha d'ocupar el mínim nombre de blocs.

Una millora consisteix a llegir tants blocs com sigui possible de la relació més petita, i reservar un bloc per a la relació interna i un altre per a la relació resultant. Si la memòria intermèdia[3] permet emmagatzemar un nombre de blocs, *nBuffer*, llavors es llegeixen (*nBuffer* – 2) blocs de *R* i un bloc de *S* cada cop. Amb aquesta tècnica, l'estimació del cost seria:

(3)En anglès, *buffer*.

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

També podríem considerar que la memòria és suficient perquè, després de la primera lectura, tots els blocs de *R* es puguin llegir de la memòria intermèdia de la base de dades. En aquest cas, l'estimació del cost seria:

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

Combinació de cicle imbricat indexat

El cost de la combinació de cicle imbricat indexat[4] varia segons el mètode d'indexació que s'empri. Si existeix un índex o funció resum[5] sobre els atributs de combinació de la relació interna llavors en el bucle interior no es fa un recorregut sobre totes les pàgines de la taula sinó que, mitjançant l'índex o la funció resum, es va directament a la pàgina adequada.

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

(5)En anglès, *hash function*.

Si es té un índex arbre B+ sobre l'atribut *A*, per a trobar el cost de la consulta caldrà conèixer *nLevelsA*(*I*), el nombre de nivells de l'arbre B+ de l'índex *I* sobre l'atribut *A*; i $CS_A(R)$, la cardinalitat de selecció de l'atribut *A* de la taula *R* i del nombre de files que caben en una pàgina *blockFactor*(*R*). Si es tracta d'un índex sobre una clau candidata de *R*, el cost serà:

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

Si l'atribut és un índex d'agrupació, aleshores l'estimació del cost serà:

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

Combinació per ordenació per fusió

En una combinació per ordenació per fusió[6], la combinació més eficient s'assoleix quan les relacions estan ordenades segons els atributs de combinació. En cas que no sigui així, caldrà un pas previ per a ordenar-les. Una vegada ordenades, només cal anar a llegir seqüencialment cadascuna de les taules i afegir als resultats els valors que coincideixen. Si considerem que la combinació és de tipus "molts a molts", és a dir, que hi ha diferents tuples de *R* i de *S*

(6)En anglès, *sort-merge join*.

amb el mateix valor de combinació i a més assumim que cada conjunt de tuples amb el mateix valor pot cabre en la memòria intermèdia, llavors tan sols caldrà llegir cada bloc de la relació un sol cop. Així, l'estimació del cost seria:

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

Si cal ordenar una de les relacions, aleshores s'hi ha d'afegir el cost d'ordenació:

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

Tenint en compte que s'arrodoneix per excés el càlcul de cada logaritme.

Combinació per funció resum

L'algorisme de combinació per funció resum[7] es basa a emprar una funció resum (*hash*) que presenti característiques d'uniformitat i aleatorietat per a dividir les files de totes dues taules en conjunts que tinguin el mateix valor de la funció resum. Després es fa la combinació de les files de cada partició.

[7] En anglès, *hash join*.

Funcions resum

Una funció resum és un algorisme o operació que permet obtenir un sumari o resum a partir d'un conjunt de dades. Aquest sumari o resum està associat a les dades originals, i qualsevol canvi en les dades originals ha de repercutir en el sumari o resum.

El cost de la combinació per funció resum és:

- $CE = 3 \cdot (nBlocks(R) + nBlocks(S))$, si l'índex *hash* s'emmagatzema a memòria i no es tenen en compte els desbordaments.
- $CE = 2 \cdot (nBlocks(R) + nBlocks(S)) \cdot [\log_{nBuffer-1}(nBlocks(S)) - 1] + nBlocks(R) + nBlocks(S)$, en un altre cas.

Exemple d'estimació de cost per a una operació de combinació

Per a assolir el propòsit d'aquest exemple, es duen a terme els supòsits següents referents a les relacions *Emp* i *Jobs*.

- Hi ha un índex *hash* sense desbordament sobre l'atribut de clau principal *depId*.
- Hi ha uns 100 blocs de memòria intermèdia de la base de dades.
- Tenim les estadístiques següents emmagatzemades en el catàleg del sistema:

```
nTuples(Emp) = 3.000
blockFactor(Emp) = 30) ⇒ nBlocks(Emp) = 100
nTuples(Jobs) = 50
blockFactor(Jobs) = 10) ⇒ nBlocks(Jobs) = 5
```

Es vol avaluar el cost de les diverses estratègies per a la combinació següent:

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

1) La combinació de cicle imbricat

- La memòria intermèdia tan sols conté un bloc de *Emp* i *Jobs*:

$$CE = nBlocks(Emp) + nBlocks(Emp) \cdot nBlocks(Jobs) = 100 + 100 \cdot 5 = 600$$

- El nombre de blocs que es tractaran per *Emp* és $(nBuffer - 2)$:

$$CE = nBlocks(Emp) + nBlocks(Jobs) \cdot nBlocks(Emp)/(nBuffer - 2) =$$

$$= 100 + 100 \cdot 5/98 = 106$$

- Tots els blocs de la relació *Emp* caben en la memòria intermèdia:

$$CE = nBlocks(Emp) + nBlocks(Jobs) = 100 + 5 = 105$$

2) La combinació de cicle imbricat indexat

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

3) La combinació per ordenació per fusió

- Tuples desordenats:

$$CE = nBlocks(Emp) + nBlocks(Jobs) + nBlocks(Emp) \cdot \log_2(nBlocks(Emp) +$$

$$+ nBlocks(Jobs) \cdot \log_2(nBlocks(Jobs)) = 100 + 5 + 100 \cdot 7 + 5 \cdot 3 = 820$$

- Tuples ordenats:

$$CE = nBlocks(Emp) + nBlocks(Jobs) = 100 + 5 = 105$$

4) La combinació per funció resum. Si l'índex *hash* cap a la memòria:

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

1.4.6. Operacions de conjunts d'àlgebra relacional

Les operacions d'unió $R \cup S$, intersecció $R \cap S$ i diferència $R - S$ es poden implementar ordenant primerament les dues relacions, atenent un mateix criteri, per a posteriorment produir el resultat.

Els resultats per a cada operació es calculen de la manera següent:

- Quan es duu a terme la unió de R i S ($R \cup S$), es realitza una lectura concurrent dels tuples de les dues relacions i si es detecta el mateix tuple en les dues relacions, s'elimina el duplicat.

- Quan es duu a terme la intersecció de R i S ($R \cap S$), tan sols s'emmagatzemen els tuples que apareguin com a duplicats en les dues relacions.

- Quan es duu a terme la diferència $R - S$, s'emmagatzemen els tuples de R que no apareixen a S.

Tot això tindria un cost $CE = nBlocks(R) + nBlocks(S)$ més el cost de l'ordenació de R i S.

1.4.7. Agregació

Qualsevol operació d'agrupació es pot implementar d'una manera semblant a l'eliminació de duplicats, però, en lloc d'eliminar els tuples que tenen el mateix valor, es reuneixen en grups i s'aplica l'operació corresponent per a obtenir el resultat.

Operacions d'agrupació

Són operacions d'agrupació les funcions *min*, *max*, *sum*, *count* i *avg*.

1.5. Optimització física

L'optimitzador físic de consultes és el component de l'SGBD que fa que els plans d'execució de les consultes es duguin a terme en el mínim temps. Hi ha diverses possibilitats per a aconseguir-ho:

- Que el cost de CPU sigui mínim.
- Que la necessitat de memòria sigui mínima.
- Minimitzar els accessos a disc.
- Altres possibilitats.

Els algorismes d'optimització física es poden classificar en dues grans famílies:

1) Optimització heurística. També es coneix com a *optimització basada en regles*. Consisteix a escollir un pla d'execució físic amb els algorismes que normalment són més eficients.

2) Optimització basada en costos. Consisteix a trobar totes les implementacions físiques, una vegada es té tot l'espai de possibles realitzacions de la consulta. Per a cada pla se'n calcula el cost (temps, nombre d'accessos a disc, etc.) i finalment s'escull el que té el cost més petit.

Generalment, els SGBD comercials permeten escollir un tipus d'implementació o un altre, si bé la tendència és oferir només optimització basada en costos, ja que, encara que l'heurística és molt ràpida de calcular, pot triar plans d'execució molt poc eficients.

1.5.1. Optimització heurística

Per a poder realitzar una optimització heurística, cal que l'SGBD tingui definit un ordre d'eficiència per a les diferents operacions lògiques i pugui escollir el que acostuma a tenir un cost més baix.

Com la majoria d'opcions de disseny, l'optimització heurística té virtuts i defectes. La virtut més important és que no cal efectuar cap tipus de càlcul, l'obtenció d'un bon pla d'execució es fa d'una manera immediata; el defecte principal és que, de vegades, els plans proposats poden diferir molt dels òptims.

En una base de dades complexa, una consulta pot implicar diverses taules, cadascuna amb diversos índexs i condicions de selecció complexes. Aquesta complexitat significa que hi podria haver una gran quantitat d'opcions resultants i el senzill conjunt de regles utilitzades per l'optimitzador basat en regles no podria diferenciar les eleccions prou bé per a assegurar la millor elecció.

Una altra de les debilitats en l'optimitzador basat en regles és la resolució de les opcions d'optimització realitzades en el cas d'obtenir dos o més plans d'execució amb el mateix cost. En aquest cas, l'optimitzador mira la sintaxi de la sentència SQL per resoldre l'empat. La ruta d'execució guanyadora es basa en l'ordre en què les taules es presenten en la sentència SQL.

1.5.2. Optimització basada en costos

Aquest mètode selecciona la ruta d'execució que requereix el nombre més petit d'operacions lògiques d'E/S. Amb aquesta finalitat s'utilitzen estadístiques, que es consideren rellevants, recollides sobre la composició de les estructures de dades donades. D'aquesta manera, l'optimitzador basat en costos pot saber quina taula és més gran i la pot seleccionar com a la taula de la dreta per iniciar la consulta, independentment de la sintaxi de la sentència SQL. Així redueix el nombre d'iteracions i, si escau, el nombre d'accessos a disc.

Exemple de reducció del nombre d'iteracions

Es pot entendre l'impacte potencial sobre l'elecció de l'ordre en les taules si s'observa una situació senzilla, en què es fa una combinació d'una petita taula amb 10 registres, *SmallTable*, amb una taula amb 10.000 registres, *LargeTable*.

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

La sentència anterior obtindria un resultat equivalent al d'aquesta altra:

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

La mida de les taules i l'ordre en què aquestes es llegeixen té repercussió en el temps d'execució de la consulta:

- Si l'optimitzador tria *SmallTable* en primer lloc, l'SGBD llegeix les 10 files i després *LargeTable*, 10 vegades, per trobar les files coincidents de cadascuna de les 10 files.
- Si l'optimitzador tria *LargeTable* en primer lloc, l'SGBD ha de llegir les 10.000 files de *LargeTable* i després les de *SmallTable* 10.000 vegades per trobar els registres coincidents.

A més a més, en el primer cas, les files de *SmallTable* probablement es podrien emmagatzemar en memòria cau, amb la qual cosa es reduiria l'impacte de lectures de disc.

Independentment de l'ordre en què apareixen les taules en la sentència SQL, gràcies a la informació obtinguda de les estadístiques, l'optimitzador escollirà la taula més gran com si aquesta aparegués l'última en la sentència, és a dir, més a la dreta.

A partir de les estadístiques, se seleccionen els operadors físics i es determina l'estratègia d'execució.

Les estadístiques i els histogrames

Les **estadístiques** registren la distribució de les dades i les característiques d'emmagatzematge de les taules, columnes, índexs i particions a partir de les quals l'optimitzador estima la quantitat d'accessos a disc i memòria que cal per a executar un pla físic particular.

Vegeu també

Alguns dels paràmetres que emmagatzemen els estadístics es veuen en el subapartat 1.4.1 d'aquest mòdul didàctic.

Els paràmetres estadístics que els SGBD emmagatzemen més sovint són els següents:

1) Taula: nombre de files; nombre de pàgines; nombre de pàgines buides; llargada mitjana d'una fila.

2) Columna: nombre de valors diferents en la columna; nombre de valors NULL en la columna; histograma de freqüència d'aparició de cadascun dels valors.

3) Índex: nombre de pàgines; nivells; factor d'agrupament.

A l'SGBD comercial Oracle es pot emprar el paquet PL/SQL DBMS_STATS per a generar i gestionar les estadístiques de les taules, les columnes, els índexs, les particions i altres objectes de la base de dades.

Generar i gestionar estadístiques

Per exemple, podem recopilar estadístiques per a l'esquema *'COMPANY'* emprant la instrucció SQL següent, en què també s'especifica, en el segon paràmetre, que l'abast de la mida de la mostra el calculi automàticament.

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

Hi ha diverses opcions a l'hora de recopilar estadístiques que permeten especificar el **tipus de mostratge**:

Reflexió

També es pot fer que Oracle recopili les estadístiques quan es creen o es reconstrueixen índexs, especificant l'opció *COMPUTE STATISTICS* en les sentències *CREATE INDEX* o *ALTER INDEX*.

- **Mostratge basat en files**, de manera que s'ignora la ubicació física en el disc.
- **Mostratge basat en blocs**, de manera que s'escull una mostra aleatòria de blocs i les estadístiques es generen a partir de les dades emmagatzemades en aquests blocs.

Generalment, un mostratge utilitza menys recursos que si es calcula el valor exacte de l'estructura completa.

Les estadístiques s'emmagatzemen en el diccionari de dades.

Un **histograma** és una estructura de dades que es pot utilitzar per a millorar les estimacions que pot disposar l'optimitzador.

Un histograma recull un conjunt de valors juntament amb les freqüències relatives corresponents i proporciona a l'optimitzador les millors estimacions en presència d'una distribució uniforme. Hi ha dos **tipus d'histogrames**:

Reflexió

A Oracle l'usuari és qui crea i manté els histogrames per a les columnes adequades emprant el paquet PL/SQL DBMS_STATS.

- **Histograma equilibrat en amplada**, que divideix les dades en un nombre fix de rangs de la mateixa amplada i per a cadascun dels quals s'efectua el càlcul del nombre de valors que hi pertanyen.

- **Histograma equilibrat en altura**, en què es determinen els rangs un cop distribuïts equinumèricament els valors i genera rangs de diferent amplada, però que tenen un mateix valor de freqüència relativa.

Operadors físics i estratègies d'execució

El terme *operador físic* s'utilitza per a referir-se a un algorisme específic que implementi l'operació lògica de la base de dades. Per exemple, es pot escollir l'operador físic de combinació mitjançant un bucle imbricat indexat amb encarrilament[8].

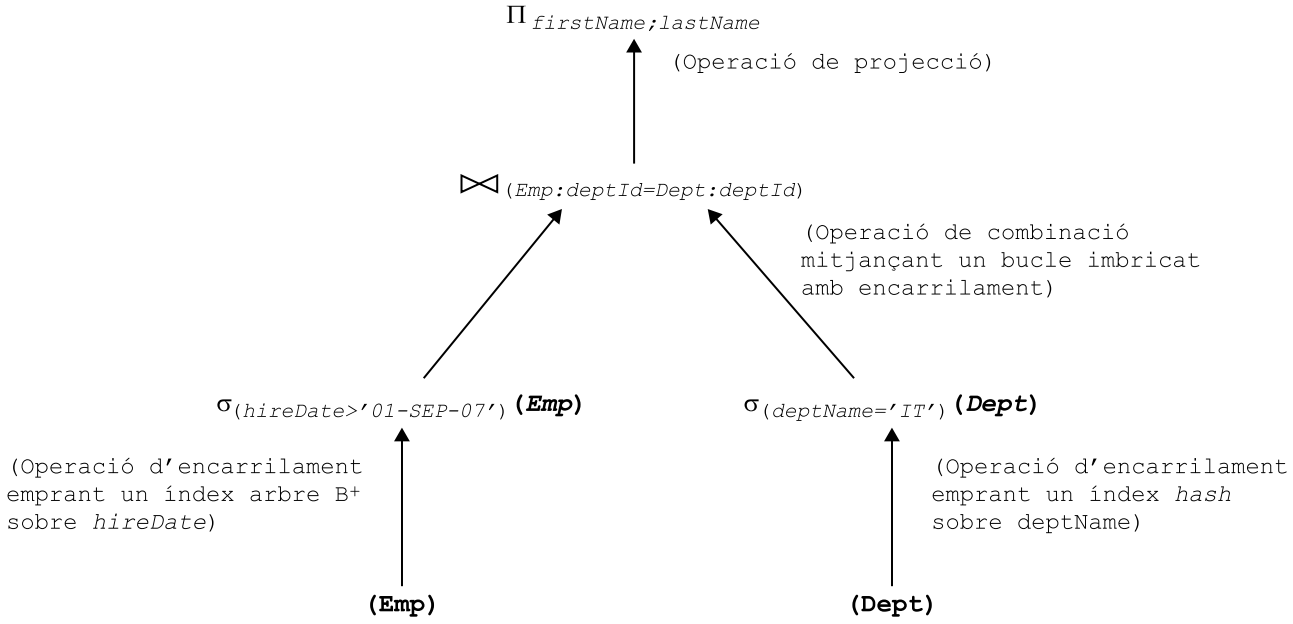
(8)En anglès, *pipelining*.

L'encarrilament

L'encarrilament (*pipelining*), com es veurà més endavant, és una tècnica que consisteix a aprofitar la sortida d'una operació com a entrada de la següent amb l'objectiu d'incrementar l'eficiència de les operacions, ja que estalvia el cost de lectura i escriptura de les relacions intermèdies.

Substituir les operacions lògiques en un arbre d'àlgebra relacional per operadors físics produeix una **estratègia d'execució**, **pla d'avaluació de la consulta** o **pla d'accés per la consulta**.

Considerem la consulta d'exemple del subapartat 1.1.1:



Encara que els diversos SGBD tenen implementacions diferents per als diferents operadors físics, podem considerar diferents operadors abstractes per implementar les funcions especificades en cada node de l'arbre.

1) ***tableScan*****(*R*)**: Exploració de la taula. Els diferents blocs de *R* es llegeixen en un ordre arbitrari.

2) ***sortScan*****(*R*, *L*)**: Exploració ordenada. Els tuples de *R* es llegeixen per ordre d'acord amb els criteris d'ordenació especificats a la llista d'atributs *L*.

3) ***indexScan*****(*R*, *P*)**: Exploració indexada. El predicat *P* especifica una comparació entre el valor de l'atribut *A* i un valor *c*. La manera d'accedir a les taules és mitjançant un índex sobre l'atribut *A*.

4) ***indexScan*****(*R*, *A*)**: Exploració indexada. *A* és un atribut de *R*. S'extreu la relació *R* completa emprant un índex sobre l'atribut *A*.

A més a més, l'estratègia d'execució suporta una interfície iteradora uniforme que permet ocultar els detalls d'implementació de cada operador.

1) ***open***. Inicia l'estat de l'iterador abans d'extreure el primer tuple i assigna un espai de memòria intermèdia per les entrades i sortides. Es poden definir diferents paràmetres que permetin definir condicions de selecció de manera que modifiquin el comportament de l'operador.

2) ***getNext***. Retorna el tuple següent del resultat i el col·loca en la memòria intermèdia de sortida. L'estat de l'iterador s'actualitza per a reflectir quantes d'aquestes accions s'han realitzat.

3) ***close***. En el moment en què s'ha finalitzat el procés d'iteració, és a dir, quan s'han generat totes les sortides sol·licitades, l'operador efectua les operacions de neteja i desassigna espai de memòria.

Quan defineix aquest tipus d'operadors, el sistema pot implementar de manera natural el concepte d'encarrilament o materialitzar els valors intermedis.

1) Materialització. Amb aquest enfocament s'inicia l'avaluació per a les operacions de nivell més baix a partir de l'arbre d'operadors. S'executen les operacions emprant els algorismes estudiats i posteriorment s'emmagatzemen els resultats en relacions temporals. Així doncs, el cost de tota l'expressió és la suma dels costos de cadascuna de les operacions més el cost dels resultats intermedis en el disc.

2) Encarrilament. Consisteix a aprofitar la sortida d'una operació com a entrada de la següent; d'aquesta manera ens estalviem el cost de lectura i escriptura de les relacions intermèdies. Per a cada operació de l'encarrilament es modelitza un procés aïllat[9], que pren un flux de tuples de les entrades i en produeix un altre per a la sortida. Per a això cal definir una memòria intermèdia per a l'entrada i una altra per a la sortida de cada operació. Amb l'encarrilament

[9] En anglès, *thread*.

també es pot aconseguir que diverses operacions s'executin paral·lelament, amb la qual cosa es guanya velocitat a canvi de necessitar més memòria per a dur a terme els diferents processos.

Visualització del pla d'execució

Oracle permet visualitzar el pla d'execució que escollirà l'optimitzador mitjançant la sentència *EXPLAIN PLAN*.

Aquesta sentència és força útil quan es volen veure les raons per les quals l'eficiència d'una consulta no és l'esperada. La sortida d'aquesta sentència s'escriu de manera predeterminada en la taula *PLAN_TABLE*.

Vegeu també

Podeu consultar la sintaxi i la funcionalitat de la sentència *EXPLAIN PLAN* a la documentació d'Oracle.

2. Processament de vistes

Des d'un punt de vista teòric, una vista es pot definir com el resultat dinàmic d'una o més operacions relacionals sobre taules per a produir una relació nova.

Les vistes són relacions virtuals que només estan representades pel nom i la definició; no existeixen físicament a la base de dades. Només serveixen a partir del moment en què l'usuari les fa servir.

La **sintaxi de creació de vistes** segons l'estàndard SQL:2011 és la següent:

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

L'estàndard SQL

Des que el llenguatge SQL va ser acceptat com un estàndard primerament per l'ANSI, el 1986, i posteriorment per l'ISO, el 1987, s'han anat afegint funcionalitat i característiques noves, que s'han recollit en les diferents versions, documentades en la referència ISO/IEC 9075.

La clàusula *WITH CHECK OPTION* assegura que mai no podrem actuar sobre parts d'una taula que no estan a la vista. Si s'inclou la clàusula *LOCAL*, es valida la integritat de la vista per a les operacions d'inserció o actualització que s'hi efectuïn, mentre que si s'utilitza la clàusula *CASCADED* també es valida la integritat de totes les altres vistes que en depenen.

Exemples de creació de vistes

Si es té en compte la base de dades *COMPANY* definida anteriorment, algunes definicions de vistes són les següents:

```
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
   ROM View01);
```

D'una manera opcional, es pot assignar nom a les diverses columnes de la vista. En cas contrari, les columnes prendran el mateix nom que tenen en la subconsulta. En cas que la subconsulta inclogui columnes calculades o funcions, serà imprescindible definir un nom per a aquestes columnes.

Fixeu-vos que, tal com es pot veure en el tercer exemple, en la subconsulta d'una vista pot aparèixer una altra vista.

D'altra banda, què passaria si es fes l'actualització següent?

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

Es donaria el cas inversemblant que es trauria una fila fora de la vista. Tindríem una situació semblant si es fes la inserció següent:

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

Si la vista s'hagués creat amb la clàusula *WITH CHECK OPTION*, les operacions s'haurien comprovat prèviament per a assegurar que les noves files inserides o actualitzades complien la condició de la vista.

Encara que una vista no ocupa gaire lloc a la base de dades, recordeu que només és una definició i que també es pot destruir.

La **sintaxi d'eliminació de vistes** segons l'estàndard SQL:2011 és la següent:

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

La clàusula opcional *CASCADE* destruirà tots els objectes creats a partir d'aquesta vista. En canvi, si es fa servir la clàusula *RESTRICT* impedirà l'eliminació de la vista en cas que hi hagi altres objectes que en depenen.

Exemple de destrucció d'una vista

Amb la sentència següent destruirem la primera de les vistes creada a l'exemple anterior:

```
DROP VIEW View01 CASCADE;
```

Quan s'executa aquesta sentència, a més d'eliminar la vista *View01*, també s'elimina la vista *View03*, ja que en la definició corresponent s'utilitza *View01*. En cas d'haver emprat la clàusula *RESTRICT*, no s'hauria eliminat cap de les dues vistes per aquesta dependència.

2.1. Mecanismes d'implementació de vistes

Hi ha dues estratègies fonamentals per a implementar les vistes: reescriure la consulta o materialitzar la vista.

L'**estratègia de reescriptura** és la més utilitzada per l'SGBD. Segons aquesta estratègia, la consulta es reescriu de manera que referenciï les taules de la base de dades.

Reescriptura o càlcul *inline*

L'estratègia d'implementació de vistes per reescriptura també es coneix amb el nom de *càlcul sota comandament* o, en anglès, *inline*.

Exemple d'implementació de vistes per reescriptura

Considereu la base de dades definida pel diagrama relacional de la base de dades *COMPANY* i les vistes següents.

```
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 com la que presentem a continuació:

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

es convertiria en el següent:

```
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 realment s'executa amb aquesta estratègia és molt més complexa que la que es planteja inicialment, i aquesta complexitat farà que les consultes requereixin molt temps d'execució. El súmmum de la ineficiència s'esdevé quan cal fer diferents consultes sobre una mateixa vista d'una manera seguida. També pot ser difícil la conversió en el cas de consultes molt complexes.

Amb la segona estratègia, anomenada **materialització de la vista**, quan es fa la consulta es crea una taula temporal amb el contingut físic de la vista.

Durada de les vistes

Si durant un període de temps la vista no es consulta, el mateix sistema la destrueix i la torna a construir de nou, si cal, en un altre moment.

Exemple d'implementació de vistes per materialització

Si s'apliqués l'estratègia de materialització de vistes en l'exemple anterior, es crearia la taula temporal *DeptSummaryView* amb les dades del resultat de la vista i les consultes es farien físicament sobre aquesta taula.

Segons aquesta estratègia, el cas de mínima eficiència que presentava la primera estratègia es transformaria en màxima eficiència, ja que només hauria de calcular la taula una sola vegada.

En contrast amb aquest avantatge, l'estratègia de materialització de la vista presenta un inconvenient: si les taules bàsiques a partir de les quals es construeix la vista varien de contingut, caldrà modificar la taula temporal (actualització incremental). Això pot resultar força costós, sobretot per a alguns casos de funcions agregades o en què apareguin clàusules *GROUP BY*.

Caldria esperar d'un bon SGBD que utilitzés la primera de les estratègies per a les consultes sobres vistes simples, i la segona per a les complexes.

2.2. Actualització de vistes

Les vistes sempre es poden consultar, però no sempre es poden actualitzar. Considerem, per exemple, una taula amb les dades dels empleats i una vista amb el nom del departament al qual estan assignats i el salari mitjà del departament. Quin significat tindria augmentar un 10% la mitjana dels salaris d'un departament? Aquest augment es pot aconseguir de moltes maneres, apujant i abaixant els salaris dels diversos membres del departament. Per tant, estem davant una ambigüitat.

Reflexió

Qualsevol operació d'actualització sobre una vista haurà de respectar les restriccions d'integritat definides en la taula original.

D'una manera més formal, sigui D una base de dades i V una vista que es construeix amb la funció $V = F(D)$. Si es fa una operació d'actualització A sobre la vista $A(V) = A(F(D))$, caldrà trobar una operació d'actualització A', de manera que $A(V) = F(A'(V))$. En molts casos hi haurà moltes operacions A' que compliran la igualtat anterior, encara que deixaran la base de dades en un estat diferent.

Només és possible assegurar que una vista serà actualitzable si està construïda sobre una única taula i els atributs de la vista contenen una clau candidata de la taula original. Les vistes definides sobre més d'una taula acostumen a no ser actualitzables. Les vistes que inclouen clàusules *GROUP BY* o funcions agregades mai són actualitzables.

El concepte *actualitzable* té una base semàntica, no sintàctica; es poden trobar definicions de vistes que *a priori* semblen no actualitzables, però sí que ho són.

Vegeu també

Podeu consultar a l'annex les consideracions referents a les vistes actualitzables en l'SGBD Oracle 11g.

Expressions actualitzables equivalents

La vista que presentem a continuació sembla no actualitzable:

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

Però sí que ho és, atès que és equivalent a aquesta altra:

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

I aquesta expressió sí que és actualitzable.

2.2.1. Actualització amb disparadors de substitució

Hi ha diversos mecanismes per a trencar les possibles ambigüitats en les accions d'actualització sobre vistes i es basen en el fet que s'emmagatzema el criteri d'actualització adequat conjuntament amb la definició de la vista. L'estàndard SQL:2008 incorpora l'ús dels disparadors de substitució[10].

[10] En anglès, *triggers instead of*.

Els **disparadors de substitució** són disparadors[11] que no s'executen ni abans ni després, sinó "en comptes de" l'ordre de modificació que es vol realitzar per mitjà de la vista. Aquesta classe de disparadors permeten efectuar accions d'inserció, eliminació i actualització sobre vistes que no són actualitzables. Només es poden definir sobre vistes.

[11] En anglès, *triggers*.

Exemple d'actualització de vistes amb disparadors de substitució

Considerem l'exemple clàssic d'implementació de vistes. Suposem que s'hi defineix la vista següent.

```
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
```

Com hem vist, aquesta vista és clarament no actualitzable. Ara bé, si considerem que la política de l'organització és efectuar increments de sou proporcionals, un disparador que fes actualitzable aquesta vista podria ser el següent (la sintaxi utilitzada és 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 aquest exemple es pot veure que, per a actualitzar el salari a un valor nou, el salari de cada empleat s'incrementa proporcionalment a l'augment que tindria la mitjana de salaris en tots ells. Si la política d'actualització de salaris de la nostra organització és proporcional, l'actualització funcionarà. De la mateixa manera, si es canvia el nom d'un departament, es modificarà en la taula corresponent.

Si la política de l'organització és que quan desapareix un departament tots els empleats que l'integren queden a l'espera de destinació –dit d'una altra manera, amb el camp departament a nul–, el disparador corresponent serà el següent:

```
CREATE TRIGGER deletingDeptAvgSalary
INSTEAD OF UPDATE 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;
```

L'ús d'aquesta classe de disparadors facilita el disseny intern i, també, l'encapsulació de la informació.

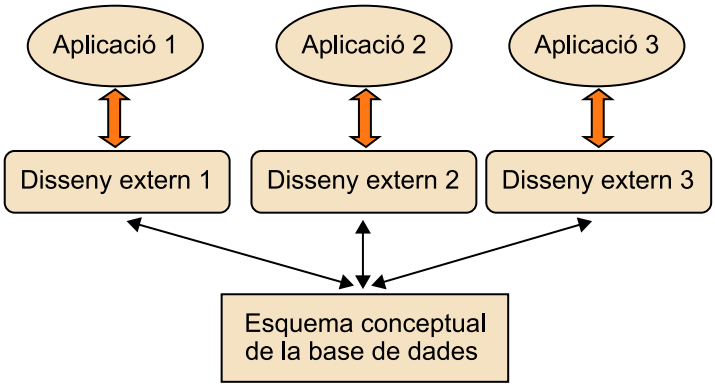
2.3. La vista com a element de disseny extern

Les bases de dades sempre s'haurien de dissenyar tenint en compte com és l'organització o el sistema, mai segons la utilització que es farà de les dades, si no és que cal per raons d'eficiència. Una base de dades corporativa serà utilitzada per un gran nombre d'aplicacions, i a totes els agradaria que la base de dades fos dissenyada a mida de les necessitats que tenen (si més no, als seus dissenyadors). La manera d'obtenir aquest objectiu doble rep el nom de *disseny extern*.

El **disseny extern** consisteix a crear una interfície entre la base de dades i l'aplicació.

En l'esquema següent es pot veure una representació gràfica d'aquesta situació.

Figura 2. Funció del disseny extern en una base de dades corporativa



Implementació del disseny extern

Per a implementar el disseny extern, es podran fer servir (segons les possibilitats de l'SGBD) vistes, consultes, vistes objecte, procediments, funcions, disparadors de substitució, mètodes, etc.

Una manera força senzilla i molt segura d'implementar el disseny extern és fer servir una col·lecció de vistes amb els disparadors de substitució respectius per a cadascuna de les aplicacions client; d'aquesta manera, la complexitat de l'esquema conceptual de la base de dades és transparent a les diferents aplicacions i aquestes es "fan la il·lusió" que tenen una base de dades dissenyada a mida, en la qual tota la informació es troba tal com els convé.

Exemple d'implementació de disseny extern amb vistes

Podem considerar que una aplicació podria tenir assignat en el seu esquema la vista *DeptSummaryView*, que hem vist en els exemples anteriors, sense cap tipus de disparador de substitució perquè només haurà de fer consultes.

Podem també tenir una altra aplicació que "veurà" la base de dades com si només hi hagués la taula *DeptAvgSalaryView*, sobre la qual pot esborrar i actualitzar gràcies als disparadors de substitució que hi ha implementats, però no podrà realitzar cap acció d'inserció.

Així s'assoleix la **independència lògica de les dades**, ja que, en cas que l'esquema conceptual sigui modificat, la visió que en tenen les diferents aplicacions no ha de veure-se'n necessàriament afectada.

En aquest apartat es fan servir la notació i les facilitats de l'SGBD comercial Oracle11g.

2.4. Les taules derivades

La major part dels SGBD incorporen unes estructures molt semblants a les vistes, les taules derivades, també conegudes com a *instantànies*[12], *agregats* o *vistes materialitzades*.

[(12)]En anglès, *snapshots*.

Exemple de taula derivada

En un supermercat es podria tenir una taula amb la suma de les vendes de cada secció. En aquesta taula no hi inseriríem cap fila, sinó que s'actualitzaria automàticament a mesura que s'anirien inserint files a la taula de vendes.

Les vistes materialitzades són taules que a més d'emmagatzemar la definició de la vista, també emmagatzemen les dades calculades (o derivades) de l'execució de la sentència *SELECT* definida a la vista. Aquesta taula es pot definir emprant els mateixos paràmetres d'emmagatzematge que en una taula normal, com, per exemple, l'espai de taules o la utilització d'índexs.

Comparació entre vistes i taules derivades

Les vistes són simples definicions que s'utilitzen quan cal (per a materialitzar o reescriure una vista). En canvi, les taules derivades existeixen físicament.

Quan una sentència SQL o PL/SQL accedeix a una vista materialitzada, l'SGBD es dirigeix directament a les dades de la vista que estan emmagatzemades en lloc d'utilitzar les dades de les diferents taules emprades en la definició de la vista.

És convenient emprar vistes materialitzades quan la definició de la vista fa referència a moltes taules enllaçades de manera complexa, així com en les vistes que s'utilitzen amb molta freqüència, ja que permet millorar-ne el rendiment atès que la sentència SQL s'executarà un sol cop.

D'altra banda, cal considerar si la vista materialitzada es reutilitzarà en el futur. Això implicarà actualitzar-ne el contingut, ja que probablement el contingut de les taules base haurà estat modificat.

A l'hora de determinar si cal definir una vista com a materialitzada o no, s'han de valorar els costos d'executar la sentència SQL que defineix la vista respecte dels costos d'emmagatzematge i actualització de la vista materialitzada.

Per a **definir una vista materialitzada**, el patró de sentència SQL és el següent:

```
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 clàusula *BUILD* permet triar si en el moment de crear la vista, a banda de la creació de la taula que emmagatzemarà els resultats, aquesta s'informa o no. En el primer cas cal fer servir *IMMEDIATE*; en el segon, *DEFERRED*, en què la taula no s'omplirà de dades fins que no s'efectuï la primera actualització de la vista materialitzada.

La clàusula *REFRESH* permet indicar el mecanisme que utilitzarà l'SGBD per a actualitzar la vista materialitzada. El mètode d'actualització que s'esculli dependrà de la freqüència d'actualització de les taules base. Aquest mètode pot ser:

- *COMPLETE* ('complet'). Es recalcula tota la taula derivada segons la consulta que la defineix.

- *FAST* ('ràpid'). S'especifica un mètode de refresc incremental; els canvis s'efectuaran afegint les noves dades que s'han incorporat a les taules base.

- *FORCE* ('forçat'). Aplica, si és possible, el refresc ràpid; en cas contrari, aplica el refresc complet.

- *NEVER* ('mai'). Indica que la vista materialitzada mai no serà refrescada.

Reflexió

Cal tenir en compte que les vistes que fan servir funcions d'agregació *SUM*, *AVG*, *MAX*, *MIN* o *COUNT* no admeten l'actualització *FAST*.

L'actualització es pot produir en dos moments diferents:

1) Actualització manual. Cal definir la vista amb la restricció *ON DEMAND*. El refresc es produeix quan l'usuari executa manualment algun procés d'actualització sobre la vista materialitzada. Les actualitzacions manuals de les vistes materialitzades es duen a terme emprant el paquet PL/SQL estàndard DBMS_MVIEW.

Reflexió

Tots aquests procediments i funcions admeten paràmetres addicionals que es poden consultar a la documentació d'Oracle 11g.

El paquet PL/SQL DBMS_MVIEW

Aquest paquet inclou un conjunt de funcions i procediments que permeten gestionar les vistes materialitzades. Podem destacar els següents:

- *DBMS_MVIEW.REFRESH*('MaterializedViewName'). Actualitza una vista materialitzada a partir del seu nom.

- *DBMS_MVIEW.REFRESH_DEPENDENT*('Table1, Table2, ...'). Actualitza totes les vistes materialitzades que utilitzin com a taula base alguna de les taules o vistes indicades a la llista.

- *DBMS_MVIEW.REFRESH_ALL_MVIEWS*(*n*). Actualitza totes les vistes materialitzades retornant un nombre (*n*) que indica el nombre de registres que s'han actualitzat.

2) Actualització automàtica. Aquesta forma d'actualització pot ser:

- *ON COMMIT*. El refresc es produeix quan la transacció que modifica alguna de les taules base es confirma. Això significa que l'execució del *COMMIT* tindrà un cost temporal més gran, que pot afectar el rendiment.

- Actualització programada. L'actualització es programa perquè succeeixi en un instant determinat.

Exemple d'actualització programada

Una vista es pot programar perquè s'actualitzi de manera automàtica en determinats moments mitjançant l'ús de les clàusules *START WITH* i *NEXT*. La clàusula *START WITH* indica la data de la primera actualització automàtica, i la clàusula *NEXT* permet indicar l'interval entre dues actualitzacions automàtiques.

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

Actualització amb taules derivades

Considerem el següent model relacional de la base de dades operativa d'un supermercat.

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 taula derivada corresponent als ingressos realitzats en les comandes per línia de producte es podria crear amb la sentència següent:

```
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 clàusula *ENABLE/DISABLE QUERY REWRITE* serveix per a determinar si l'optimizador de consultes pot reescriure o no les sentències SQL de manera que, si és possible, s'utilitzi la vista materialitzada en lloc de les taules base emprant el mecanisme de reescriptura. Aquesta opció tan sols està disponible quan s'utilitza l'optimitzador basat en costos, ja que l'SGBD empra les estadístiques per a determinar si l'execució de la sentència SQL o la reescriptura d'aquesta usant la vista materialitzada és la que té un cost més baix.

Mecanismes de reescriptura de consultes

Seguint amb la base de dades del supermercat de l'exemple anterior, imaginem la consulta següent.

```
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 podria aprofitar la vista materialitzada i reescriuria la consulta d'aquesta manera:

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

Les vistes materialitzades sempre es calculen a partir de les taules bàsiques i l'única operació que s'hi pot fer és la consulta; no té sentit parlar d'actualització, esborrament o inserció. Com que s'acostumen a fer servir per a emmagatzemar i precalcular dades agregades, sovint són anomenades *resums*.

Taules derivades en entorns distribuïts

En entorns distribuïts, les taules derivades es poden utilitzar per a replicar dades a les diferents seus, sincronitzant les actualitzacions sense crear cap conflicte.

2.5. Taules temporals

Un altre tipus de taules especials són les taules temporals o, més exactament, *taules de contingut temporal*.

En les taules temporals, el contingut té vigència en el transcurs d'una sessió (o una transacció) i desapareix en el moment en què aquesta finalitza.

La **sentència SQL per a la creació d'una taula temporal** és la següent:

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

Aquestes taules són molt útils per a emmagatzemar resultats intermedis d'una transacció.

Exemple d'utilització d'una taula temporal

Considerem la creació de la taula següent referent a les qualificacions que han obtingut els alumnes en una assignatura.

```
CREATE LOCAL TEMPORARY TABLE SubjectMarks (
   student VARCHAR2(50),
   mark NUMBER(3,1))

ON COMMIT DELETE ROWS;
```

Quan es faci la primera inserció, la taula es materialitzarà i serà possible treballar-hi fins al moment en què s'executi la clàusula *COMMIT*; aleshores, la taula es destruirà i només en restarà la definició en el diccionari de dades.

2.6. Avantatges i inconvenients en la utilització de vistes

Com en qualsevol decisió de disseny, la utilització de vistes presenta una sèrie d'avantatges i inconvenients. Els avantatges més importants són els següents:

a) Independència de les dades i les aplicacions. Si es fan servir vistes com a interfície, es pot arribar a una independència completa entre l'estructura real de les dades i la que veuen les aplicacions. En qualsevol moment es pot canviar el nom d'una taula, afegir-hi noves columnes o canviar-ne completament l'estructura. Tots aquests canvis resulten invisibles per a les aplicacions clients. Només caldria redefinir la interfície, és a dir, les vistes.

b) Simplificació de l'ús per a l'usuari. La utilització de les vistes permet tenir emmagatzemades consultes força complexes amb múltiples combinacions de taules, condicions i funcions agregades, i fer-ne ús amb una consulta molt simple.

c) Millora de la seguretat. L'usuari no coneix les taules i les columnes que formen realment la base de dades, de manera que no pot ni tan sols intentar accedir-hi; només té noticia de la informació i l'estructura que se li dóna a partir de les vistes.

d) Integritat de les dades. Amb la clàusula *WITH CHECK OPTION*, l'usuari no pot portar dades fora dels límits que té marcats; tots els canvis que faci sobre la vista (si és actualitzable) hauran de deixar la fila consistent amb les condicions de la vista.

e) Rendiment. Si és possible determinar la classe de consultes que es faran a partir de la vista, també ho és determinar camins d'accés que millorin l'eficiència de la consulta.

Però no tot són avantatges; els inconvenients més importants són els següents:

a) Restriccions d'actualització. No totes les vistes són actualitzables (en realitat, la majoria no ho són); no és possible fer un disseny extern només amb vistes (llevat que es facin servir disparadors de substitució).

b) Restriccions d'estructura. Alguns SGBD, seguint una norma ISO, no permeten construir una vista a partir de qualsevol consulta: una vista creada a partir d'una sentència *SELECT* no tindrà en compte les columnes noves que s'hagin afegit a la taula original.

3. La seguretat

En un sistema d'informació, generalment, les diferents aplicacions i usuaris de l'organització fan servir un únic conjunt de dades (base de dades corporativa) amb l'SGBD. D'una banda, això resol problemes de redundància, inconsistència i independència entre les dades i els programes i, de l'altra, fa que la seguretat esdevingui un dels problemes més importants en aquests entorns.

La paraula **seguretat** incorpora diferents conceptes. Els més importants són els següents:

a) Confidencialitat. Cal protegir l'ús de la informació per persones no autoritzades. Això implica que un usuari només ha de poder llegir la informació per a la qual té autorització i que no podrà inferir informació secreta a partir de la informació a la qual té accés.

b) Integritat. La informació s'ha de protegir de modificacions no autoritzades; això inclou tant la inserció de dades falses com la destrucció de dades.

c) Disponibilitat. La informació ha d'estar disponible en el moment que faci falta a l'usuari.

Problemes de disponibilitat

Els problemes de disponibilitat inclouen la seguretat física (per problemes de maquinari), la no-saturació del sistema, la denegació de servei (DoS) per la xarxa, etc.

Les **violacions de la base de dades** consisteixen en lectures o modificacions incorrectes de les dades. Per *modificació* entenem les altes i baixes d'informació i les modificacions de la informació existent. Les conseqüències d'aquestes violacions es poden agrupar en tres categories:

a) Alliberament incorrecte de la informació. Causat per la lectura de dades per usuaris impropis mitjançant un accés intencionat o accidental. En aquesta categoria s'inclouen les violacions del secret derivades de les deduccions d'informació secreta a partir de lectures d'informació autoritzada.

b) Modificació impròpia de les dades. Correspon a totes les violacions de la integritat de les dades per tractaments o modificacions fraudulentes d'aquestes dades. Les modificacions impròpies no impliquen necessàriament lectures no autoritzades, ja que les dades es poden falsificar sense ser llegides.

c) Denegació de serveis. Correspon a accions que puguin impedir que els usuaris accedeixin a les dades o utilitzin uns recursos determinats.

Les **amenaces a la seguretat** es poden classificar d'acord amb la manera en què es produeixen.

1) Amenaces no fraudulentes. Són accidents casuals, entre els quals es poden distingir els següents:

a) Desastres accidentals o naturals, com ara terratrèmols, inundacions o foc, que originen accidents que danyen el maquinari del sistema.

b) Errors en el maquinari o en el programari que poden desencadenar accessos no autoritzats.

c) Errors humans causats d'una manera no intencionada quan s'introdueixen dades o s'utilitzen aplicacions.

2) Violacions intencionades o violacions fraudulentes. Són causades per dos tipus d'usuaris diferents:

a) Usuaris autoritzats que abusen dels privilegis que tenen.

b) Agents hostils, usuaris impropis (interns o externs) que executen accions de vandalisme sobre el programari i el maquinari del sistema, o lectures o escriptures de dades; en tots dos casos, els usos legals de les dades i les aplicacions poden emmascarar el propòsit fraudulent real.

A continuació es detallen els principals **mecanismes de seguretat**.

1) Identificació i autenticació. Són mecanismes que identifiquen l'usuari i s'asseguren que és qui diu que és.

2) Control d'accés. Aquests mecanismes s'asseguren que els usuaris accedeixen només als llocs als quals estan autoritzats amb l'objectiu de fer accions per a les quals tenen autorització.

Vegeu també

El control d'accés a l'SGBD comercial Oracle 11g es pot consultar en el subapartat 4.3 d'aquest mòdul didàctic.

3) Integritat i consistència. Són mecanismes perquè la base de dades resti sempre en un estat que compleixi totes les regles del negoci del model de dades, encara que es produeixin canvis.

4) Auditoria. Aquests mecanismes permeten saber qui ha fet què, és a dir, portar un registre de qui fa tots els canvis i consultes a la base de dades. Més que un mecanisme per a proporcionar seguretat, es tracta d'un mecanisme que permet monitorar els usuaris del sistema.

3.1. Identificació i autenticació

Els sistemes d'informació i les dades que emmagatzemen i processen són recursos molt valuosos que cal protegir. Un dels primers passos cap a la seguretat en un sistema d'informació és la capacitat de verificar la identitat dels usuaris. Aquest procés es compon de dues parts: identificació (veure qui és) i autenticació (comprovar que és qui diu que és).

La **identificació** implica la manera en què proporciona un usuari la seva identitat al sistema.

La identitat ha de ser única perquè el sistema la pugui diferenciar entre els diversos usuaris. Segons els requisits operacionals, una identitat pot descriure un individu, més d'un individu, o un o més individus només una part del temps.

L'**autenticació** és el procés d'associar un individu amb la seva identitat única. És a dir, és el procés que verifica que un usuari és qui diu que és.

Àmbit de la identitat

Una diferència important entre la identificació i l'autenticació és que les identitats són públiques, mentre que la informació d'autenticació es guarda en secret. Això proporciona el recurs pel qual una persona prova que és realment qui diu que és.

Hi ha tres **recursos d'autenticació bàsics** per a poder demostrar qui és realment un individu:

1) Una cosa que una persona coneix: una contrasenya, un número d'identificació personal, etc.

2) Una cosa que una persona posseeix: una targeta, una clau, etc.

3) Una cosa que caracteritza una persona: l'empremta dactilar, la veu, etc.

Aquests mètodes bàsics es poden utilitzar individualment o es poden combinar per a obtenir un nivell de seguretat més alt.

Les **contrasenyes** són el mecanisme més clàssic d'autenticació. Les contrasenyes són paraules (o, més ben dit, combinacions de caràcters) que només coneix un usuari. La seguretat d'un esquema de contrasenyes depèn de la capacitat de mantenir-les en secret. Una contrasenya s'ha de triar de manera que sigui fàcil de recordar i difícil d'endevinar.

Criteris per a la selecció de contrasenyes

A continuació presentem uns quants criteris que convé tenir en compte a l'hora de triar una contrasenya.

1) Seleccionar contrasenyes llargues: vuit caràcters és una mida adequada.

2) Combinar diferents tipus de caràcters: majúscules, minúscules, nombres, espais en blanc i caràcters de puntuació.

3) No fer servir paraules amb significat.

4) Utilitzar contrasenyes diferents per a accedir a sistemes diferents.

5) Canviar la contrasenya d'una manera periòdica.

6) No escriure la contrasenya en llocs on una altra persona pugui accedir.

7) Que no sigui la mateixa que l'identificador d'usuari.

8) Quan es canvia, que difereixi de l'anterior en tres caràcters, com a mínim.

9) Canviar la contrasenya la primera vegada que s'inicïi una sessió.

Les **targetes** donen una seguretat més gran. Poden ser simples trossos de plàstic amb una banda magnètica o, fins i tot, poden incorporar xips, com fan les targetes intel·ligents. En tots dos casos, una contrasenya personal ha de coincidir amb la que hi ha escrita a la targeta, o bé la contrasenya més alguna informació que consta a la targeta han de coincidir amb la que hi ha a l'ordinador.

La tendència actual és anar cap a **sistemes biomètrics**, és a dir, mètodes automatitzats de reconeixement d'una persona que es basen en una característica fisiològica o de comportament. Aquests sistemes es poden fer servir com a mètode d'identificació, en què s'identifica una persona dins una població mitjançant una característica seva registrada en una base de dades, i se'n cerca la coincidència. També es poden utilitzar a manera de verificació: el sistema autentica la identitat reclamada d'una persona amb el seu patró enregistrat prèviament.

Per a la **validació d'un usuari** d'un sistema gestor de bases de dades es poden fer servir quatre mètodes.

El sistema d'autenticació de la xarxa Kerberos

En sistemes en què es requereix certa seguretat és usual la utilització de sistemes externs (màquines diferents d'aquella a la qual ens volem connectar) com a serveis d'autenticació. El més conegut és Kerberos, el qual, utilitzant diferents tècniques de xifratge i emissió de tiquets (*tickets*), proporciona un bon nivell de seguretat per a la majoria de sistemes.

1) Autenticació pel mateix SGBD. És la més utilitzada, ja que els comptes són més fàcils de controlar i gestionar, i el mateix SGBD té recursos que permeten l'administració de petites comunitats d'usuaris.

2) Autenticació pel sistema operatiu. És una forma de validació externa. Només és possible en els sistemes que permetin la validació d'usuaris (UNIX, Windows NT...).

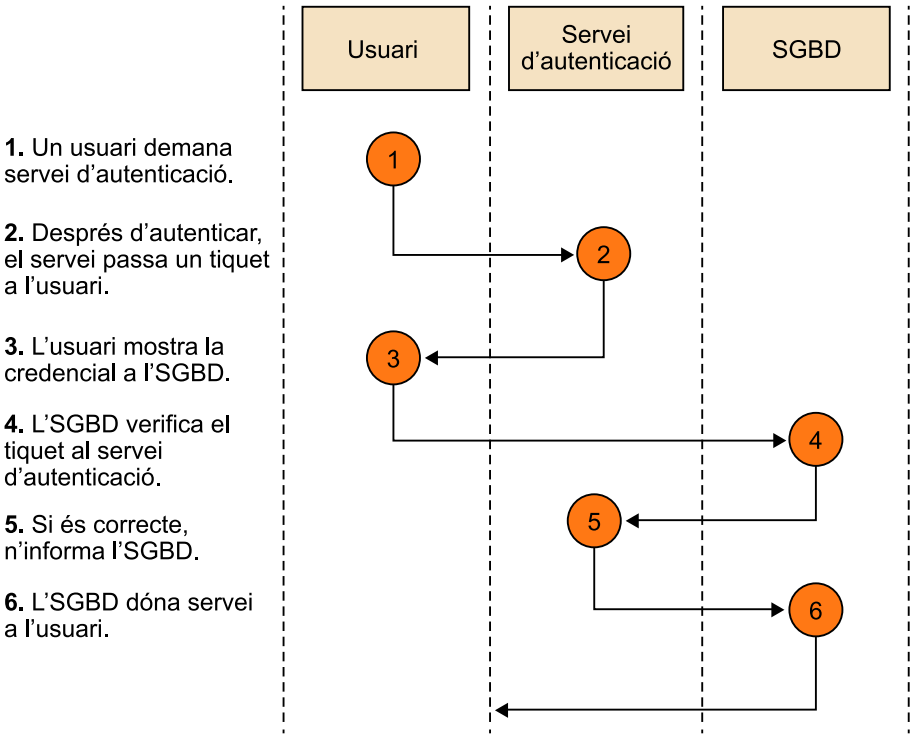
3) Autenticació pel servei de xarxa. Fa servir productes especialitzats de xarxa, com Kerberos, CyberSafe, Identix, Radius o altres.

(13)En anglès, *middleware*.

4) Autenticació per una capa intermèdia. En un sistema client/servidor de tres nivells, la capa intermèdia podria ser el programari intermediari[13] o l'aplicació mateix.

En tots els casos, la comunicació pot ser xifrada.

Figura 3. El sistema d'autenticació de la xarxa Kerberos



La figura mostra el sistema d'autenticació per la xarxa Kerberos, que fa servir una màquina específica per a dur a terme l'autenticació i, d'aquesta manera, allibera l'SGBD d'aquesta tasca.

3.2. Control d'accés

S'anomena **control d'accés** la part de l'SGBD que té la funció d'assegurar que els accessos al sistema concorden amb els models i les regles fixades per la política de protecció.

El control d'accés controla la interacció (lectura, escriptura...) entre els subjectes (usuaris, aplicacions, processos...) i els objectes als quals accedeixen.

El control d'accés es pot considerar que està format per dos components:

1) Polítiques d'accés. Defineixen els principis segons els quals s'autoritza un usuari, es denega o es filtra l'accés específic a un objecte.

Filtratge

El filtratge retorna només una part de la informació demanada.

2) Mecanismes de seguretat. Són procediments que s'apliquen a les consultes dels usuaris perquè compleixin les normes anteriors.

Les diferents implementacions de les polítiques d'accés es poden classificar en *control d'accés discrecional* i *control d'accés obligatori*.

3.2.1. Control d'accés discrecional

El control d'accés discrecional (DAC[14]) es basa en la identitat dels usuaris o grups d'usuaris per a restringir l'accés a objectes. El control discrecional és el mecanisme de control més comú que s'implementa en els sistemes d'informació actuals.

(14)DAC és la sigla de *discretionary access controls*.

Ús del DAC

El control d'accés discrecional és el més utilitzat en els SGBD comercials.

Les **polítiques discrecionals** es fonamenten en el coneixement dels drets que cada usuari té sobre cada objecte. Aquestes polítiques podran ser definides per l'administrador de la base de dades o pel propietari de l'objecte.

Els drets de l'usuari

Els drets de l'usuari poden incloure l'administració de la seguretat de l'objecte.

La millor manera de representar l'estructura del control d'accés discrecional és la matricial:

	Objecte 1	**Objecte 2**	**...**	**Objecte *n***
Usuari 1			...	
Usuari 2			...	Drets de l'usuari 2 sobre l'objecte *n*
...	...	...	...	
Usuari *n*			...	

La intersecció entre files i columnes indica els drets de cada usuari sobre cada objecte.

Una ampliació usual de la taula anterior consisteix a incorporar-hi els grups d'usuaris com si es tractés d'un altre usuari i els privilegis sobre el sistema com si es tractés d'un objecte. La descentralització de les autoritzacions fa que apareguin problemes de propagació tant d'autoritzacions com de revocacions. Si un usuari *M* ha rebut una autorització d'un usuari *N* que tenia el dret d'administrar la seguretat sobre un objecte *P*, quins drets tindrà l'usuari *M* si a *N* se li revoquen tots els permisos? El sistema haurà de preveure polítiques d'autorització i revocació en cadena.

3.2.2. Control d'accés obligatori

El control d'accés obligatori (MAC[15]) s'acostuma a fer servir en les bases de dades en què les dades tenen una estructura de classificació molt rígida i estàtica, com, per exemple, les bases de dades militars i governamentals.

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

Ús del MAC

El control d'accés obligatori s'acostuma a utilitzar en els SGBD que treballen amb informació sensible (d'alta seguretat).

Les **polítiques de control d'accés obligatori** es basen en la idea que cada dada té un nivell de classificació (per exemple, molt secret, secret, confidencial...) i cada usuari té un nivell d'acreditació (amb les mateixes possibilitats que el nivell de classificació). Els diferents nivells estan ordenats de manera estricta i incremental: molt secret > secret > confidencial...

Les normes de funcionament del control d'accés obligatori són les següents:

- Un usuari pot veure un objecte solament si té un nivell d'acreditació més gran o igual que el nivell de classificació de l'objecte.

- Un usuari pot modificar un objecte només si té un nivell d'acreditació igual que el nivell de classificació de l'objecte.

Els usuaris, primerament, hauran de tenir l'accés discrecional autoritzat i els privilegis adequats sobre l'objecte de la base de dades abans que es comprovi el sistema d'accés obligatori. El sistema de seguretat MAC es basa en el concepte d'*etiqueta*.

Una **etiqueta** indica el nivell de l'usuari (acreditació) o de l'objecte (classificació). Els valors baixos designen informació no classificada o menys sensible; els valors alts denoten informació més restrictiva. Les etiquetes es poden refinar afegint-hi altres subnivells.

Exemple d'utilització d'etiquetes

A l'exèrcit es podria tenir una classificació com la que es presenta en la taula següent, en què, a cada etiqueta, hi correspon un valor.

El sistema gestor de bases de dades comprova, en primer lloc, totes les autoritzacions d'accés discrecional; en cas que l'usuari compleixi tots els nivells de seguretat, el sistema d'accés obligatori compara les etiquetes de l'usuari i de l'objecte per tal de decidir qui hi té accés:

Nom	Valor	Descripció
General	80	Alt secret, personal
Oficial	60	Alta seguretat, no distribuir
Suboficial	40	Moderadament segur
Militar	20	Nivell bàsic, no massa sensible

Nom	Valor	Descripció
Civil	1	Coneixement públic, distribució lliure

3.2.3. Classificació dels sistemes de seguretat

Els sistemes de seguretat es classifiquen en quatre nivells, que, de menys a més protecció, són els següents:

Els sistemes de seguretat habituals

Per bé que alguns productes comercials proporcionen un nivell de seguretat B1, el més normal és que només arribin al nivell C2.

1) Protecció mínima. La classe D no incorpora cap mecanisme de seguretat.

2) Protecció discrecional. La classe C suporta el control d'accés discrecional, té les subclasses C1 i C2. La classe C1 requereix separació entre les dades i els usuaris. La classe C2 requereix, a més, processos de registre, auditoria i aïllament de recursos.

3) Protecció estructurada. La classe B suporta el control d'accés obligatori i té les subclasses B1, B2 i B3. La classe B1 requereix protecció de seguretat etiquetada: tots el objectes estan etiquetats amb un nivell de seguretat. La classe B2 requereix, a més, una sentència formal per a cada cosa i, també, que els canals de cobertura siguin identificats i eliminats. La classe B3 requereix, també, suports de recuperació i auditoria.

Exemple de cobertura

Un exemple de cobertura podria ser inferir la resposta d'una consulta il·legal a partir d'una consulta legal.

4) Protecció verificada. La classe A, la més segura, requereix una demostració matemàtica que els mecanismes de seguretat són consistents i adequats per a suportar la política de seguretat especificada.

La **classificació dels sistemes de seguretat** que s'ha presentat aquí va ser creada pel Pentàgon com a sistema de classificació estàndard. Aquesta classificació es troba definida en l'*Orange Book* ('llibre taronja'), en el qual es defineixen els requisits de seguretat per a qualsevol TCB[16], i en el *Lavender Book*, en què es defineix la interpretació dels requisits per a un sistema gestor de bases de dades específic.

(16) *TCB* és la sigla de l'expressió anglesa *trusted computing base*.

3.3. Implementació del control d'accés discrecional a SQL:2011

L'estàndard actual defineix la sintaxi d'autoritzacions següent:

```
<sentència d'autorització> ::=
   <sentència autorització privilegis>
   | <sentència autorització rols>
<sentència autorització privilegis> ::=

GRANT <privilegis> TO <autoritzat> [{ , <autoritzat>}]
   [WITH HIERARCHY OPTION]
   [WITH GRANT OPTION]
```

```
   [GRANTED BY <autoritzador>]
<privilegis> ::=
   <privilegis d'objecte> ON <nom objecte>
<privilegis d'objecte> ::=
   ALL PRIVILEGES|<acció>[{ , <acció>}]
<acció> ::=
   DELETE |
   SELECT [(<nom columna> [ , <nom columna>] ... ])]|
   SELECT [(<nom rut ina> [ , <nom rutina>] ... ])]|
   INSERT [(<nom columna> [ , <nom columna>] ... ])]|
   UPDATE [(<nom columna> [ , <nom columna>] ... ])]|
   REFERENCES [(<nom columna> [ , <nom columna> ... ])]|
   USAGE |
   UNDER |
   TRIGGER |
   EXECUTE

<nom objecte> ::=
   [ TABLE ] <nom taula>
   |DOMAIN <nom domini>
   | COLLATION <nom compaginador de caràcters>
   | CHARACTER SET <nom joc de caràcters>
   | TRANSLATION <nom transcripció>
   | TYPE <nom tipus>
   | SEQUENCE <nom generador de seqüències>
   | <designador específic de rutina>
   | MODULE <nom mòdul>

<autoritzat> ::=
   PUBLIC | <nom autoritzat>

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

Els significats dels privilegis que es poden donar a un usuari sobre un objecte són aquests:

Clàusules d'autorització de privilegis

L'opció *WITH GRANT OPTION* permet transmetre els privilegis que es tenen a un altre. La clàusula *WITH HIERARCHY OPTION* permet traslladar els permisos a totes les subtaules.

Exemples d'autoritzacions

A continuació presentem un exemple d'implementació de les autoritzacions que s'han tractat en aquest subapartat.

```
GRANT SELECT ON TABLE Empleat TO Auditor2;
GRANT USAGE ON DOMAIN d_Telefon TO PUBLIC;
```

Privilegis	Definició
INSERT [(llista-noms-columnes)]	Inserir valors en les columnes relacionades d'una fila.
UPDATE [(llista-noms-columnes)]	Actualitzar valors en les columnes relacionades d'una fila.

Privilegis	Definició
DELETE	Esborrar files.
SELECT [(llista-noms-columnes)]	Consultar valors en les columnes relacionades.
REFERENCES [(llista-noms-columnes)]	Fer referència a valors de les columnes relacionades.
TRIGGER	Crear un disparador sobre una taula.
EXECUTE	Executar un procediment incorporat.

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

És relativament freqüent associar els mateixos privilegis a diferents usuaris, per exemple, a tot el departament de comptabilitat. Per a facilitar aquesta classe d'autorització, es fa servir el rol.

Un **rol** es pot definir com un conjunt de zero o més privilegis. Si s'autoritza un rol a un usuari, automàticament se li autoritzen tots els privilegis que incorpora el rol.

La **sintaxi del rol** és la següent:

```
<definició rol> ::=
   CREATE ROLE <nom rol>
   [WITH ADMIN <autoritzador>]

<sentència autorització rol> ::=
   GRANT <nom rol> [ { , <nom rol>}]
   TO <autoritzat> [ { , <autoritzat>}]
   [WITH GRANT OPTION]
   [GRANTED BY <autoritzador>]
```

La clàusula *WITH ADMIN* permet indicar qui administrarà el rol.

Exemples de creació i utilització de rols

Considereu els exemples de creació i utilització de rols següents.

```
CREATE ROLE Auditors WITH ADMIN CURRENT_USER;
GRANT SELECT, UPDATE (Salari, Comissio) TO Auditors;
GRANT Auditors TO Pere, Joan, Maria;
```

De la mateixa manera que es donen autoritzacions, es poden revocar.

La **sintaxi per a revocar rols** és la següent:

```
<sentència revocar privilegis> ::=
   REVOKE [{GRANT OPTION FOR|HIERARCHY OPTION FOR}]
```

```
      <privilegis> FROM <autoritzat> [{ , <autoritzat>} ... ]
   [ FROM {CURRENT_USER|CURRENT_ROLE}]
   [ GRANTED BY <autoritzador>]
   { RESTRICT|CASCADE}

<sentència revocar rol> ::=
   REVOKE [ADMIN OPTION FOR] <nom rol> [{ , <nom rol> } ... ]
   FROM <autoritzat> [ { , <autoritzat >} ... ]
   [FROM {CURRENT_USER|CURRENT_ROLE}]
   [GRANTED BY <autoritzador>]
   {RESTRICT|CASCADE}
```

Problemes de propagació de privilegis

Les opcions *RESTRICT* i *CASCADE* tenen una importància especial per a resoldre els problemes de propagació. Quan A autoritza un privilegi a B (amb l'opció d'administrar-lo) i B l'autoritza a C, si a A se li prenen els privilegis que tenia, B i C restaran amb els privilegis "abandonats". Si l'opció triada quan es revoquen els privilegis a A és *CASCADE*, els permisos de B i C també restaran revocats. Si l'opció és *RESTRICT*, només es podran revocar els permisos d'A, si prèviament s'han eliminat els permisos candidats a ser abandonats. Si no s'utilitza cap de les opcions, B i C restaran amb els permisos, i es correrà el risc que B torni a donar privilegis a A.

3.4. Auditoria

L'auditoria és el registre i monitoratge d'algunes accions específiques d'usuaris sobre la base de dades.

La càrrega del treball d'auditoria

El treball d'auditoria pot representar una sobrecàrrega superior a la del nivell del treball normal.

L'auditoria s'utilitza normalment per als casos següents:

1) La investigació d'una activitat sospitosa.

Exemple d'investigació d'una activitat sospitosa

Si un usuari no autoritzat intenta esborrar dades de les taules, l'administrador de seguretat podrà decidir d'auditar totes les connexions de la base de dades i tots els intents (amb èxit o no) d'esborrar dades.

2) El monitoratge i la recollida d'activitats específiques de la base de dades.

Exemple de monitoratge de la base de dades

L'administrador de la base de dades pot recollir dades sobre les taules que s'actualitzen o el nombre d'usuaris que estan connectats en moments punta.

El sistema d'auditoria ha de permetre diverses formes d'utilització:

- Auditar sentències. L'auditoria[17] indicarà quan i qui ha utilitzat un tipus de sentència concreta. Per exemple, auditar totes les insercions o els esborraments.

[17] Auditar significa esbrinar qui és l'autor de qualsevol canvi a la base de dades.

- Auditar objectes. El sistema registrarà cada vegada que es realitzi alguna operació sobre un objecte determinat.

- Auditar sentències sobre objectes, una versió combinada de les dues anteriors.

- Auditar usuaris o grups.

També es pot decidir si es vol fer un únic registre per sessió, per sentència o per accés; i si només es vol registrar quan té èxit, fracassa o en tots dos casos. La informació que acostuma a registrar l'auditoria és el nom de l'usuari, l'identificador de sessió, l'identificador de terminal, el nom de l'objecte al qual s'ha accedit, l'operació executada o intentada, el codi complet de l'operació, la data i l'hora.

La creació d'auditories

Alguns SGBD incorporen sentències SQL que permeten generar una auditoria de manera declarativa; en altres casos caldrà crear disparadors que vagin emplenant les taules d'auditoria a mesura que s'hi produeixen esdeveniments.

3.5. La Llei de protecció de dades de caràcter personal

La legislació espanyola recull una sèrie de drets i deures relatius a la utilització de les dades personals en l'àmbit dels sistemes d'informació. Les principals fonts de drets són les següents: la Llei orgànica 5/1992, de 29 d'octubre, de regulació del tractament automatitzat de les dades de caire personal (LORTAD) i el Reial decret 1332/1994, de 30 de juny, que desenvolupa la llei anterior. Aquesta llei ha estat derogada per la Llei orgànica 15/1999, de 13 de desembre, de protecció de dades de caràcter personal (LOPDCP).

Les lleis LORTAD i LOPDCP s'han fet en compliment del mandat constitucional contingut en l'article 18.4, que diu: "la llei limitarà l'ús de la informàtica per garantir l'honor i la intimitat personal i familiar dels ciutadans i el ple exercici dels seus drets".

Les normatives actuals també segueixen la directiva del Parlament europeu 95/46, de 24 d'octubre, relativa a la protecció de les persones físiques pel que fa al tractament de dades personals i a la lliure distribució d'aquestes dades.

3.5.1. Principis generals de protecció de dades

1) Qualitat de les dades. Les dades de caràcter personal només es poden recollir per al tractament corresponent, i es poden sotmetre a aquest tractament quan sigui adequat, pertinent i no excessiu en relació amb l'àmbit i amb les finalitats per a les quals s'han obtingut.

Finalitat del tractament de les dades

Les dades de caràcter personal objecte del tractament no es poden fer servir per a finalitats incompatibles amb aquelles per a les quals s'han recollit.

2) Dret d'informació en la recollida de dades. S'ha d'informar amb antelació els interessats a qui se sol·licitin les dades personals dels punts següents:

- De l'existència d'un fitxer o tractament de dades de caràcter personal.

- Del caràcter obligatori o facultatiu de la resposta donada a les preguntes que els siguin plantejades.
- De les conseqüències de l'obtenció de les dades o de la negativa a subministrar-les.
- De la possibilitat d'exercitar els drets d'accés, rectificació, cancel·lació i oposició.
- De la identitat i direcció del responsable del tractament o, segons el cas, del seu representant.

3) Consentiment de l'afectat. Llevat que la llei disposi una altra cosa, per al tractament de les dades de caràcter personal es requereix el consentiment inequívoc dels afectats.

4) Dades especialment protegides. Les dades de caràcter personal que revelen ideologia, afiliació sindical, religió i creences només poden ser objecte de tractament amb el consentiment, exprés i per escrit, de l'afectat.

5) Dades relatives a la salut. Les institucions i els centres sanitaris públics i privats i els professionals corresponents poden procedir al tractament de les dades de caràcter personal relatives a la salut de les persones sempre que siguin facilitades pel titular per motiu d'assistència sanitària.

6) Seguretat de les dades i deure secret. Tant el responsable del fitxer com l'encarregat del tractament han d'adoptar les mesures tècniques i organitzatives necessàries que garanteixin la seguretat de les dades de caràcter personal i n'evitin l'alteració, la pèrdua, el tractament o l'accés no autoritzat.

Secret professional de les dades personals

Les dades de caràcter personal que fan referència a l'origen racial, a la salut i a la vida sexual es poden tractar o cedir només quan així ho disposa la llei.

7) Comunicació de dades. Les dades de caràcter personal objecte de tractament només poden ser comunicades a un tercer per al compliment de finalitats directament relacionades amb les funcions legítimes del cedent i del cessionari, si hi ha consentiment de l'interessat.

8) Accés a les dades per terceres persones. Els tractaments que realitzin terceres persones han d'estar regulats en un contracte per escrit o en alguna altra forma que permeti acreditar-ne la subscripció i el contingut. L'accés d'un tercer a aquestes dades no es considera comunicació de dades si és necessari per a la prestació d'un servei al responsable del tractament.

Les persones tenen els drets següents:

1) Impugnació de valoracions. Els ciutadans tenen dret a no veure's sotmesos a una decisió amb efectes jurídics que es fonamenten únicament en un tractament de dades destinades a avaluar aspectes de la seva personalitat.

Impugnació de valoracions

L'afectat pot impugnar els actes administratius o decisions privades que impliquin una valoració del seu comportament.

2) Dret d'accés. L'interessat té dret a sol·licitar i obtenir gratuïtament informació de les seves dades de caràcter personal sotmeses a tractament, l'origen d'aquestes i les comunicacions efectuades o que es prevegi fer.

Dret d'accés

El Registre General és de consulta pública i gratuïta.

3) Dret de rectificació i cancel·lació. Si les dades de caràcter personal són inexactes o incompletes, s'han de rectificar o cancel·lar. El responsable del tractament té l'obligació de fer efectiu aquest dret de rectificació o cancel·lació de l'interessat.

4) Dret a una indemnització. Si el responsable o l'encarregat del tractament incompleixen la llei i si, a conseqüència d'aquest incompliment, els interessats tenen algun perjudici o lesió en els seus béns o drets, els interessats tenen dret a ser indemnitzats.

Tutela dels drets

Les actuacions contràries a la present Llei poden ser objectes de reclamació pels interessats davant l'Agència de Protecció de Dades.

3.5.2. Els nivells de seguretat

Segons el Reial decret 994/1999, s'estableixen tres nivells de seguretat: el bàsic, el mitjà i l'alt. L'aplicació d'un nivell o un altre dependrà de la naturalesa de la informació.

Per a l'aplicació de les mesures de seguretat dels fitxers amb dades de caràcter personal, s'han d'observar els aspectes que es consideren a continuació.

Nivell bàsic

Tots els fitxers que contenen dades de caràcter personal han de seguir les mesures de seguretat del nivell bàsic.

Les **mesures del nivell bàsic de seguretat** s'han de definir en el document de seguretat que ha d'elaborar i implementar el responsable del fitxer. Aquest document ha d'incloure els punts següents:

El document de seguretat

El document de seguretat s'ha de mantenir sempre actualitzat i s'ha de revisar sempre que es produeixin canvis rellevants en el sistema d'informació o en l'organització d'aquest. El document s'ha d'adequar sempre a la normativa vigent en matèria de protecció de dades de caràcter personal.

a) L'àmbit d'aplicació del document amb especificació detallada dels recursos protegits.

b) Les mesures, normes, procediments, regles i estàndards adreçats a garantir el nivell de seguretat que exigeix el Reial decret 994/1999.

c) Les funcions i les obligacions del personal.

d) L'estructura dels fitxers que continguin dades de caràcter personal i la descripció dels sistemes d'informació que les tracten.

e) El procediment de notificació, gestió i resposta de les incidències (tant tecnològiques com funcionals i d'accés). El procediment de notificació i gestió d'incidències ha d'incloure un registre en què faci constar com a mínim el tipus d'incidència, el moment en el qual s'ha produït la incidència, la persona que fa la notificació, la persona a qui es notifica la incidència i els efectes derivats d'aquesta.

f) Els procediments de realització de còpies de suport i de recuperació de dades.

g) La periodicitat amb la qual s'han de substituir les contrasenyes dels usuaris.

h) El personal autoritzat per a concedir, modificar o alterar els accessos autoritzats a les dades i recursos i, també, els criteris per a dur a terme aquestes accions.

i) El personal autoritzat per a accedir al lloc on s'emmagatzemen dades de caràcter personal.

Nivell mitjà

Els fitxers que contenen dades relatives a la comissió d'infracció penals o administratives, la Hisenda pública i serveis financers (solvència patrimonial i crèdit, compliment o incompliment de les obligacions monetàries) han de seguir les mesures del nivell mitjà.

Les **mesures de seguretat de nivell mitjà** inclouen les del nivell bàsic i, a més, exigeixen el compliment dels punts següents:

a) Identificació del responsable o responsables de seguretat.

El responsable de seguretat

Molt sovint, l'administrador de la base de dades (ABD) és la persona responsable de la seguretat i de mantenir el document de seguretat.

b) Controls periòdics (auditoria) per a verificar el compliment del que està establert en el mateix document de seguretat.

Auditories

Els informes d'auditoria s'han de dipositar a l'Agència de Protecció de Dades.

c) Mesures que cal adoptar quan un suport que conté dades de caràcter personal està a punt de ser destruït o reutilitzat.

d) Personal autoritzat per a accedir als locals on els sistemes d'informació amb dades de caràcter personal es trobin físicament ubicats.

Nivell alt

Fitxers amb continguts de dades relacionades amb la ideologia, la religió, les creences, l'origen racial, la salut o la vida sexual han de seguir les mesures del nivell alt.

Les **mesures de seguretat de nivell alt** inclouen les del nivell mitjà i, a més, exigeixen el compliment dels punts següents:

a) Les dades de caràcter personal que s'han de distribuir en qualsevol tipus de suport s'han de xifrar.

b) Per a cada accés s'ha de guardar, com a mínim, la identificació de l'usuari, la data i l'hora en què s'ha fet l'accés, el nom del fitxer al qual s'ha accedit i el tipus d'accés.

c) S'ha de guardar un còpia de suport i dels procediments de recuperació de les dades en un lloc diferent de la ubicació física dels sistemes d'informació.

d) Les dades de caràcter personal que s'han de transmetre per la xarxa de telecomunicacions s'han de xifrar.

3.5.3. L'Agència de Protecció de Dades

L'Agència és un ens de dret públic, amb personalitat jurídica pròpia i plena capacitat pública i privada. Actua amb total independència de les administracions públiques en l'exercici de les funcions que duu a terme. La finalitat principal de l'Agència de Protecció de Dades consisteix a vetllar pel compliment de la legislació sobre protecció de dades personals i controlar-ne l'aplicació, especialment pel que fa als drets d'informació, accés, oposició, rectificació i cancel·lació de dades.

L'agència de protecció de dades a Internet

Podeu trobar les darreres actualitzacions de totes les normatives, i fer notificacions i registres d'inscripció dels fitxers de dades personals a l'adreça d'Internet de l'Agència de Protecció de Dades.

La pàgina també disposa de les FAQ (preguntes més freqüents) amb la finalitat que el ciutadà pugui exercir els seus drets i, també, aclarir com han de complir les organitzacions amb els deures que tenen.

4. Annexos

4.1. Regles d'equivalència d'operacions d'àlgebra relacional

Les regles d'equivalència s'utilitzen per a reestructurar l'arbre d'operacions d'àlgebra relacional que s'ha obtingut en el procés de descomposició de la consulta.

A continuació definim algèbricament cadascuna de les regles i en mostrem alguns exemples aclaridors.

1) Les operacions conjuntives de selecció es poden transformar en una cascada d'operacions individuals de selecció.

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

Exemple

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

2) Les operacions de selecció són commutatives.

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

Exemple

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

3) En una seqüència d'operacions de projecció tan sols és necessària la última projecció de la seqüència.

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

Exemple

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

4) Commutativitat de la selecció i la projecció.

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

Exemple

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

5) Commutativitat de la combinació theta (i del producte cartesià).

Combinacions theta

Són combinacions que utilitzen els operadors de comparació com a condició de combinació.

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

Exemple

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

6) Commutativitat de la selecció i de la combinació tetha (i del producte cartesià).

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

En cas que el predicat sigui de la forma $(p \wedge q)$, en què p correspon als atributs de R, i q, als de S, llavors les operacions són commutatives de la manera següent:

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

Exemple

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

7) Commutativitat de la projecció i de la combinació tetha (i del producte cartesià).

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

Exemple

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

8) Commutativitat de la unió i de la intersecció, però no de la diferència.

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

9) Commutativitat de la selecció i de les operacions de conjunts (unió, intersecció i diferència).

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

10) Commutativitat de la projecció i de la unió.

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

11) Associativitat de la combinació tetha (i del producte cartesià).

$$((R \bowtie S) \bowtie T) = (R \bowtie (S \bowtie T))$$
$$((\mathrm{R} \times \mathrm{S}) \times \mathrm{T}) = (\mathrm{R} \times (\mathrm{S} \times \mathrm{T}))$$

Exemple

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

12) Associativitat de la unió i de la intersecció (però no de la diferència de conjunts).

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

4.2. Consideracions sobre vistes en l'SGBD Oracle 11g

En aquest apartat de l'annex considerarem diferents aspectes referents a l'ús de les vistes en l'SGBD Oracle 11g:

1) Les vistes del diccionari de dades.

2) Les operacions d'actualització sobre vistes.

4.2.1. Vistes del diccionari de dades

El diccionari de dades està format per tot un conjunt de taules i vistes que donen informació sobre el contingut d'una base de dades:

- Estructures d'emmagatzematge.
- Usuaris i els seus drets.
- Els objectes: taules, vistes, índexs, procediments i funcions.

El diccionari de dades es carrega a la memòria i és utilitzat internament per al tractament de les consultes. Hi ha dos grups de taules o vistes en el diccionari de dades.

1) Vistes estàtiques. Estan basades en taules emmagatzemades a l'espai de taules[18] *SYSTEM*. Hi ha tres grans grups de vistes estàtiques:

(18)En anglès, *tablespace*.

a) *USER_%*. Contenen informació sobre els objectes que pertanyen a l'usuari.

b) *ALL_%*. Contenen informació sobre els objectes als quals l'usuari té accés.

c) *DBA_%*. Contenen informació sobre tots els objectes de la base de dades.

A continuació del prefix, el seu nom descriu la informació a la qual s'accedeix quan es fa una consulta sobre la vista.

2) Vistes dinàmiques de rendiment. Per bé que contenen informació que no està basada en taules ni en altres vistes, sinó que es captura de la memòria o de parts del fitxer de control, aquestes vistes permeten fer consultes sobre el rendiment de la base de dades o de l'SGBD. Es caracteritzen perquè tenen el prefix *V$* i, a continuació, el seu nom descriu la informació que representa. Aquestes vistes són accessibles tan sols si l'usuari té el privilegi DBA.

4.2.2. Operacions d'actualització sobre vistes

Reflexió

Per a cada SGBD, caldrà conèixer molt bé les operacions d'actualització que permet sobre les vistes.

Les operacions d'actualització (*INSERT*, *DELETE* i *UPDATE*) són un tema conflictiu per als diversos SGBD, ja que les vistes es basen en sentències *SELECT*, en què poden intervenir moltes o poques taules i, fins i tot, altres vistes, i per tant cal decidir a quina d'aquestes taules o vistes correspon l'operació d'actualització sol·licitada.

En l'SGBD Oracle 11g, el concepte de taula *key-preserved* és fonamental per a entendre les restriccions en les actualitzacions sobre vistes basades en operacions de combinació (*JOIN*).

Una taula és *key-preserved* en una operació de combinació (*JOIN*) si cada valor clau de la taula també pot ser un valor clau en el resultat del *JOIN*.

La propietat *key-preserved* d'una taula en un JOIN no depèn de les dades de les taules, sinó que és una propietat deduïda a partir de la seva definició.

Exemple

En la vista *EmpDeptView*, referida en aquest mòdul, la taula *Emp* és *key-preserved* en un *join* amb la taula *Dept*, ja que la columna *empId* (clau primària de la taula *Emp*) continua essent única en el resultat de la combinació. En canvi, la taula *Dept* no és *key-preserved*, ja que la columna *deptId* (clau primària de la taula *Dept*) no és única en el resultat de la combinació.

L'SGBD Oracle 11g proporciona la vista *USER_UPDATABLE_COLUMNS* que permet conèixer totes les columnes que podem actualitzar en les taules i vistes a les quals tenim accés.

Si analitzem el descriptor de la vista *USER_UPDATABLE_COLUMNS* podem veure que aquesta ens mostra totes les columnes de totes les taules i vistes a les quals l'usuari té accés, juntament amb les operacions que hi pot executar.

Descriptor d'una taula o vista

Per a visualitzar la descripció d'una taula o vista s'utilitza la sentència SQL:

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

- La columna *owner* mostra l'esquema que conté la taula o vista.
- La columna *table_name* conté els noms de les taules i de les vistes a les quals l'usuari té accés.
- La columna *column_name* ens indica les diverses columnes.

Exemples de columnes actualitzables en una vista

Recordem les dues vistes creades sobre l'esquema *COMPANY*. En cas que vulguem saber quines operacions podem realitzar sobre les columnes d'aquestes dues vistes, és possible 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'));
```

Fixem-nos que la columna *deptName* de la vista *EmpDeptView* no és actualitzable ni tampoc s'hi poden efectuar insercions ni eliminacions, és a dir:

- Si executem una sentència *DELETE* sobre la vista *EmpDeptView*, eliminarem files de la taula *Emp* (a la qual pertanyen les columnes que tenen *YES* com a suprimible), però no eliminarem cap fila de la taula *Dept* (a la qual pertany la columna *deptName*).
- Podem executar *INSERT* sobre la vista *EmpDeptView* per emplenar files de la taula *Emp*, però no pas per a emplenar cap fila en la taula *Dept*.
- Podem executar *UPDATE* sobre la vista *EmpDeptView* per modificar el contingut de les columnes provinents de la taula *Emp*, però no ho podrem fer per a la columna *deptName* provinent de la taula *Dept*.

Condicions perquè una vista sigui actualitzable

La vista *USER_UPDATABLE_COLUMNS* només mostra les vistes que el sistema considera que poden ser actualitzables. Perquè una vista tingui aquest reconeixement, s'han de donar les condicions següents:

a) Cada columna de la vista ha de correspondre amb una columna d'una taula simple.

b) La vista no pot contenir cap d'aquests components:

- Operador *DISTINCT*.
- Funcions d'agrupament.
- Clàusules *GROUP BY*, *ORDER BY*, *CONNECT BY* o *START WITH*.
- Subconsultes en la clàusula *SELECT*.
- Vista creada amb l'opció *WITH READ ONLY*.
- Operacions de combinació, amb algunes excepcions que es poden localitzar en la documentació de l'Oracle.

c) A més a més, si una vista actualitzable conté pseudocolumnes o expressions, l'operació d'actualització no pot fer referència a cap de les pseudocolumnes o expressions.

d) Perquè una vista basada en un *JOIN* sigui actualitzable, s'han de complir totes les condicions següents:

- La sentència d'actualització només pot afectar una única taula de les que formen part de l'operació de combinació.
- Per a una sentència *INSERT*, la vista no pot haver estat creada amb *WITH CHECK OPTION*, i totes les columnes per a les quals s'inseriran valors han de pertànyer a una taula *key-preserved*.
- Per a una sentència *UPDATE*, la vista no pot haver estat creada amb *WITH CHECK OPTION* i totes les columnes modificades han de pertànyer a una taula *key-preserved*.
- Per a les sentències *DELETE* sobre vistes basades en *JOIN*, si el *JOIN* és format per més d'una *KEY-PRESERVED TABLE*, s'efectua l'eliminació sobre la primera taula indicada en la clàusula *FROM*.

Exemple d'operacions d'actualització sobre vistes

Suposem que efectuem algunes actualitzacions de departaments parells per mitjà de la vista *EvenDeptView*.

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

Aquesta instrucció provoca la inserció d'una fila sense cap problema en la taula *Dept*.

Ara bé, si la vista *EvenDeptView* hagués estat creada amb l'opció *WITH CHECK OPTION* en executar qualsevol de les sentències següents:

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

no es podrien realitzar les accions sol·licitades, ja que no verifiquen la clàusula *WHERE* de la definició de la vista.

4.3. Aspectes referents a la seguretat en l'SGBD Oracle 11g

4.3.1. Gestió d'usuaris

A) Creació d'usuaris

Rol DBA

El rol DBA és el paper, rol, que permet a un usuari realitzar tasques que corresponen a l'administrador de la base de dades.

La sentència *SQL CREATE USER* permet crear un nou usuari.

La sintaxi és la següent:

```
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}];
```

Perquè un usuari es pugui connectar, cal donar-li els drets sobre el que pot fer, en aquest cas atribuint-li els privilegis de sistema *CREATE SESSION*.

Espais de taules *SYSTEM* i *SYSAUX*

Quan es realitza la instal·lació de l'SGBD Oracle, es creen automàticament els espais de taules *SYSTEM*, en què s'emmagatzemen els objectes del sistema, i *SYSAUX*, que s'utilitza per a fer operacions auxiliars del sistema.

Descripció dels paràmetres:

- *IDENTIFIED*. Aquesta clàusula indica si l'usuari està identificat pel sistema operatiu (*EXTERNALLY*) o per contrasenya (*BY* contrasenya). Des de la versió 11, les contrasenyes són per defecte sensibles a majúscules i minúscules (paràmetre *SEC_CASE_SENSITIVE_LOGON = TRUE* per defecte). Això vol dir que podem tenir creats comptes d'usuari sense permís de connexió. Encara que pot ser útil en la preparació dels comptes d'usuari sense activar-les immediatament i així deshabilitar temporalment l'accés a un usuari, actualment es bloqueja o desbloqueja explícitament un compte mitjançant *ACCOUNT LOCK|UNLOCK*.

- *DEFAULT NAMESPACE*. Aquesta clàusula indica en quin espai de taules es crearan per defecte els segments de l'usuari en cas que no s'expliciti cap clàusula *TABLESPACE* en el moment de la creació del segment. Si aquesta clàusula s'omet, roman l'espai de taules per defecte que hi hagi definit a la base de dades.

- *TEMPORARY NAMESPACE*. Aquesta clàusula indica l'espai de taules en què es crearan per defecte els segments temporals de l'usuari, per exemple creats en el moment d'execució d'una operació d'ordenació. Si aquesta clàusula s'omet, roman l'espai de taules temporal per defecte que hi hagi definit a la base de dades.

- *QUOTA*. Aquest concepte permet limitar l'espai que un usuari pot emprar en un espai de taules. Aquesta funcionalitat tan sols afecta els usuaris que poden crear segments, i en cap cas els usuaris finals d'una aplicació, atès que aquests es limiten a manipular dades. Per defecte, els usuaris no tenen cap quota en cap espai de taules, i en canvi els DBA tenen una quota il·limitada en tots els espais de taules. En tot cas, cal evitar donar quotes als usuaris en el *TABLESPACE SYSTEM* ni en el *SYSAUX*.

- *PROFILE*. Aquesta clàusula indica el perfil assignat a l'usuari.

- *PASSWORD EXPIRE*. Aquesta clàusula permet forçar la modificació de la contrasenya en el moment de la primera connexió. No té sentit si l'usuari s'identifica pel sistema operatiu.

- *ACCOUNT*. Aquesta clàusula admet un dels dos paràmetres següents: *LOCK*, si el compte existeix però evitem que l'usuari es pugui connectar, o *UNLOCK*, en què la connexió està autoritzada.

Exemple

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

B) Modificació d'un usuari

La sentència SQL *ALTER USER* permet modificar un usuari. Les clàusules són les mateixes que per a la creació.

Exemple

```
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 exemple canvia la contrasenya d'un usuari i força el canvi en la primera connexió. El segon exemple canvia l'espai de taules i hi assigna una quota sense límit. El tercer exemple prohibeix temporalment la connexió a l'usuari especificat.

C) Eliminació d'un usuari

La sentència SQL *DROP USER* permet eliminar un usuari.

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

Si un usuari és propietari d'un conjunt d'objectes, l'opció *CASCADE* és necessària per a eliminar aquests objectes. En cas contrari, ens retornarà el codi d'error ORA-01922.

Reflexió

No es pot eliminar un usuari que està connectat. En aquest cas ens retorna l'error ORA-01940.

També cal recordar que amb *DROP USER* no hi ha possibilitat de *ROLLBACK* perquè és una sentència DDL.

4.3.2. Definició de perfils

Un perfil és un conjunt de limitacions de recursos identificades per un nom que es poden assignar a un usuari.

A) Creació d'un perfil

La sentència SQL *CREATE PROFILE* permet crear un nou perfil.

```
CREATE PROFILE profile_name LIMIT constraint_resources
```

Les limitacions poden ser de restriccions de recursos o per la gestió de contrasenyes.

Paràmetres per a definir restriccions:

1) Restriccions de recursos

- *SESSION_PER_USER*: Nombre de sessions simultànies.
- *CPU_PER_SESSION*: Assignació de CPU total per sessió.
- *CPU_PER_CALL*: Assignació de CPU total per crida.
- *CONNECT_TIME*: Durada total de la connexió, en minuts.
- *IDLE_TIME*: Temps d'inactivitat.
- *LOGICAL_READS_PER_SESSION*: Nombre de lectures lògiques per sessió.
- *LOGICAL_READS_PER_CALL*: Nombre de lectures lògiques per crida.
- *COMPOSITE_LIMIT*: Suma ponderada de *CPU_PER_SESSION*, *CONNECT_TIME*, *LOGICAL_READS_PER_SESSION* i *PRIVATE_SGA*. La vista *RESOURCE_COST* permet consultar les ponderacions emprades i la sentència *ALTER RESOURCE COST* permet modificar les ponderacions.

2) Restriccions sobre contrasenyes

- *FAILED_LOGIN_ATTEMPTS*: Nombre d'intents de connexió fracassats com a pas previ al bloqueig de compte.
- *PASSWORD_LOCK_TIME*: Durada del bloqueig.
- *PASSWORD_LIFE_TIME*: Durada de vida de la contrasenya.
- *PASSWORD_GRACE_TIME*: Període de gràcia després de la caducitat de la contrasenya.
- *PASSWORD_REUSE_TIME*: Nombre de canvis de contrasenya abans que una contrasenya es pugui reutilitzar.

- *PASSWORD_VERIFY_FUCTION*: Funció de verificació de la complexitat de la contrasenya. L'*script utlpwdmg:sql* del repositori $ORACLE_HOME/rdms/admin conté un exemple de funció de verificació.

En la majoria de restriccions esmentades es poden emprar també les paraules clau *UNLIMITED* i *DEFAULT*.

B) Modificació d'un perfil

La sentència *ALTER PROFILE* permet modificar un perfil. Per exemple:

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

Els valors dels altres paràmetres conserven el valor que tenen per defecte (*UNLIMITED*).

C) Assignació d'un perfil a un usuari

Es pot assignar un perfil a un usuari en les situacions següents.

1) En el moment de la creació d'un usuari. Per exemple:

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

2) Quan es modifica un usuari. Per exemple:

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

D) Informació referent a usuaris i els perfils corresponents

Per a obtenir informació sobre usuaris i perfils es poden consultar les vistes del diccionari de dades següents:

- *DBA_USERS*: Informació sobre els usuaris.
- *DBA_TS_QUOTAS*: Informació sobre quotes d'usuaris.
- *DBA_PROFILES*: Informació sobre els perfils.

4.3.3. Identificació d'usuaris

Un usuari es pot identificar des de l'SGBD Oracle o bé des del mateix sistema operatiu.

1) Identificació per Oracle. L'usuari es connecta a la base de dades emprant un nom i una contrasenya. Per exemple:

```
SQL > CONNECT joan/GNB9175$

Connectat.
```

2) Identificació pel sistema operatiu. Oracle no verifica la contrasenya. Cal configurar el paràmetre *OS_AUTHEN_PREFIX = OPS* si es vol tenir aquesta funcionalitat habilitada.

4.3.4. Gestió dels privilegis

En una base de dades Oracle, els drets dels usuaris es gestionen a partir del concepte de *privilegi*. Un privilegi és:

- El dret d'executar una sentència SQL general, com pot ser per a crear una taula. Aquest concepte s'anomena **privilegi de sistema**.

- El dret d'accedir a un objecte d'un altre usuari. Aquest concepte s'anomena **privilegi d'objecte**.

1) Privilegis de sistema

Un privilegi de sistema és el dret d'executar una sentència SQL en general. Cada sentència SQL té com a mínim un privilegi de sistema associat que s'anomena de la mateixa manera que la sentència SQL. Així doncs, la sentència SQL *CREATE TABLE* té un privilegi de sistema associat anomenat *CREATE TABLE* que atorga el dret de crear una taula en el seu propi esquema. També cal tenir en compte que el privilegi *CREATE ANY TABLE* atorga el dret per crear taules en qualsevol esquema de la base de dades.

Els privilegis de sistema són una font de risc, sobretot els relacionats amb la gestió d'usuaris i els drets corresponents (*CREATE USER*, *ALTER USER*, *DROP USER*, *GRANT ANY PRIVILEGE*, *GRANT ANY ROLE*) i tots els que permeten eliminar objectes (*DROP ANY TABLE*, *DROP TABLESPACE*, etc.).

Alguns privilegis

- *CREATE SESSION*: Atorga a un usuari el dret de connectar-se. Si un usuari no té aquest privilegi, es retorna l'error ORA-01045.
- *SELECT ANY DICTIONARY*: Permet consultar qualsevol objecte que pertanyi al diccionari de dades de l'esquema SYS.

Els privilegis de sistema s'utilitzen principalment per a controlar l'ús de les sentències DDL i normalment estan destinats a administradors o desenvolupadors i al compte propietari de l'aplicació, i poques vegades a usuaris finals.

La **sentència d'atorgament de privilegis** és la següent:

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

Un privilegi es pot assignar a un usuari, a un grup d'usuaris o a tothom (*PUBLIC*). El privilegi assignat està actiu immediatament. La clàusula *WITH ADMIN OPTION* atorga al beneficiari el dret de transmetre aquest privilegi de sistema.

Un usuari que vulgui assignar o revocar un privilegi de sistema és necessari que prèviament hagi rebut:

- El mateix privilegi amb la clàusula *WITH ADMIN OPTION*.
- El privilegi de sistema *GRANT ANY PRIVILEGE*.

Per a revocar un privilegi a un usuari s'utilitza la sentència *REVOKE*.

La **sentència de revocació de privilegis** és la següent:

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

No hi ha cascada en la revocació d'un privilegi de sistema que hagi estat transmès gràcies a la clàusula *WITH ADMIN OPTION*.

Exemple

Si s'assigna a en Joan un privilegi amb l'opció *WITH ADMIN OPTION* i aquest l'ha transmès a en Jordi, revocar aquest privilegi a en Joan no té cap efecte sobre el privilegi transmès d'en Joan a en Jordi.

Si s'ha assignat un privilegi a un usuari amb l'opció *WITH ADMIN OPTION* i es vol eliminar aquesta opció, cal revocar el privilegi i assignar-lo novament sense l'opció *WITH ADMIN OPTION*.

També cal tenir en compte que la sentència *REVOKE* permet revocar privilegis que un usuari hagi rebut directament, no els que l'usuari rep via *PUBLIC*. Es poden revocar tots els privilegis de sistema mitjançant la sentència següent:

```
REVOKE ALL PRIVILEGES FROM autoritzat;
```

2) Privilegis d'objecte

Un privilegi sobre un objecte és el dret d'accedir a un objecte d'un altre usuari. Per defecte, l'únic que té dret a accedir a l'objecte és el propietari.

Els privilegis d'objecte s'utilitzen fonamentalment per a permetre als usuaris finals d'una aplicació accedir als objectes de l'aplicació creats en un compte propietari de l'aplicació. Perquè un altre usuari pugui accedir a l'objecte, el propietari cal que li assigni un privilegi objecte:

Taula 6. Exemples de privilegis

Privilegi	Definició	Taula	Vista	Seqüència	Programa
`SELECT [(columnes)]`	Dret de lectura de dades	√	√	√	√
`INSERT [(columnes)]`	Dret de creació de dades	√	√		
`UPDATE [(columnes)]`	Dret d'actualització de dades*	√	√		
`DELETE`	Dret d'eliminació de dades*	√	√		
`EXECUTE`	Dret d'execució d'un programa				√

* Per a assolir els privilegis *UPDATE* i *DELETE*, cal concedir també el privilegi *SELECT*.

La sentència SQL *GRANT* permet assignar un privilegi objecte.

La **sentència d'atorgament de privilegis** és la següent:

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

Per als privilegis *INSERT* i *UPDATE* es poden especificar les columnes amb l'objectiu de limitar el privilegi a les columnes indicades.

Un privilegi es pot assignar a un usuari, a un grup d'usuaris o a tothom (*PUBLIC*).

La clàusula *WITH GRANT OPTION* atorga al beneficiari el dret de transmetre aquest privilegi objecte.

Per a assignar o revocar un privilegi objecte és necessari:

- Ser el propietari de l'objecte.
- Haver rebut el mateix privilegi amb la clàusula *WITH ADMIN OPTION*.
- Haver rebut el privilegi de sistema *ANY OBJECT PRIVILEGE*.

Quan volem accedir a un objecte del qual s'han rebut privilegis amb l'opció *WITH GRANT OPTION*, cal qualificar aquest objecte amb el nom del propietari, perquè l'SGBD assumeix per defecte que aquest objecte es troba en el propi esquema.

Per a facilitar l'escriptura de sentències i fer transparent l'esquema propietari dels objectes, cal emprar els sinònims. L'existència d'un sinònim, encara que aquest sigui públic, no dóna cap dret sobre l'objecte subjacent.

La sentència SQL *REVOKE* permet revocar un privilegi objecte.

La **sentència de revocació de privilegis** és la següent:

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

També cal tenir en compte que la sentència *REVOKE* permet revocar privilegis que un usuari hagi rebut directament, no els que l'usuari rep via PUBLIC.

Es poden revocar tots els privilegis de sistema mitjançant la sentència següent:

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

Hi ha cascada en la revocació d'un privilegi d'objecte que hagi estat transmès gràcies a la clàusula *WITH GRANT OPTION*.

Exemple

Si s'assigna a en Joan un privilegi amb l'opció *WITH GRANT OPTION* i aquest l'ha transmès a en Jordi, el fet de revocar després aquest mateix privilegi a en Joan en comporta la revocació immediata sobre en Jordi.

Si s'ha assignat un privilegi a un usuari amb l'opció *WITH GRANT OPTION* i es vol eliminar aquesta opció, cal revocar el privilegi i assignar-lo novament sense l'opció *WITH GRANT OPTION*.

Algunes consideracions sobre privilegis en vistes i programes emmagatzemats

El fet que un usuari tingui un dret sobre una vista no implica que tingui cap dret sobre els objectes subjacents a la vista. En canvi, per defecte, un programa emmagatzemat s'executa amb els drets del propietari. El comportament desitjat es defineix en el moment de la creació del programa emmagatzemat gràcies a la clàusula *AUTHID*.

La sintaxi de la clàusula *AUTHID* és la següent:

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

Consulta d'informació al catàleg sobre privilegis

Hi ha diferents vistes al catàleg sobre els privilegis de sistema.

- *DBA_SYS_PRIVS*: mostra els privilegis de sistema assignats als usuaris o als rols.
- *SESION_PRIVS*: mostra els privilegis de sistema actualment actius a la sessió, tant si són obtinguts directament o mitjançant un rol.
- *SYSTEM_PRIVILEGE_MAP*: llista tots els privilegis de sistema.

També podem obtenir del diccionari de dades informació sobre els privilegis objecte per mitjà de les vistes següents.

- *DBA_TAB_PRIVS*: mostra els privilegis objecte assignats als usuaris o als rols sobre la totalitat de l'objecte.
- *DBA_COL_PRIVS*: mostra els privilegis objecte assignats únicament sobre certes columnes de l'objecte.
- *TABLE_PRIVILEGE_MAP*: mostra la llista de tots els privilegis d'objecte.

4.3.5. Rols

Dins d'una organització, els rols es creen per a modelitzar funcions de treball diferents. Els permisos per a fer determinades operacions estan assignats a rols específics. Als usuaris del sistema se'ls assigna uns rols concrets i per mitjà d'aquests rols adquireixen els diferents permisos que els permeten exercir funcions específiques en el sistema informàtic. Atès que els usuaris no tenen permisos d'una manera directa, sinó que els adquireixen a partir del seu rol, la gestió dels drets de cada usuari es converteix simplement en una qüestió d'assignar els rols adequats al compte de l'usuari, cosa que simplifica les operacions més comunes, com ara afegir un usuari o canviar les funcions d'un usuari.

El model basat en rols RBAC defineix les tres normes primàries següents:

1) Assignació de rols. Un subjecte pot exercir un permís només si el subjecte ha estat seleccionat o se li ha assignat un rol.

2) Autorització de rol. El rol actiu d'un subjecte cal que estigui autoritzat pel subjecte. Amb l'article 1, aquesta norma garanteix que els usuaris puguin assumir rols només per als quals estan autoritzats.

3) Autorització de permís. Un usuari pot exercir un permís només si el permís és autoritzat pel rol actiu de l'usuari. Amb les normes 1 i 2, aquesta norma garanteix que els usuaris poden exercir només els permisos per als quals estan autoritzats.

Com a restriccions addicionals, els rols es poden combinar en una jerarquia de nivells, en què els rols de nivell superior assumeixen els permisos de propietat dels seus subrols.

Figura 4. Model RBAC

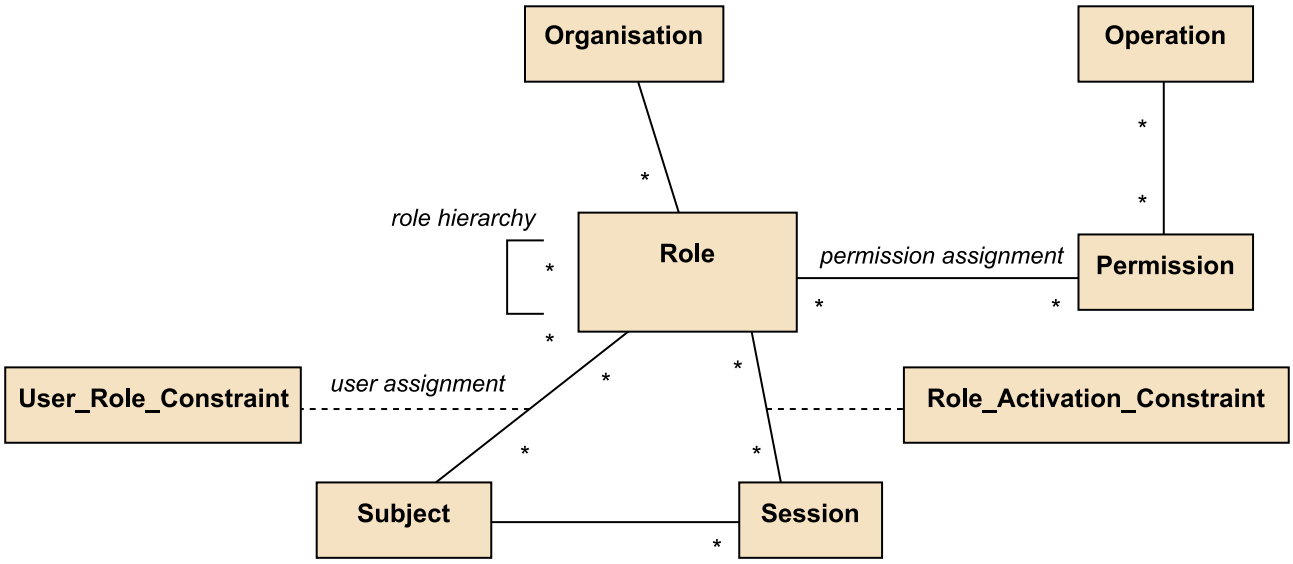


Figura 4

A la figura podem veure les regles següents:

- Un subjecte pot tenir diversos rols.
- Un rol pot tenir diversos subjectes.
- Un rol pot tenir molts permisos.
- Un permís pot ser assignat a múltiples rols.
- Una operació pot tenir assignats molts permisos.
- Un permís pot estar assignat a moltes operacions.

Els privilegis es poden assignar directament als usuaris o mitjançant rols. L'ús d'RBAC per a administrar els privilegis d'usuari està àmpliament acceptat com una bona pràctica. Oracle permet modelitzar RBAC.

Un **rol** és un agrupament de privilegis definit per un nom que es pot atribuir a un usuari de manera que l'usuari rep automàticament els privilegis continguts en el rol. Els rols s'utilitzen per a la gestió dels drets.

La sentència SQL *CREATE ROLE* permet crear un rol.

Reflexió

Per a crear un rol cal que l'usuari que el crea tingui el privilegi de sistema *CREATE ROLE*.

La **sentència de creació d'un rol** té la sintaxi següent:

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

En el moment de creació del rol és possible precisar el mecanisme amb què es podrà activar, tant mitjançant una contrasenya, l'autenticació externa (per exemple, mitjançant el sistema operatiu) com mitjançant un paquet.

Els requisits i la sintaxi referents a l'assignació o la revocació de privilegis de sistema i d'objecte per rols són els mateixos que per als usuaris. Els privilegis són immediatament assignats o revocats segons quina sigui la instrucció i tenen un efecte immediat sobre els usuaris connectats que tenen el rol actiu.

Reflexió

Un usuari pot tenir diversos rols i en aquest cas els privilegis s'acumulen (no hi ha efecte "negatiu").

La **sentència d'assignació de rols** a usuaris té la sintaxi següent:

```
GRANT role_name [ , ... ]
TO {user_name|role_name|PUBLIC} [ , ... ]
[WITH ADMIN OPTION];
```

Els requisits i la sintaxi referents a la revocació d'un rol a un usuari o a un altre rol, i sobre l'eliminació d'un rol, són semblants als que hem explicat amb anterioritat.

Un rol assignat a un usuari s'activa automàticament per defecte en el moment de la connexió de l'usuari. Si l'usuari ja estava connectat en el moment de l'assignació, l'activació immediata del rol no és automàtica. Cal activar el rol amb la sentència *SET ROLE*.

Tenir la possibilitat d'emprar diferents rols sense que estiguin actius al mateix temps és interessant perquè:

- El paràmetre *MAX_ENABLED_ROLES* (per defecte 30) limita el nombre de rols actius simultanis per a un usuari.

- Els rols protegits amb contrasenya es poden assignar als usuaris encara que romanguin inactius, i sense donar la contrasenya a l'usuari, fer que les aplicacions siguin les encarregades d'activar els rols quan calgui proporcionant les contrasenyes.

La sentència *ALTER USER* permet definir els rols per defecte d'un usuari.

La **sintaxi de la sentència *ALTER USER*** és la següent:

```
ALTER USER user_name
DEFAULT ROLE {role_name [ , ... ]|ALL {EXCEPT role_name [ , ... ]|NONE};
```

La sentència *SET ROLE* permet activar o desactivar un rol.

La **sintaxi de la sentència *SET ROLE*** és la següent:

```
SET ROLE {role_name [IDENTIFIED BY password][ ,... ]|ALL {EXCEPT role_name[ , ... ]|NONE};
```

Consulta al catàleg per a obtenir informació sobre rols

Hi ha diferents vistes del diccionari de dades a més de les vistes anteriorment (DBA_SYS_PRIVS, DBA_TAB_PRIVS i DBA_COL_PRIVS) que permeten obtenir informació sobre els rols:

- *DBA_ROLES*: llista dels rols existents a la base de dades.
- *DBA_APLICATION_ROLES*: descripció dels rols que tenen el sistema d'activació per mitjà d'un paquet.
- *DBA_ROLE_PRIVS*: rols assignats a usuaris o a altres rols.
- *ROLE_SYS_PRIVS*: privilegis de sistema assignats a rols.
- *ROLE_TAB_PRIVS*: privilegis d'objecte assignats a rols.
- *ROLE_ROLE_PRIVS*: rols assignats a altres rols.
- *SESION_ROLES*: rols actualment actius a la sessió.

Resum

En aquest mòdul hem estudiat el processament de consultes i vistes, i la seguretat en bases de dades.

Hem vist que és responsabilitat de l'SGBD transformar la consulta introduïda per un usuari en una altra d'equivalent, però que es pugui calcular d'una manera més eficient. Aquest procés de cerca d'una bona estratègia per a processar una consulta s'anomena *optimització de consultes*. Primerament es comprova la correctesa de la consulta SQL des dels punts de vista semàntic i lèxic, i després es transforma en un arbre que en permet l'anàlisi i l'optimització. Posteriorment es decideix quina és la millor estratègia d'implementació física.

Una vista és una relació virtual que no existeix físicament en la base de dades, sinó que es genera cada vegada que un usuari efectua una sol·licitud. El mecanisme de les vistes contribueix a la seguretat de manera que permet ocultar els detalls de la base de dades a certs usuaris. Amb l'ús de disparadors de substitució es pot fer que les vistes siguin actualitzables.

Hem exposat el concepte de *seguretat d'una base de dades* com el conjunt de mecanismes que protegeixen la base de dades davant d'amenaces intencionades o accidentals. La majoria de SGBD proporcionen un mecanisme anomenat *control d'accés discrecional* (DAC) que gestiona els privilegis emprant llenguatge SQL. Encara que alguns SGBD proporcionen **tècniques de control d'accés obligatori** (MAC) basades en polítiques de nivell de sistema que no poden ser alterades pels usuaris, l'estàndard SQL no inclou suport per a MAC. Finalment, hem vist una breu pinzellada sobre la legislació vigent de protecció de dades, a més de la necessitat de tenir una cura especial amb totes les dades de caràcter personal que han d'estar protegides per llei.

Glossari

amenaça *f* Situació o succés intencionat o accidental que pot afectar d'una manera adversa el sistema i, en conseqüència, l'organització.

autenticació *f* Mecanisme pel qual es determina si un usuari és qui diu que és.

autorització *f* Concessió d'un dret o privilegi que permet a un subjecte accedir legítimament al sistema o a un objecte del sistema.

heurístic -a *adj* Dit del mètode que utilitza el raonament i les experiències passades per a trobar la millor solució a un problema.

mecanisme de còpia de seguretat *m* Procés de realitzar periòdicament una còpia de la base de dades, de l'arxiu de registre i possiblement d'algun programa i emmagatzemar-ho en un dispositiu d'emmagatzematge fora de línia.

optimització *f* Procés pel qual es transforma una consulta en una altra d'equivalent, però més eficient, i que es pot realitzar des dels punts de vista semàntic, sintàctic i físic.

pla d'execució *m* Seguit d'operacions lògiques o físiques que cal efectuar per a obtenir el resultat d'una consulta.

registre *m* Procés de mantenir un diari on s'emmagatzemin els canvis efectuats en la base de dades amb l'objectiu de realitzar la recuperació d'una manera efectiva en cas de fallida del sistema.

vista *f* Resultat dinàmic d'una o més operacions relacionals sobre una base de dades amb l'objectiu de produir una altra relació.

xifratge *f* Codificació de dades mitjançant un algorisme especial que fa que aquestes dades no puguin ser llegibles per cap programa que no disposi de la clau de desxifratge.

Bibliografia

Connolly, T.; Begg, C. (2005). *Sistemas de bases de datos*. 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*. Madrid: McGraw-Hill.

Weinberg, P.; Groff, J.; Oppel, A. (2010). *SQL. The complete reference* (3a. ed.). McGraw-Hill.