Disseny físic de bases de dades

PID_00253055

Jordi Conesa Caralt
M. Elena Rodríguez González

Temps mínim de dedicació recomanat: 14 hores

Universitat Oberta de Catalunya

Índice

Introducción

Les fases de disseny conceptual i lògic d'una base de dades tenen com a objectiu construir un model (a diferents nivells d'abstracció) que satisfaci els requisits d'informació dels usuaris. Com a resultat, es proporciona un esquema de base de dades que permeti representar tota la informació necessària correctament. Però si pensem en el sistema gestor de bases de dades (SGBD) on s'implementarà la base de dades, en el maquinari en què l'SGBD estarà instal·lat, en quins dispositius d'emmagatzematge no volàtil es guarden les dades, en quins programes accediran a la base de dades i en el volum de dades que haurà d'emmagatzemar, sorgeix la pregunta de com ajustar la base de dades perquè funcioni eficientment en l'entorn per al qual s'ha dissenyat. El disseny físic de bases de dades és el procés que s'encarrega de respondre a aquesta pregunta.

El disseny físic forma part del procés general de disseny d'una base de dades. En particular, es realitza després del disseny lògic d'aquesta. Quan la base de dades es basa en el model de dades relacional, a la fase de disseny lògic es generen un conjunt de taules normalitzades a partir de l'esquema conceptual (el qual, com ja sabem, és independent de la tecnologia que s'utilitzarà per implementar la base de dades) creat a la fase de disseny conceptual de la base de dades. El disseny físic comença en el moment en què aquestes taules són creades i se centra a definir els mètodes d'emmagatzematge i accés a les dades que permetin operar amb la base de dades amb una eficiència esperada. Un cop fet el disseny físic, el pas següent és implementar la base de dades i monitorar-la. Aquesta tasca de monitoratge s'encarregarà de comprovar si la base de dades satisfà el rendiment esperat. Si no és així, s'hauran de fer modificacions per millorar-lo. Potser també caldrà modificar l'esquema per ajustar la base de dades a nous requeriments. El responsable de realitzar les tasques prèviament descrites és l'administrador de la base de dades, que té a la seva disposició diferents eines subministrades pels venedors d'SGBD.

Model lògic

Alguns fabricants d'SGBD utilitzen el terme *model lògic* per referir-se a l'esquema conceptual i el terme *model físic* per referir-se a la implementació de les taules normalitzades a l'SGBD específic. En aquest cas, la creació de les taules normalitzades a partir de l'esquema estaria dins de la fase de disseny físic.

Taula normalitzada

Una taula està normalitzada quan representa un concepte únic del món real. Si el disseny conceptual de la base de dades és correcte, les taules resultants en el disseny lògic compleixen aquesta condició. Una taula no normalitzada pateix redundàncies, és a dir, repeticions de les dades que serien evitables.

En aquest mòdul s'introdueix el disseny físic i les seves característiques principals. Per fer-ho es mostren els elements que s'hi utilitzen per millorar el rendiment de la base de dades. Per a cada un d'aquests elements es mostra, a tall d'exemple, com gestionar-los en Oracle i PostgreSQL.

Objetivos

Aquests materials permeten que l'estudiant adquireixi les competències següents:

1. Entendre què és el disseny físic de bases de dades, el seu objectiu, motivació i com fer-ho.

2. Conèixer, de manera general, les diferents tècniques i elements utilitzats en el disseny físic de bases de dades i saber en quins casos té sentit aplicar-los.

3. Conèixer la diferència entre espai virtual i físic de la base de dades i com gestionar-los.

4. Conèixer els tipus d'índexs més habituals, com funcionen i en quins casos són necessaris.

5. Saber com funcionen els mètodes d'optimització de consultes utilitzats als SGBD relacionals i com utilitzar índexs per optimitzar consultes.

6. Saber què són les vistes materialitzades, quan són necessàries i com gestionar-les.

1. Disseny físic de bases de dades

Conceptualment, podem definir el **disseny físic** d'una base de dades com un procés que, a partir del seu disseny lògic i d'informació sobre l'ús que se n'espera, crearà una configuració física de la base de dades adaptada a l'entorn en què s'allotjarà i que permeti l'emmagatzematge i l'explotació de les dades de la base de dades amb un rendiment adequat.

A continuació, estudiarem detalladament aquesta definició per entendre a fons el concepte de disseny físic:

a) Entrades: el disseny físic parteix de la informació següent:

- **L'esquema lògic**: és el resultat del disseny lògic de la base de dades i conté la llista de taules necessàries per emmagatzemar la informació rellevant, incloent-hi almenys les seves columnes, claus primàries i foranes.

- **Informació sobre l'ús esperat de la base de dades**: estimació sobre els volums de dades i transaccions que el sistema haurà de processar. També ha de contenir una estimació sobre les operacions SQL que s'utilitzaran, sobre quines dades, amb quina freqüència i la qualitat de servei es vol tenir.

b) Sortida: es prendran un conjunt de decisions sobre les estructures físiques més adequades que caldrà utilitzar, fet que generarà una configuració física de la base de dades. Aquesta configuració podrà estar composta per una combinació adequada dels espais d'emmagatzematge, un conjunt d'índexs per millorar el rendiment de les consultes, una partició de taules adequada, un conjunt de vistes materialitzades i altres elements addicionals, com ara disparadors que regulin regles de negoci complexes o procediments emmagatzemats a la base de dades.

c) Adaptat a l'entorn: el disseny físic es pot veure influït per l'SGBD en què s'implementi la base de dades, els dispositius d'emmagatzematge no volàtil on es guardin les dades de la base de dades i l'entorn de maquinari on s'allotgi l'SGBD.

- **SGBD**: el pas d'un disseny lògic a un de físic requereix un coneixement profund de l'SGBD on s'implementarà la base de dades. En particular, entre altres, s'haurà de tenir un coneixement dels elements següents:
  - Suport que s'ofereix a la integritat referencial.
  - Tipus d'índexs disponibles.
  - Tipus de dades disponibles.
  - Tipus de restriccions d'integritat disponibles.
  - Paràmetres de configuració que puguin afectar el rendiment, com ara la mida de pàgina i la gestió de dades i de concurrència utilitzades.
  - Construccions SQL disponibles de suport al disseny físic.
  - Particularitats de l'SGBD utilitzat (i, en conseqüència, del llenguatge SQL) en relació amb la definició d'elements relacionats amb el disseny físic de la base de dades.

Sentències SQL

A diferència de les sentències SQL relacionades amb la manipulació de les dades i amb el disseny lògic de les taules, les sentències SQL de suport al disseny físic de la base de dades no apareixen a l'estàndard SQL. Per aquest motiu, hi ha molta variació entre les sentències SQL utilitzades en els SGBD per gestionar els elements relacionats amb el disseny físic.

- **Entorn d'emmagatzematge**: no és el mateix emmagatzemar la base de dades en un PC que en un servidor amb múltiples discs o altres dispositius d'emmagatzematge no volàtils, o en un sistema distribuït. El maquinari disponible i les seves capacitats (velocitat d'accés, sistemes de replicació, etc.) permetran definir diferents configuracions físiques per millorar el rendiment de la base de dades.

d) Que permeti l'emmagatzematge i l'explotació de les dades amb un rendiment adequat: l'objectiu del disseny físic és obtenir un bon rendiment de la base de dades en un entorn real. Amb *rendiment* ens referim bàsicament al temps de resposta a operacions de consulta o actualització, a la càrrega de transaccions que cal processar i a la disponibilitat de la base de dades.

1.1. Elements de disseny físic

Els elements oferts per l'SGBD per tal d'afinar el model físic de la base de dades són principalment els següents:

- Espais per a taules
- Índexs
- Vistes materialitzades
- Particions

1.1.1. Espais per a taules

Un **espai per a taules**, *tablespace* en anglès, és un component de l'SGBD que indica on s'emmagatzemaran les dades de la base de dades i en quin format.

L'espai per a taules permet associar un fitxer físic a un conjunt d'elements de la base de dades (taules, índexs, etc.). Aquest fitxer contindrà les dades corresponents als elements de la base de dades associats.

El nombre d'espais per a taules que s'han de crear en una base de dades dependrà de les necessitats del disseny físic. Els espais per a taules podran ser simples quan afectin un sol element de la base de dades, o compostos quan afectin diferents elements de la base de dades.

A més d'indicar el fitxer on s'emmagatzemaran les dades, els espais per a taules també permeten definir com serà aquest fitxer (espai assignat inicialment, espai incremental quan l'espai assignat s'esgota, memòria intermèdia disponible), si s'utilitzaran tècniques de compressió sobre les dades emmagatzemades, com gestionar l'espai lliure del fitxer, etc.

1.1.2. Índexs

Els **índexs** són estructures de dades que permeten millorar el temps de resposta de les consultes sobre les dades d'una taula.

De manera intuïtiva, es pot establir un paral·lelisme entre els índexs usats en una base de dades amb els índexs de conceptes clau que podem trobar en la majoria de llibres de text. La idea és organitzar (alfabèticament en el cas de l'índex del llibre) una sèrie de valors d'interès (els conceptes clau triats en el cas del llibre), conjuntament amb les pàgines físiques (les pàgines del llibre) que contenen dades sobre el valor que està sent indexat (en el cas del llibre, s'indiquen les pàgines del llibre que tracten sobre cada concepte clau que es decideix indexar).

Els índexs mantenen els valors d'una o més columnes d'una taula en una estructura que permet l'accés a les files de la taula. Cada possible valor v té correspondència amb l'adreça (o adreces) física que conté les files de la taula que tenen v com a valor de la columna indexada. Aquesta estructuració de les dades permet un accés ràpid quan es fan cerques per valor o quan es requereix l'ordenació de les files d'una taula d'acord amb els valors de la columna indexada.

En cas de no disposar d'índexs, qualsevol consulta sobre una taula de la base de dades requerirà, en el pitjor dels casos, un recorregut complet del contingut de la taula.

Els índexs es poden utilitzar també com a sistema de control de les restriccions d'integritat d'unicitat (claus primàries i alternatives), definint un índex únic sobre les columnes amb aquesta restricció. De manera similar, també serveixen per garantir les restriccions associades a les claus foranes.

Tot i que l'ús d'índexs millora el rendiment de la consulta de dades d'una base de dades, cal tenir en compte que tenen associat un cost de manteniment respecte als canvis a les dades. En conseqüència, cal estudiar acuradament quants i quins índexs s'han de crear.

Hi ha diferents tipus d'índex, com, per exemple, els índexs basats en arbres B+, en mapes de bits, en tècniques de dispersió (*hash* en anglès), en arbres R, etc. L'índex més utilitzat és el basat en arbres B+, ja que funciona relativament bé en tot tipus de situacions. No obstant això, no hi ha un índex universal que funcioni bé en tots els casos. En funció de cada cas s'haurà d'escollir entre un tipus d'índex o un altre, i amb una configuració adequada.

1.1.3. Vistes materialitzades

Els resultats de les consultes sobre una o més taules es poden emmagatzemar en **vistes materialitzades**. A diferència de les vistes convencionals, el conjunt de files que forma el contingut d'una vista materialitzada s'emmagatzema físicament a la base de dades, com les files que formen el contingut d'una taula. Aquest tipus de vistes poden ser molt útils per millorar el rendiment de consultes que s'executen repetidament o de consultes basades en dades agregades. En tots dos casos, l'ús d'una vista materialitzada permet calcular la informació *a priori*, amb la qual cosa ens estalviem de calcular el contingut associat a la vista (o a part d'ella) cada vegada que l'usuari ho sol·liciti.

El temps de resposta de la base de dades pot disminuir significativament quan s'implementen les vistes materialitzades adequades. No obstant això, en la seva definició cal tenir en compte els seus costos: requereix espai extra per emmagatzemar el contingut associat a les vistes materialitzades i requereix mantenir el contingut actualitzat. Aquest últim fet, per exemple, desaconsella l'ús de les vistes materialitzades en casos en què les dades d'origen tinguin una freqüència d'actualització alta.

1.1.4. Particions

Les **particions** permeten distribuir les dades d'una taula en diferents espais físics. Les particions es poden fer d'acord amb multitud de criteris: distribuint les files de la taula en funció dels valors d'una columna o d'un conjunt de columnes (fragmentació horitzontal), distribuint les dades de les files d'una

taula entre diferents espais en funció de les columnes a les quals pertanyen (fragmentació vertical), i fins i tot agrupant dades amb característiques comunes, com és el cas de la distribució de les dades agrupades per dimensions en els *data warehouse*.

Un bon disseny de les particions permet reduir la càrrega de treball dels components del maquinari del sistema. Els motius són que la partició de taules facilita que diferents transaccions s'executin concurrentment, fet que incrementa el rendiment de l'SGBD i que les consultes a taules de grans dimensions es puguin resoldre mitjançant consultes més simples que s'executen de manera concurrent. Addicionalment, en el cas que la base de dades estigui distribuïda, les particions permeten apropar i adequar les dades a les necessitats dels usuaris/aplicacions, fet que fomenta el paral·lelisme i disminueix el tràfic de dades per mitjà de la xarxa de comunicacions. També permet distribuir les dades en dispositius d'emmagatzematge més o menys ràpids segons la importància de les dades.

1.1.5. Altres mètodes

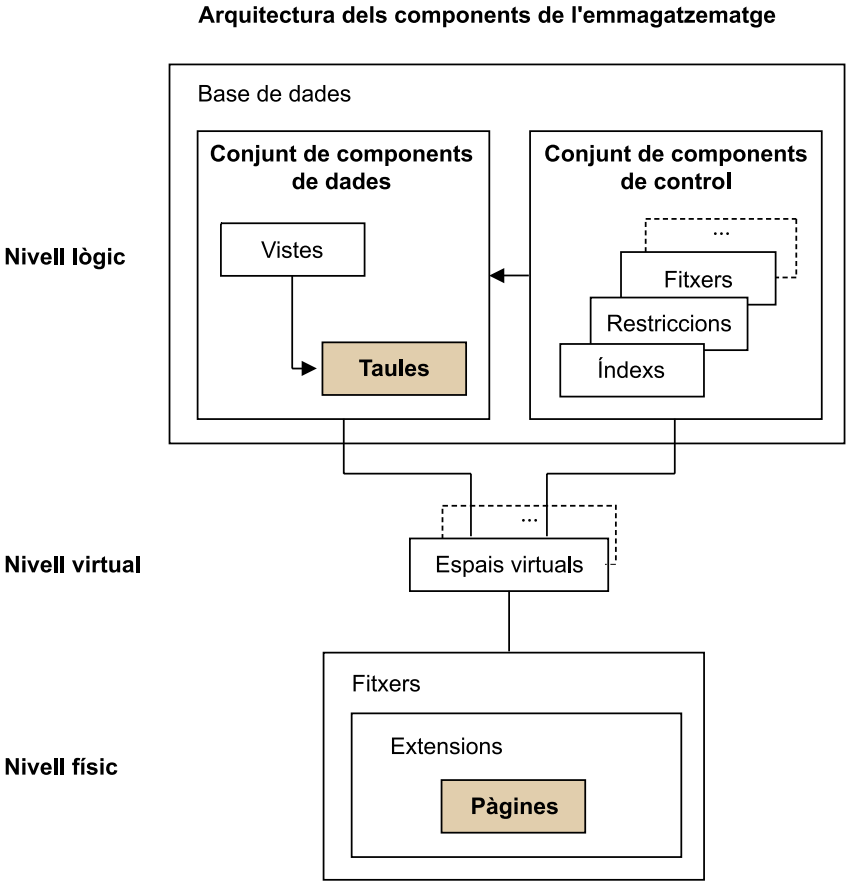
També hi ha altres mètodes per fer l'accés a les dades més eficient. A més, és important mantenir-se degudament actualitzat sobre les noves versions d'SGBD, ja que cada nova versió pot afegir més elements per realitzar un bon disseny físic i un ajustament posterior de la base de dades.

Un altre mètode que pot ser utilitzat en el disseny físic és la compressió de dades, que permet reduir la mida requerida per emmagatzemar les dades i, en conseqüència, millorar el temps de resposta a l'hora de recuperar-les. En utilitzar-lo, s'ha de tenir en compte que fer-ne ús implica una petita sobrecàrrega a l'hora de descomprimir les dades quan es recuperen i a l'hora de comprimir-les quan s'actualitzen. Un altre dels mètodes utilitzats és el *data striping*, que permet distribuir dades que s'han de consultar de manera atòmica en diferents dispositius d'emmagatzematge no volàtil per permetre un nivell més elevat de paral·lelisme i de balanceig de càrrega. Normalment, aquest mètode permet incrementar el nombre de transaccions per segon. Una altra manera de millorar la disponibilitat de la base de dades és el *mirroring*, que permet reproduir (o replicar) la base de dades. Aquesta reproducció obliga que es mantinguin les còpies actualitzades i exigeix més espai d'emmagatzematge. Per això, el *mirroring* és convenient quan disposem de dades que s'actualitzen molt rarament.

1.2. Components d'emmagatzematge d'una base de dades

Abans de poder abordar el disseny físic d'una base de dades és important conèixer com emmagatzema i gestiona les dades un SGBD. La figura següent mostra, de manera simplificada, l'arquitectura d'emmagatzematge que utilitza.

Figura 1. Arquitectura dels components d'emmagatzematge d'un SGBD



Tal com es pot veure a la figura, l'emmagatzematge de dades persistents en una base de dades es realitza en una arquitectura de tres nivells:

- **Nivell lògic**: són els elements de la base de dades amb què treballa l'usuari (taules, vistes, índexs, restriccions d'integritat, etc.).

- **Nivell físic**: són les estructures de dades utilitzades per emmagatzemar les dades de la base de dades en dispositius no volàtils. Aquest nivell inclourà els fitxers, les extensions i les pàgines.

- **Nivell virtual**: és un nivell intermedi entre els nivells físic i lògic que proporciona a l'SGBD una visió simplificada del nivell físic. Això facilita el disseny físic de la base de dades.

Des d'un punt de vista teòric, en el model relacional, les dades s'emmagatzemen en **relacions**, que normalment corresponen a conceptes del domini d'interès. Cada relació conté **tuples**, que es corresponen a les instàncies (o ocurrències) d'un concepte. Les tuples estan compostes per **atributs**, que són els que permeten representar les característiques (o propietats) dels conceptes. En implementar el model anterior en un SGBD relacional, les relacions es representen mitjançant **taules**, que contenen diferents **files**, que corresponen a les tuples de la relació. Cada taula té un conjunt de **columnes**, que repre-

senten els atributs del concepte que s'ha de representar. Quan s'implementen les taules en **fitxers**, les files es representen mitjançant **registres** i les columnes mitjançant **camps**. La taula següent mostra un resum d'aquesta nomenclatura i la relació entre termes. Al llarg del mòdul farem servir principalment els termes corresponents a les dues últimes files.

Taula 1. Nomenclatura

Nivell	Concepte	Instància	Característica
Model relacional	Relació	Tupla	Atribut
SQL	Taula	Fila	Columna
Físic	Fitxer	Registre	Camp

Ja coneixeu els components del nivell lògic. A continuació introduirem els components dels nivells virtual i físic, als quals conjuntament anomenarem components d'emmagatzematge, perquè controlen la disposició física de les dades en el suport de l'emmagatzematge no volàtil (en general, disc).

1.2.1. Nivell físic

Les dades s'emmagatzemen en suports no volàtils controlats pel sistema operatiu (SO) de la màquina en què s'allotgen. L'SO és l'encarregat d'efectuar les operacions de lectura i escriptura física de les dades. Els SGBD aprofiten les rutines existents de l'SO per llegir i escriure les dades de les bases de dades.

L'SO gestiona les dades a partir d'unes estructures globals anomenades **fitxers**. Normalment l'SO no reserva una gran quantitat d'espai destinat a satisfer totes les necessitats futures d'emmagatzematge al dispositiu d'emmagatzematge no volàtil, sinó que realitza una assignació inicial i va adquirint més espai a mesura que ho necessita. La unitat d'adquisició s'anomena **extensió**. Una extensió és una agrupació de pàgines consecutives en el dispositiu d'emmagatzematge no volàtil. Al seu torn, les pàgines són el component físic més petit. Les **pàgines** són els elements que finalment contenen i emmagatzemen les dades del nivell lògic.

La pàgina és el component més important del nivell físic, ja que és la unitat mínima d'accés i de transport del sistema d'entrada i sortida (E/S) d'un SGBD. Això vol dir que:

- La pàgina és la unitat mínima de transferència entre la memòria externa (no volàtil) i la memòria interna (o memòria principal).

- En els SGBD l'espai en el dispositiu d'emmagatzematge no volàtil sempre s'assigna en un nombre múltiple de pàgines.

- Cada pàgina pot ser adreçada individualment.

El fet que la pàgina sigui la unitat mínima de transferència té implicacions importants en el disseny físic. Quan treballem amb un SGBD és important saber amb quina mida de pàgina treballa, per conèixer quantes files d'una taula o entrades d'un índex cabran a cada pàgina. Fer una bona gestió de les pàgines d'una base de dades pot estalviar molts accessos d'E/S innecessaris i fer-ne incrementar enormement el rendiment. En veurem alguns exemples al llarg d'aquest mòdul.

Les pàgines poden classificar-se, principalment, en pàgines de dades i pàgines d'índex. Les primeres contenen dades (entre altres) de taules o vistes materialitzades, mentre que les segones contenen dades sobre els índexs. Cada tipus de pàgina requereix una estructura interna de pàgina diferent.

L'extensió és una agrupació de pàgines consecutives que l'SO proporciona per emmagatzemar dades. Quan l'SGBD detecta que es necessita més espai, l'SO atorga una nova extensió. Les diferents extensions d'un mateix fitxer no tenen per què ser consecutives en el dispositiu d'emmagatzematge no volàtil. L'adquisició de noves extensions és automàtica i transparent per als usuaris de la base de dades.

Un fitxer està format per un conjunt d'extensions i és la unitat lògica que fa servir l'SO per gestionar l'espai en els dispositius d'emmagatzematge no volàtil.

1.2.2. Nivell virtual

En aquest punt ens podríem preguntar: realment és necessari el nivell virtual en una base de dades? Què proporciona realment aquest nivell?

En una primera aproximació podríem pensar que cada taula s'emmagatzema en un fitxer, i que cada fitxer només emmagatzema dades d'una taula. És a dir, podríem pensar que sempre hi ha una relació biunívoca entre taula i fitxer, amb la qual cosa desapareixeria la necessitat d'un nivell intermedi que faci correspondre components lògics amb components físics, ja que aquesta correspondència seria fixa.

La realitat, però, és més complexa que la suposició anterior. A continuació en presentem alguns casos:

a) Hi pot haver taules molt grans en què ens pot interessar definir particions, de manera que cada partició s'emmagatzemi en un fitxer diferent (potser localitzat en un dispositiu d'emmagatzematge) per millorar-ne l'accés.

b) Per contra, podem trobar un conjunt de taules molt petites, que convingui guardar en un mateix fitxer amb l'objectiu de no consumir tants recursos del sistema.

c) De vegades, una taula pot estar tan relacionada amb una altra o altres que en la majoria dels casos l'accés a la primera, per part dels usuaris o de les aplicacions, comporta també un accés a la segona, o a les altres. En aquests casos, ens interessarà que les dades d'aquestes taules es trobin tan a prop físicament com sigui possible per minimitzar el temps d'accés global i augmentar-ne el rendiment de la gestió. En definitiva, darrere d'aquesta opció, la idea és tenir les operacions de combinació (en anglès, *join*) entre les taules relacionades preconstruïdes en el dispositiu d'emmagatzematge no volàtil.

d) Es poden fer servir taules que, a més de contenir columnes amb dades de tipus tradicional (quantitats numèriques, cadenes de caràcters, dates, etc.), en tenen d'altres que emmagatzemen tipus de dades diferents, com gràfics, imatges o alguns minuts d'àudio o de vídeo. Aquestes dades, que necessiten molts més *bytes* per a ser emmagatzemades, es denominen objectes grans (en anglès *large objects*, *LOB*). Aquests objectes grans s'emmagatzemen en fitxers diferents als que es fan servir per a les dades de tipus més tradicional, per millorar els temps d'accés.

e) Tot i que només hem esmentat les taules, hi ha altres components, com per exemple, els índexs. Els fitxers que guarden informació sobre els índexs, de manera similar al cas anterior, estan estructurats internament de manera diferent per poder obtenir-ne el màxim rendiment.

Tots aquests exemples han de servir per ser conscients de la utilitat de disposar del nivell virtual. Ens proporciona un grau elevat de flexibilitat (o independència) per assignar components lògics als components físics, ja que no sempre es vol establir una correspondència biunívoca entre taula i fitxer. Concretament, ens permet decidir on s'emmagatzemarà cada taula o partició de taula. D'aquesta manera escollim en quina màquina, en quin dispositiu d'emmagatzematge no volàtil o en quin tros d'aquest tindrem les diferents dades de la base de dades. El nivell virtual és normalment conegut sota les denominacions de *tablespace* o *dbspace* en els SGBD comercials.

2. Espai per a taules

Els espais per a taules (*tablespace* en anglès) permeten definir on s'emmagatzemaran les dades dels objectes de la base de dades. Un cop creat, un *tablespace* es pot referenciar pel seu nom quan es crea un nou objecte a la base de dades.

L'ús dels *tablespaces* permet tenir més control sobre com es distribuiran les dades de la base de dades en els dispositius d'emmagatzematge no volàtil disponibles (en general, discos). Faciliten en gran mesura la redistribució de les dades de la base de dades en cas de necessitat, com, per exemple, quan un *tablespace* s'ubica en un disc que està a punt d'omplir-se. En aquest cas, es podria canviar fàcilment la ubicació dels objectes del *tablespace* assignant-li simplement una nova ubicació física. A part, els *tablespaces* ens permeten distribuir fàcilment els diferents objectes de la base de dades en diferents discos per millorar el rendiment. Així doncs, podríem situar les dades d'un índex que s'utilitzi freqüentment en un disc d'alta velocitat (com, per exemple, en un disc d'estat sòlid) per accelerar-ne l'accés. D'altra banda, podríem ubicar les taules que no es facin servir freqüentment en un disc més lent.

Tal com hem comentat, els *tablespaces* poden assignar-se a qualsevol objecte de la base de dades. En cas d'assignar-se a taules, vistes materialitzades i índexs, indica l'espai físic on s'emmagatzemaran les dades d'aquests objectes. En cas d'assignar-se a una base de dades, es pot utilitzar per a diferents finalitats: per emmagatzemar les dades del catàleg de la base de dades, per emmagatzemar les dades temporals de la base de dades, com *tablespace* per defecte, etc. La necessitat d'emmagatzemar dades temporals per part de l'SGBD pot esdevenir, per exemple, en la resolució de les consultes formulades pels usuaris i també en la gestió de vistes. Aquestes dades temporals es guarden en els anomenats espais de taules temporals (en anglès, *temporary tablaspaces*).

Com que no estan inclosos en l'estàndard SQL, la definició i l'ús de cada *tablespace* és diferent en cada SGBD. Per tant, en cada cas s'haurà de comprovar la semàntica i la sintaxi concreta en la documentació de cada SGBD. A continuació veurem com definir-los i utilitzar-los a Oracle i PostgreSQL.

2.1. Definició i ús d'espai de taules a Oracle

A continuació mostrem una versió simplificada de la sintaxi de creació de *tablespaces* a Oracle:

Notació

A la notació que farem servir a partir d'ara s'utilitzaran:

- Paraules en majúscules per denotar clàusules SQL.
- Paraules en minúscules per denotar noms de variables que han de ser informats per l'usuari.
- Claudàtors [] per a indicar opcionalitat (clàusules que poden no informar-se).
- Claus {} per a indicar un conjunt de clàusules separades per «|» de les quals només se'n pot triar una.

```
CREATE TABLESPACE tablespace_name
   DATAFILE 'filename' [SIZE nn]
   [AUTOEXTEND {OFF|ON [NEXT integer] [MAXSIZE {UNLIMITED|integer}]]
   [BLOCKSIZE integer]
   DEFAULT [{COMPRESS|NOCOMPRESS}]
   ...
```

Oracle permet una gran capacitat de personalització a l'hora de crear els espais de taules. En particular, els elements que permet definir són:

- `Tablespace_name`: nom del *tablespace*. És el nom que s'utilitzarà per identificar el *tablespace* i assignar-lo als diferents objectes de la base de dades.
- `Filename`: permet definir el fitxer on s'emmagatzemarà la informació associada al *tablespace* i la mida d'aquest.
- `Autoextend`: indica què cal fer quan el fitxer assignat al *tablespace* s'ompli. Es pot no fer res (`OFF`), estendre el fitxer perquè permeti noves dades (`ON NEXT nn`, en què `nn` indica la mida de l'extensió). També se'n pot indicar la mida (`size`) màxima del fitxer.
- `Blocksize`: permet definir la mida de pàgina que utilitzarà el *tablespace* per emmagatzemar la informació.
- `Compress/nocompress`: indica si les dades s'han d'emmagatzemar comprimides, o no.

Acrònims de mides

Quan s'indiquen mides s'ha d'indicar la unitat de mida utilitzant un acrònim. Les unitats que es poden utilitzar i els seus acrònims són: K per indicar *kilobytes*, M per indicar *megabytes*, G per indicar *gigabytes* i T per indicar *terabytes*.

Definició de mida de pàgina

A Oracle la mida de pàgina es defineix quan es crea la base de dades. No obstant això, aquesta grandària de pàgina per defecte pot canviar-se per tenir més flexibilitat a l'hora de treballar amb *tablespaces*.

Per exemple, la sentència següent crearia un *tablespace* per a emmagatzemar els elements de la base de dades que no requereixin un accés gaire ràpid. Com podem veure, s'indica la ubicació del fitxer, el nom del *tablespace* (`tbs_slow_access`), la mida inicial del fitxer (20 *megabytes*) i que el sistema ha d'estendre el fitxer en increments de 10 *megabytes* quan aquest s'ompli fins a arribar a un màxim de 200 *megabytes*.

```
CREATE TABLESPACE tbs_slow_access
   DATAFILE '/dev/slowdisks/disc1/tbs_slow_access.dat'
      SIZE 20M
```

```
       AUTOEXTEND ON NEXT 10M MAXSIZE 200M;
```

Un cop creat el *tablespace*, si volguéssim assignar-li una taula, només hauríem d'indicar-ho a la definició. Així, per exemple, la sentència següent assigna la taula ciutat (`City`) al *tablespace* denominat `tbs_slow_access`.

```
CREATE TABLE City (
   cityZip  CHAR(5),
   cityName VARCHAR(50)
) TABLESPACE tbs_slow_access;
```

2.2. Definició i ús d'espai de taules a PostgreSQL

La definició de *tablespaces* a PostgreSQL és més simple i permet menys personalització.

```
CREATE TABLESPACE tablespacename [OWNER nomusuari] LOCATION 'directory'
```

Com podem veure, en aquest cas només es permet definir el nom del *tablespace*, el propietari i la ubicació. Molts dels paràmetres que hem vist al *tablespace* d'Oracle també poden configurar-se a PostgreSQL, però amb sentències separades. Per exemple, la mida de la pàgina es pot modificar quan s'instal·la (o recompila) l'SGBD mitjançant el paràmetre `with-blocksize`.

Si haguéssim de crear un *tablespace* com el de la secció anterior i assignar-lo a la taula ciutat (`City`) ho faríem de la manera següent:

```
CREATE TABLESPACE tbs_slow_access
  LOCATION '/dev/slowdisks/disc1/tbs_slow_access.dat';

CREATE TABLE City (
   cityZip  CHAR(5),
   cityName VARCHAR(50)
) TABLESPACE tbs_slow_access;
```

3. Índexs

La gran majoria d'operacions sobre una base de dades es realitzen per valor. Al seu torn, l'accés per valor pot ser directe o seqüencial. Més concretament:

- L'**accés directe per valor** consisteix a obtenir totes les files que contenen un determinat valor per a una columna.

- L'**accés seqüencial per valor** consisteix a obtenir diverses files per l'ordre dels valors d'una columna.

A continuació es mostren exemples sobre una taula de clients que conté les columnes següents: codi del client (clau primària), nom del client, població i edat.

a) Exemples d'accés directe per valor:

```
SELECT * FROM client WHERE poblacio='Arenys de Mar'

UPDATE client SET edat=50 WHERE codi_client=20

DELETE FROM client WHERE codi_client=100
```

b) Exemples d'accés seqüencial per valor:

```
SELECT * FROM client ORDER BY edat

SELECT * FROM client WHERE edat>=40 AND edat<=50

DELETE FROM client WHERE poblacio IN ('Barcelona','Tarragona')
```

Els exemples previs mostren accessos per valor (seqüencials o directes) sobre una única columna. Però també s'hi pot accedir pel valor que mostren diverses columnes. Aquests accessos es denominen **accessos per diversos valors**, i poden ser directes, seqüencials i mixtos (en el cas que una mateixa operació o sentència combini alhora accessos per valor directe i seqüencial). A continuació es mostren alguns exemples sobre la taula de clients:

a) Accés directe per diversos valors:

```
SELECT * FROM client WHERE poblacio='Arenys de Mar' AND edat=30

UPDATE client SET edat=edat+1 WHERE codi_client='Barcelona' AND edat=40
```

b) Accés seqüencial per diversos valors:

```
SELECT * FROM client ORDER BY edat, poblacio

SELECT * FROM client WHERE edat>30 AND codi_client<100
```

c) Accés mixt per diversos valors:

```
SELECT * FROM client WHERE ciutat='Arenys de Mar' ORDER BY edat

DELETE FROM client WHERE edat<25 AND ciutat='Barcelona'
```

Per defecte, i a falta d'estructures de dades que donin suport a les operacions mostrades als exemples, qualsevol operació sobre una taula de la base de dades significarà fer un recorregut de tota la taula i examinar, per a cada fila recuperada, si compleix o no les condicions de cerca. Si aquest sistema ja pot resultar molt costós per si mateix, la situació s'agreuja en funció del nombre de files que conté la taula (per exemple, podrien ser desenes de milers), o si en la petició s'inclouen combinacions de taules (operacions de *join*) i/o agrupacions (consultes amb clàusula `GROUP BY`). En tots aquests casos, l'eficiència pot caure en picat.

Per solucionar aquesta problemàtica s'han creat els **índexs**, que són estructures de dades que permeten millorar el temps de resposta de les peticions que impliquin un accés per valor.

El concepte d'índex no és nou, sinó que són estructures centenàries que formen part de la nostra vida quotidiana. Exemples d'índexs fora de l'àmbit de les bases de dades serien l'índex de capítols d'un llibre, l'índex de conceptes clau d'un llibre esmentat a l'inici d'aquest mòdul i el mapa de lloc d'una web.

Cal destacar que els SGBD creen automàticament alguns índexs per gestionar les restriccions d'integritat de la base de dades. Entre els índexs creats per defecte pels SGBD trobem els creats sobre la clau primària i les claus alternatives.

En la seva versió més simple, un índex permet accedir a les files d'una taula a partir dels valors d'una de les seves columnes. La columna sobre la qual es construeix l'índex s'anomena columna indexada. Cada valor diferent *v* de la columna indexada es fa correspondre amb l'identificador de registre (RID[1]) que apunta a la fila de la taula que té a *v* com a valor de la columna indexada. Les diferents parelles (*v*, *RID*) que s'emmagatzemen en l'índex reben el nom d'entrades de l'índex. Aquesta estructuració de les dades permet accedir-hi ràpidament quan es realitzen accessos directes o seqüencials per valor o es requereix l'ordenació de les files d'una taula d'acord amb els valors de la columna indexada. L'índex descrit en aquest paràgraf respon a un dels tipus d'índexs

(1)El RID (o *record identifier* en anglès) és un identificador que apunta al registre físic on s'emmagatzema la fila d'una taula.

més simples: l'índex únic, en què la columna indexada és una clau candidata, és a dir, el seu valor identifica unívocament cada fila de la taula amb independència que aquesta sigui clau primària o alternativa de la taula.

A partir d'aquí, els índexs es poden anar sofisticant segons el tipus i nombre de columnes indexades i el tipus d'estructura de dades utilitzada en la implementació. A continuació descrivim els diferents tipus d'índex en funció del tipus i nombre de columnes indexades i als apartats següents veurem algunes de les estructures de dades més utilitzades per crear índexs en les bases de dades.

En alguns casos interessarà indexar una taula per una columna que no prengui valors únics, com, per exemple, la població dels clients, que pot repetir valor en diferents files de la taula. Aquests casos impliquen modificar lleugerament l'estructura de l'índex, per exemple, fent que l'entrada de l'índex associada a cada possible valor v de la columna indexada tingui associats N *RID* ($\{RID_1, ..., RID_N\}$, $N>0$), on cada RID_i apuntarà cap a una fila de la taula que té el valor v a la columna indexada.

També ens podrem trobar amb la necessitat de definir índexs sobre múltiples columnes: per exemple, un índex sobre les columnes nom i cognoms de la nostra taula de clients. Aquests índexs poden ser únics o no únics. En aquest cas, cada entrada de l'índex estarà representada per una combinació dels valors indexats i apuntarà al (els) RID de les files de la taula que tenen columnes amb un valor indicat a l'entrada de l'índex. A l'hora de definir l'índex, l'ordre de les columnes pot ser rellevant, és a dir, un índex definit sobre les columnes nom i cognoms pot ser diferent a un índex definit sobre les columnes cognoms i nom, de tal manera que cadascun d'aquests índexs ajudarà a resoldre de manera eficient diferents tipus de consulta.

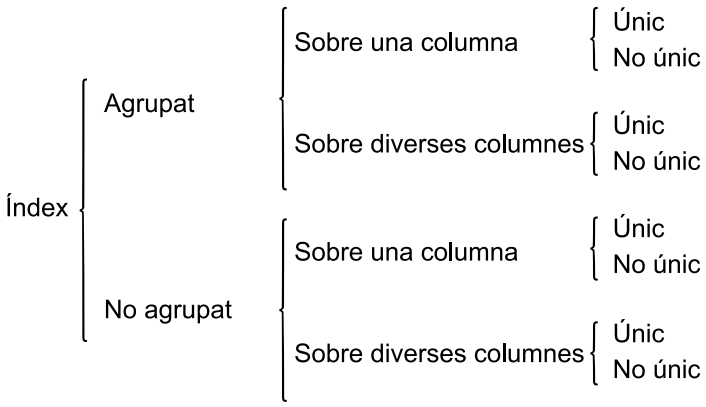
Nombre d'operacions d'E/S

El nombre d'operacions d'E/S és un factor que s'ha de reduir, ja que permet millorar significativament el rendiment del processament de les consultes.

Els índexs comentats fins ara s'anomenen índexs ordenats perquè les columnes indexades (en general, alfanumèriques) permeten l'ordenació dels seus valors, fet que facilita l'estructuració de l'índex. No obstant això, les dades a què fan referència, és a dir, les files de la taula, no tenen per què estar ordenades. Una alternativa a aquests índexs seria emmagatzemar les files de la taula de manera ordenada d'acord amb les columnes indexades. Això és el que es coneix amb el nom d'índex agrupat (o *clustered index* en anglès). Ordenar les files d'una taula pot millorar significativament el temps de resposta en la resolució de peticions que impliquen un accés seqüencial per valor, ja que poden minimitzar el nombre d'operacions d'E/S. Fixeu-vos que només és possible tenir un índex agrupat per taula.

Tal com es mostra a la figura següent, els índexs explicats fins ara es poden classificar en funció de si les dades es guarden de manera ordenada al fitxer que els contenen, del nombre de columnes que s'indexen i de la unicitat de valors d'aquestes.

Figura 2. Classificació d'índexs



A part dels índexs prèviament presentats, tenim altres tipus d'índexs que ens permeten indexar informació no textual, com poden ser els índexs geogràfics. Aquests permeten consultar elements geogràfics dins de mapes basant-se en la seva posició, i faciliten l'ús de funcions, com la distància entre punts o la intersecció de figures geomètriques.

Hi ha diferents estructures de dades que poden utilitzar-se per crear índexs en bases de dades, com ara els arbres B+, les funcions de dispersió (en anglès *hash*), els mapes de bits (en anglès *bitmap*), els índexs parametritzables (GIST i GIN), els arbres R, els Quadtree, etc. A l'apartat següent veurem les estructures de dades més utilitzades per emmagatzemar i gestionar índexs de bases de dades. Després veurem com crear aquests índexs a Oracle i PostgreSQL. A continuació introduirem breument els conceptes bàsics associats a l'optimització de consultes, i acabarem amb una sèrie de recomanacions que ajudin el lector a saber quins índexs s'han de crear en cada cas concret.

3.1. Tipus d'índexs més habituals

L'estructura més utilitzada en els índexs de bases de dades són els arbres B+, perquè es comporten bé en la majoria de situacions. No obstant això, no hi ha un tipus d'índex universal que funcioni bé en tots els casos. Altres tipus d'índexs, com ara els basats en funcions de dispersió o en mapes de bits, poden ser més adequats en certes circumstàncies. A continuació explicarem què són i com funcionen els índexs basats en arbres B+, en funcions de dispersió i en mapes de bits. Així mateix, quan parlem de PostgreSQL, s'introduiran un parell de tipus d'índexs més particulars d'aquest SGBD, els índexs GiST i GIN.

3.1.1. Arbres B+

L'estructura d'arbre B+ és la més utilitzada per implementar índexs en bases de dades relacionals. De fet, de tots els tipus d'índexs que hem presentat, els arbres B+ (tal com els explicarem o variants similars) són els únics que estan disponibles en tots els SGBD relacionals.

Els índexs basats en arbres B+ són índexs ordenats. Per tant, tal com s'indica a la classificació de la figura 2, permeten indexar una o múltiples columnes, independentment de si contenen valors únics o no i de manera ordenada (arbre B+ agrupat) o no ordenada (arbre B+ no agrupat). Aquests índexs poden ajudar a resoldre eficientment qualsevol accés per valor, sigui directe o seqüencial. També són especialment útils per retornar resultats de manera ordenada, com per exemple en les operacions SQL que contenen la clàusula `ORDER BY`.

Els arbres B+ són un tipus particular d'arbre de cerca, l'objectiu primordial dels quals és minimitzar les operacions d'E/S en les cerques.

Un arbre B+ es compon de nodes que estan interrelacionats entre ells. Les relacions entre nodes són dirigides i relacionen un node pare (l'origen de la relació) amb un node fill (el destí de la relació). De fet, es tracta de relacions d'ordre entre els valors indexats que es troben distribuïts entre els diferents nodes de l'arbre. En un arbre no es permeten cicles, és a dir, un node no pot estar interrelacionat amb si mateix, ni directament ni indirecta. Cada node de l'arbre, excepte un node especial anomenat arrel, té un node pare i diversos (zero o més) nodes fill. El node arrel no té pare, els nodes que no tenen fills es diuen nodes fulla i els nodes que no són fulles es diuen nodes interns.

Un arbre B+ té associades les característiques següents:

- **Els nodes fulla i els nodes no fulla** (node arrel i nodes interns) **tenen estructures diferents**. Els nodes no fulla contenen valors per dirigir la cerca cap als nodes fulla i apuntadors cap a nodes fill (que poden ser nodes interns o nodes fulla). Per la seva banda, els nodes fulla contenen les entrades de l'índex (és a dir, les parelles valor i RID). A més, cada node fulla pot contenir fins a dos apuntadors més. Un apuntador al node fulla següent i un apuntador al node fulla anterior. Aquests apuntadors tenen com a objectiu facilitar la resolució d'accessos seqüencials (parcials o complets) per valor.

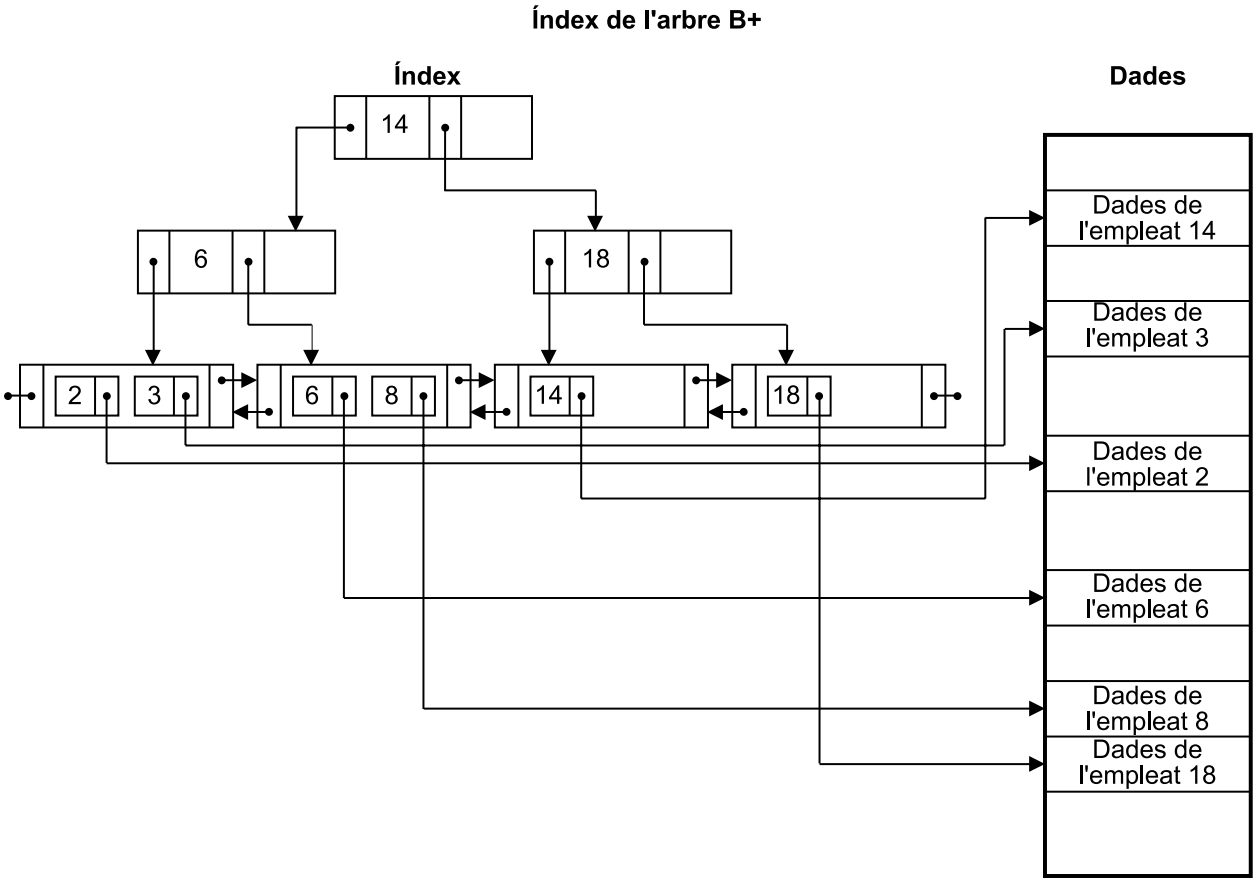
- **Un arbre B+ té un nombre d'ordre *d* que indica la capacitat dels seus nodes i**, en conseqüència, indica també **el nombre de nodes fill que pot tenir cada node intern**. Els nodes tenen com a màxim *2d* valors o entrades, depenent de si es tracta d'un node intern o fulla, respectivament. Addicionalment, els arbres B+ també imposen una ocupació mínima per als nodes. Aquesta ocupació mínima és del 50%. Hi ha un node, en concret el node arrel, que està exempt de complir aquesta condició. Per tant, en un arbre B+ d'ordre *d*, tots els nodes interns (a excepció de l'arrel) tenen una ocupació mínima de *d* valors (i un mínim de *d+1* apuntadors a nodes fill) i una ocupació màxima de *2d* valors (i un nombre màxim de *2d+1* apuntadors a nodes fill). En el cas de nodes fulla, tindran entre *d* i *2d* entrades (i un màxim de dos apuntadors). La condició d'ocupació mínima causa que un arbre B+ sigui equilibrat.

- Tots **els valors d'un node intern han d'estar presents en algun node fulla i no poden estar repetits**.

- **Els nodes fulla han de contenir totes les entrades**. És a dir, tots els valors que en un moment donat existeixen de la columna (o columnes) sobre la qual es construeix l'índex, tenen l'entrada corresponent en un node fulla.

La figura següent mostra un exemple d'índex d'arbre B+ definit sobre una columna de tipus numèric:

Figura 3. Exemple d'arbre B+ per indexar una columna numèrica



Estructura d'un arbre B+

La figura mostra un índex arbre B+ d'ordre *d*=1 que serveix per accedir a dades d'empleats segons el valor de l'atribut `IDEmpleat`. Fixeu-vos que els RID que apunten a les dades només es troben en els nodes fulla, que els nodes interns serveixen per buscar els valors de les fulles i que el fet que les fulles estiguin connectades entre elles facilita l'accés seqüencial per valor.

Suposeu que la figura anterior es correspon amb un índex únic definit sobre la clau primària (`IDEmpleat`) d'una taula que guarda dades sobre empleats. Suposeu també que un usuari planteja la consulta següent:

```
SELECT *
 FROM EMPLEAT
 WHERE IDEmpleat > 16;
```

En aquest cas l'SGBD detecta que s'estan filtrant els valors de la consulta en funció de la columna `IDEmpleat` per a la qual hi ha un índex en forma d'arbre B+. Suposem que en aquest punt l'SGBD decideix fer servir l'índex. En aquest cas, l'SGBD carregaria en memòria el node arrel de l'arbre B+. A continuació consultaria l'únic valor que conté (14) i el compararia amb el valor que es vol cercar (16). Com que 14 és menor que 16, s'ha de seguir l'apuntador de la dreta del valor per accedir al node següent (recordeu que l'objectiu dels nodes intermedis és conduir la cerca cap als nodes fulla). Després de carregar-se en memòria, es consultaria el valor que conté aquest node, que és 18. Com que 18 és més gran que 16, se seguirà l'apuntador de l'esquerra del valor per accedir al node fulla. Aquest node fulla ha de contenir valors inferiors a 18. En arribar al node fulla es llegeix l'única entrada que conté. Aquesta entrada té el valor de 14, que com que és inferior de 16 no és d'interès per a la consulta. Com que el node fulla no conté més entrades, per tornar els valors amb identificador d'empleat superior a 16 se segueix l'apuntador al node fulla següent de l'arbre. En aquest node es troba una nova entrada de l'índex amb el valor de 18. Com que 18 és més gran que 16 el sistema utilitza el RID de l'entrada per accedir a les dades de l'empleat i tornar-les a la consulta. Com que no hi ha més entrades al node fulla i tampoc altres nodes fulla a continuació, la consulta acaba i es retornen les dades de l'empleat amb identificador d'empleat 18.

Els arbres B+ són útils per:

- **Realitzar accessos directes per valor**: primer cal localitzar la fulla que té l'entrada del valor buscat, i després, utilitzar el RID de l'entrada per trobar les dades a què es vol accedir.

- **Realitzar accessos seqüencials per valor**: primer cal localitzar la primera entrada de la seqüència i utilitzar el seu RID per tornar el primer resultat. Posteriorment s'obtindran els valors següents de manera ordenada i els seus RID navegant pels nodes fulla de l'índex.

Les explicacions prèvies assumeixen que l'arbre B+ és no agrupat. En el cas que fos agrupat, podem guanyar eficiència, especialment en els accessos seqüencials per valor. En aquest cas, un cop s'ha localitzat la primera entrada d'interès en un node fulla de l'arbre B+, seguint el seu RID podem accedir al primer resultat en el fitxer que guarda les files que estan sent indexades. Com que aquest fitxer està ordenat físicament segons el valor que pren la columna (o columnes) sobre la qual s'ha construït l'arbre B+, n'hi ha prou a fer la lectura de les pàgines d'aquest fitxer i, per tant, no cal consultar els nodes fulla restants de l'arbre B+.

De manera similar, en les explicacions, la taula sobre la qual s'ha construït l'arbre B+ es troba en un únic fitxer que només emmagatzema dades d'aquesta taula. Alguns SGBD (per exemple, aquest seria el cas d'Oracle i PostgreSQL) permeten crear índexs particionats sobre taules que han estat, al seu torn, fragmentades horitzontalment. En aquest cas, cada fragment de la taula es

troba emmagatzemat en un fitxer diferent que només emmagatzema dades d'aquesta taula, i cada un d'aquests fragments disposa del seu índex en forma d'arbre B+, que únicament indexa les dades contingudes en el fragment. Per finalitzar, un altre tipus d'índex que pot ser útil en el disseny físic és el basat en funcions. Aquest tipus d'índex no indexa els valors d'una columna (o un conjunt de columnes), sinó els valors d'una funció (o la composició d'un conjunt d'elles) que s'aplica sobre una o més columnes. Això pot ser molt útil quan es fa un ús intens d'una determinada funció sobre les mateixes columnes d'una taula.

Cost de localització d'una entrada

Una pràctica habitual a l'hora de crear arbres B+ és fer coincidir els nodes de l'arbre amb la mida de la pàgina. Així, el nombre d'accessos d'E/S necessaris per localitzar qualsevol entrada de l'arbre és *h*, en què *h* és l'altura de l'arbre.

Com que l'altura de l'arbre és directament proporcional al nombre d'accessos necessaris per trobar una entrada, és important crear arbres que tinguin una alçada tan petita com sigui possible. Per tant, cal que els nodes de l'arbre siguin grans però sense sobrepassar la mida d'una pàgina, perquè, en cas contrari, no es podrien consultar amb una única operació d'E/S. Addicionalment, ens interessa que els nodes estiguin tan plens com sigui possible, és a dir, ens interessa que l'ordre *d* de l'arbre sigui tan gran com sigui possible. La idea no és només que els nodes siguin grans, sinó també que estiguin plens, és a dir, que tinguin un índex d'ocupació elevat.

Exemple: càlcul de l'ordre d'un arbre B+ per ajustar els nodes a les pàgines de dades

Suposeu que el nostre SGBD treballa amb pàgines de 8 Kb (com en el cas de PostgreSQL) i que volem indexar la columna DNI d'una taula que guarda informació de persones. La columna DNI ocupa 8 *bytes* i és la clau primària de la taula persones. Considerem que la mida del RID i dels apuntadors als nodes de l'arbre és de 4 i 3 *bytes*, respectivament.

Segons aquestes dades, el càlcul de l'ordre *d* de l'arbre B+ que permet una ocupació màxima dels nodes, s'efectua de la manera següent:

- **Node fulla**: en un node fulla hi caben un màxim de *2d* entrades i fins a dos apuntadors a nodes fulla. En el nostre exemple, per emmagatzemar cada entrada necessitem 12 *bytes* (8 + 4). I per emmagatzemar els dos apuntadors, 6 *bytes* (2 * 3). Cada node fulla té una mida de 8.192 *bytes* (8 Kb). En conseqüència:
  $2d * 12 + 6 = 8192 \Rightarrow d = 341{,}0833 \Rightarrow d = 341$, atès que *d* és un nombre enter.

- **Node no fulla**: en un node no fulla hi caben un màxim de *2d* valors i *2d + 1* apuntadors a nodes fill. Cada node no fulla té una mida de 8.192 *bytes* (8 Kb). En conseqüència:
  $2d * 8 + (2d + 1) * 3 = 8192 \Rightarrow d = 372{,}2272 \Rightarrow d = 372$, atès que *d* és un nombre enter.

L'ordre *d* d'un arbre B+ és únic i ha de satisfer les necessitats d'emmagatzematge dels nodes fulla i no fulla. Per tant, en el cas del nostre exemple, l'ordre màxim *d* de l'arbre B+ és 341.

A més, donat l'ordre de l'arbre B+ i les propietats comentades anteriorment, és fàcil calcular quantes files de dades es poden indexar en un índex en forma d'arbre B+ en funció de la seva altura. A continuació presentem un exemple que indica el nombre de files que pot indexar l'arbre B+ anterior amb altures 1, 2 i 3, respectivament.

Exemple: càlcul del nombre de files que es poden arribar a indexar en funció de l'altura de l'arbre B+

Continuant amb l'arbre B+ anterior, l'ordre *d* del qual és 341, la qüestió és quantes files diferents de la taula de persones podem arribar a indexar com a màxim. Això dependrà de l'altura (*h*) de l'arbre B+.

- **Si h=1**, l'arbre B+ té un únic node (que és alhora node arrel i fulla) que conté 341 * 2 entrades, és a dir, permet indexar 682 persones diferents.
- **Si h=2**, l'arbre B+ té un node arrel que conté 341 * 2 valors i 341 * 2 + 1 apuntadors a nodes fill, que alhora són nodes fulla. Per tant, tenim 683 nodes fulla, cada un amb 682 entrades. En conseqüència, tindrem un total de 683 * 682 entrades, és a dir, l'arbre B+ permet indexar 465.806 persones diferents.
- **Si h=3**, l'arbre B+ té un node arrel que conté 682 valors i 683 apuntadors a nodes interns. Tenim, doncs, 683 nodes interns. Cada node intern, al seu torn, té 682 valors i 683 apuntadors a nodes fulla. Per tant, tenim un total de 466.489 (683^2) nodes fulla. El nombre total d'entrades serà 466.489 * 682, és a dir, l'arbre B+ permet indexar 318.145.498 persones diferents (noteu que aquest nombre ja excediria el total d'habitants d'Espanya).

L'altura d'un índex en forma d'arbre B+ condiciona el nombre d'operacions d'E/S necessàries per donar resposta als accessos per valor. En particular, en una consulta d'accés directe per valor, només cal consultar *h* nodes, en què *h* és l'altura de l'arbre, per identificar si hi ha alguna fila d'una taula que satisfà els criteris de cerca. En cas que sigui així, consultar les dades de la fila requerirà una nova operació d'E/S per carregar les dades de la fila en memòria. A continuació es mostra mitjançant un exemple el nombre d'operacions d'E/S necessàries per resoldre una consulta SQL en l'índex creat anteriorment.

Exemple: càlcul d'operacions necessàries d'E/S per respondre una consulta d'accés directe per valor utilitzant un arbre B+

Imaginem-nos que el nostre arbre B+ (vist en els dos exemples anteriors) té una alçada *h* igual a 3, ordre *d* de 341, i que els nodes presenten ocupació màxima. Suposem que volem resoldre la consulta següent:

```
SELECT *
 FROM persones
 WHERE DNI = '46742377';
```

Quantes operacions d'E/S són necessàries per localitzar la persona amb DNI 46742377?

Suposant que a la taula de persones hi hagi una persona amb el DNI indicat, serien necessàries 4 operacions d'E/S, 3 a l'índex, per localitzar el node fulla que conté l'entrada que ens interessa, més 1 operació d'E/S addicional en el fitxer de dades per recuperar les dades de la persona desitjada. Si no hi ha cap persona amb el DNI indicat serien necessàries 3 operacions d'E/S (les de l'índex). Imaginem-nos el nombre d'operacions d'E/S per localitzar aquesta persona entre un total de 318.145.498 persones si no tinguéssim l'arbre B+. O fins i tot pitjor, imaginem què passaria si la persona amb el DNI que busquem (46742377) no existís a la base de dades.

Cost de manteniment

Els arbres B+ són arbres equilibrats. Aquest tipus d'arbres tenen característiques molt interessants que faciliten la cerca d'elements, però mantenir els arbres equilibrats té un cost associat.

Com ja hem vist, els arbres B+ han de satisfer la restricció que tots els nodes (excepte el node arrel) han de contenir com a mínim *d* valors o entrades, depenent del tipus de node, en què *d* és l'ordre de l'arbre. Aquest fet requereix que la modificació dels valors de la columna indexada i la inserció i eliminació de files de la taula indexada impliquin reestructurar l'índex sovint. Per això en taules i/o columnes amb una freqüència d'actualització molt alta l'ús d'aquest tipus d'índex pot tenir un cost de manteniment molt elevat.

3.1.2. Taules de dispersió

Un altre tipus d'estructura de dades que s'utilitza en la creació d'índexs és la de dispersió (o *hash* en anglès). Aquest tipus d'estructures permet un accés directe per valor a les dades molt eficient, fins i tot més eficient que fent servir arbres B+ en molts casos. No obstant això, aquest tipus d'índexs no donen suport a l'accés seqüencial per valor, és a dir, no resulta d'utilitat en cerques que utilitzin operadors diferents de la igualtat (>, >=, <, <=, <>).

La filosofia d'aquests índexs és la mateixa que hi ha darrere de les taules de dispersió utilitzades per emmagatzemar dades en memòria interna. Tot i que en el nostre cas l'objectiu és utilitzar funcions de dispersió per minimitzar el nombre d'operacions d'E/S.

En els índexs basats en dispersió, hi ha una funció de dispersió (normalment anomenada *h*) que identifica la pàgina de dades en què s'ubicaran les files de la taula indexada. La funció de dispersió *h* és una funció que té per domini el conjunt dels possibles valors de la columna indexada i, com a rang, les referències de les pàgines disponibles a la base de dades per a la taula indexada: $h(v) \rightarrow p$.

Per a cada fila de la taula indexada la funció de dispersió calcula, a partir del valor de la seva columna indexada, la pàgina de dades on s'haurà d'emmagatzemar la fila.

Com que el valor de la funció de dispersió estableix la ubicació de les dades de la taula, en principi només és possible utilitzar un índex de dispersió per taula. No obstant això, en alguns casos s'utilitzen taules auxiliars per permetre una indirecció i, així, suportar l'ús de més d'un índex de dispersió per taula, a canvi d'incrementar en una unitat el nombre d'operacions d'E/S necessàries en la recuperació de les dades.

A causa de les seves característiques, no té sentit que els índexs de dispersió siguin agrupats, ja que la funció de dispersió retorna directament la pàgina on emmagatzemar les dades i no garanteix que dades amb valors propers tinguin valors d'*h* pròxims. Hi ha altres característiques que també en complicarien l'agrupació, com la gestió de sinònims que veurem més endavant.

Normalment, el nombre de valors possibles que retorna la funció de dispersió ha de ser menor que el nombre de valors possibles de la columna indexada. Això es deu a dos fets:

1) El nombre de possibles valors pot ser molt superior al nombre de pàgines disponibles.

2) La majoria dels valors possibles no s'utilitzen.

Suposeu ara un identificador d'empleat de 32 bits. Els possibles valors d'aquest identificador són superiors a 4.000 milions (concretament, igual a 4.294.967.296 = 2^{32}). En cas de tenir una funció de dispersió que retorna el mateix nombre de valors que la columna que indexa, necessitaríem 4.294.967.296 pàgines per emmagatzemar la taula indexada. Si la mida de pàgina del nostre SGBD fos de 8 Kb es requeriria 32 *terabytes* (2^{32} * 8 *kilobytes*) per emmagatzemar la taula. A més, cada pàgina només contindria una fila, amb la qual cosa s'estaria malbaratant espai. Com es pot comprovar, aquesta estratègia d'emmagatzematge no és precisament eficient.

Per tot això, les funcions de dispersió acostumen a reduir el nombre de valors possibles i, per tant, no són injectives, sinó exhaustives: és a dir, dos valors diferents (v_i i v_j sent $v_i \neq v_j$) poden tenir el mateix valor de *h*: $h(v_i) = h(v_j)$. Quan això passa, els valors v_i i v_j s'anomenen sinònims.

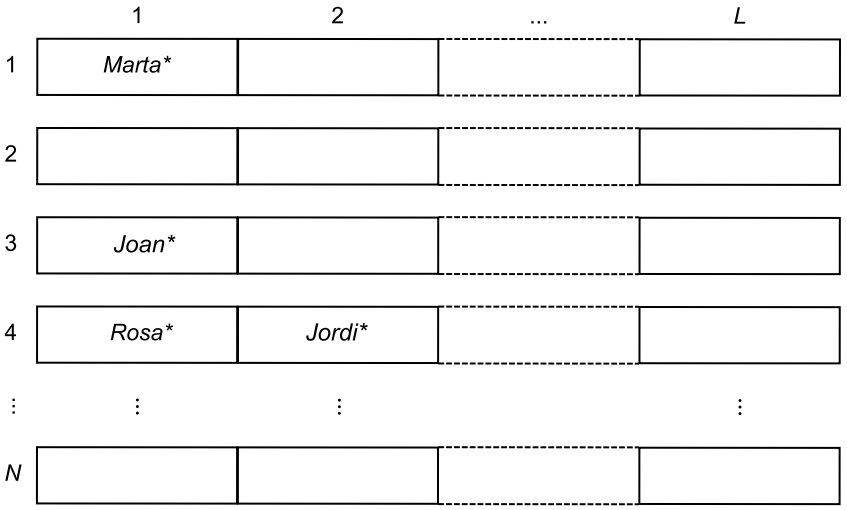
Les files dels valors sinònims s'emmagatzemen a la mateixa pàgina. Quan la pàgina està plena se'n crea una de nova, anomenada pàgina d'excedents, on s'emmagatzemaran les files dels nous sinònims, i es crea un apuntador a la pàgina original que apunta a la nova pàgina d'excedents. En cas que la pàgina d'excedents s'ompli, s'encadenarà una nova pàgina d'excedents, i així successivament. Una situació similar a la descrita també pot esdevenir quan l'índex de dispersió es construeix sobre una columna que pot prendre valors repetits. La pàgina inicial que hauria de contenir les dades d'interès un cop aplicada la funció de dispersió s'anomena pàgina primària, en contraposició a les pàgines d'excedents.

Tal com deveu haver intuït, l'habilitat de la funció *h* per evitar valors sinònims afectarà significativament al nombre d'operacions de E/S requerides per a la recuperació de les dades d'interès.

Exemple: localització d'una fila de dades utilitzant índexs de dispersió

Suposem que volem indexar una taula d'empleats amb un índex de dispersió segons la columna del nom de pila. Suposem també que *h* té els valors següents per als noms de pila següent (*Joan → 3*, *Marta → 1*, *Rosa → 4*, *Jordi → 4*). La figura següent ens mostra la distribució de les dades de la taula a les pàgines de dades de la base de dades. En particular, *N* és el nombre de pàgines primàries que tenim disponibles (que ve determinat pel nombre de valors possibles de la funció de dispersió) i *L* és el nombre de files que hi caben per pàgina. *Nom** representa les dades completes de la fila de la taula amb valor *Nom*. Per tant, per exemple, a la quarta pàgina es troben les dades de les files que corresponen als noms de *Rosa* i *Jordi*.

Figura 4. Distribució de les dades d'una taula seguint un índex de dispersió



Suposem que l'usuari fa la consulta següent:

```
SELECT *
 FROM EMPLEAT
 WHERE nom = 'Jordi';
```

L'SGBD identificaria que la columna *nom* està indexada segons l'índex anteriorment mostrat i calcularia *h ('Jordi')*. El resultat d'aquest càlcul seria el número de pàgina en què es troben les dades de la fila (la pàgina 4) o, en el pitjor dels casos, la pàgina des d'on es pot accedir a les seves dades. El sistema carregaria en memòria la quarta pàgina, consultaria la primera fila d'aquesta, que equival a un sinònim (*Rosa*). Posteriorment consultaria la segona fila de la pàgina per adonar-se que és la que estava buscant. En aquest moment podrien retornar les dades i finalitzar amb la consulta. El nombre d'operacions d'E/S per resoldre la consulta seria 1.

Cost de localització d'una entrada

Quan s'utilitza un índex de dispersió, el cost de localitzar una fila varia bastant en funció de si aquesta es troba en una pàgina d'excedents o no. Si una entrada no es troba en una pàgina d'excedents només cal fer una operació d'E/S per obtenir-la. En cas contrari, caldrà fer diferents operacions d'E/S fins a trobar la pàgina d'excedents que contingui la fila buscada.

En conseqüència, per aconseguir un bon rendiment de l'índex interessa que hi hagi poques pàgines d'excedents. Això es pot aconseguir de dues maneres diferents:

- Escollir una funció de dispersió que tingui en compte la distribució de valors de la columna (o columnes) indexada per minimitzar el nombre de sinònims.

- Dissenyar l'índex per reduir el nombre de pàgines d'excedents.

Hi ha molta bibliografia que tracta sobre com escollir funcions de dispersió adequades per a cada cas. Tractar aquest tema queda fora dels objectius d'aquest material, principalment per la dificultat de definir criteris que serveixin per a la majoria de casos.

Per veure com podem aconseguir reduir el nombre de pàgines d'excedents, introduirem el concepte de factor de càrrega. S'anomena factor de càrrega el nombre d'entrades que s'espera tenir, dividit pel nombre d'entrades que caben a les pàgines primàries:

$$C = M / (N \times L)$$

M representa el nombre d'entrades que esperem tenir, *N* el nombre de pàgines primàries (o nombre de possibles valors de la funció de dispersió) i *L* és el nombre de files que caben a cada pàgina. El valor de *L* és senzill de calcular, ja que es fa dividint la mida de pàgina per la grandària de cada fila de la taula indexada. Idealment també s'hauria de conèixer el valor de *M*, si no exactament, almenys sí aproximadament.

El factor de càrrega és un element de disseny que permet ajustar els valors de *N* (modificant la funció de dispersió) o de *L* (canviant la mida de la pàgina de dades) per reduir el nombre de pàgines d'excedents necessàries.

Un factor de càrrega baix indica que hi haurà menys excedents, fet que minimitzaria el nombre d'operacions d'E/S necessàries per obtenir les dades. En contrapartida, requerirà més pàgines primàries. D'altra banda, un factor de càrrega elevat permetrà reduir el nombre de pàgines primàries, però augmentarà el nombre de pàgines d'excedents. Això pot causar que es necessitin més operacions d'E/S per resoldre les consultes.

Els índexs de dispersió que hem explicat, que requereixen fixar *a priori* el nombre *N* de pàgines primàries, es coneixen també sota la denominació d'índexs de dispersió estàtics. Com a alternativa existeixen els índexs de dispersió dinàmics. En aquests índexs no cal que el nombre *N* de pàgines primàries estigui fixat *a priori*. En altres paraules, el nombre de pàgines primàries creix en funció de les necessitats, a mesura que s'afegeixen més files a la taula sobre la qual s'ha construït l'índex. Entre els índexs de dispersió dinàmics més coneguts hi ha l'*Extendible hashing* i el *Linear hashing*. Aquest últim està disponible, per exemple, a l'SGBD Berkeley DB. Així mateix, aquest tipus d'índexs, inicialment dis-

senyats per a l'àmbit de les bases de dades, també han estat transferits a llenguatges de programació, com seria el cas de Python, que disposa d'*Extendible hashing*.

Cost de manteniment

En general, les insercions, modificacions i les eliminacions implicaran inserir o eliminar files en les pàgines primàries o en les d'excedents. El nombre d'operacions d'E/S dependran del factor de càrrega.

Les eliminacions de files es poden fer de manera lògica (només marcant la fila afectada com a esborrada) o de manera física. L'eliminació física d'una fila pot implicar moure files de les pàgines d'excedents a les pàgines primàries, fet que redueix la longitud de les cadenes d'excedents.

Les operacions de modificació sobre una columna indexada implicaran moure les files afectades a pàgines de memòria diferents i, en alguns casos, reestructurar la pàgina primària amb files de les pàgines d'excedents. En aquest tipus d'operacions el manteniment pot arribar a ser més alt que en el cas dels índexs basats en arbres B+, ja que s'han de moure les files de dades de pàgina. En els índexs en forma d'arbre B+ no agrupats es pot realitzar l'actualització simplement modificant la informació de l'índex.

3.1.3. Mapa de bits

Un altre tipus d'índexs útils per indexar columnes amb pocs valors possibles que es repeteixen sovint són els índexs de mapes de bits (o índexs *Bitmap* en anglès). El principal avantatge d'aquest tipus d'índex és que ocupa poc, quan s'usa amb les condicions adequades, fet que permet fins i tot tenir tot l'índex carregat en memòria i evitar operacions d'E/S per consultar-lo. Aquest tipus d'índex permet identificar quines files compleixen una determinada condició utilitzant només operacions de bits. Com que les operacions sobre seqüències de bits són molt més ràpides que les seves alternatives sobre altres tipus de dades, aquest índex permet identificar si una fila compleix les condicions especificades més ràpidament.

En un índex de mapa de bits es crea una seqüència de bits per a cada fila de la taula que indica el valor de la columna indexada. La manera de crear aquesta seqüència pot variar, però en aquests materials explicarem la manera més simple: tindrà tants bits com possibles valors pot tenir la columna indexada. El bit i-èssim de la seqüència de bits relativa a una fila tindrà un 1 en cas que la fila tingui el valor corresponent a la posició i i un 0 en cas contrari. El valor de l'índex per a tota una taula estarà format per una matriu de N files i M columnes, en què N és el nombre de files i M són els possibles valors de la

columna indexada. Normalment s'anomena mapa de bits la seqüència de bits relacionada amb el possible valor d'un índex (les columnes de la matriu), que indica quines files de la taula tenen aquest valor.

Per exemple, suposem que tenim una taula d'empleats com la que es mostra a la figura 5, amb les columnes DNI, nom, vehicle propi i zona geogràfica. Suposem també que la columna de vehicle propi admet valors nuls o bé els valors SÍ/NO, i la columna zona geogràfica és obligatòria i admet només quatre valors (Barcelona, Girona, Lleida i Tarragona). Com que les columnes vehicle propi i zona geogràfica tenen pocs valors possibles, podem considerar adient crear dos índexs de mapa de bits, un per a cada columna.

Figura 5. Exemple de representació de dos índexs de tipus de mapa de bits (columnes de vehicle propi i zona geogràfica)

Empleat				Vehicle propi		Zona geogràfica			
DNI	Nom	Vehicle propi	Zona	Sí	No	Barcelona	Girona	Lleida	Tarragona
88775997	José	Sí	Barcelona	1	0	1	0	0	0
46781222	María	Sí	Lleida	1	0	0	0	1	0
E-998272	Jordi	NULL	Girona	0	0	0	1	0	0
87271923	Neus	Sí	Barcelona	1	0	1	0	0	0
88928187	Elena	No	Tarragona	0	1	0	0	0	1

A la figura anterior es pot veure gràficament la creació d'aquests índexs. Hi haurà dos mapes de bits de l'índex sobre la columna vehicle, un per a cada possible valor. Cada mapa de bits tindrà 5 bits, un per a cada fila de la taula. Podríeu pensar que es podria utilitzar un únic mapa de bits. No obstant això, en aquest cas és obligatori utilitzar dos mapes de bits, ja que la columna admet valors nuls i, per tant, que una fila no tingui un valor cert no implica necessàriament que sigui fals (tal com mostra la tercera fila del nostre exemple). A la figura també podem veure l'índex de mapa de bits resultant d'indexar la columna zona geogràfica. Fixeu-vos que en aquest cas hi ha quatre mapes de bits que es corresponen amb els possibles valors de la columna zona geogràfica. Cada mapa de bits consta de 5 bits.

Un dels avantatges dels índexs de mapes de bits és l'escassa mida que ocupen quan el nombre de valors diferents que pren la columna indexada són pocs. A l'exemple anterior necessitaríem 10 bits (2 valors × 5 files) per emmagatzemar el primer índex i 20 bits (4 valors × 5 files) per emmagatzemar el segon. A més, els mapes de bits es poden comprimir i les seves ràtios de compressió són molt elevades a causa de la distribució homogènia dels seus valors.

Aquest tipus d'índex només és aplicable quan es realitzen operacions d'igualtat i desigualtat (=, <>), d'inclusió (`IN`, `NOT IN`) o de lògica (`AND`, `OR`, `NOT`). En conseqüència, no és adequat quan es realitzen operacions de comparació sobre les columnes indexades (<, <=, >, >=).

Exemple: Resolució d'una consulta utilitzant índexs de mapes de bits

Suposem que l'usuari vol consultar els empleats que són responsables de la zona de Barcelona o Lleida i que tenen vehicle propi.

```
SELECT *
 FROM EMPLEAT
 WHERE vehicle = 'SI' AND zona IN ('Barcelona', 'Lleida');
```

L'SGBD, en rebre aquesta consulta, comprovaria que les columnes utilitzades a la clàusula `WHERE` estan indexades per índexs de mapes de bits (vehicle propi i zona). Suposant que l'SGBD decidís utilitzar aquests índexs, els passos que podria realitzar per identificar les files que s'han de tornar serien els següents:

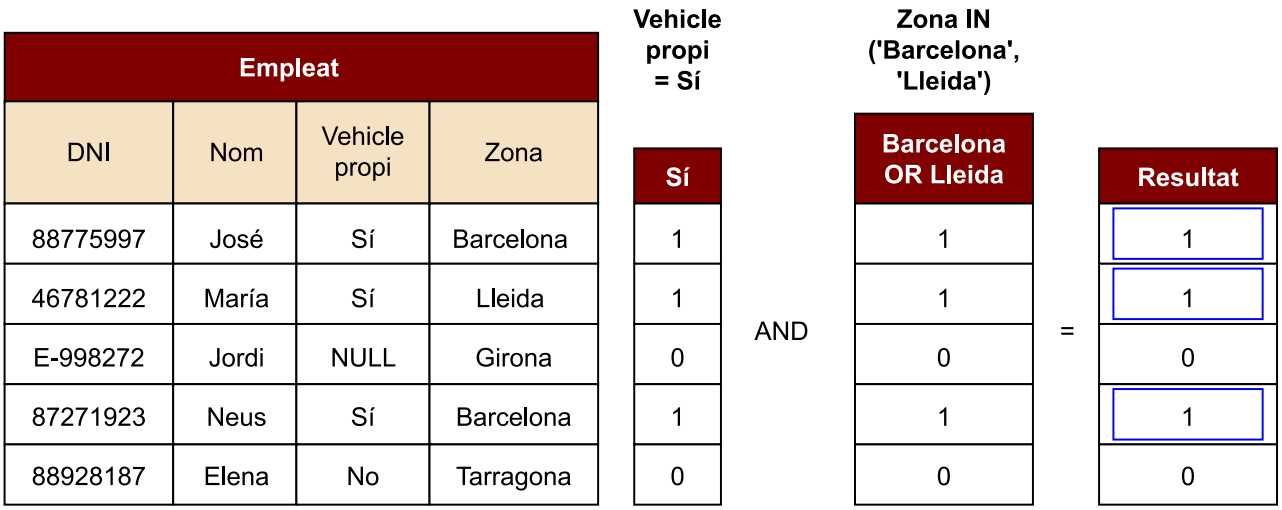
Per identificar els empleats amb vehicle propi s'hauria d'obtenir les files que tenen un 1 a la primera posició del mapa de bits, és a dir, les files amb un 1 al mapa de bits del valor *Sí*. El resultat inclou la primera, la segona i la quarta fila.

Per identificar els empleats de les zones de Barcelona i Lleida, s'han d'identificar les files amb un 1 al primer o el tercer mapa de bits. Això es pot fer amb una operació `OR` lògica entre els mapes de bits dels valors Barcelona i Lleida. El resultat inclouria les files 1, 2 i 4.

Finalment, s'identifiquen les files que han satisfet les dues condicions: les files 1, 2 i 4, que es podrien calcular fent una operació `AND` lògica entre els dos mapes de bits resultants dels passos 1 i 2.

Podem veure les operacions realitzades i el resultat gràficament a la figura següent:

Figura 6. Exemple de com resoldre una consulta utilitzant índexs de mapes de bits



Cost de localització d'una entrada

L'accés als índexs de mapa de bits és normalment molt ràpid quan el nombre de possibles valors de la columna indexada és limitat. A mesura que el nombre de valors possibles de la columna indexada augmenta, la mida del mapa de bits, i en conseqüència el temps de resposta, s'incrementa exponencialment.

Cost de manteniment

Calcular en quins casos és aconsellable un índex del tipus d'un mapa de bits pot ser una tasca complicada. Per prendre una decisió amb rigor, s'hauria de tenir en compte, entre altres, el nombre de files de la taula, el nombre de columnes que cal indexar, i el nombre de possibles valors per a cada columna. Aquesta informació s'hauria de fer servir per estimar la mida de l'índex. Normalment, i per simplificar, es tendeix a evitar aquest tipus d'índexs quan el nombre de valors possibles per a la columna que cal indexar està per sobre dels cent, ja que el seu rendiment decreix molt ràpidament i la seva mida s'incrementa de manera exponencial.

A més, aquest tipus d'índex només és recomanable per a taules estàtiques, és a dir, per a taules amb poques actualitzacions. El motiu és que quan es realitza una modificació sobre una columna indexada, es bloqueja tot l'índex de mapa de bits. Aquest bloqueig impedeix que es pugui tornar a utilitzar l'índex fins que no es resolgui la transacció que va originar el bloqueig. Per tant, si en una taula d'un milió de files modifiquéssim el valor d'una fila, bloquejaríem l'índex sencer, fet que provocaria que no es pogués utilitzar per accedir a cap valor de la taula i alentiria enormement altres operacions que poguessin estar executant-se concurrentment a la base de dades.

No obstant això, aquest tipus d'índex és molt apropiat per a *data warehouses* que s'actualitzen poques vegades i en horaris en què no hi ha activitat, com ara a la nit. Si a la taula sobre la qual volem crear l'índex hi accedeixen freqüentment múltiples usuaris, val més evitar-ne l'ús.

3.2. Definició d'índexs en sistemes gestors de bases de dades

Els índexs són elements del disseny físic de la base de dades i, com a conseqüència, no estan coberts en l'estàndard SQL. No obstant això, la sentència `CREATE INDEX` està present en tots els SGBD, tot i que amb paràmetres diferents en funció de l'SGBD amb el qual treballem.

A continuació veurem de manera esquemàtica com es creen índexs a Oracle i PostgreSQL.

3.2.1. Definició d'índexs a Oracle

Els índexs utilitzats per Oracle són els basats en arbres B+ i els de mapes de bits. Els índexs en forma d'arbre B+ són els utilitzats per defecte. Oracle també inclou altres tipus d'índex en algunes de les seves extensions, per exemple, l'extensió d'Oracle per tractar dades espacials (*Oracle Spatial*), inclou índexs basats en *Quadtrees* i arbres R.

Lectures addicionals

Informació sobre les *index-organized tables*.

Podeu consultar com simular un índex *clustered* a:

http://www.dba-oracle.com/data_warehouse/clustered_index.htm.

Oracle no permet crear índexs agrupats (o *clustered*) o índexs de dispersió directament, però permet simular-los utilitzant altres estructures de dades. Per exemple, les taules organitzades per índexs (*index-organized tables* en anglès) es poden utilitzar per simular índexs *clustered*.

A continuació mostrem de manera simplificada la sintaxi de la sentència per crear índexs a Oracle:

```
CREATE [{UNIQUE|BITMAP}] INDEX index_name ON
   table (index_expr [{ASC | DESC}] [, index_expr [{ASC|DESC}] ]...)
   [ { TABLESPACE { tablespace | DEFAULT }
      | key_compression
      | ...
      }, ...
   ]
```

Algunes de les opcions que cal definir en la creació d'un índex són:

- `UNIQUE`: es pot indicar si l'índex es realitza sobre una columna que no permet repetits. Si no s'utilitza la clàusula `UNIQUE` l'índex permetrà valors duplicats.

- `BITMAP`: permet crear índexs de tipus mapa de bits. És important notar que no es permet definir aquest tipus d'índexs sobre columnes que prenen valors únics. En cas de no informar aquesta clàusula, l'índex creat serà de tipus arbre B+.

- `Table (index expr [ASC|DESC])...`: indica les columnes indexades i si l'índex guarda els valors de manera ascendent o descendent.

- `TABLESPACE`: indica l'espai de taules que cal utilitzar per emmagatzemar l'índex.

- `Key compression`: permet indicar si es vol comprimir els valors que s'han d'indexar i com s'ha de realitzar la compressió. Aquesta opció no és necessària en els índexs de mapes de bits perquè sempre es comprimeixen a Oracle. Per això aquesta clàusula s'orienta a la compressió de les entrades dels índexs en forma d'arbre B+.

Per exemple, posem per cas una taula de ciutats com la següent:

```
CREATE TABLE City (
   cityZip       CHAR(5) NOT NULL,
   cityName      VARCHAR(50),
   cityRegion    CHAR(2),
   cityPopulation NUMBER(10)
```

```
) TABLESPACE tbs_slow_access;
```

Les sentències que es mostren a continuació permeten crear tres índexs sobre la taula de ciutats. Un únic índex sobre la clau primària (`cityZip`), un en forma d'arbre B+ sobre les columnes ciutat (`cityName`) i província (`cityRegion`) i l'últim de tipus de mapa de bits sobre la columna província (`cityRegion`).

```
CREATE UNIQUE INDEX indexOnZip
   ON City(cityZip)
   TABLESPACE tbs_indexs;

CREATE INDEX indexOnNameAndRegion
   ON City(cityName, cityRegion)
   TABLESPACE tbs_indexs;

CREATE BITMAP INDEX indexOnRegion
   ON City(cityRegion)
   TABLESPACE tbs_indexs;
```

Fixeu-vos que si a la taula de ciutats haguéssim definit la clau primària (que és l'opció conceptualment correcta) l'índex únic sobre la columna `cityZip` s'hagués creat automàticament, sense necessitat d'executar una sentència separada de creació de l'índex.

3.2.2. Definició d'índexs a PostgreSQL

Els índexs permesos a PostgreSQL són en forma d'arbre B+, de dispersió, GiST i GIN. Per defecte, si no s'indica el contrari, s'utilitzen índexs basats en arbres B+. Els índexs de tipus GiST i GIN s'explicaran més endavant.

Tot i que PostgreSQL no ofereix índexs de tipus mapa de bits, utilitza mapes de bits internament per accelerar les consultes. En particular, utilitza un sistema de mapa de bits per processar operacions lògiques entre diferents índexs sobre una mateixa taula. Per fer-ho, el sistema recorre cada índex involucrat en la consulta i prepara un mapa de bits en memòria indicant les localitzacions de les files de la taula que satisfan les condicions de l'índex. Posteriorment, es realitzen les operacions d'`AND` i `OR` necessàries sobre els mapes de bits per identificar i retornar les files que formen part del resultat de la consulta formulada.

És important notar que, com en el cas d'Oracle, PostgreSQL també inclou altres tipus d'índexs a les seves extensions. Per exemple, l'extensió de PostgreSQL per tractar dades espacials (PostGIS) també inclou índexs basats en *Quadtrees* i arbres R.

A continuació mostrem de manera simplificada la sintaxi de la sentència per crear índexs a PostgreSQL:

```
CREATE [UNIQUE] INDEX [CONCURRENTLY] [name] ON table_name [USING method]
   ({column_name} [opclass] [ASC|DESC] [,...])
   [TABLESPACE tablespace_name]
```

Algunes de les opcions que cal definir en la creació d'un índex són:

- `UNIQUE`: permet indicar que la columna indexada no permet valors duplicats.

- `CONCURRENTLY`: normalment, en la creació d'un índex, PostgreSQL bloqueja la taula indexada per no permetre operacions d'inserció, esborrat i modificació mentre l'índex s'està creant. En alguns casos, com per exemple en taules de grans dimensions, la indexació pot trigar hores. L'opció `CONCURRENTLY` permet no bloquejar la taula indexada mentre es crea l'índex. Això permet que la taula continuï sent accessible mentre s'indexa, però requereix que la taula sigui analitzada dues vegades (una quan es crea l'índex i una altra per veure els canvis que hi ha hagut des de la creació de l'índex). Per tant, aquesta opció és una arma de doble tall que cal saber fer servir adequadament, ja que garanteix la disponibilitat de la taula indexada però requereix més temps d'indexació i més càrrega de CPU per crear l'índex.

- `Method`: és el tipus d'índex que s'ha d'utilitzar. Els valors possibles són `btree`, `hash`, `gist`, `spgist` i `gin`.

- `Opclass`: permet definir un tipus d'operador diferent per establir l'ordre dels valors indexats, com ara un operador específic per ordenar nombres reals o coordenades. Es pot definir un operador per a cada columna indexada.

- `TABLESPACE`: indica l'espai de taules que cal utilitzar per emmagatzemar l'índex.

Els índexs agrupats no existeixen explícitament a PostgreSQL, sinó que cal simular-los amb la sentència `CLUSTER`. Aquesta sentència permet ordenar físicament les files d'una taula d'acord amb un índex creat anteriorment. La sentència `CLUSTER` ordena la taula quan s'executa, però un cop completada la seva execució, les noves files són afegides a la taula de manera seqüencial segons van arribant. Per tornar a reordenar la taula cal tornar a executar la sentència `CLUSTER` sense paràmetres. A continuació mostrem la sintaxi de la sentència:

```
CLUSTER [VERBOSE] table_name [USING index_name]
```

La clàusula `VERBOSE` permet mostrar el resultat de l'ordenació de la taula. D'altra banda, el paràmetre `index_name` indica l'índex (en concret el seu nom) que s'ha de fer servir per ordenar la taula.

A continuació en veurem uns exemples per mostrar com es creen índexs a PostgreSQL. Recordem la taula de ciutats vista anteriorment:

```
CREATE TABLE City (
   cityZip        CHAR(5) NOT NULL,
   cityName       VARCHAR(50),
   cityRegion     CHAR(2),
   cityPopulation  NUMBER(10)
) TABLESPACE tbs_slow_access;
```

Les sentències següents permeten crear tres índexs sobre la taula de ciutats. Un únic sobre la clau primària (`cityZip`) de tipus dispersió, un en forma d'arbre B+ sobre les columnes ciutat (`cityName`) i província (`cityRegion`) i l'últim també en forma d'arbre B+ sobre la província (`cityRegion`). L'última sentència permet ordenar la taula segons la columna `cityRegion`, emulant un índex agrupat sobre la província.

```
CREATE UNIQUE INDEX indexOnZip
   ON City USING hash (cityZip)
   TABLESPACE tbs_indexs;

CREATE INDEX indexOnNameAndRegion
   ON City(cityName, cityRegion)
   TABLESPACE tbs_indexs;

CREATE INDEX indexOnRegion
   ON City USING btree (cityRegion)
   TABLESPACE tbs_indexs;

CLUSTER City USING indexOnRegion;
```

A continuació introduirem els índexs GiST, SP-GiST i GIN que proporciona PostgreSQL. Estudiar aquests índexs a fons excedeix les pretensions d'aquest material.

Lectura complementària

Podeu trobar més informació sobre els índexs GiST, SP-GIST i GIN als capítols 55, 56 i 57 del manual de PostgreSQL:
http://www.postgresql.org/docs/9.3/static/index.html.

Índexs GiST

Els índexs GIST no són exactament un tipus d'índex, sinó una infraestructura que proporciona diferents estratègies d'indexació i que permet implementar noves estratègies. GiST són les sigles d'arbre de cerca generalitzat (o *generalized search tree* en anglès). Aquest tipus d'índex proporciona un mètode d'accés en una estructura d'arbre equilibrat que serveix com a patró per implementar esquemes d'indexació arbitraris. Es poden utilitzar aquests tipus d'índexs, per exemple, per crear índexs en forma d'arbre B+ o arbre R.

Tradicionalment, implementar un nou mètode d'accés a un índex era una tasca complexa, ja que requeria un coneixement profund dels detalls interns de funcionament de la base de dades: bloquejos, registres del dietari (en anglès, *log*), transaccions, etc. La interfície GiST proporciona un nivell d'abstracció superior que permet als desenvolupadors implementar un nou mètode d'accés en un índex indicant només la semàntica del tipus de dades que cal indexar. Per tant, aquest tipus d'índex permet personalitzar els índexs amb nous tipus de dades, de manera que experts en el tipus de dades que es vulguin indexar puguin fer-ho, encara que no siguin experts en el funcionament de la base de dades. PostgreSQL ofereix alguns mètodes d'accés implementats a GiST, com són els de nombres reals, hipercubs, parells de <*clau, valor*> i textos amb operadors que permeten calcular la similitud entre textos.

Lectura complementària

Podeu trobar més informació sobre com estendre els índexs de GiST a:

http://www.postgresql.org/docs/current/interactive/gist.html.

Segurament us deveu estar preguntant per què és necessari un mecanisme d'extensió com aquest havent-hi índexs basats en arbres B+ i de dispersió a PostgreSQL. La resposta és que els índexs GiST també es poden fer servir per donar suport a nous operadors, i no només als operadors permesos per defecte en els índexs en forma d'arbre B+ (<, <=, =, >, >=) i en els de dispersió (=). Per tant, usant índexs de tipus GiST es pot redefinir la semàntica dels operadors existents o crear operadors nous. Per exemple, es podria definir un nou índex per indexar imatges que proporcionés operadors per identificar la similitud entre dues imatges, per identificar quan una imatge està sobreexposada (és a dir, té massa llum), o per segmentar una imatge.

Les últimes versions de PostgreSQL (des de la 9.2) incorporen també els índexs de tipus SP-GiST. La filosofia d'aquest tipus d'índexs és la mateixa que en el cas dels índexs GiSP, però per a arbres de cerca no equilibrats. Aquest tipus d'índexs permeten implementar índexs que no era possible implementar amb GiST, com ara *Quadtrees* o *K-D trees*.

Índexs GIN

Els índexs *generalized inverted index* (GIN) són índexs de tipus invertit, que poden gestionar valors que contenen més d'un valor clau, on l'element que s'ha d'indexar es concep o pot ser vist com a multivalent. Això els fa especialment interessants per indexar valors compostos en cerques que requereixin identificar les files que contenen determinats valors components. Per exemple, els elements que s'han d'indexar poden ser documents, i les consultes tenen com a objectiu recuperar els documents que contenen un conjunt específic de paraules.

En aquest tipus d'índexs s'utilitza el terme *ítem* per referir-se al valor compost que s'ha d'indexar (el document) i la paraula *clau* per referir-se als valors que cal cercar (les paraules). Els índexs GIN emmagatzemen un conjunt de parells (*clau*, {RID_0 ,..., RID_N}) que indiquen que el valor de la clau es troba en les files les adreces de les quals són RID_0 ,..., RID_N.

De la mateixa manera que en els índexs GiST i SP-GiST, els índexs GIN poden donar suport a la creació d'estratègies d'indexació per part de l'usuari.

3.3. Optimització de consultes

L'optimització de consultes és un dels aspectes més importants que cal considerar quan es dissenya i construeix un SGBD relacional, ja que les tècniques que s'utilitzen per optimitzar consultes condicionen el rendiment global del sistema. En concret, això afecta el temps que necessita l'SGBD per respondre les consultes realitzades pels usuaris.

El procés d'optimització de consultes està inclòs dins d'un procés més general que es coneix com a processament de consultes. El **processament de consultes** consisteix en la transformació d'una consulta (expressada amb un llenguatge de consultes relacional, com SQL) en un conjunt d'instruccions de baix nivell. Aquest conjunt d'instruccions de baix nivell constitueix una estratègia per accedir a les dades amb un consum de recursos mínim. Precisament, l'objectiu del procés d'optimització de consultes consisteix a trobar aquesta estratègia.

Els usuaris d'un SGBD relacional accedeixen a les bases de dades gestionades per l'SGBD mitjançant consultes expressades en un llenguatge relacional, generalment en SQL. Com ja sabeu, SQL es caracteritza principalment per ser un llenguatge declaratiu. La implicació principal d'aquest fet és que, quan un usuari realitza una consulta, indica quines són les dades que desitja, quines condicions han de complir i en quines taules es troben, però no diu com han de trobar-se aquestes dades ni en quin ordre s'han d'executar les operacions especificades en la consulta.

L'SGBD té la missió de determinar com s'ha d'executar la consulta: ha de saber com estan emmagatzemades les dades i quins camins d'accés (índexs) s'han definit sobre aquestes. A més, ha de determinar en quin ordre s'han d'executar les operacions que es demanen a la consulta. Tenim moltes alternatives per resoldre una consulta; l'SGBD haurà d'avaluar el cost de cadascuna i escollir la de cost més baix. Aquesta alternativa rep el nom de **pla d'accés de la consulta**.

Per poder optimitzar una consulta, l'SGBD necessita passar d'un llenguatge declaratiu a un llenguatge procedimental. A continuació veurem de manera esquemàtica com es realitza aquesta traducció dins el processament de consultes.

3.3.1. L'àlgebra relacional en el processament de consultes

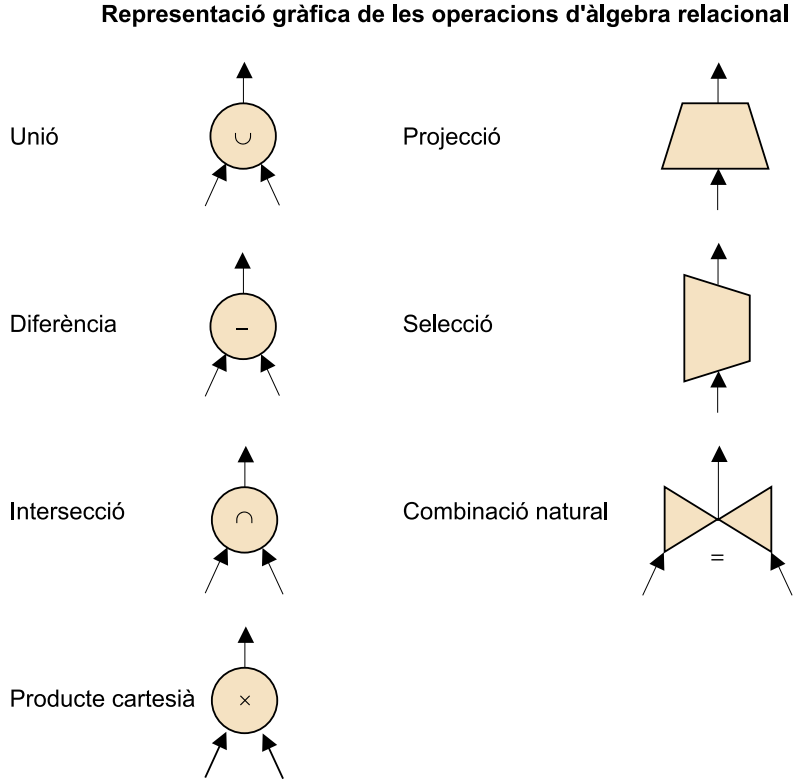
Donada una consulta formulada en llenguatge SQL, és possible trobar diverses consultes equivalents expressades en àlgebra relacional. Cadascuna d'aquestes consultes en àlgebra relacional es pot representar gràficament mitjançant una estructura en forma d'arbre, la qual es coneix amb el nom d'arbre sintàctic de la consulta.

L'**arbre sintàctic d'una consulta** és una estructura en forma d'arbre que correspon a una expressió d'àlgebra relacional, en què:

- Les **fulles de l'arbre** representen les taules que intervenen en la consulta.
- Els **nodes intermedis** són les operacions d'àlgebra relacional que intervenen en la consulta original. L'aplicació de cadascuna d'aquestes operacions dona lloc a una taula intermèdia.
- L'**arrel de l'arbre** constitueix la resposta a la consulta formulada.

Totes les operacions d'àlgebra relacional es poden representar gràficament. A la figura següent trobareu la representació de cadascuna de les operacions que ja coneixeu:

Figura 7. Representació gràfica de les operacions d'àlgebra relacional



Exemple: creació dels possibles arbres sintàctics d'una consulta

Imaginem-nos una base de dades d'una empresa que vol tenir constància de qui són els seus treballadors, quins projectes es desenvolupen en un moment determinat i quins empleats estan assignats a cada projecte. A continuació mostrem els esquemes de les taules que emmagatzemen aquestes dades:

- EMPLEATS(num_empl, nom_empl, categoria_laboral, divisió, sou, cap)
- PROJECTES(num_proj, nom_proj, producte, durada)
- ASSIGNACIONS(num_empl, num_proj, dedicació)

Les columnes subratllades són les claus primàries de cada taula. També tenim les claus foranes següents:

- La columna que representa el cap de la taula EMPLEATS és una clau forana que referencia la taula EMPLEATS.
- La columna de número d'empleat (*num_empl*) de la taula ASSIGNACIONS és una clau forana que referencia la taula EMPLEATS.
- La columna de número de projecte (*num_proj*) de la taula ASSIGNACIONS és una clau forana que referencia la taula PROJECTES.

Suposem que un usuari vol conèixer informació sobre els empleats que dediquen més de dues-centes hores a un projecte. En concret, l'usuari vol trobar el nom i el número d'aquests empleats i el número dels projectes als quals han dedicat més de dues-centes hores. Una possibilitat seria formular la consulta SQL següent:

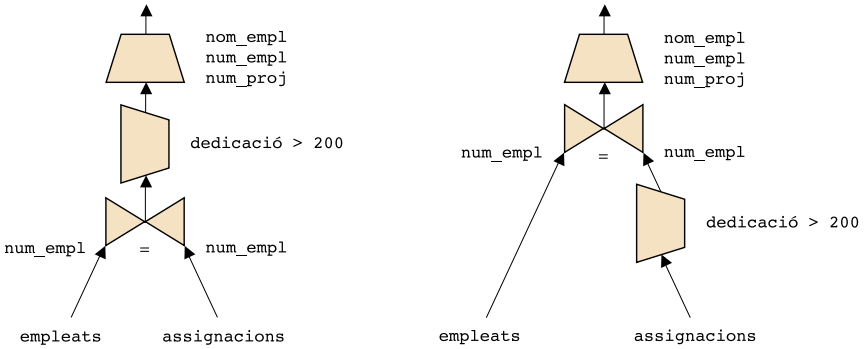
```
SELECT empleats.nom_empl, empleats.num_empl,
       assignacions.num_proj
  FROM empleats, assignacions
  WHERE empleats.num_empl=asignacions.num_empl AND
       asignacions.dedicacio>200;
```

Si ens hi fixem, veurem que en aquesta consulta s'inclouen les operacions d'àlgebra relacional següents:

- Combinació natural: entre les columnes *num_empl* de les taules EMPLEATS i ASSIGNACIONS.
- Selecció: seleccionar les assignacions amb dedicació superior a 200.
- Projecció: interessen les columnes nom empleat, número d'empleat i número de projecte.

Si combinem aquestes operacions, podem obtenir, per exemple, els arbres sintàctics que presentem a continuació:

Figura 8. Exemples de dos possibles arbres sintàctics que donen solució a una mateixa consulta SQL



Podeu observar que les fulles de tots dos arbres són les taules implicades en la consulta; que els nodes intermedis són les operacions d'àlgebra relacional especificades en la consulta, i que l'arrel de tots dos arbres representa la resposta a la consulta.

Com podem veure, els dos arbres sintàctics són equivalents a la consulta original formulada en SQL. La diferència existent entre els dos ve determinada per l'ordre en què s'executen les operacions d'àlgebra relacional. En el cas de l'arbre de l'esquerra, en primer lloc s'executa l'operació de combinació natural, mentre que en l'arbre de la dreta, primerament es resol l'operació de selecció.

3.3.2. El procés d'optimització sintàctica de consultes

L'objectiu de l'optimització de consultes és trobar el pla d'accés de la consulta; és a dir, l'estratègia d'execució de la consulta que té associat un **cost mínim**. Per cost mínim entenem que l'interval de temps necessari per trobar la resposta a la consulta formulada per l'usuari ha de ser el més breu possible.

Hi ha diferents **factors de cost** que influeixen en el temps necessari per resoldre una consulta. De tots ells, el més important és el nombre d'accessos (o operacions d'E/S) que s'han de realitzar en el dispositiu d'emmagatzematge no volàtil, en general a disc, que depèn bàsicament de les cardinalitats de les taules, les relacions existents entre taules que intervenen en la consulta i de la longitud de les seves files.

Factors de cost

Altres factors de cost serien el temps necessari per accedir a la memòria principal i el temps d'unitat de procés (CPU) necessari per executar les operacions sol·licitades en la consulta.

A continuació veurem amb un exemple la importància d'optimitzar consultes. Avaluem primerament el cost aproximat (en termes de nombre de files a les quals s'ha accedit) de tres estratègies d'execució possibles per a la consulta presentada en l'exemple del subapartat anterior:

```
SELECT empleats.nom_empl, empleats.num_empl, assignacions.num_proj
 FROM empleats, assignacions
```

```
    WHERE empleats.num_empl = assignacions.num_empl AND assignacions.dedicacio > 200;
```

Imaginem-nos que tenim les dades següents:

- card(EMPLEATS) = 100
- card(PROJECTES) = 20
- card(ASSIGNACIONS) = 400
- Aproximadament un 30% dels empleats que participen en projectes els dediquen més de dues-centes hores.

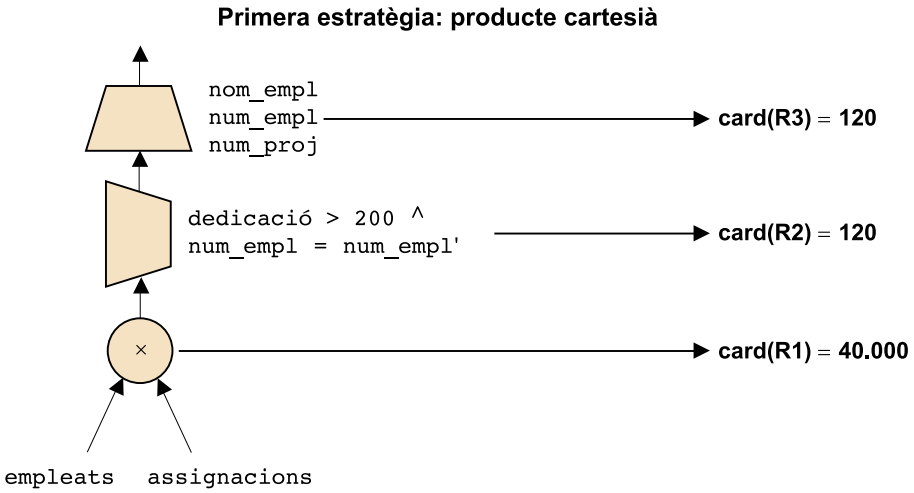
Recordeu

La cardinalitat d'una taula T qualsevol es representa com a card(T), i és el nombre de files que pertanyen a l'extensió de T.

Podem implementar, entre altres, les tres estratègies següents:

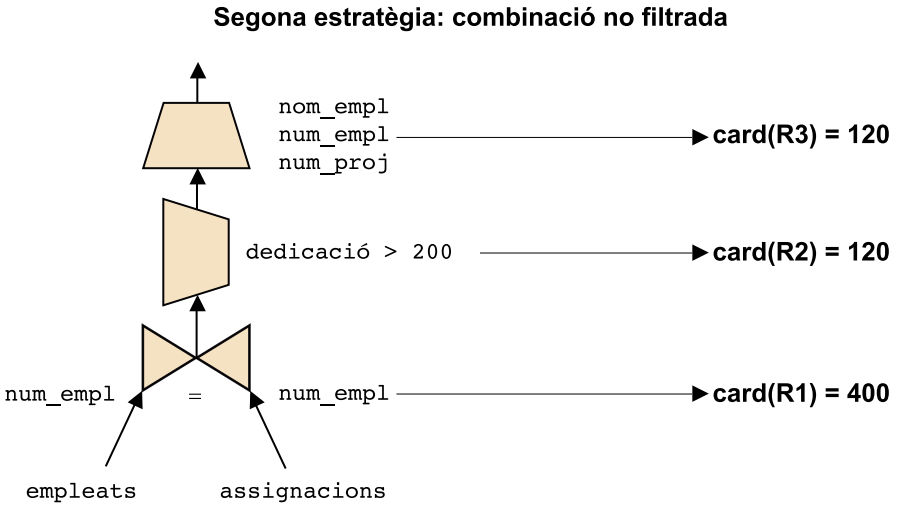
1) Producte cartesià

Figura 9. Exemple d'estratègia de consulta utilitzant el producte cartesià



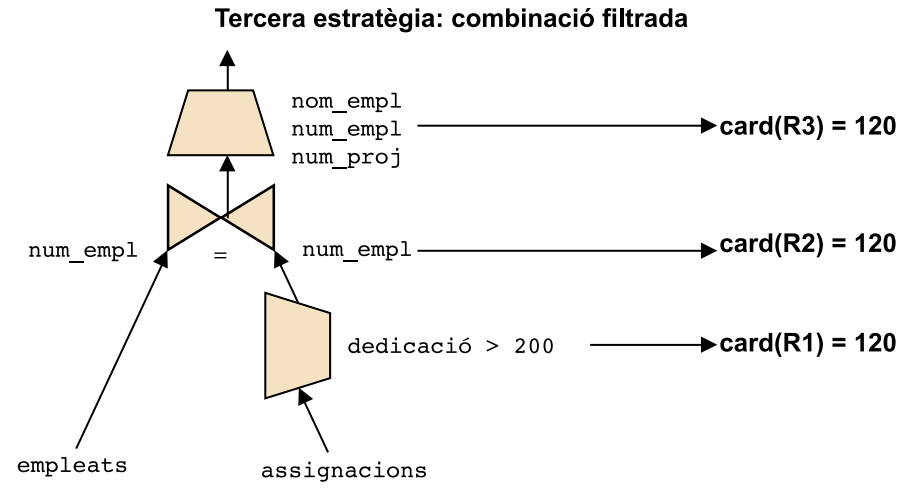
2) Combinació no filtrada

Figura 10. Exemple d'estratègia de consulta començant amb una operació de combinació



3) Combinació filtrada

Figura 11. Exemple d'estratègia de consulta utilitzant una operació de combinació després d'haver filtrat les files innecessàries



Podem observar que cada estratègia té un cost diferent. La millor estratègia és la combinació filtrada (la tercera), perquè és la que genera taules intermèdies de menor cardinalitat i, per tant, serà l'estratègia que implicarà un nombre més reduït d'operacions d'E/S. Aquesta estratègia executa les instruccions tal com es presenta a continuació:

- Primerament s'executa l'operació de selecció; aquesta operació dona com a resultat una taula intermèdia amb cent-vint files, ja que sabem que un 30% dels empleats que participen en projectes els dediquen més de dues-centes hores.

- A continuació, s'executa l'operació de combinació natural, que genera una taula intermèdia amb cent-vint files, ja que com que la columna número d'empleat (*num_empl*) de la taula ASSIGNACIONS és la clau forana que re-

ferencia la taula EMPLEATS, tota fila de la taula intermèdia anterior trobarà parella a la taula EMPLEATS.

- Finalment s'executa l'operació de projecció, que proporciona la resposta a la consulta original formulada per l'usuari. Com que la columna número d'empleat (*num_empl*) és la clau primària de la taula EMPLEATS, no tindrem dues files iguals i, per tant, la resposta estarà composta per cent-vint files.

L'ordre d'execució de les operacions formulades dins d'una consulta repercuteix directament en el cost d'execució de la consulta. Per tant, és important trobar el millor ordre d'execució possible per a les operacions de cada consulta. Malauradament, trobar aquest ordre d'execució requereix molt de temps si les consultes no són simples. Per aquesta raó, els SGBD relacionals apliquen mètodes heurístics per trobar una estratègia d'execució raonablement adequada per resoldre una consulta determinada.

L'**optimització sintàctica** d'una consulta és el procés que determina un ordre d'execució raonablement adequat de les operacions que inclou una consulta.

L'optimització sintàctica de la consulta comença amb la traducció de la consulta original en un arbre sintàctic equivalent i finalitza quan s'ha trobat l'arbre sintàctic òptim. Per trobar l'arbre sintàctic òptim, l'SGBD relacional aplica els mètodes heurístics següents:

- Executar les operacions d'àlgebra relacional que disminueixin la cardinalitat dels resultats intermedis tan aviat com sigui possible. Les operacions d'àlgebra relacional que disminueixen la cardinalitat de la taula resultant són la selecció, la projecció, la diferència i la intersecció.

- Endarrerir al màxim l'execució de les operacions que incrementen la cardinalitat dels resultats intermedis. Les operacions d'àlgebra relacional que incrementen la cardinalitat de la taula resultant són el producte cartesià, la unió i la combinació.

Si llegim detingudament les condicions anteriors, veiem que l'optimització sintàctica reordena simplement les operacions d'àlgebra relacional que figuren a la consulta. En canvi, en el nostre exemple hem considerat la cardinalitat de les taules intermèdies. De fet, l'avaluació del cost de l'arbre sintàctic òptim obtingut com a resultat de l'optimització sintàctica es realitza posteriorment, en l'etapa d'optimització física de la consulta.

3.3.3. El procés d'optimització física de consultes

L'objectiu de l'**optimització física** d'una consulta és avaluar el cost total d'execució de l'arbre sintàctic òptim associat a una consulta d'un usuari. Aquest cost d'execució d'un arbre sintàctic és igual a la suma dels costos de totes les seves operacions.

El cost de cada operació inclou tant el cost d'execució de l'operació en si com el d'escriure el resultat de l'operació en una taula intermèdia. Per poder aproximar el cost de les diferents operacions, l'SGBD relacional necessita conèixer la informació següent:

1) Les estructures d'emmagatzematge definides a l'esquema intern. D'aquestes estructures, és important tenir-ne informació sobre els aspectes següents:

- En quins fitxers s'han emmagatzemat les taules implicades a la consulta i on es troben aquests fitxers.
- Si s'han definit estructures d'agrupació per a les files d'una taula segons el valor d'una o més columnes.
- Si s'han definit vistes materialitzades sobre les taules implicades a la consulta.
- Els índexs (i el seu tipus) que s'han definit a la base de dades.

2) Dades estadístiques sobre les taules (i les seves columnes) i índexs definits a la base de dades i que s'emmagatzemen en el catàleg de la base de dades. Alguns exemples serien:

- Taula: cardinalitat, quantes pàgines ocupa, l'ocupació mitjana de cada pàgina, nombre de pàgines buides, mida mitjana de les files.
- Columna: nombre de valors diferents a la columna, nombre de valors nuls a la columna, valor mínim i màxim de la columna, histograma de freqüència d'aparició de cada un dels valors, etc. Aquestes dades són especialment rellevants en el cas de columnes sobre les quals s'hagin construït índexs.
- Índex: les dades dependran del tipus d'índex que es tingui en consideració. Per exemple, en el cas d'un arbre B+, interessa saber el nombre de nodes fulla (recordem que cada node equival a una pàgina) que conté, l'ocupació mitjana dels nodes fulla, l'alçada i ordre de l'arbre B+.

Dades estadístiques

La gestió de dades estadístiques és crucial perquè el procés d'optimització física es realitzi correctament. Els SGBD proporcionen sentències SQL que ajuden a la recollida i actualització de les dades estadístiques dels objectes definits a la base de dades, com ara la sentència `ANALYZE` que existeix a Oracle i PostgreSQL i que permet obtenir estadístiques sobre elements de la base de dades. En el cas de PostgreSQL, com que les files esborrades no s'eliminen físicament de la taula, es pot afegir l'ordre `VACUUM` a `ANALYZE` per eliminar-les abans d'analitzar les estadístiques.

3) Els algorismes d'implementació de les operacions d'àlgebra relacional i d'altres extensions proposades per SQL (per exemple, les clàusules `ORDER BY`, `DISTINCT`, funcions d'agregació, etc.) que hi ha disponibles. Aquests algorismes també es coneixen com a mètodes d'accés.

La informació prèviament descrita permet que un SGBD relacional pugui decidir quins algoritmes són aplicables i quins no. L'aplicabilitat d'aquests algorismes depèn dels elements exposats al punt 1. Per acabar, l'SGBD haurà d'estimar el cost de cada un dels algoritmes aplicables i escollir l'algorisme que tingui el cost més baix, basant-se en les dades estadístiques. Això s'ha de realitzar per a totes les operacions que formen part de la consulta, i dona lloc al que s'anomena el pla de la consulta.

Exemple: quin algorisme hem d'escollir?

Per l'arbre sintàctic òptim trobat a l'exemple anterior, l'SGBD necessita avaluar el cost de les operacions de combinació natural i de selecció que hi figuren. Suposem que sobre la columna dedicació de la taula ASSIGNACIONS s'ha definit un índex.

En aquest cas, per implementar l'operació de selecció física, l'SGBD podria escollir entre dos algorismes:

- Un algorisme que no fes ús de l'índex. En aquest cas, realitzaríem un recorregut seqüencial del fitxer que emmagatzema la taula ASSIGNACIONS i comprovaríem la condició demanada a la consulta (dedicació superior a dues-centes hores). Per cada fila que compleixi la condició, projectaríem les columnes que necessitéssim (*num_empl* i *num_proj*), i les guardaríem en una taula intermèdia R1.

- Un algorisme que fes ús de l'índex. Llavors, mitjançant l'índex, accediríem només a les files que verifiquessin la condició, projectaríem les columnes que ens interessessin i guardaríem el resultat en una taula intermèdia R1.

Dels dos algorismes, intuïtivament veiem que el segon segurament tindrà associat un cost més baix, i seria l'algorisme que escolliria l'SGBD.

Com hem comentat, cada SGBD té un conjunt d'algoritmes o mètodes d'accés per a resoldre les operacions (tant les d'àlgebra relacional com els que s'han afegit per SQL) incloses en les consultes que formulen els usuaris. Els mètodes d'accés més comuns són:

- *Table scanning*: quan no s'utilitza un índex i la consulta es resol fent un recorregut seqüencial de tota la taula. Es pot utilitzar, per exemple, en la resolució de consultes que únicament incorporen operacions de l'àlgebra relacional de selecció i/o projecció.

- *Sorting algorithms*: permeten ordenar les dades de les taules de la base de dades o de taules intermèdies que es generen durant el processament de la consulta i que serveixen de suport per a algun altre mètode d'accés (per exemple, els algorismes de *join* que introduirem posteriorment). També pot ser útil en la resolució de consultes que incorporen les clàusules `ORDER BY` i `GROUP BY`, i per a l'eliminació de duplicats (clàusula `DISTINCT` d'SQL o per a la resolució d'operacions d'unió d'àlgebra relacional).

- *Index / index-only scanning* i *block or row index ANDing*: aquests mètodes ajuden principalment a resoldre consultes que incorporen alguna operació de selecció de l'àlgebra relacional sobre columnes per a les quals s'ha definit un índex. El mètode *index / index-only scanning* es basa en l'ús d'un únic índex, mentre que *block or row index ANDing* es basa en l'ús de diversos índexs i serveix per fusionar les entrades d'aquests índexs. L'*index / index-only scanning* també pot resultar d'utilitat en la resolució de consultes que incorporen les clàusules `ORDER BY`, `GROUP BY` o funcions d'agregació (per exemple, `COUNT`, `MIN` o `MAX`).

- *Join*: permeten resoldre consultes que incorporen operacions de l'àlgebra relacional de combinació entre taules. Hi ha diferents variants d'aquest mètode d'accés. A continuació es descriuen les més rellevants (el pseudocodi associat es mostra a la figura 12):
  - *Nested loop*: consisteix en el recorregut seqüencial de les taules implicades en la combinació, per mitjà de dos bucles imbricats (un per cada taula). Al final, per a cada fila de la taula que es recorre en el bucle exterior s'obtenen totes les files de la taula que es recorre en el bucle interior que compleixen la condició expressada en la combinació. Aquest mètode sempre es pot aplicar, amb independència d'operador de comparació (=, <>, <, <=, >, >=) indicat a la combinació. La taula de menys grandària és la que s'associa al bucle exterior.

  - *Index nested loop*: és una variant del mètode anterior, en què en el bucle interior no es realitza un recorregut complet de la taula a la qual s'associa, sinó que únicament s'accedeix a les files que compleixen la condició expressada en la combinació mitjançant un índex. Si l'índex és de dispersió, només es podrà aplicar quan l'operador de comparació usat en la combinació sigui la igualtat (equicombinació i combinació natural). Si l'índex es basa en un arbre B+, el mètode sempre es podrà aplicar.

  - *Sort-Merge*: consisteix en la lectura seqüencial de les taules implicades en l'operació de combinació, prenent les files que verifiquin la condició expressada en la combinació. Requereix que les taules estiguin ordenades físicament segons el valor de les columnes sobre les quals es realitza la combinació. Si no estan ordenades, es poden fer servir els *sorting algorithms* prèviament presentats. Únicament és aplicable quan l'operador és la igualtat (equicombinació i combinació natural).

  - *Hash join*: aquest mètode, en la seva forma més simple, es basa en la construcció d'un índex de dispersió sobre la taula que conté menys files (en concret, es construeix sobre la columna implicada en la combinació). A continuació, sobre l'altra taula que intervé en la combinació es realitza un recorregut seqüencial i s'usa l'índex de dispersió construït per saber quines files formen part del resultat de la combinació. Només es pot aplicar en equicombinació i combinació natural. Un cop

construït l'índex de dispersió, aquest mètode es pot considerar un cas particular d'*index nested loop*.

Figura 12. Pseudocodi mètodes de *join*

Nested loop: R[AθB]S, θ∈{=,<>,>=,>,<,<=}, card(R)<=card(S)

```
per a cada pàgina de R
        transferir pàgina de R de disc a memòria principal
        per a cada pàgina de S
            transferir pàgina de S de disc a memòria principal
          per a cada fila t de la pàgina de R
            per a cada fila s de la pàgina de S
                si t.AθsB , llavors, generar resultat
                fsi
            fpera
          fpera
        fpera
fpera
```

Index nested loop: R[AθB]S, existeix índex I_B sobre la columna B de S, θ depèn del tipus d'índex.

```
per a cada pàgina de R
        transferir pàgina de R de disc a memòria principal
        per a cada fila t de la pàgina de R
          usar I_B per buscar entrades amb valor t.A
        si existeixen entrades en I_B que compleixin les condicions
          de join, llavors generar resultat (pot implicar accés a S)
        fsi
        fpera
fpera
```

Sort-Merge: R[A=B]S

```
ordenar R segons A (si R no està ordenada)
ordenar S segons B (si S no està ordenada)
recórrer pàgines de R i S aplicant algorisme de merge
per a generar resultat
```

Hash join: R[A=B]S, card(R)<=card(S)

```
Crear índex hash H_A sobre R.A
per a cada pàgina de S
        transferir pàgina de S de disc a memòria principal
        per a cada fila s de la pàgina de S
          usar H_A per buscar entrades amb valor s.B
        si existeixen entrades en H_A
           llavors generar resultat (pot implicar accés a R)
        fsi
        fpera
fpera
```

Els SGBD poden suportar tots o una part dels mètodes d'accés que acabem d'explicar. També poden introduir variants, per la qual cosa serà imprescindible consultar la documentació del fabricant. Les taules intermèdies que es generin durant l'execució de les consultes, si no es poden emmagatzemar en memòria interna per la seva mida, es guarden en memòria externa (per exemple, en disc). Per fer-ho, l'SGBD utilitza espais de taula temporals (en anglès, *temporary tablespaces*).

Finalment, els SGBD disposen d'eines que faciliten l'anàlisi de l'estratègia (o pla de la consulta) seguida a la resolució de cada consulta. Exemples d'aquestes són les sentències `EXPLAIN PLAN` d'Oracle o `EXPLAIN` de PostgreSQL, que permeten analitzar el pla de consulta d'una sentència SQL. A causa de la similitud `EXPLAIN` i `EXPLAIN PLAN`, presentarem únicament el cas particular de PostgreSQL.

Anàlisi del pla de consulta a PostgreSQL

A continuació veurem com es comprova el pla d'una consulta SQL. A PostgreSQL els plans de consulta es poden consultar mitjançant la sentència `EXPLAIN`, que rep una sentència SQL com a paràmetre i retorna el pla de la consulta en el format següent:

Sentència `EXPLAIN`

Vegeu més informació de la sentència `EXPLAIN` a: http://www.postgresql.org/docs/9.3/static/using-explain.html.

```
-- Suposem la taula Country següent
```

```
CREATE TABLE Country (
   name     VARCHAR(50) UNIQUE,
   region   CHAR(2),
   ...
);
-- A continuació es consulta la sentència següent mitjançant la sentència EXPLAIN
EXPLAIN SELECT * FROM Country;

QUERY PLAN
-------------------------------------------------------
Seq Scan on Country (cost=0.00..458.00 rows=10000 width=244)
```

Tal com podem veure, aquesta sentència analitza la consulta passada per paràmetre i retorna el pla d'execució. Com que la consulta d'exemple no té clàusula `WHERE`, l'SGBD ha de consultar totes les files de la taula, per tant, el sistema ha triat utilitzar un mètode d'accés seqüencial per recórrer la taula (*seq scan*). Els números entre parèntesis tenen el significat següent:

- Temps estimat d'inicialització: és el temps necessari abans que el mètode d'accés sigui capaç de tornar la primera fila del resultat. Per exemple, en cas de requerir una ordenació, seria el temps necessari per ordenar les dades.

- Temps estimat total: temps necessari per completar l'operació del node del pla de consulta, és a dir, per tornar totes les files que conformen el resultat de la consulta.

- Nombre estimat de files de sortida per a aquest node del pla de consulta.

- Mida mitjana estimada de files produïdes en aquest node del pla de consulta (en *bytes*).

Hem vist a l'exemple anterior que a PostgreSQL el mètode de *table scanning* s'anomena *seq scan*. Per facilitar al lector la comprensió dels resultats de la sentència `EXPLAIN`, a continuació presentem els principals mètodes d'accés utilitzats a PostgreSQL i la seva equivalència amb els mètodes d'accés presentats anteriorment:

PostgreSQL

Al capítol titulat «Performance Tips» (http://www.postgresql.org/docs/9.3/static/performance-tips.html) del manual de PostgreSQL es pot trobar més informació sobre aquest tema en el cas particular de PostgreSQL.

- *Seq scan* és l'equivalent a *table scanning* i realitza un recorregut seqüencial de tota la taula.

- *Index scan* i *index only scan* són els equivalents a *index scanning* i *index-only scanning*, respectivament, i es basen en l'ús d'arbres B+.

- *Bitmap index scan*, *bitmap heap scan* i *recheck cond* serien els equivalents a l'*index scanning* per als índexs de mapes de bits.

- Operacions de combinació:

– *Nested loop* és l'equivalent a *nested loop* i *index nested loop*, ja que PostgreSQL utilitza, sempre que sigui possible, un índex per consultar les files de la segona taula que estan relacionades amb la primera.

– *Hash join* i *merge join* són l'equivalent als mètodes *hash join* i *sort-merge* prèviament presentats.

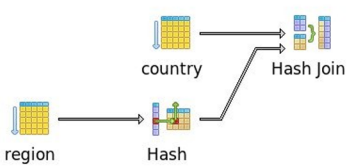
Per exemple, suposem que volem analitzar el pla de consulta de la sentència SQL següent:

```
-- Suposem la taula Region següent
CREATE TABLE Region (
   id   CHAR(2) UNIQUE,
   name VARCHAR(50),
   ...
);

-- Suposem la taula Country següent
CREATE TABLE Country (
   name   VARCHAR(50) UNIQUE,
   region CHAR(2),
   ...
);

-- La sentència SQL que s'ha d'analitzar és la següent
SELECT c.name as country_name , r.name as region_name
FROM   Country c, Region r
WHERE  r.id=c.region;
```

PostgreSQL ens permet visualitzar el pla de consulta textualment o de manera gràfica, tal com podem veure a la figura següent.

Figura 13. Visualització d'un pla de consulta, de manera textual i gràfica, a PostgreSQL

	QUERY PLAN text
1	Hash Join (cost=1.09..2.19 rows=4 width=24)
2	Hash Cond: (c.region = r.id)
3	-> Seq Scan on country c (cost=0.00..1.04 rows=4 width=15)
4	-> Hash (cost=1.04..1.04 rows=4 width=13)
5	-> Seq Scan on region r (cost=0.00..1.04 rows=4 width=13)



A continuació interpretarem els resultats obtinguts. El mètode *hash join* que apareix a l'arrel de l'arbre s'utilitza per realitzar la combinació (en concret, es tracta d'una equicombinació) entre les taules `REGION` i `COUNTRY`. Per realitzar aquesta combinació, es crea una taula temporal que conté les dades de `REGION` i un índex de tipus dispersió (*hash*) sobre la seva clau primària que faciliti fer la combinació. Per això és necessari que primer es faci un accés seqüencial (de tipus *seq scan*) sobre la taula `REGION`, per, posteriorment, obtenir l'índex de dispersió de la clau primària. Un cop fet això, es pot fer una lectura seqüencial (de tipus *seq scan*) de les files de la taula `COUNTRY`. Per a cada fila de país es podrà accedir a la seva regió utilitzant l'índex de dispersió creat i així fer la combinació corresponent.

Com podem veure, a l'exemple anterior l'SGBD ha creat un índex de dispersió temporal per augmentar la velocitat de la consulta. Ens podem preguntar, què hagués passat si ja existís un índex sobre la clau primària de regió? Suposem que creéssim aquest índex i tornéssim a consultar el pla de la consulta. El resultat seria el següent:

Figura 14. Visualització d'un pla de consulta. Exemple amb *nested loop*

	QUERY PLAN text
1	Nested Loop (cost=10000000000.00..10000000008.16 rows=4 width=24)
2	-> Seq Scan on country c (cost=10000000000.00..10000000001.04 rows=4 width=15)
3	-> Index Scan using region_id_index on region r (cost=0.00..1.77 rows=1 width=13)
4	Index Cond: (id = c.region)

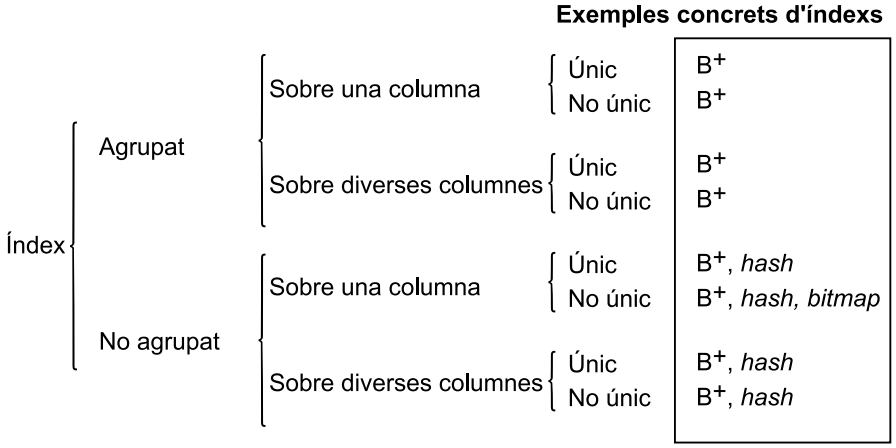
Ara no cal crear un índex temporal perquè la columna `id` de la taula `REGION` ja està indexada. Per tant, en aquest cas la consulta es resol aplicant un mètode de *nested loop*. O sigui, s'han de consultar les files de la taula `COUNTRY` de manera seqüencial i, per a cada fila, s'ha de buscar la fila de la taula `REGION` associada per mitjà de l'índex existent.

3.4. Seleccionar quins índexs i de quin tipus els volem crear

Les peculiaritats dels índexs fan que sigui aconsellable gestionar-los a part de les dades de les taules. Per aquest motiu, hi ha un tipus d'espai virtual per contenir els índexs anomenat espai d'índexs.

Tal com es resumeix a la figura següent, en aquests materials hem introduït els índexs en forma d'arbre B+, de dispersió i de mapes de bits, però hi ha altres índexs (n'hem esmentat alguns) i variacions dels presentats aquí que poden ser molt útils en determinats casos. L'índex en forma d'arbre B+ és l'índex per excel·lència en SGBD relacionals, per la seva aplicabilitat en tot tipus de situacions i el seu bon rendiment. No obstant això, l'ús d'un altre tipus d'índexs en circumstàncies concretes pot millorar el rendiment de la base de dades de manera significativa.

Figura 15. Classificació dels índexs presentats en aquests materials



Hem après a definir índexs i en quins casos és convenient fer-ho. A continuació estudiarem una sèrie de consideracions que cal tenir en compte des del punt de vista del rendiment.

Abans de plantejar-se crear un nou índex és important saber quins índexs creen els SGBD per defecte. És important consultar la documentació per al cas concret de cada SGBD, però per norma general podem assumir que l'SGBD crearà automàticament índexs per a totes les claus primàries i alternatives que s'hagin definit a les taules.

Resulta convenient definir un índex en els casos següents:

- Garantir la unicitat d'algunes columnes. Definir un índex únic és potser la manera més fàcil i eficient de garantir que no hi ha duplicitat de valors a la columna.

- Garantir l'ordenació de les files dins de la taula. Alguns processos poden requerir accedir a les taules mitjançant accés seqüencial per valor. En aquests casos, i quan la taula té un nombre elevat de files, l'elecció d'un índex agrupat té una importància crítica.

- Optimitzar les comprovacions de les restriccions d'integritat referencial. L'SGBD ha de garantir que les restriccions d'integritat referencial de la base de dades no es violen. Per fer-ho ha de comprovar l'existència de la clau relacionada en l'altra taula quan s'insereix o s'esborra una fila. En aquests casos, és recomanable disposar d'índexs sobre les claus foranes per tal d'optimitzar els accessos de comprovació del mateix SGBD.

- Columnes freqüentment referenciades en clàusules `WHERE`. En general, aquestes columnes són bones candidates per a un índex.

- Ús de les mateixes columnes en clàusules `ORDER BY` o `GROUP BY`. En el cas que diverses columnes apareguin repetidament, o en consultes molt

freqüents, referenciades en aquest tipus de clàusules, és convenient crear un índex. Així eliminem la necessitat que l'SGBD ordeni les files, ja que l'índex proporciona automàticament l'ordre desitjat o facilita l'agrupació requerida.

- Combinacions entre dues o més taules. En un procés de combinació el punt més crític és establir l'enllaç entre les dues taules per mitjà dels predicats de la clàusula `WHERE`. Si definim un índex en cada taula amb les columnes que participen en l'operació de combinació, aconseguirem, en la majoria dels casos, que l'optimitzador esculli els índexs per resoldre l'operació, de manera que es millora el temps d'execució.

- Grans volums de dades. Quan tenim taules que emmagatzemen grans volums de dades, com pot passar en un *data warehouse*, l'ús d'índexs en forma d'arbre B+ pot generar estructures de dades molt pesades d'emmagatzemar i processar. En aquests casos pot resultar més eficient utilitzar índexs de tipus mapa de bits, que permeten definir estructures més compactes, menys pesades i que faciliten una execució ràpida de les operacions.

El nombre i la grandària dels índexs varia en les diferents bases de dades, però normalment s'aplica una regla general que diu que els índexs han d'ocupar entre el 10% i el 20% de l'espai d'emmagatzematge de la base de dades. En els casos en què ocupen més d'un 25%, els experts aconsellen estudiar l'estructura de la base de dades a fons per assegurar que no hi ha índexs superflus. A més, tenint en compte que els índexs tenen un cost de manteniment no menyspreable, s'han de seguir un conjunt de regles bàsiques per assegurar-se que el nombre d'índexs és adequat. A continuació enumerem les més importants:

- Evitar i eliminar índexs redundants: en alguns casos hi pot haver índexs difícils d'utilitzar pel planificador de consultes. Un cas comú és quan tenim diferents índexs que comparteixen columnes. En aquests casos ens hem d'assegurar que el planificador de consultes del nostre SGBD és capaç d'utilitzar els índexs creats en els casos requerits. Per fer-ho es pot utilitzar la sentència `EXPLAIN` que hem vist abans o alguna sentència equivalent que ofereixi l'SGBD amb el qual treballem.

- Afegir índexs només quan sigui necessari: l'avantatge d'afegir un nou índex ha de ser clar i estar justificat.

- Afegir o eliminar columnes d'índexs compostos per millorar el rendiment. Els índexs compostos, és a dir, aquells que es defineixen sobre múltiples columnes, poden ser molt útils i minimitzar el nombre d'ordenacions i combinacions necessàries en cerques per múltiples columnes. No obstant això, com més columnes té un índex, més difícil és la seva aplicació. Per tant, de vegades és convenient minimitzar el nombre de columnes indexades en un índex per maximitzar les consultes on es pot aplicar. Evident-

ment, aquesta regla no s'aplica en el cas que l'índex sigui sobre una clau primària, alternativa o forana composta.

- Sospesar si és viable indexar columnes que s'actualitzen freqüentment. Modificar el valor d'una columna indexada implica reestructurar l'índex. Per tant, quan un índex es crea sobre una columna que s'actualitza sovint, ens hem de preguntar si el temps que necessitarà l'SGBD per reestructurar l'índex és menor que el temps que guanyarem a utilitzar-lo en les consultes.

- Realitzar el manteniment dels índexs regularment, ja que les actualitzacions de la base de dades pot haver provocat que alguns índexs deixin de ser rendibles. Aquest manteniment requerirà esborrar índexs quan es detecti que ja no contribueixen a millorar el rendiment o quan clarament castiguin el rendiment de la base de dades.

4. Vistes materialitzades

És probable que a la nostra base de dades tinguem consultes que s'executin molt freqüentment o consultes que, tot i que s'executin menys freqüentment, requereixin de càlculs molt complexos. En ambdues situacions sorgeix la mateixa pregunta, per què no emmagatzemem el resultat de la consulta per al futur? Així, quan els usuaris tornin a fer la mateixa consulta, l'SGBD no haurà de tornar a calcular el resultat, podrà tornar el resultat desat prèviament. Amb aquesta filosofia es van crear les vistes materialitzades.

Una **vista materialitzada** és un objecte de la base de dades emmagatzemat al dispositiu d'emmagatzematge no volàtil que conté el resultat d'una consulta precalculada. Les vistes materialitzades permeten un accés més eficient a les dades a canvi d'incrementar la mida de la base de dades.

Una vista materialitzada és semblant a una vista convencional. La diferència rau en el fet que en una vista convencional el contingut és virtual i, per tant, el contingut es calcula cada vegada que se sol·licita el seu valor, per a la qual cosa s'aplica un procés de traducció, mentre que una vista materialitzada sí que s'emmagatzema en el dispositiu d'emmagatzematge no volàtil i els seus valors són, en conseqüència, precalculats.

Per tant, podríem veure una vista materialitzada com una taula, però els valors són redundants i precalculats a partir d'altres dades (disponibles en taules o vistes materialitzades) de la base de dades. Les taules (o vistes materialitzades) sobre les quals es defineix la vista materialitzada reben la denominació genèrica de taules de base. El fet que les vistes materialitzades es comportin com a taules permet que es puguin crear índexs sobre les columnes que incorporen. En alguns casos serà possible modificar les dades directament en les vistes materialitzades, fet que propagaria els canvis a les taules de base usades en la definició de les vistes materialitzades. En aquests casos, es diu que la vista materialitzada és actualitzable. Igual que en el cas de vistes convencionals, la definició de vistes materialitzades actualitzables no és sempre possible.

Es pot accedir a les vistes materialitzades de manera explícita o implícita. Direm que a una vista materialitzada s'hi accedeix de manera explícita quan aquesta apareix explícitament a les sentències SQL de l'usuari. Les sentències SQL que s'han d'executar indiquen explícitament que s'ha d'accedir a la vista materialitzada com si fos una taula més. Aquesta opció obliga l'usuari a conèi-

xer les vistes materialitzades existents i les dades que contenen i pot limitar les opcions de l'optimitzador de consultes per escollir els plans de consulta més adequats.

S'accedeix a una vista materialitzada de manera implícita quan el seu nom no apareix de manera explícita a les sentències SQL que cal executar. En aquest cas, l'usuari escriu les sentències SQL com si la vista materialitzada no existís. L'optimitzador de consultes és l'encarregat de descobrir en quins casos s'ha d'utilitzar la vista materialitzada. Aquest mètode allibera l'usuari de saber quines vistes materialitzades existeixen i quan cal utilitzar-les. Tanmateix, la seva eficàcia depèn totalment del fet que l'optimitzador de consultes sigui capaç de detectar quan es pot fer servir una vista materialitzada per resoldre una consulta i en quins casos és beneficiós fer-ne ús. Desafortunadament, no tots els SGBD tenen aquest nivell de sofisticació.

Les dades d'una vista materialitzada s'hauran d'actualitzar periòdicament perquè no quedin obsoletes. Això és així perquè els canvis que es realitzen en les taules de base no es veuen per defecte automàticament reflectits en la vista materialitzada. Per això, freqüentment es diu que les vistes materialitzades constitueixen fotografies (*snapshots* en anglès) de la base de dades.

Determinar quan i com es calcula el contingut d'una vista materialitzada és crític i pot marcar la diferència entre una vista materialitzada eficient, o una que té uns costos de manteniment més elevats que els beneficis que se n'obtenen. Des d'un punt de vista teòric, les vistes materialitzades poden seguir diferents estratègies de manteniment:

- **Actualització automàtica**: les vistes materialitzades es poden actualitzar automàticament sense la intervenció de l'usuari. Hi ha diferents opcions, però les més comunes són l'actualització segons calendari (és a dir, marcar la periodicitat de l'actualització amb independència de les modificacions produïdes en les taules de base), actualització segons dades (l'SGBD detecta quan s'han modificat les dades en les taules de base i propaga els canvis a les vistes materialitzades afectades). La freqüència i el tipus de transaccions sobre les taules base i el tipus de funcions d'agregació utilitzades en la vista seran determinants per escollir una opció o una altra.

- **Actualització sota demanda**: l'usuari ha d'especificar quan s'ha d'actualitzar la vista materialitzada de manera explícita. És una opció poc recomanable en termes generals, però pot ser útil utilitzar-la en casos excepcionals. Per exemple, aquest seria el cas d'actualització segons calendari si hi ha hagut molts canvis abans de la data d'actualització.

- **Actualització completa o incremental**: en una actualització completa es destrueix la vista materialitzada per tornar-la a crear des de zero. En contraposició, l'actualització incremental només introdueix canvis en les dades que s'hagin vist afectades pels canvis en les taules de base. L'actualització

completa és menys eficient que la incremental, però té l'avantatge que es pot realitzar sempre. És més, en alguns casos pot ser l'única opció. Per exemple, si la vista materialitzada inclou funcions agregades, és possible que aquestes no es puguin calcular de manera incremental, de tal manera que sigui obligatori realitzar el càlcul des de zero en modificar les dades de les taules de base. Aquest seria el cas de la mitjana aritmètica.

En general, les vistes materialitzades són molt utilitzades en *data warehouses*. Això es deu al fet que permeten precalcular operacions d'agregació molt costoses i sobre gran quantitat de dades. També ajuden a precalcular agregacions per diferents nivells de granularitat del *data warehouse*, fet que facilita les operacions de *drill up* i *drill down* sobre aquest. Les decisions sobre la freqüència i la manera d'actualització de les vistes materialitzades, tal com s'ha argumentat anteriorment, seran d'importància primordial. Per exemple, si una vista materialitzada s'usa en un *data warehouse* en el qual el procés d'extracció, transformació i càrrega (*extraction, tranformation and load* o ETL en anglès) es realitza cada nit, serà suficient amb actualitzar la vista materialitzada després del procés ETL.

4.1. Definició i ús de vistes materialitzades a Oracle

A continuació mostrem de manera esquemàtica la sintaxi de creació de vistes materialitzades a Oracle:

```
CREATE MATERIALIZED VIEW materialized_view
  [column_alias [, column_alias]...]
  [[{physical_attributes_clause|TABLESPACE tablespace}]...]
  {REFRESH create_mv_refresh}
  [ENABLE QUERY REWRITE]
   ...
  AS subquery;
```

Oracle va ser el precursor de les vistes materialitzades, i com a tal permet un alt grau de personalització. Els elements principals en la creació d'una vista materialitzada són:

a) `Materialized_view`: nom de la vista materialitzada.

b) `Column_alias...`: Permet definir el nom de les columnes de la vista materialitzada.

c) `Physical_atributes_clause`: permet indicar els paràmetres físics de la vista materialitzada i el *tablespace* on assignar-la.

d) `REFRESH create_mv_refresh`: permet indicar quan i com actualitzar la vista materialitzada. Algunes de les opcions permeses són:

- Actualització completa o incremental:

Lectura recomanada

Més informació al manual d'Oracle
http://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_6002.htm#i2064161.

  - FAST: indica que les dades de la vista materialitzada s'actualitzaran de manera incremental.

  - COMPLETE: permet indicar que les dades de la vista materialitzada s'actualitzaran completament.

  - Si no s'especifica cap opció, l'SGBD intentarà realitzar l'actualització incremental. Si l'SGBD no pot realitzar l'actualització de manera incremental, llavors farà una actualització completa.

- Actualització automàtica segons dades, segons calendari o actualització sota demanda:
  - ON COMMIT: permet indicar que les dades de la vista materialitzada es modifiaran en modificar les dades de les taules de base.

  - START WITH... NEXT: permet indicar la data de càrrega inicial de dades (START WITH) i configurar l'actualització periòdica segons calendari. En concret, la sentència NEXT permet indicar l'interval de temps que especifica cada quant s'han de realitzar les càrregues periòdiques.

  - ON DEMAND: indica que les dades de la vista materialitzada es modificaran manualment. És l'opció per defecte si no s'usa l'opció ON COMMIT.

  - ENABLE QUERY REWRITE: permet indicar si volem que l'SGBD tingui en compte la vista materialitzada en el procés de reescriptura de consultes (en parlarem a les línies següents). Si no s'indica aquesta clàusula, la vista materialitzada no es tindrà en compte en el procés de reescriptura de consultes.

e) Subquery: indica la sentència SQL que permet calcular la vista materialitzada.

Per exemple, la sentència següent crearia una vista materialitzada per emmagatzemar les vendes d'una empresa agrupades per any i producte. La vista es crearia per primera vegada el dia posterior a l'execució de la sentència SQL, a les 11 del matí, i es refrescaria cada dilluns a les 15 de la tarda de manera incremental.

```
-- Donades les taules de times, products i sales:

CREATE TABLE times (
   time_id          DATE        PRIMARY KEY,
   calendar_year    VARCHAR(50),
   calendar_month   VARCHAR(50),
```

```
   calendar_day     VARCHAR(50),
   ...
);

CREATE TABLE products (
   prod_id          NUMBER(6)   PRIMARY KEY,
   prod_name        VARCHAR(50),
   prod_description VARCHAR(2000),
   ...
);

CREATE TABLE sales (
   prod_id          NUMBER(6),
   cust_id          NUMBER,
   time_id          DATE,
   quantity_sold    NUMBER(3),
   amount_sold      NUMBER(10,2),
   ...
);

-- Creem vista materialitzada que permet precalcular el total
de vendes agrupat per producte i any.
CREATE MATERIALIZED VIEW sales_mv
   TABLESPACE tbs_materialized_views
   REFRESH FAST START WITH ROUND(SYSDATE + 1) + 11/24
   NEXT NEXT_DAY(TRUNC(SYSDATE), 'MONDAY') + 15/24
   AS SELECT t.calendar_year, p.prod_id,
      SUM(s.amount_sold) AS sum_sales
      FROM times t, products p, sales s
      WHERE t.time_id = s.time_id AND p.prod_id = s.prod_id
      GROUP BY t.calendar_year, p.prod_id;
```

Oracle permet accedir a les vistes materialitzades de manera implícita i explícita. No obstant això, aconsella evitar l'accés explícit (fer servir el nom de la vista materialitzada en les sentències SQL), ja que això limita la possibilitat d'introduir canvis en les vistes materialitzades en un futur. Per utilitzar vistes materialitzades de manera explícita, només s'ha d'afegir la vista materialitzada en les sentències SQL. Per utilitzar vistes materialitzades de manera implícita, Oracle utilitza la funcionalitat de reescriptura de consultes (*query rewrite* en anglès).

La reescriptura de consultes és una tècnica d'optimització que transforma una sentència SQL que utilitza taules de la base de dades en una altra sentència SQL que és equivalent semànticament i que inclou una o més vistes materialitzades. Conceptualment, podem veure aquest procés com un procés en què

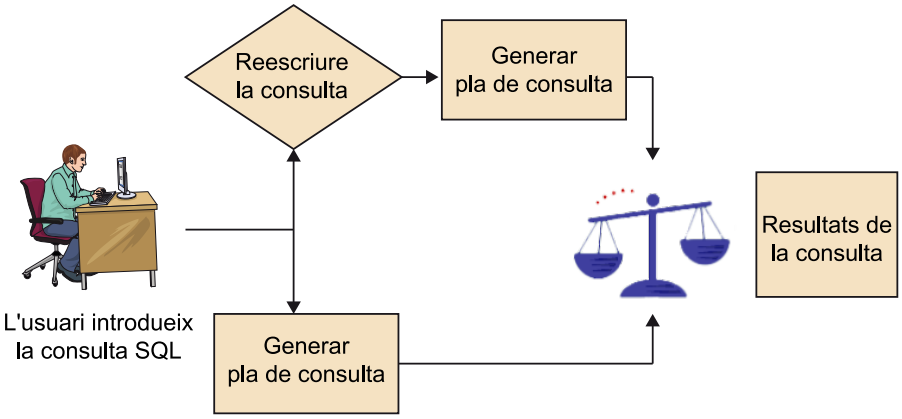
es reformula la sentència SQL perquè accedeixi a les dades per mitjà de les vistes materialitzades, en comptes de fer-ho per mitjà de les taules originalment indicades en la consulta.

La figura següent mostra el procés que segueix Oracle per identificar quan és necessari utilitzar una vista materialitzada de manera implícita. Quan arriba una consulta SQL, Oracle comprova si aquesta consulta es pot reescriure utilitzant vistes materialitzades. Si és així, Oracle reescriu la consulta i calcula el pla de consulta òptim per a la nova consulta SQL. A més, també calcula el pla de consulta òptim per a la consulta original. Finalment, compara els dos plans de consulta i tria el pla de consulta més eficient.

Lectura complementària

Més informació sobre les vistes materialitzades a Oracle i com funciona el procés de reescriptura de consultes en els capítols 9 (vistes materialitzades), 18 (reescriptura de consultes simple) i 19 (reescriptura de consultes avançada) del llibre *Oracle Database Warehousing Guide*: (http://docs.oracle.com/cd/E11882_01/server.112/e25554/toc.htm).

Figura 16. Mètode per escollir si s'ha d'utilitzar o no una vista materialitzada en la resolució d'una consulta



Si volem accedir a una vista materialitzada de manera implícita cal indicar-ho a la seva definició afegint la clàusula `ENABLE QUERY REWRITE`. Així mateix, l'ús de la reescriptura imposa certes restriccions sobre la vista materialitzada que hauran de ser comprovades i tingudes en compte en cada cas.

4.2. Definició i ús de vistes materialitzades a PostgreSQL

L'ús de vistes materialitzades a PostgreSQL no és tan complet com a Oracle. El motiu és que les vistes materialitzades no van ser incloses a PostgreSQL fins al setembre del 2013, a la versió 9.3.

La sintaxi de creació de vistes materialitzades a PostgreSQL és:

```
CREATE MATERIALIZED VIEW name
    [(column_name [, ...])]
    [WITH (storage_parameter [= value] [, ... ])]
    [TABLESPACE tablespace_name]
    AS query
    [WITH [NO] DATA]
```

Els elements principals en la creació d'una vista materialitzada són els següents:

- `name`: nom de la vista materialitzada.
- `(column_name [,...])`: permet definir el nom de les columnes de la vista materialitzada.
- `storage_parameter...`: permet indicar els paràmetres físics de la vista materialitzada.
- `TABLESPACE`: permet indicar el *tablespace* que s'ha d'utilitzar.
- `query`: indica la sentència SQL que permet calcular la vista materialitzada.
- `WITH DATA` o `WITH NO DATA`: permet especificar si la vista materialitzada ha de ser poblada (`WITH DATA`) o no (`WITH NO DATA`) en temps de creació.

Versió de PostgreSQL

La versió de PostgreSQL utilitzada per realitzar aquests materials és la 9.3. És possible que elements de disseny físic presentats en aquest document s'actualitzin i s'hi afegeixin noves funcionalitats en versions posteriors.

A PostgreSQL, les vistes materialitzades s'han d'actualitzar manualment. Per tant, caldrà indicar mitjançant una sentència SQL quan s'han d'actualitzar les dades de la vista materialitzada. La sentència SQL utilitzada és la que es presenta a continuació.

```
REFRESH MATERIALIZED VIEW name
    [WITH [NO] DATA]
```

Per defecte, la sentència actualitza les dades de la vista materialitzada a partir de la sentència SQL indicada en la seva creació. L'addició de noves dades no es fa de manera incremental, sinó que es buida la vista materialitzada i es torna a poblar de nou. És possible afegir la clàusula `WITH NO DATA` a l'hora d'executar la sentència `refresh`. En aquest cas, les dades de la vista actualitzada s'eliminaran, però aquesta no es poblarà amb noves dades.

El `refresh` de vistes materialitzades a PostgreSQL realitza un bloqueig exclusiu i total de la vista materialitzada. Això causa que aquesta no estigui disponible mentre s'actualitza. Això és un inconvenient important atès que limita l'ús de les vistes materialitzades a PostgreSQL, ja que l'actualització es realitza totalment (no és incremental) i que el temps d'actualització pot ser considerable.

Per exemple, la sentència següent crearia una vista materialitzada per emmagatzemar les vendes d'una empresa agrupades per any i producte. La vista materialitzada es crearia i es poblaria en el moment de la seva creació.

```
-- Donades les taules següents de times, products i sales:
CREATE TABLE times (
   time_id          DATE     PRIMARY KEY,
   calendar_year    VARCHAR(50),
```

```
   calendar_month   VARCHAR(50),
   calendar_day     VARCHAR(50),
   ...
);

CREATE TABLE products (
   prod_id          NUMBER   PRIMARY KEY,
   prod_name        VARCHAR(50),
   prod_description VARCHAR(2000),
   ...
);

CREATE TABLE sales (
   prod_id          NUMBER,
   cust_id          NUMBER,
   time_id          DATE,
   quantity_sold    NUMBER(3),
   amount_sold      NUMBER(10,2),
   ...
);

-- Creem una vista materialitzada que permet precalcular el total de vendes agrupat per producte i any
per producte i any
CREATE MATERIALIZED VIEW sales_mv
   TABLESPACE tbs_materialized_views
   AS SELECT t.calendar_year, p.prod_id,
      SUM(s.amount_sold) AS sum_sales
      FROM times t, products p, sales s
      WHERE t.time_id = s.time_id AND p.prod_id = s.prod_id
      GROUP BY t.calendar_year, p.prod_id
   WITH DATA;
```

Respecte a com accedir a les vistes materialitzades, PostgreSQL, de moment, només permet accedir de manera explícita. Per tant, per utilitzar vistes materialitzades en una consulta, cal incloure les seves referències explícites a la consulta SQL.

Resumen

Els continguts d'aquest mòdul han estat creats amb l'objectiu d'introduir el lector en el món del disseny físic de les bases de dades del qual és responsable l'administrador de la base de dades.

El mòdul comença definint què és el disseny físic, comentant quins efectes té un bon disseny físic a la base de dades i presentant els components d'emmagatzematge d'una base de dades.

A continuació es descriuen detalladament alguns dels elements de disseny físic de bases de dades més significatives. Es comença amb els *tablespaces*, indicant quin és el seu objectiu i com s'han de fer servir. Es continua amb els índexs, descrivint els índexs basats en arbres B+, en dispersió i en mapes de bits, introduint els conceptes d'optimització i plans de consultes amb l'objectiu de mostrar al lector com avaluar l'impacte dels índexs en una base de dades, i presentant algunes regles que cal tenir en compte en el disseny d'índexs. Finalment, es presenta el concepte de vista materialitzada, indicant quina és la seva funció i què s'ha de tenir en compte a l'hora de crear-les. Tots els elements de disseny físic presentats es descriuen primer de manera teòrica i posteriorment s'indica com s'han de fer servir en les últimes versions d'Oracle i PostgreSQL.

En cas que algun lector vulgui aprofundir en el disseny físic de bases de dades, pot consultar les referències que hem anat indicant al llarg del mòdul o els llibres indicats en l'apartat de bibliografia.

Ejercicios de autoevaluación

1. Tenint en compte la taula següent (clau primària subratllada) que guarda informació sobre diferents centres de *fitness*:

Centre (nom, adreça, població, m2)

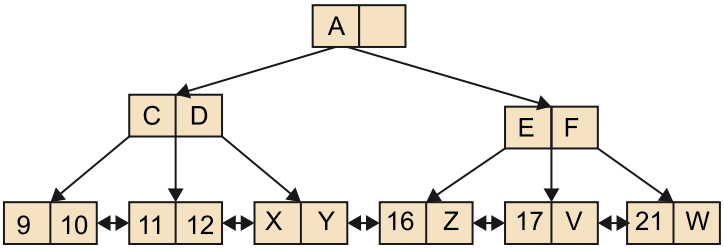
Digueu quin tipus d'accés implica les consultes que es mostren a continuació:

1) `SELECT * FROM centre ORDER BY poblacio`

2) `SELECT * FROM centre WHERE m2> 100 AND m2 <150`

3) `SELECT FROM centre WHERE poblacio = 'Barcelona' AND m2> 100`

2. Tenint en compte l'arbre B+ següent, d'ordre *d* igual a 1 (d = 1), que correspon a un índex definit sobre una taula *T* per a una columna de tipus enter que és clau primària de la taula *T*:



Es demana:

a) Indiqueu quins són els valors possibles per a X, Y, Z, V i W.

b) Indiqueu quins són els valors possibles per a C i D.

c) Indiqueu quins són els valors possibles per a E i F.

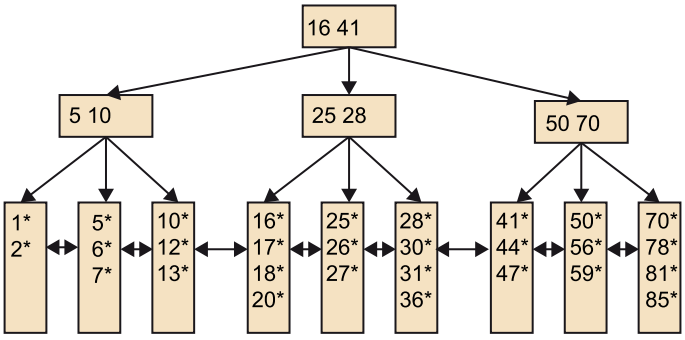
d) Indiqueu quins són els valors possibles per a A.

e) Sabent que els valors que cal indexar sobre l'arbre B+ ocupen 32 *bytes*, els apuntadors a registres de dades (RID) 4 *bytes*, els apuntadors a nodes 2 *bytes* i les pàgines són de 4 Kb, indiqueu quin seria l'ordre *d* més òptim (el valor més gran possible per a *d*) de l'arbre B+.

3. Tenim una taula que guarda dades sobre palaus de congressos amb l'esquema següent (clau primària subratllada):

palaus_congressos(codi_palau, ciutat, importància, capacitat)

Les dades de la taula prèvia estan emmagatzemades en un sol fitxer, que únicament emmagatzema dades d'aquesta taula. Aquest fitxer consta de deu pàgines. Sobre la columna *codi_palau* (que és de tipus enter) s'ha construït l'arbre B+ d'ordre d = 2 que es mostra a la figura següent. Cada node de l'índex constitueix una pàgina del fitxer que guarda les dades de l'índex.



a) Suposant que l'arbre B+ és no agrupat, digueu si l'arbre B+ ajuda o no a resoldre les consultes següents. Indiqueu també quin serà el cost mínim i màxim (en nombre d'operacions d'E/S) de resolució de les consultes.

1) `SELECT * FROM palaus_congressos WHERE codi_palau = 80`

2) `SELECT * FROM palaus_congressos WHERE codi_palau BETWEEN 20 AND 26`

3) `SELECT * FROM palaus_congressos WHERE ciutat = 'Barcelona' AND importancia> 50`

b) Suposant ara que l'arbre B+ és agrupat, canviaríeu alguna de les respostes formulades a l'apartat anterior?

4. Suposem que tenim un índex de dispersió amb la funció de dispersió següent: h(X) = (X mod 3) + 1, en què X representa el valor que s'ha d'inserir. A cada pàgina només hi caben dos valors, i disposem de tres pàgines primàries.

a) Indiqueu com quedaria l'índex si inserim els valors següents: 5, 3, 47, 94, 123 i 214. Calculeu el factor de càrrega després de la inserció del conjunt de valors.

b) Quina solució proposaríeu si a continuació dels sis valors anteriors hem d'afegir deu milions més de valors? Calculeu el factor de càrrega associat a la solució donada.

c) I si aquesta inserció de deu milions de valors fos només temporal, per a la realització d'uns càlculs puntuals, i després s'esborressin el 75% d'aquests valors perquè ja no són necessaris?

5. Suposem que hem de gestionar una base de dades de venda de productes per internet. La taula de vendes de la base de dades té un nombre de files important i les consultes que es realitzen sobre ella comencen a demorar-se en excés. La definició de la taula de vendes és la següent:

Sale(saleID, saleCostumer, salePrice, saleDate, saleCountry, saleDayOfWeek, salePaymentMethod, saleHour)

En què *saleID* és la clau primària, *saleCostumer* és el client que va realitzar la venda, *salePrice* indica el preu de la venda, *saleDate* indica la data en què es va realitzar la venda, *saleCountry* indica el país on es va realitzar la venda, *saleDayOfWeek* indica el dia de la setmana (diumenge, dilluns, dimarts, etc.) en què es va realitzar la venda, *salePaymentMethod* indica el mètode de pagament amb què es va realitzar la venda i *saleHour* és un enter entre 1 i 24 que indica l'hora en què es va realitzar la venda.

Ens demanen el següent:

a) Creeu els índexs de mapa de bits necessaris perquè els accessos a la taula de vendes siguin el més ràpids possibles. Cal tenir en compte les dades següents:

1) La taula de vendes té cinc milions de files.

2) Les consultes que es volen optimitzar són les que realitza el departament de *màrqueting*, que són del tipus següent:

- Les deu vendes més nombroses per país, per forma de pagament o per dia de la setmana en un període de temps determinat.
- El nombre de vendes i la quantia de les vendes setmanals en funció del país d'origen, de la forma de pagament i del dia de la setmana.

3) La clau primària i la columna *saleDate* ja estan indexades amb un índex basat en arbre B+.

4) Hi ha una mitjana de vint vendes per minut.

b) Suposeu que l'empresa canvia l'SGBD per un que és tan avançat que permet realitzar modificacions a les columnes indexades amb índex de mapa de bits sense bloquejar l'índex.

1) Quins índexs crearíeu sota aquest supòsit?

2) Quant ocuparia cada índex?

3) Indiqueu els valors dels índexs creats per les files de la taula:

saleID	saleCostumer	salePrice	saleDate	saleCountry	saleDayOfWeek	salePaymentMethod	saleHour
54478	112	630	9/12/2013	Espanya	Diumenge	Credit card	02
54479	114	6	9/12/2013	Andorra	Diumenge	PayPal	04
54480	115	3	9/12/2013	Portugal	Diumenge	PayPal	05
54481	212	1200	9/12/2013	Alemanya	Diumenge	Credit card	05
54482	154	50	9/12/2013	Xina	Diumenge	Check	05
54483	25	680	9/12/2013	Vietnam	Diumenge	PayPal	07

e) Indiqueu com s'utilitzarien els índexs creats a l'execució de la consulta següent prenent només com a referència les dades de la taula anterior.

```
SELECT salePaymentMethod, count(*)
FROM sale
WHERE saleCountry IN ( 'Xina', 'Vietnam')
GROUP BY salePaymentMethod
```

6. Donades les taules següents (claus primàries subratllades) d'una base de dades del departament postvenda d'una cadena de botigues:

Clients (codi_client, nom, edat, sexe)

Compres (num_compra, codi_client, codi_producte, codi_botiga, data, quantitat)

codi_producte és clau forana a *Producte* i *codi_botiga* és clau forana a *Botiga*.

Incidencies_postvenda (num_incidencia, codi_producte, tipus_incidencia, data, resultat)

codi_producte és clau forana a *Producte* i *tipus_incidencia* és clau forana a *Tipus_incidencia*. No es manté informació del client que ha generat la incidència. Les incidències estan tipificades (*tipus_incidencia*), hi ha només deu incidències diferents i es troben descrites a la taula *Tipus_incidencia*.

Les taules *Tipus_incidencia*, *Producte* i *Botiga* no són rellevants per a la resolució de l'exercici, per això no se n'inclou l'esquema. La taula *Clients* ocupa 100.000 pàgines, la taula *Compres* 2.000.000 de pàgines i la taula *Incidencies_postvenda* ocupa 300.000 pàgines. Les dades de cada taula s'emmagatzemen en un sol fitxer de dades que únicament guarda dades de la taula amb la qual es relaciona.

Després d'observar les consultes d'accés per valor que es fan, veiem que els criteris de selecció més freqüents són els següents:

a) 10% de consultes seleccionen respecte del valor de la columna *Clients.codi_client*.

b) 35% de consultes seleccionen respecte del valor de la columna *Clients.nom*.

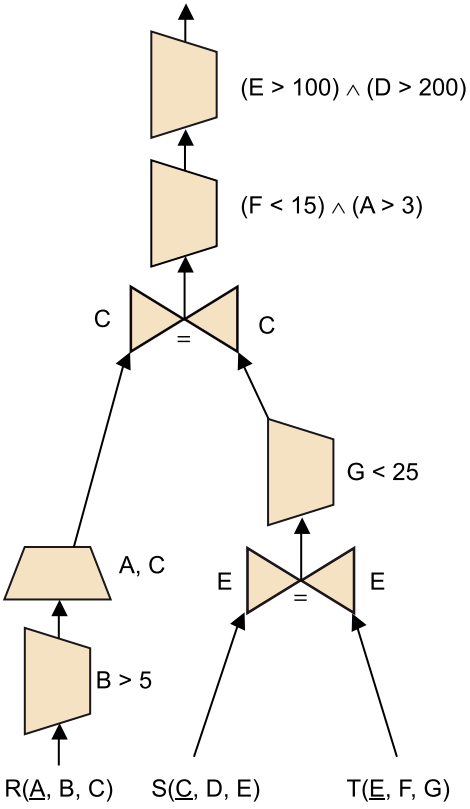
c) 40% de consultes seleccionen respecte del valor de la columna *Compres.codi_producte*.

d) 5% de consultes seleccionen respecte el valor de la columna *Incidencies_postvenda.tipus_incidencia*.

e) 10% de consultes seleccionen respecte del valor de la columna *Incidencias_postvenda.data*.

Sabem que sobre aquestes taules no hi ha cap índex definit, però es volen crear per tal d'accelerar les consultes més freqüents que es fan. Suposem que disposem dels recursos necessaris per construir dos índexs en forma d'arbre B+ no agrupat, i que amb aquests podrem aconseguir temps de resposta inferiors, i significativament millors que els obtinguts mitjançant un recorregut seqüencial. Sobre quines columnes construiríeu els dos índexs?

7. Donat l'arbre sintàctic següent, en el qual les columnes subratllades són les claus primàries de cada taula, baixeu les operacions de selecció tant com sigui possible a l'arbre per obtenir l'arbre sintàctic òptim:



8. Suposeu que acabem de crear la vista materialitzada següent en un SGBD PostgreSQL tal com es mostra a continuació:

```
-- Donades les taules següents de times, products i sales:
CREATE TABLE times (
   time_id          DATE     PRIMARY KEY,
   calendar_year    VARCHAR(50),
   calendar_month   VARCHAR(50),
   calendar_day     VARCHAR(50),
   ...
);

CREATE TABLE products (
   prod_id          NUMBER   PRIMARY KEY,
   prod_name        VARCHAR(50),
   prod_description VARCHAR(2000),
   ...
);

CREATE TABLE sales (
   prod_id          NUMBER,
   cust_id          NUMBER,
   time_id          DATE,
   quantity_sold    NUMBER(3),
   amount_sold      NUMBER(10,2),
   ...
);

-- Creem una vista materialitzada que permet precalcular el total
de vendes agrupat per producte i any
CREATE MATERIALIZED VIEW sales_mv
   TABLESPACE tbs_materialized_views
   AS SELECT t.calendar_year, p.prod_id,
      SUM(s.amount_sold) AS sum_sales
      FROM times t, products p, sales s
      WHERE t.time_id = s.time_id AND p.prod_id = s.prod_id
      GROUP BY t.calendar_year, p.prod_id
   WITH DATA;
```

Es demana:

a) Feu servir disparadors per fer que la vista materialitzada s'actualitzi cada vegada que s'afegeix/esborra/modifica una fila a les taules `times`, `products` i `sales`.

b) Indiqueu quins problemes podria comportar l'actualització de la vista que acabeu d'implementar.

Solucionario

Ejercicios de autoevaluación

1. La primera consulta és un accés seqüencial per valor, atès que cal recórrer totes les files de la taula centre considerant el valor de la columna població que ens permetrà la seva ordenació.

La segona consulta és un accés seqüencial per valor sobre la taula centre, en haver de recórrer totes les files contingudes en aquesta taula prenent en consideració únicament les que compleixen la condició que els seus m2 estan entre 100 i 150.

L'última consulta és un accés mixt sobre la taula centre, on s'efectua un accés directe per valor sobre la columna població i un accés seqüencial per valor sobre la columna m2.

2. Es proposa la solució següent per a cada un dels apartats plantejats:

Segons la definició d'arbre B+, els valors a les fulles estan ordenats i no n'hi ha de repetits, per tant, $12 < X < Y < 16$. Partint d'aquesta premissa, els elements poden prendre els valors següents:

• X = 13, llavors Y = 14 o 15

• X = 14, llavors Y = 15

• X = 15, llavors Y podria no tenir cap valor associat.

• Z no pot contenir cap valor ja que hauria de ser $16 < Z < 17$.

• V pot ser 18, 19 o 20.

• $W > 21$

Segons la definició de node intern d'un arbre B+, $C > 10$ i $C <= 11$; per tant, només pot prendre el valor 11. De manera similar, $D > 12$ i $D <= X$, però tenint en compte que tots els valors dels nodes interns han d'estar a les fulles D també ha d'estar a les fulles, ha de ser igual a X. En resum, C = 11 i D = X (cal fer notar que en realitat D no pot ser menor que X).

Els valors d'E i F, estan determinats, només poden ser 17 i 21, pel mateix argument exposat per l'element C.

El valor d'A ha de complir que ha de ser menor que tots els valors de la dreta de l'arbre i a més ha d'estar a les fulles. Per tant, necessàriament ha de ser 16.

Per saber l'ordre òptim de l'arbre, hem de calcular els valors que cabrien als nodes interns i nodes fulles:

• Nodes interns: $2d * 32 + (2d + 1) * 2 = 4.096 \Rightarrow d = 60$

• Nodes fulla: $2d * (32 + 4) + 2 * 2 = 4.096 \Rightarrow d = 56$

Per tant, l'ordre òptim de l'arbre B+ és 56.

3. Es proposa la solució següent per a cada un dels apartats plantejats:

a) Es resoldran de manera eficient la primera i la segona de les consultes, que impliquen, respectivament, un accés directe per valor i un accés seqüencial per valor. En el cas de la darrera consulta l'índex no resulta de cap utilitat, atès que l'arbre B+ no indexa valors de ciutat ni d'importància. En relació amb els costos, aquests serien:

1) El cost (mínim i màxim) seria tres operacions d'E/S, que correspon a la lectura de tres nodes de l'arbre B+ (l'últim seria el node fulla que hauria de contenir l'entrada buscada). Com que no existeix el palau amb codi 80, no es desencadena cap operació d'E/S en el fitxer que conté les dades dels palaus de congressos.

2) El cost mínim serà de cinc operacions d'E/S: quatre a l'índex en forma d'arbre B+ per recuperar les entrades d'interès més una en el fitxer que conté les dades de la taula de palaus de congressos (assumim que totes les files buscades, tres en total, es troben a la mateixa pàgina). Per la seva banda el cost màxim serà de set operacions d'E/S: quatre a l'arbre B+ més tres al fitxer que conté les dades de la taula (assumim que cadascuna de les files que conforma el resultat de la consulta està en una pàgina diferent del fitxer de dades).

3) El cost (mínim i màxim) de la darrera consulta serà de deu operacions d'E/S, atès que per resoldre la consulta cal recórrer totes les pàgines del fitxer que conté les dades dels palaus de congressos.

b) Canviaria la resposta a la consulta número 2 que es veuria beneficiada. Tenint en compte que l'arbre B+ és agrupat, les dades dels palaus de congressos estan emmagatzemades físicament al fitxer de dades segons el valor de la columna sobre la qual s'ha construït l'índex. El cost (mínim i màxim) seria de quatre o cinc operacions d'E/S, tres a l'arbre B+ (a causa de l'ordenació física del fitxer de dades només cal accedir al primer node fulla amb entrades d'interès) més una o dues al fitxer de dades (una si les tres files a recuperar estan en una única pàgina o dues si estan en pàgines consecutives, atès que podem afirmar que si el fitxer de dades té deu pàgines, de mitjana, hi ha entre dues i tres files en cada pàgina).

4. A continuació es proporcionen les respostes a cada un dels apartats plantejats:

a) Obtindríem el resultat següent:

		L	
	1	3	123
N	2	94	214
	3	5	47

El factor de càrrega C seria:

C = M / (N * L)
C = 6 valors / (3 pàgines * 2 valors en cada pàg.) = 1

Per tant, el factor de càrrega es 1.

b) Una primera aproximació a la solució ens podria portar a pensar que tots aquests valors podrien gestionar-se mitjançant pàgines d'excedents, però això no tindria sentit, hauríem de gestionar milions de files en pàgines d'excedents, fet que comportaria un nombre significatiu d'accessos (operacions d'E/S) a les pàgines d'excedents que faria ineficient aquest índex basat en la dispersió.

Per tant, la solució és reformular la funció de dispersió, que equival a canviar el nombre de pàgines primàries a considerar, un cop fixada una C objectiu (en cas contrari no podríem aïllar la N):

C = 10.000.006 / (N * 2)

On N és el nombre de pàgines primàries, i el nou interval [1,..., N] que resulta de la nova funció de dispersió.

c) En aquest cas en què l'índex creix i disminueix, està clar que un model de dispersió estàtica no ens serveix i hauríem d'anar a un model d'índex de dispersió dinàmica (*linear hashing* o *extensible hashing*).

5. A continuació es presenta la solució per a cada un dels supòsits plantejats:

a) No es crearia cap índex de tipus mapa de bits ja que la taula té un gran nombre d'insercions. En cas de crear algun índex de mapa de bits sobre la taula, cada inserció bloquejaria l'índex, evitant que altres operacions concurrents poguessin accedir-hi fins que la inserció acabés. Això podria donar problemes en un sistema que requereix processar operacions en temps real.

b) Si no hi ha la restricció comentada a l'apartat anterior llavors es crearien índexs de mapa de bits sobre les columnes següents:

saleCountry: ja que té un nombre de valors limitat i és utilitzat en les operacions de cerca.

saleDayOfWeek: ja que només té set possibles valors que es repetiran en moltes files i és utilitzat en les operacions de cerca.

salePaymentMethod: ja que té pocs valors i s'utilitza en les operacions de cerca comentades.

c) Els índexs creats ocuparien:

saleCountry: la botiga realitza vendes per Internet, per tant, poden comprar des de qualsevol país del món. Segons l'ONU hi ha 193 països al món. Per tant, la mida de l'índex serà: nombre de possibles valors (193) * nombre de files (5.000.000) = 965.000.000 bits = 115 *megabytes* aproximadament.

saleDayOfWeek: 7 possibles valors * 5.000.000 = 35.000.000 bits, uns 4 *megabytes*.

salePaymentMethod: suposant que només hi ha 3 possibles valors (*PayPal*, *credit card*, *check*) seria: 3 * 5.000.000 = uns 2 *megabytes*.

d) Els índexs creats tindrien els valors següents a la taula de referència:

Índex sobre *saleCountry*

Espanya	Andorra	Alemanya	Xina	Vietnam
1	0	0	0	0
0	1	0	0	0
1	0	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

Índex sobre *saleDayOfWeek*

Diumenge	Dilluns	Dimarts	Dimecres	Dijous	Divendres	Dissabte
1	0	0	0	0	0	0
1	0	0	0	0	0	0
1	0	0	0	0	0	0
1	0	0	0	0	0	0
1	0	0	0	0	0	0
1	0	0	0	0	0	0

Índex sobre *salePaymentMethod*

Credit card	PayPal	Check
1	0	0
0	1	0
0	1	0
1	0	0
0	0	1
0	1	0

e) Per resoldre la consulta de referència primer es realitzaria el filtre de la clàusula `WHERE`. Per identificar les files que representen vendes de la Xina i Vietnam es realitzaria una operació `OR` lògica entre els dos últims mapes de bits de l'índex sobre *saleCountry*. Podem veure el resultat a les taules:

Xina	Vietnam
0	0
0	0
0	0
0	0
1	0
0	1

Xina OR Vietnam
0
0
0
0
1
1

De l'operació resultant podem comprovar que les úniques files que ens interessen són les dues últimes. Per realitzar l'agrupació es poden utilitzar els mapes de bits de l'índex sobre *salePaymentMethod*. La manera de fer-ho seria fent una operació `AND` lògica entre el resultat anterior (*Xina OR Vietnam*) i el mapa de bits de cada grup de l'agrupació, on hi ha un grup per a cada valor de la columna *salePaymentMethod*. Podem veure el resultat d'executar l'operació sobre cada agrupació a continuació:

Credit card	PayPal	Check	Xina OR Vietnam
1	0	0	0
0	1	0	0
0	1	0	0
1	0	0	0
0	0	1	1
0	1	0	1

(Xina OR Vietnam) AND Credit card	(Xina OR Vietnam) AND PayPal	(Xina OR Vietnam) AND Check
0	0	0
0	0	0
0	0	0
0	0	0
1	1	1
1	1	1

Dels resultats obtinguts podem veure que no hi ha cap fila que satisfaci les condicions en el primer grup (*Credit card*), que en el segon grup (*PayPal*) hi ha només una fila (l'última), i que en el tercer grup (*Check*) hi ha una sola fila (la penúltima).

6. Els avantatges que s'obtenen pel fet de tenir implementat un índex sobre una columna són més significatius com més gran és el nombre de pàgines de la taula que s'ha de consultar. Aquest factor, però, també s'ha de combinar amb la freqüència d'execució de cada consulta.

Tenint en compte això, i el fet que només podem crear dos índexs, ràpidament podem veure que tenim un clar candidat: la consulta sobre la taula *Compres* és la que més vegades s'executa i és, a més, la taula més voluminosa (2.000.000 de pàgines) amb molta diferència respecte de la resta de taules.

Si mirem la resta de consultes, veurem que dues actuen sobre *Clients* i dues sobre *Incidencies_postvenda*. Si decidim crear un índex sobre *Clients* ho farem sobre la columna *nom*, atès que és la consulta que, sobre la taula *Clients*, s'executa més vegades. Si creem l'índex sobre *Incidencies_postvenda*, per motius similars als anteriors, el crearem sobre la columna *data*.

Per acabar de decidir-nos entre l'índex sobre *Clients.nom* o sobre *Incidencies_postvenda.data*, hem de fer càlculs. Si no hi ha cap índex, les consultes només es podran resoldre fent recorreguts seqüencials dels fitxers que emmagatzemen dades contingudes a les taules. En el pitjor cas, per resoldre les consultes, caldrà recórrer totes les pàgines d'aquests fitxers. De cada cent consultes, trenta-cinc actuen sobre *Clients.nom* i deu actuen sobre *Incidencies_postvenda.data*. Per tant, el nombre de pàgines accedides en cada cas serà:

Cas (b) consultes que seleccionen *Clients.nom*

$$100.000 \text{ pàgines} * 35 = 3.500.000 \text{ pàgines}$$

Cas (e) consultes que seleccionen *Incidencies_postvenda.data*

$$300.000 \text{ pàgines} * 10 = 3.000.000 \text{ pàgines}$$

Per tant, els dos índexs que cal crear seran: un primer índex sobre *Compres.codi_producte* i un segon índex sobre *Clients.nom*.

Si fem els càlculs per a totes les consultes, l'elecció és, òbviament, la mateixa:

Cas (a) consultes que seleccionen. *Clients.codi_client*

100.000 pàgines * 10 = 1.000.000 pàgines

Cas (b) consultes que seleccionen. *Clients.nom*.

100.000 pàgines * 35 = 3.500.000 pàgines

Cas (c) consultes que seleccionen. *Compres.codi_producte*

2.000.000 pàgines * 40 = 80.000.000 pàgines

Cas (d) consultes que seleccionen. *Incidencies_postvenda.tipus_incidencia*

300.000 pàgines * 5 = 1.500.000 pàgines

Cas (e) consultes que seleccionen. *Incidencies_postvenda.data*

300.000 pàgines * 10 = 3.000.000 pàgines

7. L'arbre resultant és l'òptim després d'haver baixat les operacions de selecció $(A > 3) \wedge (D > 200) \wedge (E > 100) \wedge (F > 15) \wedge (G > 25)$. Analitzant cadascuna de les restriccions de manera independent podem afirmar que:

• La restricció A > 3 pot baixar per la branca de la taula R, ja que és una columna que pertany únicament a aquesta taula.

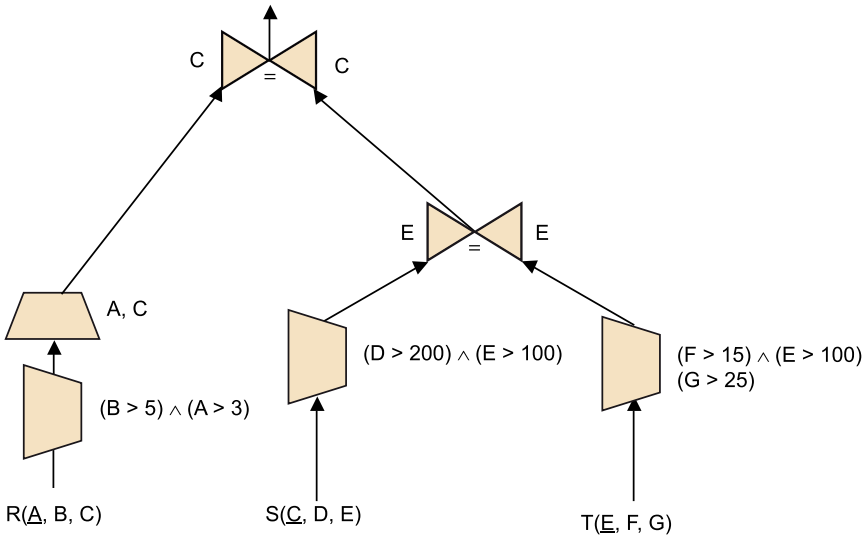
• La restricció D > 200 pot baixar per la branca de la taula S, ja que és una columna que pertany únicament a aquesta taula.

• La restricció F > 15 pot baixar per la branca de la taula T, ja que és una columna que pertany únicament a aquesta taula.

• La restricció G > 25 pot baixar per la branca de la taula T, ja que és una columna que pertany únicament a aquesta taula.

• La restricció E > 100 ha de baixar per les branques de les taules S i T, ja que afecta les dues taules. Aquestes restriccions es poden agrupar amb les baixades en els passos anteriors.

L'arbre resultant és el que es mostra a continuació:



8. Les respostes a l'exercici plantejat són les següents:

a) Primer es crearia una funció que forcés l'actualització de la vista materialitzada i posteriorment es crearia un disparador per a cada taula, indicant que en afegir/modificar/eliminar una fila sobre les taules afectades s'ha d'executar la funció que actualitza la vista materialitzada:

```
CREATE OR REPLACE FUNCTION refreshMateralizationViewSales()
   Returns trigger As '
Begin
   Refresh Materialized View sales_mv;
   Return NULL;
End

LANGUAGE plpgsql;

Create Trigger refreshMaterializedViewOnTimes
After INSERT Or UPDATE Or DELETE
On times For Each Row
   Execute Procedure refreshMateralizationViewSales();

Create Trigger refreshMaterializedViewOnProducts
After INSERT Or UPDATE Or DELETE
On products For Each Row
   Execute Procedure refreshMateralizationViewSales();

Create Trigger refreshMaterializedViewOnSales
After INSERT Or UPDATE Or DELETE
On sales For Each Row
   Execute Procedure refreshMateralizationViewSales();
```

b) Tal com s'ha comentat als materials, actualment PostgreSQL realitza càrregues completes de les vistes materialitzades i bloquejos de la vista sencera quan s'actualitza. Si hi hagués una freqüència molt alta d'actualització de les taules afectades (`times`, `sales`, `products`) podria resultar en un bloqueig permanent de la vista. Això faria impossible la consulta, mentre que suposaria un gran malbaratament de recursos (CPU i memòria RAM).

Glosario

arbre sintàctic d'una consulta *m* Representació interna en forma d'arbre d'una consulta, en què les fulles de l'arbre representen les taules implicades en la consulta, els nodes intermedis corresponen a les operacions demanades a la consulta i l'arrel de l'arbre és la resposta a la consulta formulada.

arbre B+ *m* Estructura de dades que s'utilitza per organitzar índexs que permeten implementar l'accés directe i l'accés seqüencial per valor.

arquitectura de components d'emmagatzematge *f* Esquema de tres nivells, lògic, físic i virtual, amb el qual es classifica i es descriu cada component dels SGBD, especialment els que estan relacionats amb l'emmagatzematge de les dades.

dispersió *f* Manera d'organitzar valors que es pot utilitzar en índexs que implementen l'accés directe per valor.

entrada *f* Element d'un índex que consisteix, en el cas més simple, en una parella formada per valor i identificador (o adreça física, RID) de la fila que conté aquest valor.

espai virtual *m* Seqüència de pàgines virtuals. Proporciona a l'SGBD una visió simplificada del nivell físic.

extensió *f* Unitat d'assignació d'espai en un dispositiu d'emmagatzematge no volàtil. Cada extensió és un nombre enter de pàgines consecutives i està continguda dins d'un fitxer.

factors de cost *m pl* Aspectes que influeixen en el cost total d'execució d'una consulta.

fitxer *m* Unitat de gestió de l'espai en els dispositius d'emmagatzematge no volàtil. Està format per una o més extensions. Normalment l'SO gestiona els fitxers en lloc de l'SGBD.

índex *m* Estructura de dades que els SGBD utilitzen per facilitar les cerques necessàries per implementar els accessos per valor a un o diversos valors.

índex agrupat *m* Índex que garanteix que les dades indexades també estan ordenades físicament segons l'ordre de l'accés per valor (directe o seqüencial) que proporcionen.

mapa de bits *m* seqüència de bits que indica quines files d'una taula satisfan una condició. Hi ha d'haver un bit per a cada fila de la taula, i el seu valor (0,1) ha d'indicar si la fila compleix la condició especificada o no.

memòria interna *f* Espai de la memòria principal de l'ordinador, normalment bastant gran, dedicat a contenir les pàgines que gestiona l'SGBD.

mètode d'accés *m* Algorisme que permet consultar índexs i taules per buscar les dades requerides.

nivell físic *m* Nivell que engloba els components físics (fitxer, extensió i pàgina) dins de l'arquitectura de components d'emmagatzematge.

nivell lògic *m* Nivell que engloba els components lògics (taules, vistes, restriccions, etc.) dins de l'arquitectura de components d'emmagatzematge.

nivell virtual *m* Nivell que engloba el component virtual dins de l'arquitectura de components d'emmagatzematge.

operació d'entrada/sortida *f* Procés que permet el transport de pàgines entre la memòria interna i la memòria externa de l'ordinador. En cada operació d'entrada/sortida (E/S) es transfereix un cert nombre de pàgines. En general, per simplificar els càlculs de cost, s'assumeix que en cada operació d'E/S es transfereix una pàgina.

optimització de consultes *f* Procés que porta a terme l'SGBD destinat a determinar quina és la millor estratègia d'execució d'una consulta.

optimització física d'una consulta *f* Procés que avalua el cost total d'execució d'una consulta i determina quins algorismes o mètodes d'accés s'han d'utilitzar per resoldre les diferents operacions de la consulta, tenint en compte les característiques de les estructures d'emmagatzematge i dels camins d'accés (índexs) que s'han definit per accedir a les dades.

optimització sintàctica d'una consulta *f* Procés que determina quin és l'ordre òptim d'execució de les operacions d'àlgebra relacional incloses en una consulta.

pàgina *f* Unitat mínima d'accés i de transferència de dades entre la memòria interna (o principal o intermèdia) de l'ordinador i la memòria externa (no volàtil) que conté els fitxers d'una base de dades. L'SO de la màquina porta a terme aquesta transferència i passa la pàgina a l'SGBD perquè en gestioni el contingut. La pàgina també és l'estructura que permet a l'SGBD organitzar les dades d'una base de dades. A l'àrea d'SO la pàgina rep el nom de bloc.

partició *f* Sistema d'emmagatzematge que permet distribuir les dades d'una taula en diferents espais físics per augmentar l'eficiència del sistema.

pla d'accés d'una consulta *f* Estratègia d'execució d'una consulta que té associat un cost mínim.

vista materialitzada *f* Vista de la base de dades que conté el resultat d'una consulta precalculada i s'emmagatzema en el dispositiu d'emmagatzematge no volàtil (en general, disc). S'utilitza per augmentar el temps de resposta en operacions comunes i costoses a canvi d'augmentar l'espai d'emmagatzematge.

Bibliografía

Abelló, A.; Rollón, E.; Rodríguez, M. E. (2008). *Database Design and Administration. Computer Science Engineering. Aula Politècnica* (núm. 131). Edicions UPC.

Cabré, B. (2004). *Disseny físic de bases de dades*. Material docent UOC, de l'assignatura Sistemes de gestió de bases de dades. Eureca Media, SL.

Costal, D. (2002). *Implementació de mètodes d'accés*. Material docent UOC, de l'assignatura Bases de dades II. Eureca Media, SL.

Lightstone, S.; Teorey, T.; Nadeau, T. (2007). *Physical Database Design* (3a. ed.). Morgan Kaufmann.

Liu, L.; Özsu, M. T. (Eds.) (2009). *Encyclopedia of Database Systems*. Springer.

PostgreSQL. Manuals accesibles en línia des de: http://www.PostgreSQL.org/docs/. Últim accés: desembre del 2013.

Oracle. Manuals accesibles en línia des de: http://www.oracle.com/technetwork/indexes/documentation/index.html. Últim accés: desembre del 2013.

Ortego, S. (2004). *El component de processament de consultes i peticions SQL*. Material docent UOC, de l'assignatura Sistemes de gestió de bases de dades. Eureca Media, SL.

Ramakrishan, R.; Gehrke, J. (2003). *Database Management Systems* (3a. ed.). Mc Graw Hill.

Rodríguez, M. E. (2002). *Altres temes de bases de dades*. Material docent UOC, de l'assignatura Bases de dades II. Eureca Media, SL.

Segret, R. (2002). *Components d'emmagatzematge d'una base de dades*. Material docent UOC, de l'assignatura Bases de dades II. Eureca Media, SL.