Disseny físic de bases de dades

PID_00270594

Blai Cabré i Segarra
Jordi Casas Roma
Dolors Costal Costa
Pere Juanola Juanola
Ivo Plana Vallvé
Àngels Rius Gavidia
Ramon Segret i Sala

Temps mínim de dedicació recomanat: 7 hores

Universitat Oberta de Catalunya

Blai Cabré i Segarra

Enginyer industrial per la Universitat Politècnica de Catalunya. Exerceix com a professional informàtic especialitzat en bases de dades a l'empresa IBM.

Jordi Casas Roma

Llicenciat en Enginyeria Informàtica per la Universitat Autònoma de Barcelona (UAB), màster en Intel·ligència Artificial Avançada per la Universitat Nacional d'Educació a Distància (UNED) i doctor en informàtica per la UAB. Des de 2009 exerceix de professor en els Estudis d'Informàtica, Multimèdia i Telecomunicació de la Universitat Oberta de Catalunya (UOC), i també de professor associat a la UAB. És director del màster universitari en Ciència de Dades de la UOC.

Dolors Costal Costa

Doctora en Informàtica per la Universitat Politècnica de Catalunya. Professora titular del Departament de Llenguatges i Sistemes Informàtics de la Universitat Politècnica de Catalunya, assignada a la Facultat d'Informàtica de Barcelona.

Pere Juanola Juanola

Enginyer tècnic d'Informàtica per la Universitat de Girona (UdG). Té més de 20 anys d'experiència en Oracle (administració i aplicacions). Ha administrat entorns d'entitats bancàries i ha treballat en el sector públic en la creació d'aplicacions en Oracle. Ha treballat com a responsable de seguretat informàtica i en qualitat informàtica desenvolupant l'aplicació de CMMi i ITIL. Des del 1995 fa docència tant en universitat presencial com no presencial. Actualment treballa com a cap de projecte en temes d'Oracle i bases de dades.

Ivo Plana Vallvé

Enginyer informàtic i Màster de la Societat de la Informació i el Coneixement per la Universitat Oberta de Catalunya (UOC). Professional de la Informàtica a l'empresa semipública fent les funcions de Director d'Informàtica i el de Cap de Transformació digital. Professor col·laborador de la UOC.

Àngels Rius Gavidia

Doctora en Informàtica per la Universitat Oberta de Catalunya. Actualment és professora dels Estudis d'Informàtica, Multimèdia i Telecomunicació de la UOC. Anteriorment ha estat professora del Departament de Llenguatges i Sistemes Informàtics de la UPC i personal docent col·laborador dels Estudis d'Informàtica, Multimèdia i Telecomunicació de la UOC.

Ramon Segret i Sala

Enginyer industrial i llicenciat en Informàtica. Ha exercit com a professional informàtic especialitzat en bases de dades a l'empresa IBM. Actualment és professor dels Estudis d'Informàtica i Multimèdia de la UOC.

La revisió d'aquest recurs d'aprenentatge UOC ha estat coordinada per la professora: Àngels Rius Gavidia (2020)

Sisena edició: febrer 2020

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

Índex

Introducció

El disseny físic de bases de dades constitueix la quarta etapa en el procés de disseny d'una base de dades. En les etapes anteriors s'ha realitzat l'anàlisi de requeriments, el disseny conceptual i, finalment, el disseny lògic de la base de dades. En aquest mòdul veurem el procés de transformació del model lògic, obtingut en l'etapa anterior, cap a un model físic que ens permeti obtenir una implementació sobre un sistema de gestió de bases de dades (SGBD).

Abans d'iniciar aquesta etapa, cal seleccionar el SGBD concret sobre el qual es vol implementar la base de dades. En aquest mòdul ens centrarem en les bases de dades relacionals; per tant, l'objectiu n'és veure el procés de transformació d'un model lògic relacional cap a un model físic i concretar un SGBD relacional.

Aquest mòdul s'estructura en tres parts:

1) En la primera part veurem com cal estructurar i emmagatzemar la informació de la base de dades en un suport físic no volàtil perquè pugui ser recuperada. Presentarem els nivells lògic, físic i virtual de les bases de dades (apartats 1, 2, 3 i 4).

2) En la segona part d'aquest mòdul veurem el procés de transformació del model lògic relacional cap al model físic, que ens permetrà una implementació sobre un SGBD concret de la base de dades (apartat 5).

3) Finalment, en la tercera part, veurem el funcionament dels mètodes principals d'accés a les dades (apartat 6).

Objectius

En els materials didàctics d'aquest mòdul trobarem les eines indispensables per a assolir els objectius següents:

1. Conèixer l'estructura física que utilitza la base de dades per a emmagatzemar les dades de manera no volàtil.

2. Conèixer la funcionalitat i l'estructura del nivell virtual i del nivell físic.

3. Aprendre a fer el disseny físic de la base de dades a partir del disseny lògic, adaptat a les característiques d'un SGBD concret.

4. Definir els índexs necessaris i convenients en cada taula perquè les aplicacions tinguin un bon rendiment quan accedeixen a la base de dades.

5. Conèixer els diferents mètodes d'accés que són necessaris per a poder fer consultes i actualitzacions a les dades emmagatzemades a les bases de dades.

6. Comprendre la utilitat dels índexs per a la implementació dels accessos per valor.

7. Conèixer l'estructura dels índexs d'arbres B^+.

1. Conceptes previs

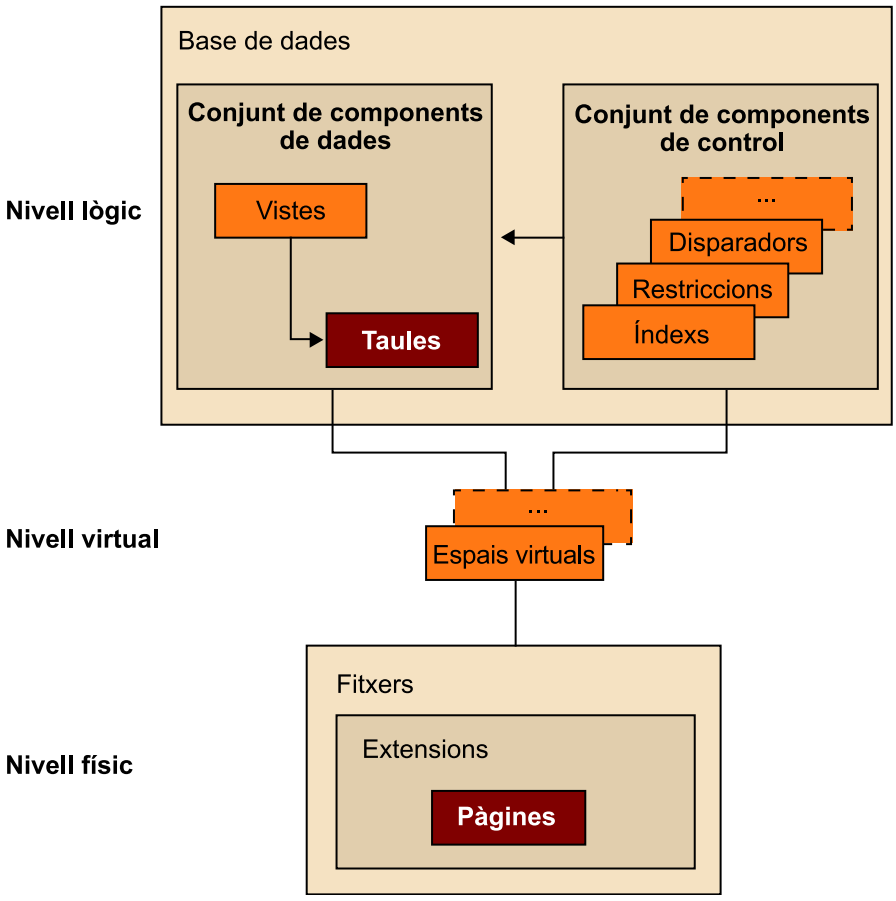
La manera més senzilla d'explicar una base de dades (BD) és dir que és un conjunt de dades persistents relacionades entre si. La característica de persistència implica que les dades no han de desaparèixer entre execucions successives de programes, encara que entre aquestes execucions transcorri un interval de temps llarg. Aquest requisit obliga, al seu torn, a tenir les dades, entre execució i execució, emmagatzemades en un medi no volàtil, normalment en un disc magnètic.

Per a comprendre quina relació hi ha entre les dades tal com les veu el programador o l'usuari final i tal com estan emmagatzemades en el suport no volàtil, ens servirem d'una arquitectura de tres nivells que anomenarem *arquitectura dels components d'emmagatzematge*, representada en la figura 1.

Què és una arquitectura?

Una arquitectura, com qualsevol altra manera d'esquematitzar la realitat, és una eina senzilla i potent ideada per a abstraure i entendre els trets fonamentals dels sistemes més complexos.

Figura 1. Arquitectura dels components d'emmagatzematge



El primer que veiem és que l'arquitectura dels components d'emmagatzematge és de tres nivells:

1) El **nivell lògic** correspon a tots els components que, d'una manera més o menys directa, són coneguts i manipulats pel programador o per l'usuari final. Aquests components formen la interfície externa o d'usuari del sistema de gestió de la base de dades (SGBD). De manera molt simplificada, podem considerar que en el nivell lògic hi ha un conjunt de dades estructurades en taules. De fet, tots els altres components lògics giren al voltant de les taules.

2) Les taules han de ser persistents; per aquest motiu s'emmagatzemen en un suport no volàtil i ho fan seguint unes estructures determinades, que són els components que constitueixen el **nivell físic**.

3) Entre els nivells físic i lògic n'hi ha un tercer que anomenarem **nivell virtual**. Aquest nivell proporciona a l'SGBD[1] una visió simplificada del nivell físic, com ja veurem.

Ja coneixeu els components del nivell lògic. En aquest mòdul presentem els components dels altres dos nivells, el virtual i el físic, que conjuntament anomenem *components d'emmagatzematge*, perquè controlen la disposició física de les dades en el suport no volàtil.

[1] Abreugem *sistema gestor de bases de dades* amb la sigla SGBD.

Terminologia

En aquest mòdul anomenem *programador* la persona que confecciona programes que interaccionen amb la base de dades, normalment mitjançant sentències d'SQL, i *usuari final*, la persona que hi interacciona directament per mitjà d'una interfície gràfica.

2. El nivell lògic

Tot i que els components del nivell lògic ja s'estudien en un altre mòdul didàctic, en recordarem breument els conceptes més importants per a poder relacionar els components lògics amb els dels nivells físic i virtual.

Vegeu també

Vegeu els components del nivell lògic en el mòdul "Disseny lògic de bases de dades" d'aquesta assignatura.

2.1. Components lògics

En la figura 1 podem veure que el component lògic més important és la taula i, per a reflectir-ho, l'hem representada amb un tramat. El conjunt de taules és el nucli fonamental de la base de dades en un doble sentit: és el conjunt de dades disponibles per a l'usuari i és, també, el conjunt més important i voluminós de dades emmagatzemades en els suports físics.

L'usuari pot accedir directament a les taules o bé pot accedir-hi indirectament per mitjà de les vistes; per això taules i vistes estan agrupades com el conjunt de components de dades.

Una base de dades té, tanmateix, altres components que no són les taules i les vistes, però que hi estan estretament relacionats. Es tracta, entre d'altres, de les restriccions[2], que defineixen certes regles que han de complir les dades, o dels disparadors[3], que indiquen accions que s'han d'executar si es compleixen certes condicions. Tots aquests altres components els anomenarem *conjunt de components de control*, ja que en certa manera permeten controlar les dades dinàmicament.

(2)En anglès, *constraints*.

(3)En anglès, *triggers*.

També es pot considerar que els índexs pertanyen al conjunt de components de control, encara que amb un matís lleugerament diferent, atès que la seva funció fonamental és de rendiment: faciliten l'accés a les dades en un temps raonable.

3. El nivell físic

Com ja hem dit, les dades s'emmagatzemen en suports no volàtils, normalment en discos magnètics en la majoria de sistemes informàtics (des dels grans sistemes fins als ordinadors personals). Aquests suports són controlats pel sistema operatiu de la màquina, que és qui realment efectua la lectura i l'escriptura física de les dades. Els SGBD no reimplementen aquestes funcions, sinó que usen les rutines especialitzades del sistema operatiu (SO) per a llegir i escriure les dades i també per a la gestió física dels dispositius.

Els sistemes operatius gestionen les dades en els discos magnètics a partir d'unes unitats globals anomenades *fitxers*[4]. Normalment, el sistema operatiu no reserva una gran quantitat d'espai de disc per a cada fitxer, sinó que en va adquirint a mesura que en necessita. La unitat d'adquisició és l'anomenada *extensió*[5], un altre dels components d'aquest nivell. Finalment, l'extensió és un múltiple enter del component físic més petit, que és la pàgina. La *pàgina*[6] és l'element que conté i emmagatzema les dades del nivell lògic, per això també l'hem tramat en la figura 1. A continuació veurem aquests tres components, del més petit al més gran.

(4)En anglès, *files*.

(5)En anglès, *extent*.

(6)En anglès, *page*.

3.1. La pàgina

Per a entendre què és una pàgina d'una BD[7] o d'un SGBD, més que començar per una definició, la descriurem des de dos punts de vista diferents però complementaris:

(7)Abreugem *base de dades* amb la sigla BD.

- Les dades s'emmagatzemen en dispositius externs però, d'altra banda, sabem que per a poder efectuar-hi qualsevol operació han de ser presents en la memòria principal de l'ordinador. Hi ha d'haver, doncs, un transport de dades entre la memòria externa (que normalment és un disc) i la memòria interna (o memòria principal). Aquest transport es fa emprant una unitat discreta de transport de dades que els SO[8] anomenen *bloc* i que en els SGBD es diu *pàgina*.

(8)Abreugem *sistema operatiu* amb la sigla SO.

- Paral·lelament al fet de ser la unitat d'entrada/sortida, la pàgina també és la unitat d'organització de les dades emmagatzemades. L'espai del disc sempre s'assigna en un nombre múltiple de pàgines, i cada pàgina pot ser adreçada individualment.

Concretant, direm que la **pàgina** és la unitat mínima d'accés i de transport del sistema d'entrada/sortida (E/S) d'un SGBD, cosa que la fa ideal per a ser, alhora, la unitat d'organització més important de les dades emmagatzemades.

Noteu que en la definició anterior hem dit "unitat mínima d'accés i de transport". La raó és que, de la mateixa manera que mai no s'accedeix a menys d'una pàgina, sí que es pot accedir de cop a un nombre enter de pàgines consecutives. Aquesta qüestió, que veurem més endavant, no invalida l'explicació que hem fet fins ara de la pàgina ni la seva importància dins de l'arquitectura de components. Es tracta només d'una opció de rendiment del sistema.

Que la pàgina sigui la unitat d'organització del nivell físic no vol dir que no tingui interiorment la seva pròpia estructura. De fet, en una pàgina hi ha diferents components. A aquests components que hi ha a l'interior de la pàgina, només hi poden accedir les rutines de l'SGBD un cop la pàgina ja és a la memòria principal de l'ordinador. En canvi, a la pàgina com a unitat hi accedeix (per a llegir i escriure) el subsistema d'E/S[9] de l'SO de la màquina.

[9]Abreugem l'expressió *entrada/sortida* amb la sigla E/S.

Un símil artístic

Imagineu-vos que la pàgina és un quadre. Llavors l'SGBD seria el pintor; és qui veu l'interior del quadre i l'omple (el pinta). Un cop llest, l'embolica i crida el seu mosso (el sistema operatiu), que agafa el quadre i, sense veure'l, el transporta al magatzem.

3.1.1. Estructura d'una pàgina

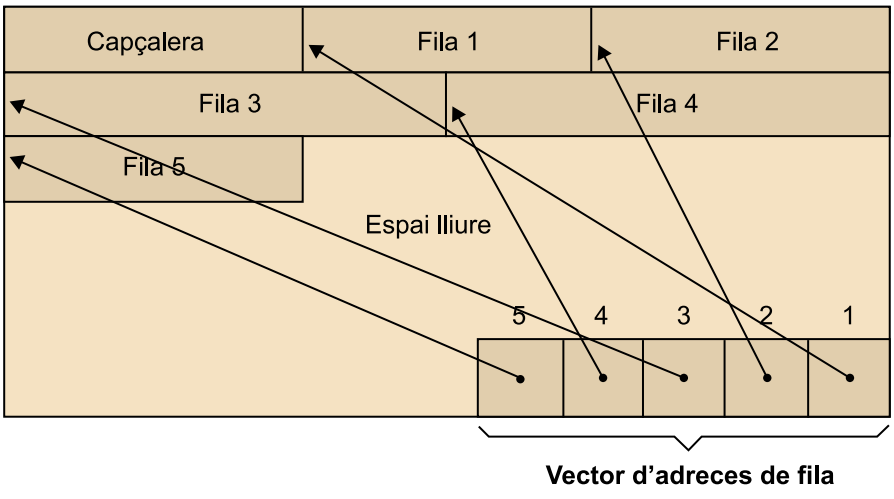
Per a facilitar l'explicació, ara ens centrarem en les pàgines que emmagatzemen taules. Aquestes pàgines se solen anomenar *pàgines de dades*. Més endavant n'ampliarem alguns detalls per a poder generalitzar l'estructura de la pàgina.

La mida d'una pàgina

Els valors més corrents de la mida d'una pàgina actualment són 2 kB, 4 kB i 8 kB. Possiblement 4 kB és la mida més habitual.

La pàgina és de longitud fixa. Hi ha SGBD que permeten escollir la mida de la pàgina entre un petit repertori. Altres tenen només una mida única. L'estructura de la pàgina, que es mostra en la figura 2, és molt similar en tots els SGBD.

Figura 2. Estructura d'una pàgina



Una pàgina consta dels elements següents, enumerats des de l'adreça més baixa de la pàgina fins a la més alta:

a) Una **capçalera**, amb informació de control de la pàgina.

b) Un cert nombre de registres, que aquí anomenarem **files** perquè corresponen a les files de les taules del nivell lògic. Aquest nombre pot variar amb el temps, i en un moment concret pot ser igual a zero.

c) Un **espai lliure**.

d) Un **vector d'adreces de fila** (**VAF**), que té tants elements com files hi ha a la pàgina. Cada element del VAF[10] conté l'adreça d'una fila dins de la pàgina (apunta a la fila). L'element que ocupa la posició d'adreça més alta en el VAF apunta a la fila que ocupa l'adreça més baixa en la pàgina, tal com podeu veure en la figura 2. La fila següent (a continuació de la primera) és apuntada per l'element del VAF a l'esquerra del primer element.

(10)Abreugem l'expressió *vector d'adreces de fila* amb la sigla VAF.

Així doncs, les files es van col·locant d'esquerra a dreta en l'ordre creixent de les adreces dins de la pàgina, mentre que els elements del VAF es van col·locant de dreta a esquerra des de la posició més alta i seguint en l'ordre decreixent de les adreces. D'aquesta manera, l'espai lliure sempre queda cap al mig de la pàgina.

3.1.2. Estructura d'una fila

L'element que hem anomenat *fila* en parlar de l'estructura de la pàgina (nivell físic) generalment registra tota la informació corresponent a una fila d'una taula (nivell lògic). Per aquesta raó, la seva estructura és senzilla. És una seqüència de camps (nivell físic), cadascun dels quals registra la informació del camp de la fila corresponent (nivell lògic).

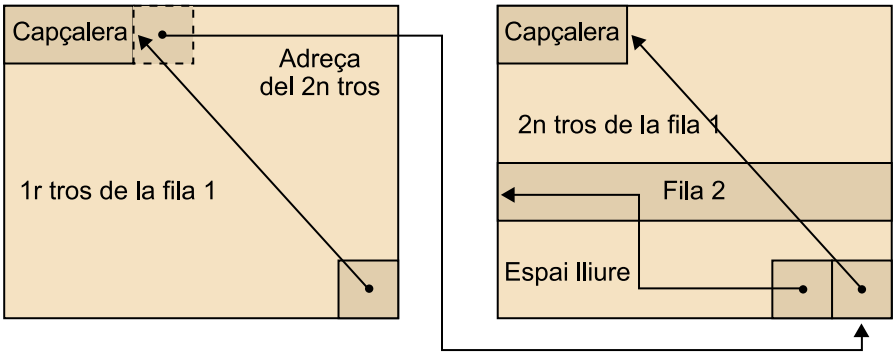
Així doncs, pel que fa a files i camps, la correspondència entre el nivell lògic i el físic és molt directa.

Pot ser que l'SGBD necessiti desar alguna informació de control referent a la fila i, per això, abans de la seqüència de camps trobem una capçalera de fila. Vegeu l'estructura de la fila en la figura 3.

Figura 3. Estructura d'una fila

Normalment, les files d'una taula són de longitud inferior a la mida de les pàgines i per això hi caben perfectament. De totes maneres, si la longitud de la fila és superior a la de la pàgina, els SGBD parteixen la fila en trossos. Cada tros s'emmagatzema en una pàgina diferent i apunta al tros següent, com podeu observar en la figura 4.

Figura 4. Emmagatzematge d'una fila més llarga que una pàgina



No us preocupeu ara si en la figura no acabeu d'entendre la manera concreta en què un tros de fila adreça el següent. Ho veureu més endavant.

3.1.3. Estructura d'un camp

Tal com hem vist, una pàgina està formada per files i una fila està formada per camps. Un camp d'una fila d'una taula (nivell lògic) s'emmagatzema com un dels camps de la fila dins de la pàgina (nivell físic).

Un camp normalment està format per una capçalera i un contingut, tal com indiquem en la figura 5.

Figura 5. Estructura d'un camp

La **capçalera** del camp permet desar certes característiques del camp, com ara:

a) La indicació de si el contingut en aquest moment és nul o no ho és. Aquesta indicació és necessària si el camp està definit de forma que admet el valor nul.

b) La longitud del camp, també en un moment concret. Aquesta informació és necessària quan el camp s'ha definit amb un tipus de dada de longitud variable, com per exemple el tipus *CHARACTER VARYING*.

CHARACTER VARYING

CHARACTER VARYING és el nom de l'estàndard SQL:1999 per al tipus de dada "cadena de caràcters de longitud variable", que en diversos SGBD comercials s'anomena *VARCHAR*.

El **contingut** del camp és el resultat d'emmagatzemar el valor del camp segons les normes de codificació de cada SGBD en cada arquitectura de màquina concreta. A continuació trobareu el format d'emmagatzematge dels tipus de dades més comuns en els diferents SGBD:

(11)Bytes codificats en codi ASCII o bé en el codi que usi la màquina on funciona l'SGBD.

- Tipus *SMALLINT*: número binari enter de 16 bits.
- Tipus *INTEGER*: número binari enter de 32 bits.
- Tipus *FLOAT*: número en coma flotant de 64 bits.
- Tipus *CHARACTER*(*n*) o, abreujadament, *CHAR*(*n*): cadena de *n*bytes[11], on *n* és la longitud definida en el tipus de dada.
- Tipus *CHARACTER VARYING*: com el cas anterior, amb l'excepció que aquí *n* és la longitud real del valor concret que el camp tingui en aquest moment determinat.
- Tipus *DATE*: encara que externament és una cadena de 8 dígits (4 per a l'any, 2 per al mes i 2 per al dia), internament es pot emmagatzemar de manera que ocupi molt menys espai, per exemple 4 bytes, seguint un patró de codificació propi de cada SGBD.

Estalviar espai en disc

Els SGBD intenten estalviar espai en el disc i temps de procés sempre que sigui possible. Per això, si el camp no admet nuls i és de longitud fixa (per exemple, *CHAR*(3)), llavors aquesta informació no cal i podem prescindir de la capçalera. En aquest cas, el camp ocupa només l'espai necessari per a emmagatzemar el seu valor.

3.1.4. Gestió de la pàgina

Després de veure com s'organitza l'espai d'una pàgina i quins són els seus continguts, descriurem breument les principals manipulacions que un SGBD fa d'aquest espai i aquests continguts. Les principals operacions que fa servir són les següents:

1) Formatació d'una pàgina. Quan l'SGBD necessita més espai dins d'un fitxer per a emmagatzemar-hi dades, adquireix una nova extensió. Les pàgines d'aquesta extensió es formaten, és a dir, s'inicialitzen d'una manera adequada perquè posteriorment les utilitzi l'SGBD. La **inicialització** consisteix fonamentalment a escriure la capçalera i deixar tota la resta de la pàgina com a espai lliure.

2) Càrrega inicial de files. Si l'extensió inicialitzada s'utilitza per a una càrrega inicial o massiva de files de taules, l'SGBD comença a fer servir l'espai lliure de la pàgina de la manera següent:

- La primera fila es col·loca darrere de la capçalera, es crea un element del VAF que apunta a aquesta fila i es col·loca al final de tot de la pàgina.
- La segona fila ocupa el lloc immediatament darrere de la primera, i el seu element del VAF ocupa el lloc just al davant de l'element de la fila anterior.

El procés es repeteix successivament, com ja heu vist en l'explicació de l'estructura d'una pàgina.

D'aquesta manera l'espai lliure va disminuint, però sempre ocupa un lloc cap al centre de la pàgina. L'administrador de la base de dades (DBA[12]) normalment haurà fixat un tant per cent d'espai lliure mínim que l'SGBD ha de respectar en una càrrega massiva de files. Això vol dir que, quan el procés de càrrega detecta que l'espai que queda lliure en una pàgina és precisament aquest espai lliure mínim, ja no hi col·loca cap més fila i comença a col·locar-les a la pàgina següent.

[12]Abreugem *administrador de la base de dades* amb la sigla DBA, de l'expressió anglesa *database administrator*.

Administrador de la BD

L'administrador de la BD (*database administrator*, DBA) és la persona, o equip de persones, encarregada, entre altres coses, de la gestió dels components d'emmagatzematge.

La finalitat d'aquesta limitació és deixar un cert espai lliure en totes les pàgines perquè sigui aprofitat per a les operacions que veurem a continuació.

3) Alta posterior d'una nova fila. Per a afegir una fila a una taula de la BD, l'SGBD, en el nivell físic, localitza primer la pàgina on ha d'afegir la fila. Aquesta pàgina s'anomena *pàgina candidata*. Un cop l'SGBD sap quina és la pàgina candidata, utilitza l'espai lliure disponible en la pàgina per a col·locar-hi la fila i deixar-la apuntada per un nou element del VAF. La fila i l'element del VAF es col·loquen segons el criteri explicat anteriorment.

Afegir una fila

Les files s'afegeixen amb la sentència *INSERT*.

Pot arribar un moment que ja no quedi espai lliure suficient en la pàgina candidata per a col·locar-hi la fila. Llavors la fila es col·loca en una altra pàgina segons algorismes propis de cada SGBD.

Reflexió

Les diferents formes de localitzar una pàgina queden fora de l'abast d'aquest mòdul.

4) Baixa d'una fila. La baixa d'una fila consisteix a alliberar l'espai ocupat per la fila i l'element del VAF. En aquest moment, o més endavant, l'SGBD reorganitza la pàgina perquè l'espai alliberat per la baixa s'uneixi a la resta d'espai lliure de la pàgina. Hi haurà un desplaçament de files i d'elements del VAF perquè l'estructura de la pàgina continuï essent tal com hem descrit anteriorment.

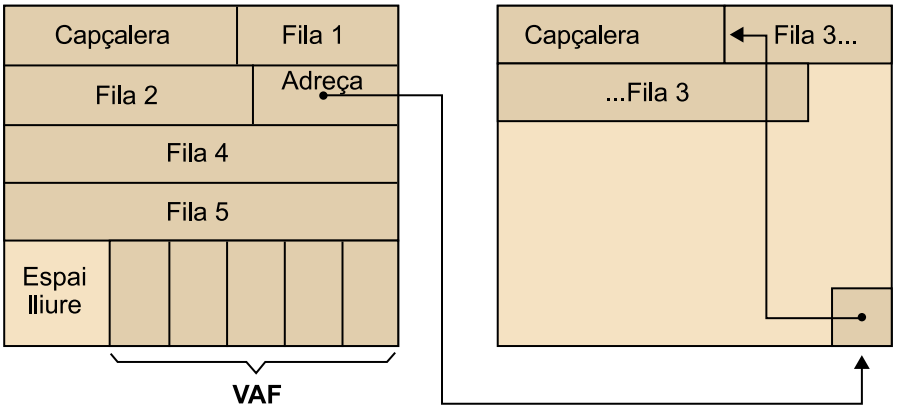
5) Canvi de longitud d'una fila. Un canvi de longitud d'una fila pot afectar de tres maneres la reorganització de l'espai de la pàgina:

a) Si el canvi té com a resultat una disminució de la longitud de la fila, l'espai alliberat s'ajuntarà amb el lliure segons el mecanisme que acabem d'explicar.

b) Si, al contrari, dóna lloc a un augment de la longitud i l'espai extra requerit està disponible en forma d'espai lliure dins la pàgina, la fila quedarà al seu lloc, ocupant més espai, i les files que la segueixen es desplaçaran cap a la dreta (cap a adreces més altes).

c) En cas que la necessitat d'espai no es pugui servir a partir de l'espai lliure de la pàgina, aquesta fila es trasllada cap a una altra pàgina que disposi d'espai suficient. En la pàgina original i en la posició original es crea una adreça que apunta cap a la nova posició de la fila, com s'indica a la figura 6.

Figura 6. Fila situada fora de la pàgina original



Com en el cas de les files que ocupen més d'una pàgina, no us preocupeu tampoc si no acabeu d'entendre, en la figura, la manera concreta en què s'adreça la fila que s'ha traslladat de la pàgina original a la nova. Ho veureu més endavant.

3.1.5. Altres tipus de pàgines

Fins ara ens hem centrat en les pàgines que contenen files de taules, les pàgines de dades. Com sabeu prou bé, en el nivell lògic hi ha altres components. De totes maneres, l'estructura de pàgina que acabeu de veure serveix de base per a entendre els altres tipus de pàgines.

En efecte, deixant els índexs a part, tots els altres components (vistes, restriccions, disparadors, etc.) són definicions que algú ha proporcionat a l'SGBD mitjançant el llenguatge SQL. Aquestes definicions, com expliquem més endavant, es desen en unes taules dissenyades especialment per a contenir-les, però que a tots els efectes que aquí ens interessen són taules com les que contenen les altres dades. Així doncs, un cop en el nivell físic, tot el que hem vist fins ara és vàlid per a les taules que contenen aquests components.

Els índexs són un cas lleugerament diferent, i també ho són les taules amb dades del tipus objecte gran. L'estructura del bloc que utilitzen és similar, però en varia el contingut.

Reflexió

Queda fora de l'abast d'aquest mòdul l'estudi en profunditat dels diferents tipus de pàgines.

3.2. L'extensió

Una **extensió** és un nombre enter de pàgines, en principi consecutives, que el sistema operatiu adquireix a petició de l'SGBD quan aquest detecta que necessita més espai per a emmagatzemar dades. Aquesta adquisició és automàtica (sense intervenció humana explícita).

Cal notar que quan, a petició de l'SGBD, l'SO gestiona les dades en el suport no volàtil dels perifèrics, ho fa sempre dins del marc d'un fitxer. Així, l'adquisició d'una extensió també es fa dins d'aquest marc. Això vol dir que quan es detecta que falta espai, aquesta manca és detectada en un fitxer determinat i s'adquireix una extensió per a aquest fitxer. L'extensió està, doncs, associada a un fitxer, és part d'un fitxer. I el fitxer és, entre altres coses, un conjunt d'extensions.

Algunes estratègies de l'SO

Algunes de les estratègies que fa servir l'SO quan no pot assignar pàgines físicament consecutives són, per exemple, partir l'extensió en uns quants trossos o retornar un avís que assenyali que no hi ha prou espai.

Les pàgines d'una extensió han de ser físicament consecutives. Si l'SO no les pot assignar perquè físicament no són al dispositiu, seguirà estratègies diferents.

La raó de la contigüitat de les pàgines d'una extensió

La raó principal per la qual les pàgines en una extensió han de ser contigües és que, d'aquesta manera, els components que són consecutius lògicament (per exemple, les files d'una taula) es poden emmagatzemar consecutivament i es poden recuperar de la mateixa manera i, així, s'afavoreixen els tractaments que impliquin la recuperació consecutiva de tots els components (tractaments que són freqüents en les BD relacionals).

D'altra banda, els mecanismes més avançats de millora de rendiment que trobareu esbossats en el subapartat que ens dóna una visió general de l'E/S en un SGBD, tenen sentit només si les pàgines sobre les quals actuen són consecutives.

3.3. El fitxer

El **fitxer** és la unitat que fan servir els SO per a gestionar l'espai en els dispositius perifèrics. També és un conjunt d'extensions.

Els SGBD aprofiten la funcionalitat que proporcionen els SO i no interaccionen directament amb els fitxers. És per aquesta raó que moltes vegades costa trobar el terme *fitxer* de manera explícita en llibres o manuals d'SGBD. Fins i tot hi ha SGBD relativament petits en què totes les dades que controlen estan emmagatzemades en un únic fitxer, fet que desvirtua la importància d'aquest component.

De totes maneres, els fitxers són sempre presents en l'emmagatzematge: les diverses dades de les BD són a les pàgines; aquestes s'agrupen en extensions, i aquestes darreres, en fitxers. En resum, podem dir que el fitxer, per a l'SGBD, és un conjunt de pàgines que s'han anat creant d'extensió en extensió.

Extensions d'un fitxer

Normalment, els SGBD permeten definir dos tipus d'extensions. L'una s'anomena **extensió primària** i l'altra, **extensió secundària**. Aquestes dues extensions poden ser, i normalment ho són, de longitud diferent, però totes dues són un múltiple de la pàgina.

Quan un fitxer necessita espai per primera vegada, s'adquireix el nombre de pàgines corresponents a l'extensió primària. A partir d'aquest moment, les extensions successives que es vagin necessitant en aquest fitxer, adquiriran el nombre de pàgines de l'extensió secundària. Per a un fitxer només hi ha una extensió primària (la primera), però diverses de secundàries (les següents). El nombre màxim d'extensions secundàries el fixa el sistema operatiu o l'SGBD, i acostuma a ser una potència de dos (16, 128, 256, etc.).

3.4. Visió general de l'E/S en un SGBD

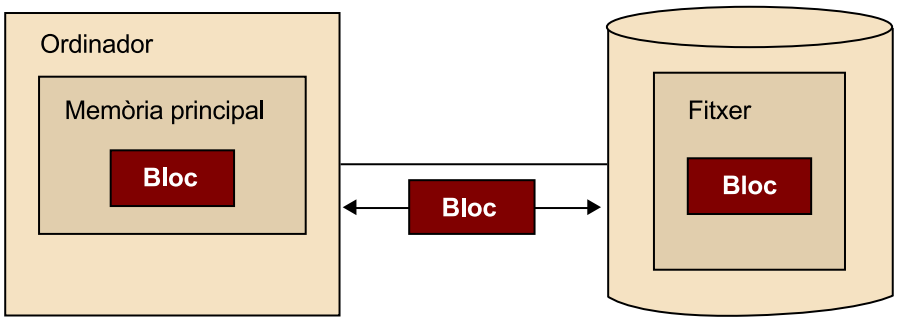
Recordeu que hem definit la pàgina com una unitat d'emmagatzematge i de transport. Com que fins ara només hem parlat d'emmagatzematge, per acabar l'exposició del nivell físic tractarem el transport.

El processament de dades persistents, que s'efectua en la memòria principal de l'ordinador, com ja heu vist, requereix un transport de dades entre unitat central i perifèrics. Tot seguit expliquem breument aquest transport i el comportament de l'entrada/sortida en el cas dels fitxers clàssics i en els SGBD.

1) Entrada/sortida de fitxers clàssics

En el cas de la gestió dels anomenats *fitxers clàssics* o *fitxers tradicionals*, com que no pertanyen a una base de dades, aquest transport és molt senzill conceptualment, tal com podeu veure en la figura 7.

Figura 7. E/S en els fitxers clàssics

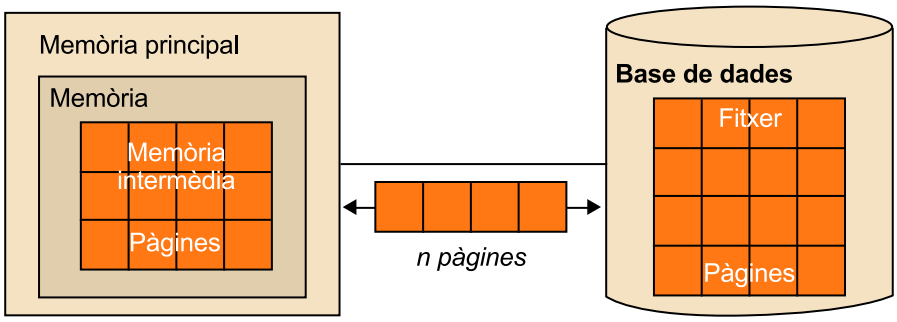


Per a llegir dades, per exemple, l'SO desencadena una operació d'E/S per a dur cap a la memòria principal el bloc que conté les dades que es necessiten. Aquest bloc és una unitat físicament delimitada i adreçable en el conjunt de les dades enregistrades en el suport no volàtil (el disc magnètic). Després, el bloc viatja pel canal que uneix el perifèric amb la unitat central de l'ordinador i, finalment, el bloc es col·loca en la memòria principal i pot ser tractat pels programes.

2) Entrada/sortida en les bases de dades

En el cas dels SGBD, aquest mecanisme, encara que és bàsicament el mateix, és una mica més elaborat. Té unes característiques pròpies que mostra la figura 8 i que descrivim tot seguit:

Figura 8. E/S en les bases de dades



a) La longitud de la pàgina és fixa. Normalment, la pàgina té la mateixa longitud en tots els fitxers que constitueixen les BD. Això es contraposa al cas dels fitxers clàssics, en què la longitud del bloc sol ser diferent per a cada un, ja que s'escull la longitud que sembla més adequada per a les dades emmagatzemades.

Llavors, i respecte als SGBD, podem plantejar-nos la pregunta següent: si els components que trobem en les pàgines (com ara les files o les entrades d'índex) són, en principi, de longitud variable, per què els encapsulem en un marc de longitud fixa (la pàgina)? La resposta és fàcil: en basar l'emmagatzematge en una unitat fixa per a tot l'SGBD, la gestió que aquest ha de fer de l'E/S pot ser molt més senzilla, cosa que finalment dóna com a resultat un rendiment millor.

b) L'optimització del temps d'E/S, que permet oferir temps raonables de processament quan es tracten quantitats importants de dades, és una altra característica dels SGBD. Els diferents SGBD han elaborat mecanismes diversos, alguns força sofisticats. N'esmentem breument uns quants:

- Aprofitar l'operació física d'E/S per a llegir i dur a la memòria més d'una pàgina consecutiva a la vegada, cosa que estalvia temps de transport per unitat llegida (pàgina).

- Avançar en el temps la lectura d'una pàgina quan es preveu que es necessitarà pròximament. Així, en el moment en què un procés la necessiti, no haurà d'esperar el temps de lectura, ja que la pàgina ja serà en la memòria.

- Retenir en la memòria principal pàgines modificades, fins i tot després d'haver-les escrit en el suport no volàtil, per a estalviar-se el fet de tornar-les a llegir si algun altre procés les necessita.

Recordeu

Quan hem definit la pàgina heu vist que, tot i que normalment és la unitat d'E/S, moltes vegades, per raons de rendiment, l'SGBD pot llegir o escriure un nombre enter de pàgines de cop.

c) Les tècniques de transmissió simultània de diverses pàgines en una sola operació d'E/S impliquen la necessitat de dedicar una part de la memòria principal de l'ordinador a rebre, gestionar i emmagatzemar pàgines de les BD. Aquesta

(13)En anglès, *buffer pool*.

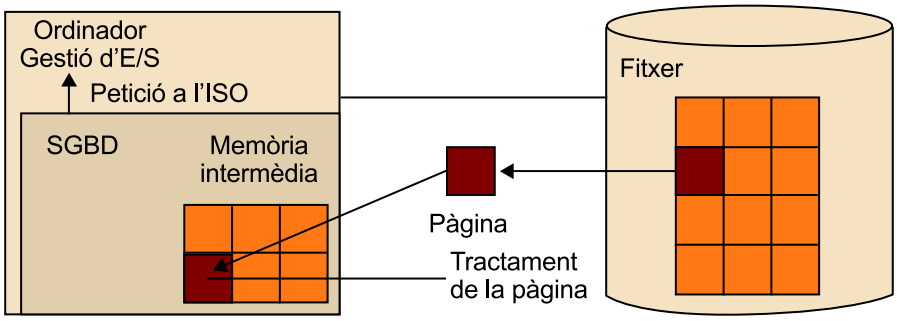
memòria, gestionada per les rutines adequades de l'SGBD, s'anomena **memòria intermèdia**[13] i està estructurada com un conjunt d'*unitats de memòria intermèdia*[14].

[14]En anglès, *buffers*.

Cada unitat de memòria intermèdia és un tros de memòria que actua com a marc per a contenir una pàgina. El conjunt de totes aquestes unitats pot ser força gran, i dimensionar-les correctament és un dels principals factors d'afinament del rendiment d'una instal·lació amb SGBD.

Finalment, per a il·lustrar les funcions pròpies de l'SGBD i les del sistema operatiu en l'entrada/sortida, passem a comentar la figura 9.

Figura 9. Funcionament de l'SO i de l'SGBD en l'entrada/sortida d'una pàgina



Observem que l'SGBD se serveix de l'SO per a certes funcions. Quan l'SGBD detecta la necessitat de tractar una pàgina que no té en la memòria intermèdia, fa una petició a les rutines de gestió d'E/S del sistema operatiu per a obtenir-la. Aquestes rutines són les que gestionen la lectura física de la pàgina i la situen en una unitat lliure del conjunt de les unitats de memòria intermèdia. Seguidament l'SGBD tracta la pàgina, ja que és qui en coneix l'organització i els continguts.

4. El nivell virtual

Per a començar, justificarem l'existència del nivell virtual i després veurem l'estructura de l'espai virtual, que és el component que materialitza les funcions del nivell virtual.

4.1. Justificació de l'existència del nivell virtual

En una primera aproximació podríem pensar que cada taula s'emmagatzema en un fitxer, és a dir, que 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 la correspondència seria fixa.

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

a) Hi ha taules molt grans que ens interessarà fragmentar i emmagatzemar cada fragment en un dispositiu diferent per a millorar-ne l'accés.

b) Contràriament, podem trobar un conjunt de taules molt petites i que convingui desar-les totes en un mateix fitxer per a no consumir tants recursos del sistema.

c) Cada vegada més es fan servir taules que, a més de camps amb dades de tipus tradicional (quantitats numèriques, cadenes de caràcters que representen noms i adreces, dates, etc.), en tenen d'altres que emmagatzemen tipus de dades diferents, com ara gràfics, imatges o uns quants minuts d'àudio o de vídeo. Aquestes dades, que necessiten molts més bytes per a ser emmagatzemades, són els objectes grans, que interessa emmagatzemar en fitxers diferents dels que s'empren per a les dades tradicionals, per a millorar els temps d'accés.

d) Malgrat que fins ara només hem esmentat les taules, tots sabeu que hi ha altres components, com ara índexs, definicions de disparadors, etc. Igual que en el punt anterior, segurament interessarà emmagatzemar cada component en fitxers estructurats de manera diferent, per a poder treure'n també el màxim rendiment.

Recursos limitats

L'interès per no voler consumir més recursos dels imprescindibles rau en el fet que hi ha SGBD que suporten només un nombre determinat de fitxers.

Tots aquests exemples ens han de fer adonar que és força útil disposar d'un nivell intermedi. Ens proporciona un grau elevat de flexibilitat (o independència) per a assignar components lògics als components físics, ja que no cal ferho d'una manera estrictament biunívoca (una taula, un fitxer). Concretament,

ens permet decidir on s'emmagatzemarà cada taula o fragment. D'aquesta manera escollim en quina màquina, en quin disc o en quin tros de disc tindrem les diferents dades de la base de dades.

4.2. L'espai virtual i les seves associacions

Hem vist que la funció del nivell virtual és proporcionar un grau elevat d'independència entre els nivells lògic i físic. Anomenem **espai virtual** (**EV**[15]) el component que implementa aquesta funcionalitat.

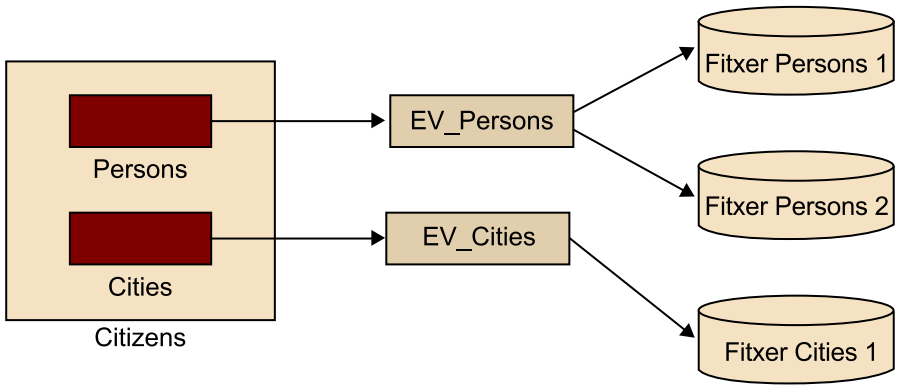
[15]Abreugem *espai virtual* amb al sigla EV.

Els diferents SGBD comercials donen diversos noms a aquest component, entre els quals els més emprats són *dbspace* i *tablespace*. També cal dir que la funcionalitat que veurem implementada per l'espai virtual és una simplificació de la realitat de molts SGBD comercials, que utilitzen una estructura una mica més complexa en aquest nivell virtual. Tanmateix, la idea del nivell virtual és simple i es pot explicar correctament a partir d'un únic component, l'espai virtual, que descrivim a continuació.

Normalment, en tots els SGBD una taula s'associa a un únic EV. Per contra, un EV pot estar associat a una o més taules. D'altra banda, l'associació entre EV i fitxer sol ser de molts a molts: un EV s'associa a un o més fitxers i un fitxer està associat a un o més EV.

L'associació entre taula i EV es fa en temps de definició de la taula. Juntament amb altres atributs, s'indica amb quin EV l'associem. L'espai virtual també s'ha de definir i en definir-lo s'esmenten el fitxer o fitxers amb els quals l'associem.

Figura 10. Exemple del nivell virtual



Exemple del nivell virtual

La figura 10 mostra una base de dades de ciutadans (*Citizens*). En aquest exemple podem veure l'associació entre el nivell lògic (amb dues taules: *Persons* i *Cities*) i el nivell físic (amb tres fitxers: *Persons1*, *Persons2* i *Cities1*) per mitjà de dos espais virtuals (*EV_Persons* i *EV_Cities*).

A continuació es mostren les sentències necessàries per a la creació d'aquestes associacions en l'SGBD Oracle:

```
CREATE TABLE Person (idPerson integer, name varchar2(30), ...)
TABLESPACE EV_Persons;

CREATE TABLESPACE EV_Persons DATAFILE '/db/files/Persons1.dbf' SIZE
100M;
```

La sentència *CREATE TABLE* permet crear una taula en un EV determinat, fent ús de la clàusula *TABLESPACE*. La segona sentència, *CREATE TABLESPACE*, permet crear un EV i assignar-hi un fitxer.

Espai virtual d'agrupació

L'espai virtual d'agrupació és un tipus d'espai virtual al qual s'associen dues o més taules i que està associat a un únic fitxer. Aquest tipus d'espai virtual permet tenir les dades de taules relacionades en un mateix fitxer, i permet tenir la combinació (operació de *JOIN*) preconstruïda en el disc. Aquest tipus d'espai virtual té sentit si l'accés a les taules es fa sempre conjunt, ja que penalitza les consultes a les taules individuals.

4.3. Estructura de l'espai virtual

L'EV és una visió diferent de les pàgines del nivell físic. Igual que una vista, en el model relacional, és una altra manera de veure les dades d'una taula sense que això representi duplicar físicament les dades de la taula; l'espai virtual és una manera diferent d'organitzar les pàgines que hem vist en el nivell físic, però sense duplicar-les materialment.

Les úniques pàgines que realment existeixen són les del nivell físic, agrupades en extensions i emmagatzemades en fitxers. Ara bé, aquestes pàgines no han d'estar necessàriament ordenades en el suport físic. Fins i tot pàgines que contenen elements que en el nivell lògic són consecutius (per exemple, les files d'una taula) poden estar separades les unes de les altres en el nivell físic (per exemple, ocupant extensions diferents); en alguns casos el resultat de la combinació de diverses taules es pot emmagatzemar en un espai virtual, si convé tenir-lo precalculat.

Estructura d'un espai virtual

L'estructura d'un espai virtual és similar a la d'altres mecanismes que ja coneixeu, com ara:

a) La correspondència que hi ha en un ordinador entre memòria virtual i memòria real.

b) Les vistes del model relacional. Les dades d'una vista són les que realment existeixen en una o més taules, però disposades d'una altra manera, sense que això representi una duplicació d'aquestes dades.

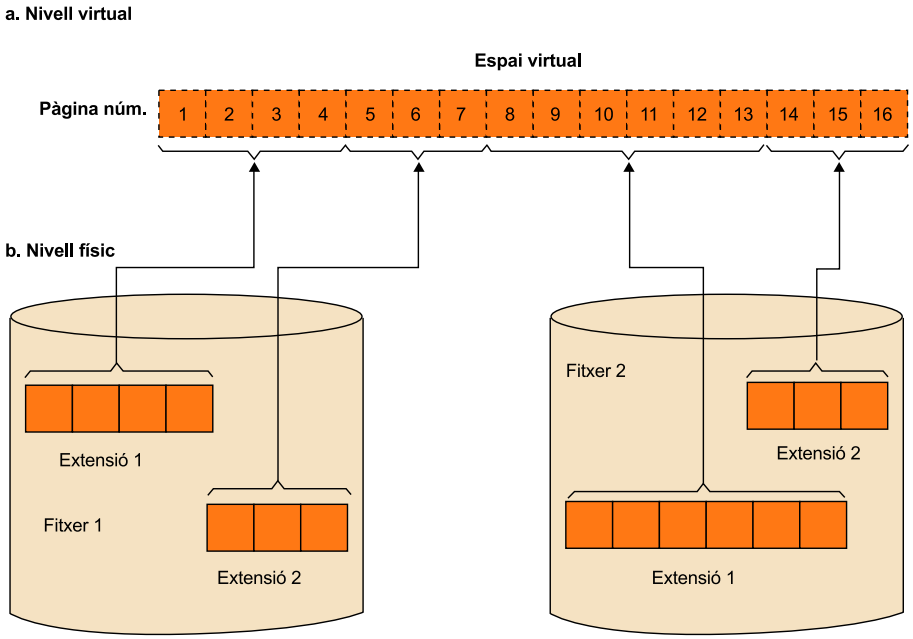
Com veurem de seguida, és convenient que les pàgines estiguin ordenades ocupant posicions consecutives. Com que això físicament no sempre és possible, els SGBD implementen una ficció, que és el que anomenen *espai virtual*.

L'espai virtual està constituït per les imatges de les pàgines reals (imatges que anomenarem *pàgines virtuals*) ordenades seqüencialment i situades contiguament (sense salts entremig).

Així doncs, podem definir l'espai virtual com una seqüència de pàgines virtuals que es corresponen, una a una, amb pàgines reals del nivell físic.

Podem il·lustrar l'estructura de l'espai virtual mitjançant la figura 11.

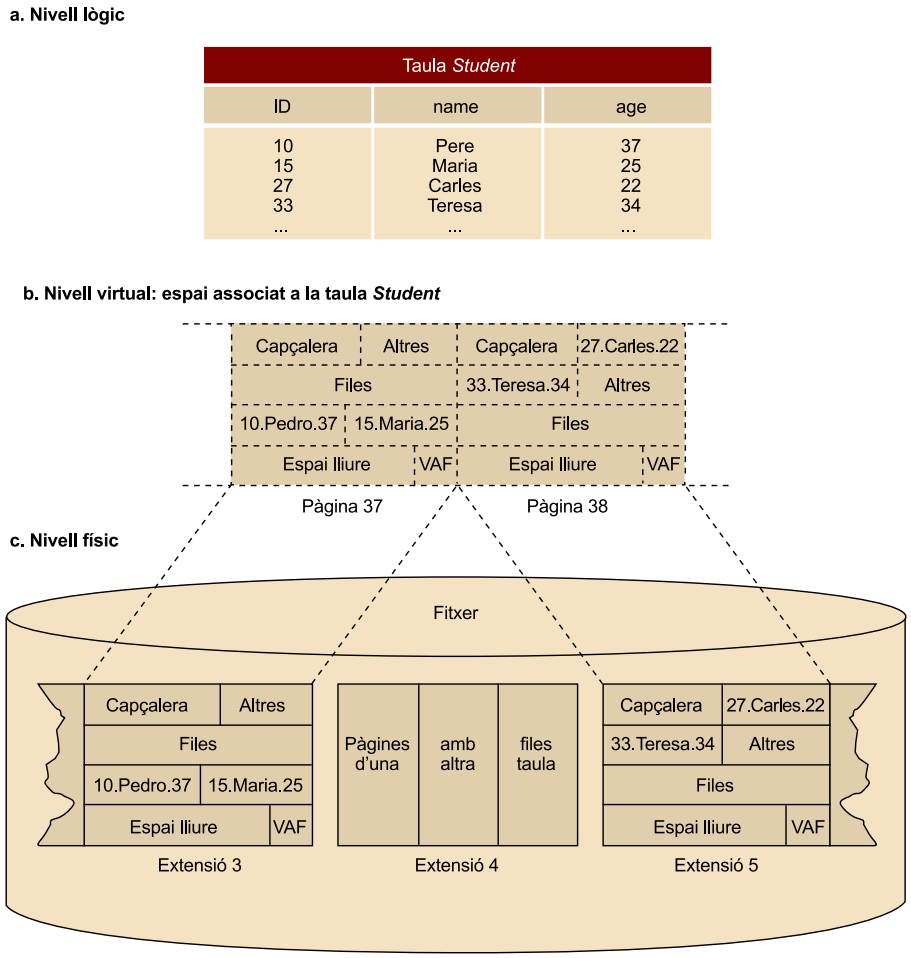
Figura 11. Correspondència entre pàgines reals i pàgines virtuals



Fixeu-vos que les diverses pàgines, que contenen, per exemple, les files d'una taula, estan distribuïdes entre extensions diferents de fitxers diferents. L'EV és una ficció per a poder veure i tractar totes aquestes pàgines com si estiguessin ordenades consecutivament.

En la figura 12 representem amb més detall, i mitjançant un exemple, la relació que hi ha entre els tres nivells de l'arquitectura que hem vist fins ara.

Figura 12. Un exemple dels tres nivells



En el nivell lògic, tenim la taula *Student* amb les seves files. Aquesta taula, en el nivell físic, s'emmagatzema en pàgines de dades. Per la manera com s'ha anat creant la taula i per la disponibilitat d'espai lliure en el disc, pot haver passat que no totes les files de la taula hagin anat a parar a pàgines consecutives. Observem que les pàgines que contenen aquestes files pertanyen a dues extensions diferents, separades per una altra extensió que pot ser d'un altre fitxer i, evidentment, pot contenir altres dades.

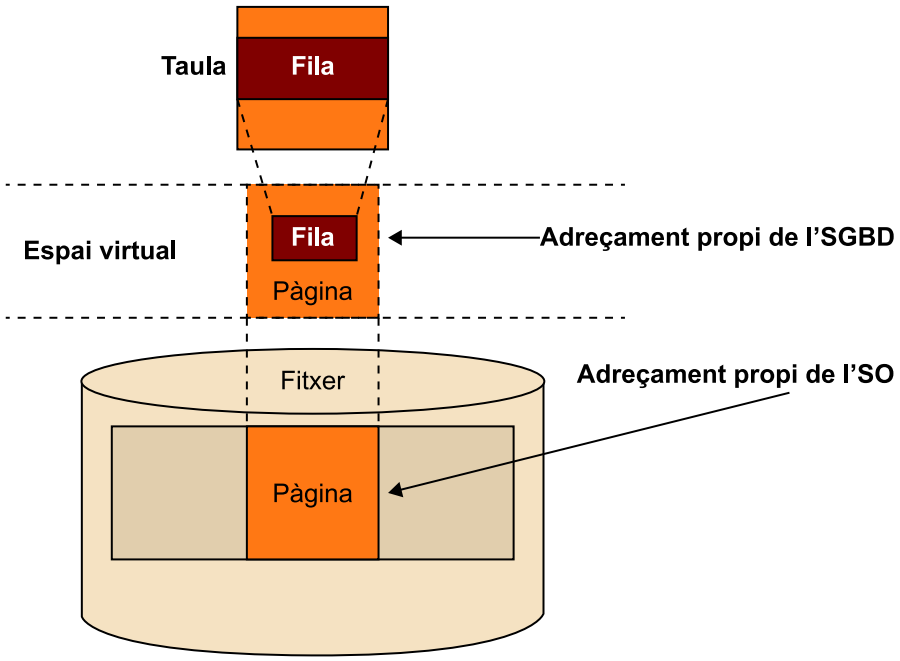
No podem assegurar la contigüitat de les files del nivell lògic en les pàgines del nivell físic. Llavors, és l'EV el que ens permet veure totes les pàgines que contenen les files d'aquesta taula de manera consecutiva i sense salts.

4.3.1. Adreçament en un SGBD

Ara estem en condicions d'analitzar com és l'adreçament en un SGBD.

En el nivell lògic, l'usuari manipula taules, files i camps. Aquests components estan emmagatzemats en fitxers. Com troba l'SGBD un d'aquests components en el nivell físic? Com l'adreça? Respondrem aquesta pregunta en dos passos, ja que en realitat hi ha un adreçament doble, que es correspon amb l'arquitectura que hem anat veient i que s'il·lustra en la figura 13.

Figura 13. Un adreçament doble



1) Primer pas: del nivell lògic al virtual

Com que la unitat més petita que adreça explícitament un SGBD és la fila, el que fa realment l'SGBD és trobar les files dins de l'espai virtual. L'adreçament de l'SGBD és, doncs, del nivell lògic al nivell virtual.

Tractament dels camps

Un cop la fila és a la memòria, l'SGBD, atès que en coneix l'estructura, pot localitzar tots els seus camps i tractar-los individualment.

Per a localitzar les files a l'EV, els SGBD usen una adreça que anomenarem **identificador de fila** (**RID**). El RID, com podem veure en la figura 14, té dues parts diferenciades:

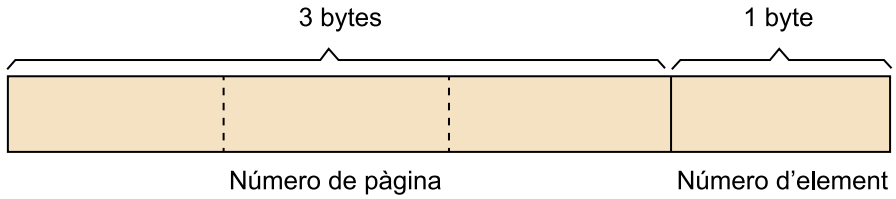
- el número de la pàgina que conté la fila dins de l'espai virtual associat a la taula a la qual pertany la fila,
- el número d'element del VAF d'aquesta pàgina que apunta a la fila.

L'identificador de fila (RID)

RID és un acrònim del terme anglès *row identifier*. Aquest identificador rep diferents noms segons les implementacions de cada SGBD. Per exemple, en el cas d'Oracle rep el nom de *ROWID*.

Tal com indiquem en la figura 14, el RID normalment és una adreça de 4 bytes, 3 per al número de pàgina i 1 per al número d'element del VAF.

Figura 14. Estructura del RID



Aquest sistema d'adreçament mereix alguns comentaris:

a) Una fila, un cop donada d'alta en una pàgina, no pot canviar de pàgina mentre estigui viva, ja que la seva adreça canviaria. Com que aquesta adreça pot estar desada en més d'un lloc de la base de dades, interessa no canviar-la per a evitar el manteniment inútil i costós d'altres estructures de dades (per exemple, en els índexs que s'hagin construït sobre aquesta taula).

b) En canvi, una fila es pot desplaçar dins de la seva pàgina original. Canviant només el contingut de l'element de VAF corresponent (que indica el desplaçament dins de la pàgina on comença la fila) tindrem sempre la fila correctament adreçada, i això sense cap canvi en el seu RID. Per això hi pot haver una gestió de l'espai lliure d'una pàgina sense canvis en els RID de les files d'aquesta pàgina. La gestió d'espai lliure, com ja heu vist, tendeix a agrupar tot l'espai lliure d'una pàgina i, per a poder-ho fer, ha de desplaçar files dins de la pàgina.

c) Pel mateix motiu que esmentàvem en el punt **a**, si una fila augmenta la seva longitud de manera que no cap en la seva pàgina original i s'ha de col·locar en una altra pàgina, ja hem vist que en la pàgina original es manté una adreça, que és un RID, que des de la pàgina original apunta a la nova pàgina. Així doncs, els RID emmagatzemats en altres estructures, com els índexs, que apuntaven a la fila, la continuen trobant sense que s'hagin hagut de modificar.

Ara ja deveu comprendre que les adreces o apuntadors que hem esmentat en punts anteriors d'aquest mòdul, en realitat són RID. És el cas de les files massa llargues que no cabien en una pàgina (que s'han partit en trossos i cada tros apunta al següent) o bé el cas de les files que creixen de longitud (que s'han de traslladar a una altra pàgina i s'apunten des de la pàgina original).

2) Segon pas: del nivell virtual al físic

En aquest pas normalment intervenen els SO i no ho veurem en detall. Direm solament que la unitat de localització en el nivell físic és la pàgina.

Així, l'SGBD demana a l'SO que li llegeixi/desi una pàgina concreta i aquest ho fa emprant una adreça molt diferent del RID. Per tant, en aquest pas hi ha una transformació d'adreces: del RID a una adreça física que depèn de l'arquitectura del dispositiu.

Abans d'acabar el tema de l'adreçament, podem reflexionar sobre la utilitat de l'adreçament propi dels SGBD, el RID.

En prescindir de molts dels detalls del nivell físic, un adreçament com el RID permet que l'SGBD pugui emprar més d'un tipus diferent de dispositius d'emmagatzematge, ja que les diferències són gestionades a un altre nivell i per un altre component (normalment, el sistema operatiu). Aquest mecanisme simplifica els SGBD i n'augmenta la potència, ja que els proporciona la independència dels dispositius físics concrets.

I, finalment, no hem d'oblidar que és el nivell virtual el que possibilita, per mitjà de l'EV, l'adreçament de tipus RID.

5. Transformació del model lògic al model físic

A partir del disseny lògic d'una base de dades hem de passar al seu disseny físic, passant pel nivell virtual. La forma d'implementar un disseny lògic en un SGBD concret depèn de les característiques pròpies de cada un d'aquests SGBD. I les característiques pròpies de cada SGBD són un reflex de l'entorn:

- Característiques del maquinari.
- Sistema operatiu i programari bàsic.
- Disseny de l'SGBD.

Cada SGBD ha desenvolupat un llenguatge propi, fet a mida pel mateix constructor, per a implementar el disseny físic de la base de dades, d'acord amb les característiques de l'entorn, i per a obtenir el màxim rendiment del maquinari, del sistema operatiu i del mateix gestor. De fet, es podria considerar com una ampliació del llenguatge SQL estàndard, amb les clàusules pròpies que cada gestor necessita per a definir els components del disseny físic. No obstant això, hi ha una gran similitud o equivalència entre bona part dels components dels diferents gestors.

L'estàndard SQL incorpora la definició de tots els components del disseny lògic de la base de dades. En canvi, no incorpora cap element del disseny físic.

En la transformació del model lògic al model físic, seguirem els següents passos:

1) Transformar les taules amb les corresponents claus primàries, claus foranes i claus alternatives, a partir del disseny lògic de la base de dades que hem obtingut en el pas anterior.

2) A continuació relacionem cada taula amb un espai per a taules, i cada índex amb un espai per a índex. És el nivell virtual.

3) Per acabar, relacionem cada espai virtual amb un fitxer físic, i en definim les característiques. Això constitueix el disseny físic de la base de dades.

Addicionalment cal completar aquesta definició amb els índexs necessaris per a garantir un accés correcte a les dades, a més de les restriccions necessàries sobre les dades (valors nuls, valors únics dels camps, etc).

A continuació veurem el procés de transformació. Com a exemple de SGBD per a la part que no queda inclosa en l'estàndard SQL, s'utilitza el sistema gestor Oracle. Tot i així, gran part de la sintaxi és similar a la que s'utilitza en altres SGBD, i només cal una consulta a un manual de referència per a veure com s'apliquen les diferents instruccions en cada SGBD.

5.1. Taula

La taula és un component del disseny lògic de la base de dades i, com a tal, està definit en l'estàndard SQL. Ara bé, és important remarcar que la part que afecta el nivell virtual i físic no està inclosa en l'estàndard i, per tant, cada SGBD implementa aquestes funcions de manera diferent.

La sentència *CREATE TABLE* de tots els SGBD incorporen les clàusules de l'estàndard SQL i, a més a més, l'enllaç amb els elements físics propis de cada un d'aquests.

```
CREATE TABLE table_name
  ( column_definition )
  <unique_constraint>
  <referential_constraint>
  <check_constraint>
  <extent_specs>
TABLESPACE table_space_name
```

Les clàusules següents de la sentència *CREATE TABLE* pertanyen a l'SQL estàndard: *column_definition*, *unique_constraint*, *referential_constraint*, *check_constraint*. Aquestes clàusules són definicions del disseny lògic. Tots els SGBD les incorporen amb una sintaxi quasi idèntica.

D'altra banda, les clàusules < *extent_specs* > i *TABLESPACE* són específiques d'Oracle. Serveixen per a relacionar la definició de la taula (nivell lògic) amb l'espai per a taules (nivell virtual), que s'anomena *Tablespace* en Oracle. A més, la clàusula <*extent_specs*> pot incorporar altres paràmetres que permeten controlar el percentatge d'ocupació de les pàgines de l'espai per a taules.

Exemple de creació d'una taula en Oracle

A continuació veurem un exemple de creació d'una taula en l'SGBD Oracle:

```
CREATE TABLE Employees (
  employeeId NUMBER(6) PRIMARY KEY,
  firstName VARCHAR2(20),
  lastName VARCHAR2(25),
  email VARCHAR2(25) NOT NULL,
  phoneNumber VARCHAR2(20),
  salary NUMBER(8,2) NOT NULL,
  commissionPct NUMBER(2,2),
  departmentId NUMBER(4),
  CONSTRAINT empSalaryMinDemo CHECK (salary > 0),
  CONSTRAINT empEmailUKDemo UNIQUE (email)
) TABLESPACE users_data;
```

En l'exemple es crea la taula *Employees* amb diferents atributs i algunes restriccions. Algunes consideracions sobre l'exemple:

- La clau primària de la taula és l'atribut *employeeId*.
- Dos dels atributs no admeten el valor nul (*email* i *salary*).
- Les restriccions que hi trobem són:
  - *CHECK*: validacions diverses, com per exemple que el camp sigui superior a zero o que compleixi determinades condicions.
  - *UNIQUE*: unicitat del camp indicat, és a dir, que no hi poden haver dos registres en aquesta taula amb el mateix valor en aquest camp.

A continuació veurem amb més detall les definicions relatives a clau primària i clau alternativa, índex i algunes de les restriccions més rellevants.

5.1.1. Clau primària i clau alternativa

Les definicions de claus primàries, foranes i alternatives actualment formen part de l'estàndard SQL. Però no sempre ha estat així; l'any 1986 encara no s'havien definit i l'estàndard SQL86 no les va incloure quan es va publicar. En l'estàndard SQL89 s'esmenten les claus primàries per primera vegada, i no va ser fins al 1992 que es defineixen i s'incorporen de fet a l'estàndard SQL92.

Com ja sabem, la clau primària ha de ser, obligatòriament, una clau única i que no admeti valors nuls. La teoria del model de bases de dades relacionals suggereix que cada taula hauria de tenir una clau primària que identifiqués de forma unívoca l'entitat que descriu. Malgrat això, l'estàndard SQL no obliga l'existència d'una clau primària a cada taula, sinó que és opcional. Lògicament, però, cada taula pot tenir com a màxim una clau primària.

La resta de claus úniques i que no admeten valors nuls són claus alternatives a la clau primària. Això tampoc no està legislat en l'estàndard SQL.

Tal com mostra l'exemple anterior, s'indica la clau primària mitjançant la instrucció *PRIMARY KEY*.

5.1.2. Índex

Els índexs són uns elements del disseny físic de la base de dades que tenen com a finalitat millorar el rendiment de les aplicacions quan accedeixen a les taules. Els índexs no formen part del disseny lògic de la base de dades i, per tant, no estan inclosos en l'estàndard SQL.

Tot i així, la sentència *CREATE INDEX* és present en tots els SGBD amb opcions molt similars. A continuació veurem el detall d'aquesta sentència sobre l'SGBD Oracle. A més a més, incorpora clàusules per a definir l'espai per a índexs i clàusules d'enllaç amb els elements físics propis (els fitxers d'índexs).

```
CREATE [UNIQUE] INDEX index_name
 ON table_name ( column [ ASC | DESC ] [ ,..n ])
 [ CLUSTER cluster_name ]
```

```
  [ < extent_specs > ]
  [ TABLESPACE table_space_name ]
  [ ..... ]

< extent_specs > ::=
  [ PCTFREE nn ]
  [ PCTUSED nn ]
  [ INITRANSnn ]
  [ MAXTRANSnn ]
  [ STORAGE < storage_clause > ]

< storage_clause > ::=
  INITIAL nn
  [ NEXT nn ]
  [ MINEXTENTS nn ]
  [ MAXEXTENTS nn ]
  [ PCTINCREASE nn ]
  [ OPTIMAL nn ]
  [ ..... ]
```

On:

- *index_name* és el nom lògic de l'índex, definit sobre la taula especificada en la clàusula *ON*.
- *UNIQUE* i *CLUSTER* són característiques de l'índex.
- *table_space_name* és el nom de l'espai per a índex i es defineix amb la sentència *CREATE TABLESPACE,* que hem vist anteriorment.
- Quan es crea l'espai per a índex (*CREATE TABLESPACE*) s'associa a un fitxer físic amb característiques pròpies de nom, grandària, ubicació física en disc, etc.
- La clàusula *<extent_specs>* especifica condicions de percentatge d'ocupació de les pàgines de l'espai per a índex.
- La clàusula *<storage_clause>* defineix característiques d'emmagatzematge, mida inicial, mida incremental, extensions mínimes i màximes, etc.

Exemple de creació d'un índex

L'exemple següent mostra com es crea un índex mitjançant la instrucció *CREATE INDEX* d'Oracle:

```
CREATE INDEX indexDepName
ON Employees (departmentId) TABLESPACE users_ind;
```

5.1.3. Restriccions

A continuació veurem algunes de les principals restriccions que podem definir sobre els atributs de les taules i com s'implementen en el SGBD d'Oracle.

Aquestes restriccions ens permeten definir una lògica que ens ajudi a validar les dades del nostre model, de manera que compleixin determinades condicions.

Alguns exemples típics que podem resoldre mitjançant aquestes restriccions són:

- Verificar que un nombre sigui superior a zero.
- Verificar que un atribut contingui una cadena de text d'una mida determinada.
- Verificar que no hi hagi valors duplicats per a un cert atribut.

a) Check

Aquesta restricció permet indicar certes condicions que ha de complir el valor del registre per a ser admès a la taula.

Per exemple, podem definir una taula amb informació sobre treballadors, que contingui com a atributs el nom i el sou. Evidentment, el sou ha de ser positiu. Per a indicar-ho, en la creació de la taula utilitzem la restricció *CHECK*.

```
CREATE TABLE Employees (
 name VARCHAR2 (30),
 salary NUMBER CHECK (salary> 0)
) TABLESPACE data_employees;
```

b) Not null

Aquesta restricció permet definir atributs que han de contenir informació obligatòriament, és a dir, que han de contenir dades. Indiquem aquesta restricció en el SGBD per mitjà del paràmetre *NOT NULL*.

```
CREATE TABLE Employees (
name VARCHAR2 (30) NOT NULL,
salary NUMBER
) TABLESPACE data_employees;
```

c) Unique

Aquesta restricció permet validar que no existeixen valors duplicats en un determinat atribut. Permet definir les claus candidates.

Continuant amb l'exemple anterior, aquesta restricció ens permet definir que cada treballador ha de tenir un identificador únic (per exemple, el DNI en el cas d'Espanya), que ha de ser únic per a cada persona.

```
CREATE TABLE Employees (
 id VARCHAR2 (9) UNIQUE,
 name VARCHAR2 (30) NOT NULL,
    salary NUMBER
) TABLESPACE data_employees;
```

d) Foreign key / References

Aquesta restricció permet crear claus foranes que referencien atributs d'altres taules.

Per exemple, si volem crear una taula que contingui els departaments de l'empresa (*Department*) i poder indicar a quin departament pertany cada treballador (*Employees*), caldrà crear la nova taula i afegir un atribut que referenciï la nova taula per a cada registre de la taula *Employees*.

```
CREATE TABLE Department
( id NUMBER PRIMARY KEY
name VARCHAR2 (50)
) TABLESPACE data_departments;

CREATE TABLE Employees
( id VARCHAR2 (9) UNIQUE,
name VARCHAR2 (30) NOT NULL,
     salary NUMBER,
departId NUMBER,
CONSTRAINT fkDep FOREIGN KEY (departId) REFERENCES Department (id)
) TABLESPACE data_employees;
```

5.2. Espai per a taules

L'espai per a taules és un component del nivell virtual. No pertany al disseny lògic de la base de dades i, per tant, no està inclòs en l'estàndard SQL. La sentència de creació d'un espai per a taules és diferent en cada SGBD i incorpora les clàusules d'enllaç amb els elements físics propis de cada un d'aquests SGBD (els fitxers).

El nivell virtual en Oracle rep el nom de *tablespace*. La sentència *CREATE TABLESPACE* permet crear espais virtuals en aquest SGBD.

```
CREATE TABLESPACE table_space_name
  DATAFILE < filespec > [ ,...n ]
  DEFAULT STORAGE < storage_clause >
```

```
< filespec > ::=
 'file_name'
 [ SIZE nn ]
< storage_clause > ::=
 INITIAL nn
 [ NEXT nn ]
 [ MINEXTENTS nn ]
 [ MAXEXTENTS nn ]
 [ PCTINCREASE nn ]
 [ OPTIMAL nn ]
 [ ..... ]
```

On:

- *table_space_name* és el nom de l'espai per a taules, que es defineix amb aquesta sentència i que està relacionada amb la clàusula *TABLESPACE* de la sentència *CREATE TABLE*.

- Cada espai per a taules s'associa a un, o més d'un, fitxer físic <*filespec*>.

- La definició del fitxer físic és donada pel seu nom extern *file_name*, la ubicació en disc i la seva mida, *size*.

- La clàusula <*storage_clause*> defineix les característiques de l'emmagatzematge, mida inicial, mida incremental, extensions mínimes i màximes, etc.

Exemple de creació d'un espai per a taules

En l'exemple següent crearem un espai per a taules amb el nom *users_data* format pel fitxer '/db/users/users_data1.dbf' i amb una mida total de 100 MB.

```
CREATE TABLESPACE users_data
DATAFILE '/db/users/users_data1.dbf' SIZE 100M;
```

Alguns paràmetres són opcionals i no cal que siguin informats en la sentència de creació. En aquests casos, s'assigna el valor per defecte del paràmetre. En l'exemple anterior la mida de la pàgina no s'especifica i, per tant, s'utilitzarà el valor per defecte.

5.3. Base de dades

Com en el cas de l'espai per a taules, la base de dades no pertany al disseny lògic de la base de dades i, per tant, tampoc no està inclosa en l'estàndard SQL.

La sentència *CREATE DATABASE* és diferent en cada SGBD i incorpora clàusules de definició d'elements físics propis de cada un d'ells (diari, catàleg, fitxers, etc.).

En Oracle, la sintaxi d'aquesta sentència és com s'indica a continuació:

```
CREATE DATABASE database_name
 [ CONTROLFILE REUSE ]
 [ LOGFILE < filespec > [ ,...n ] ]
 [ MAXLOGFILES nn ]
 [ MAXLOGMEMBERS nn ]
 [ MAXLOGHISTORY nn ]
 [ DATAFILE < filespec > [ ,...n ] ]
 [ MAXDATAFILES nn ]
 [ MAXINSTANCES nn ]
 [ CHARACTER SET charset ]
 [ ..... ]
```

On:

- *database_name* és el nom de la base de dades que s'associa a un conjunt de fitxers físics que contenen els espais per a taules que hem explicat en el subapartat anterior.

- Els fitxers físics que contenen els espais per a taules es relacionen en la clàusula *DATAFILE* < *filespec* >.

- La clàusula *LOGFILE* < *filespec* > especifica el nom dels diaris que es defineixen perquè el gestor registri totes les actualitzacions de les taules d'aquesta base de dades i possibilitar-ne, així, la recuperació en cas necessari.

- El detall de la definició dels fitxers físics < *filespec* > s'ha explicat en el subapartat dels espais per a taules.

- Altres paràmetres limiten el nombre màxim de fitxers de cada tipus, *MAXLOGFILES*, *MAXLOGMEMBERS*, *MAXDATAFILES*, etc.

Exemple de creació de bases de dades

En el següent exemple crearem una base de dades per a una universitat, amb uns diaris (*logfiles*) de 10 MB cadascun i amb uns límits de 5 *logfiles* i 100 *datafiles*.

```
CREATE DATABASE university
 USER SYS IDENTIFIED BY pass1
 USER SYSTEM IDENTIFIED BY pass2
 LOGFILE GROUP 1 ('/db/oracle/oradata/university/redo01.log')
      SIZE 10M,
   GROUP 2 ('/db/oracle/oradata/university/redo02.log') SIZE 10M,
   GROUP 3 ('/db/oracle/oradata/university/redo03.log') SIZE 10M
 MAXLOGFILES 5
```

```
    MAXDATAFILES 100;
```

6. Implementació de mètodes d'accés

En una base de dades hi ha dades emmagatzemades a les quals s'ha de poder accedir per tal de fer-hi consultes i actualitzacions. Per aquest motiu, una de les funcions dels SGBD és la de proporcionar l'accés a les dades que gestionen. La problemàtica de com oferir aquest accés és l'objecte d'estudi d'aquest apartat.

Inicialment, veurem de forma breu els diferents mètodes d'accés a les dades. A continuació, aprofundirem en l'estudi dels mètodes d'accés per posició, per valor i per diversos valors. Finalment, veurem com implementa aquestes funcions l'SGBD Oracle.

6.1. Els mètodes d'accés a una BD

Una de les funcions dels SGBD és proporcionar l'accés a les dades que gestionen. L'accés a les dades ha de permetre consultar i actualitzar les dades emmagatzemades. Sempre que es llegeix o s'enregistra alguna dada en una BD es fa mitjançant algun dels mètodes d'accés dels quals disposa l'SGBD.

Reflexió

Hem d'entendre "actualitzacions de dades" en sentit ampli; és a dir, la seva inserció, eliminació i modificació.

6.1.1. Les dades

En aquest subapartat, per tal de presentar els mètodes d'accés a les dades de manera entenedora, suposarem sempre que les dades a les quals cal accedir es perceben segons la visió que proporciona el nivell virtual i no segons la del nivell físic. Així, doncs, gairebé sempre que en aquest mòdul s'empra la paraula *pàgina* hem d'interpretar que ens referim a pàgines virtuals; si ens referim a pàgines reals, ho esmentarem explícitament.

A fi de simplificar, també suposem que tots els accessos es fan a dades que hi ha emmagatzemades en una única taula situada en un únic EV. Així, doncs, no considerem els casos en què cal combinar dades de taules (i espais) diferents per a obtenir les dades a les quals volem accedir.

6.1.2. Els accessos per posició

En aquest subapartat veurem els tipus d'accés més simples que utilitzen els SGBD.

L'accés **directe per posició** consisteix a obtenir una pàgina que té un número de pàgina determinat dins d'un espai.

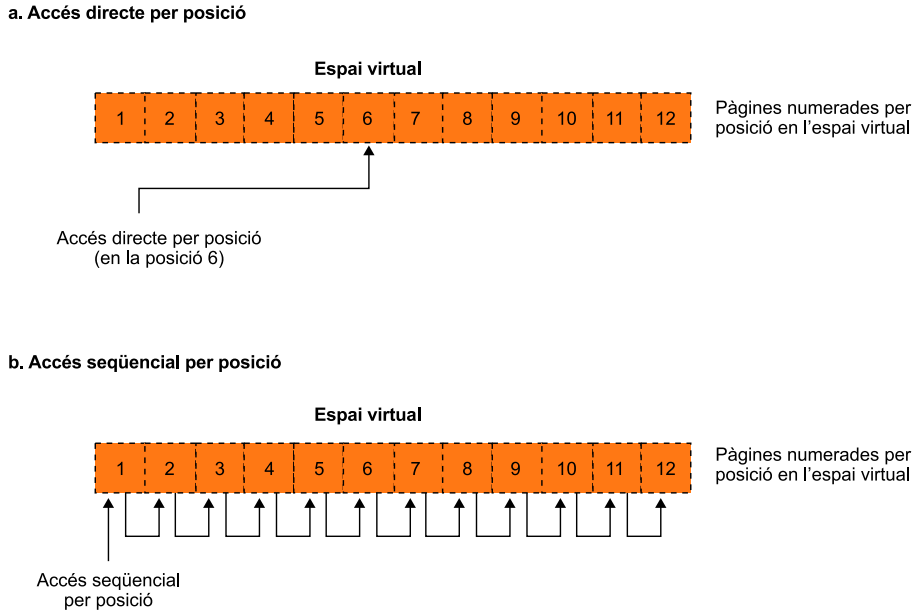
L'accés **seqüencial per posició** consisteix a obtenir les pàgines d'un espai seguint l'ordre definit pels seus números de pàgina.

Accessos per posició

Els accessos per posició són vells coneguts. L'accés seqüencial i l'accés directe per posició s'utilitzen també en la tecnologia dels fitxers. En concret, el fitxer seqüencial ofereix l'accés seqüencial per posició i el fitxer relatiu ofereix tant l'accés directe per posició com l'accés seqüencial per posició.

La figura 15 mostra aquests dos tipus d'accés per posició.

Figura 15. Els accessos per posició



Aquests tipus d'accés simples són suficients per a casos en què només calgui efectuar consultes i actualitzacions molt senzilles a la BD.

Exemple d'accés directe per posició

Suposem que disposem d'una BD que conté la taula *Employees*(*emplId, emplName, officeNum, salary*). Suposem també que aquesta taula s'emmagatzema en un espai determinat i que les files dels empleats se situen en l'espai esmentat per ordre d'inserció. Considerem ara l'actualització següent, descrita en SQL:

```
INSERT INTO Employees
VALUES (25, 'Joan Tarrago', 150, 2000);
```

Aquesta actualització es pot efectuar fàcilment. L'SGBD té enregistrat el número de la primera pàgina de l'espai amb capacitat d'absorbir noves files. Aleshores, es fa servir l'accés directe per posició per a inserir l'empleat en la pàgina esmentada (afegint-hi la fila corresponent a l'empleat nou).

Exemple d'accés seqüencial per posició

A la base de dades anterior, se li podria fer una consulta per a recuperar les dades de tots els empleats, com ara la següent:

```
SELECT * FROM Employees;
```

Per a resoldre aquesta consulta, l'SGBD pot utilitzar l'accés seqüencial per posició, mitjançant el qual l'SGBD va obtenint totes les pàgines que contenen les files dels empleats. D'aquesta manera pot proporcionar els empleats en el mateix ordre amb què els obté. Observeu que la consulta no requereix cap ordenació particular dels empleats i, aleshores, l'SGBD els pot proporcionar en el mateix ordre en què estan emmagatzemats.

6.1.3. Els accessos per valor

En aquest subapartat expliquem alguns tipus d'accés més complexos que els accessos per posició que acabem de veure.

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

L'**accés seqüencial per valor** consisteix a obtenir diverses files per l'ordre dels valors d'un atribut.

Exemples d'accés directe per valor

Considerem les sentències de manipulació següents:

- Sentència 1:

```
SELECT * FROM Employees WHERE officeNum = 150;
```

- Sentència 2:

```
UPDATE Employees SET salary = 2500 WHERE officeNum = 200;
```

- Sentència 3:

```
DELETE FROM Employees WHERE officeNum = 150;
```

Observeu que en tots tres exemples cal buscar les files dels empleats que tenen un número de despatx (*officeNum*) determinat, per a mostrar-los, modificar-los o esborrar-los, respectivament. És a dir, en tots tres casos cal fer cerques d'un valor per un atribut determinat.

D'aquests exemples es dedueix que l'accés directe per valor és necessari per a poder executar consultes i actualitzacions sobre files que contenen un determinat valor d'un atribut.

Exemples d'accés seqüencial per valor

Considerem les sentències de manipulació següents:

- Sentència 1:

```
SELECT * FROM Employees ORDER BY officeNum;
```

- Sentència 2:

```
SELECT * FROM Employees
WHERE officeNum >= 100 AND officeNum <= 300;
```

- Sentència 3:

```
UPDATE Employees SET salary = 2500
WHERE officeNum >= 100 AND officeNum <= 300;
```

- Sentència 4:

```
DELETE FROM Employees
WHERE officeNum >= 100 AND officeNum <= 300;
```

En tots aquests exemples cal obtenir algunes files de la taula d'empleats en una seqüència ordenada per l'atribut *officeNum*.

En la primera de les sentències cal obtenir els empleats ordenats pel número de despatx, a causa de la clàusula *ORDER BY officeNum*. Per a executar les tres sentències restants també s'ha de disposar de l'accés seqüencial per valor, encara que la causa és potser menys evident. La clàusula *WHERE officeNum >= 100 AND officeNum <= 300* fa que calgui localitzar tots els empleats que tenen un número de despatx entre el 100 i el 300, per a consultar-ne les dades, modificar-les o esborrar-les, respectivament. Una manera de localitzar tots els empleats és disposar d'algun mecanisme que accedeixi a cadascun d'ells per ordre de número de despatx a partir del valor 100 i fins al 300. Aquest mecanisme és l'accés seqüencial per valor.

Dels exemples anteriors es desprèn que l'accés seqüencial per valor es fa servir per a executar consultes en què el resultat ha d'estar ordenat per un atribut i per a consultes o actualitzacions que afectin un conjunt de files que contenen el valor d'un atribut que es troba dins d'un rang determinat de valors.

6.1.4. Els accessos per diversos valors

Hem considerat accessos per valor d'un sol atribut. Ens manca considerar els casos en què es vol accedir segons els valors de diversos atributs: els accessos per diversos valors. Els accessos per diversos valors poden ser, com els accessos per un sol valor, directes o seqüencials.

Els accessos per diversos valors

Els accessos per diversos valors no tenen el seu corresponent en la tecnologia dels fitxers, a diferència d'allò que passava amb els altres accessos que hem explicat.

Exemples d'accés per diversos valors

Considerem els exemples següents d'accés per diversos valors:

- Sentència 1:

```
SELECT *
FROM Employees
WHERE officeNum = 150 AND salary = 2000;
```

- Sentència 2:

```
SELECT *
FROM Employees
ORDER BY officeNum, salary;
```

- Sentència 3:

```
DELETE *
FROM Employees
WHERE officeNum >= 100 AND salary >= 1800;
```

En aquests exemples s'accedeix a les dades segons els valors de dos atributs: l'atribut *officeNum* i l'atribut *salary*. La primera sentència correspon a un accés directe per diversos valors i les dues següents a un accés seqüencial per diversos valors.

També pot passar, però, que els accessos per diversos valors siguin mixtos.

Els **accessos mixtos** per diversos valors són accessos en què es combina l'accés directe per valors d'alguns atributs i l'accés seqüencial per valors d'altres atributs.

Exemples d'accessos mixtos per diversos valors

Considerem les sentències següents:

- Sentència 1:

```
SELECT *
FROM Employees
WHERE officeNum = 150 AND salary >= 2000;
```

- Sentència 2:

```
SELECT *
FROM Employees
WHERE salary = 2000
ORDER BY officeNum;
```

- Sentència 3:

```
DELETE *
FROM Employees
WHERE officeNum >= 100 AND officeNum <= 300 AND salary = 2000;
```

La primera de les sentències combina un accés directe per valor de l'atribut *officeNum* amb un accés seqüencial per valor de l'atribut *salary*. Les dues següents combinen un accés seqüencial per valor de l'atribut *officeNum* amb un accés directe per valor de l'atribut *salary*.

6.2. Implementació dels accessos per posició

Per a la implementació dels accessos per posició, els SGBD es basen gairebé completament en l'SO. Les rutines de gestió d'E/S de l'SO permeten obtenir una pàgina real si tenim la seva adreça i també permeten enregistrar en el disc una pàgina real concreta. Podríem dir que l'SO implementa l'accés per posició a pàgines reals.

D'acord amb el punt de partida que acabem de presentar, la implementació dels accessos per posició és molt simple.

En el cas de l'accés **directe per posició**, només cal que l'SGBD transformi els números de posició de les pàgines virtuals en adreces de pàgines reals emmagatzemades al disc.

Si l'SGBD necessita fer un accés **seqüencial per posició**, el mateix procediment que hem explicat per a l'accés directe per posició li serveix per a anar accedint a totes les pàgines, des de la primera a la darrera.

6.3. Implementació dels accessos per valor

La implementació dels accessos per valor és més complexa que la dels accessos per posició. La dificultat prové del fet que si l'SGBD es basés únicament en l'SO per a implementar-los, no sempre obtindria un bon rendiment. Caldrà, doncs, que l'SGBD disposi de mecanismes propis especialitzats que els implementin de manera eficient.

En aquest apartat veurem que per a implementar d'una manera eficient l'accés directe i l'accés seqüencial per valor, els SGBD fan servir unes estructures de dades auxiliars que s'anomenen *índexs*. Per a la implementació dels accessos per valor amb índexs partirem del fet que ja disposem dels accessos per posició que hem explicat. Els SGBD fan servir els accessos per posició per a implementar els accessos per valor.

6.3.1. Necessitat dels índexs

Començarem explicant per què els índexs són necessaris si es desitja obtenir un bon rendiment quan es fan accessos per valor. Per a fer-ho, presentarem algunes implementacions que no els utilitzen i veurem els problemes que comporten.

Considerem la sentència següent, que requereix un accés directe per valor de l'atribut *officeNum*:

```
SELECT * FROM Employees WHERE officeNum = 150;
```

Cal tenir en compte que les files de la taula *EMPLEATS* són en un espai virtual. Aleshores, com que disposem de l'accés seqüencial per posició a les pàgines de l'espai virtual, el que pot fer l'SGBD és emprar aquest accés per a obtenir totes les pàgines una a una, comprovant, per a cadascuna d'elles, quines files contenen el valor buscat.

L'inconvenient de la solució anterior és que requerirà un gran nombre d'E/S, sobretot si hi ha molts empleats a la BD. Caldrà consultar totes les pàgines de la BD que tenen files d'empleats per tal d'aconseguir únicament els empleats del despatx 150, que poden ser molt pocs.

Ara analitzem un altre exemple que requereix un accés seqüencial per valor; vegeu la sentència següent:

```
SELECT * FROM Employees ORDER BY officeNum;
```

Per a obtenir els empleats per ordre de número de despatx convindria que estiguessin emmagatzemats segons aquest mateix criteri. Llavors, utilitzant l'accés seqüencial per posició que ens facilita l'SO, s'aconseguirien els empleats en l'ordre desitjat.

El problema d'aquesta solució és que l'ordenació dels empleats per número de despatx aniria molt bé per a la consulta anterior, però molt malament si també es volgués executar, per exemple, la sentència següent:

```
SELECT * FROM Employees ORDER BY emplId;
```

Com que en una BD han de conviure habitualment consultes i actualitzacions que requereixen ordenacions diferents d'unes mateixes dades, la solució que hem presentat no és satisfactòria per a totes les sentències que caldrà poder executar de manera eficient.

Les solucions que hem analitzat per a implementar els accessos per valor no ens permeten aconseguir un rendiment prou acceptable en molts casos. Tot seguit analitzarem la manera en què els índexs ens permetran millorar aquesta situació.

6.3.2. Característiques generals dels índexs

Hi ha índexs de molts tipus diferents, però tots tenen algunes característiques generals comunes que fan que siguin implementacions adequades dels accessos per valor. Els índexs dels SGBD tenen una utilitat semblant a la dels índexs que contenen els llibres per a localitzar-ne ràpidament un apartat determinat.

L'índex d'un llibre

Aquest mòdul té un índex al principi. Si volem localitzar amb rapidesa el subapartat dedicat als arbres B^+, el que hem de fer és consultar l'índex. Allà trobarem fàcilment el número de pàgina on està situat el subapartat dels arbres B^+, perquè l'índex no és gaire llarg i no trigarem a llegir-lo. Un cop conegut el número de pàgina, només ens cal localitzar la pàgina que té aquell número.

Els **índexs** que empren els SGBD són unes estructures de dades auxiliars que, com els índexs dels llibres, faciliten les cerques sobre unes determinades dades.

Un dels motius pels quals les cerques poden ser més ràpides si es fan mitjançant l'índex que si es fan directament sobre les dades, és que habitualment els índexs ocupen menys espai que les dades.

Les files que contenen les dades generalment són voluminoses perquè emmagatzemen molts atributs. Els índexs, en canvi, normalment contenen només una col·lecció de parelles, anomenades *entrades*, formades per un valor i un RID.

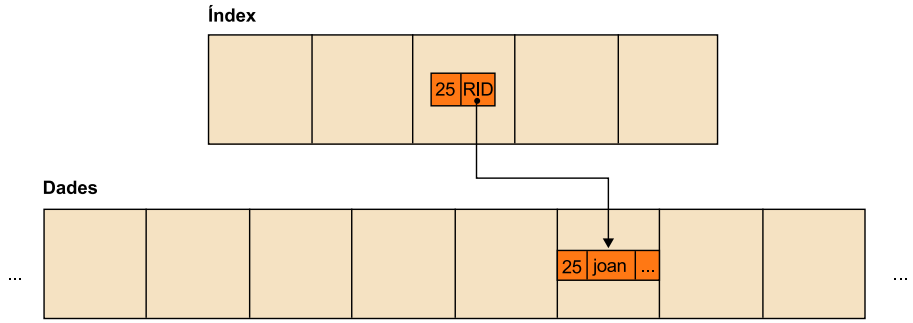
Espai ocupat pels índexs i per les dades

Les files d'una taula d'empleats podrien enregistrar el número d'empleat, el nom, el DNI, l'adreça, el telèfon, el sou, el número de despatx, el departament on està assignat, etc. En canvi, si volem tenir un índex per a accedir als empleats segons el seu número d'empleat, les entrades de l'índex hauran d'estar formades només per un número d'empleat i un RID.

Una cerca mitjançant un índex consisteix a localitzar primer els valors en l'índex per tal de conèixer-ne el RID. Un cop es disposa del RID, s'empra un accés directe per posició per a aconseguir la pàgina que conté la fila de les dades que es buscava.

La figura 16 il·lustra la consulta de l'empleat número 25 amb un índex.

Figura 16. Consulta mitjançant un índex



Dels paràgrafs anteriors podem deduir que els SGBD utilitzen l'accés per posició per tal d'implementar els accessos per valor.

Un índex ha d'estar organitzat d'alguna manera que faciliti les cerques dels valors que conté. Hi ha diverses maneres d'organitzar els índexs: els arbres B^+, les funcions de dispersió, etc. Alguns d'aquests tipus d'índex només faciliten l'accés directe per valor (per exemple, les funcions de dispersió); d'altres, serveixen tant per a l'accés directe com per a l'accés seqüencial per valor (per exemple, els arbres B^+).

Exemple

Sobre una taula que conté dades d'empleats podem tenir un índex que faciliti l'accés per número d'empleat, un altre que faciliti l'accés per nom, un altre que faciliti l'accés per número de despatx, etc.

Una característica molt útil de la majoria de tipus d'índex és que permeten la possibilitat de tenir diversos índexs sobre unes mateixes dades.

Les peculiaritats dels índexs fan aconsellable gestionar-los separadament de les dades de les taules. Per això, hi ha un tipus d'espai virtual per a contenir-los, anomenat *espai d'índexs*.

Més endavant estudiem els tipus d'índexs que fan servir més sovint els SGBD per a implementar els accessos per valor: els estructurats com a arbres B^+ i els estructurats segons funcions de dispersió. Cada SGBD té les seves particularitats pròpies en la implementació d'un tipus d'índex determinat. Atès que seria impracticable intentar explicar exhaustivament i amb tots els detalls les implementacions que hi ha, només estudiem els principis comuns a la majoria d'implementacions d'índexs estructurats com arbres B^+ i els dels índexs estructurats mitjançant funcions de dispersió. Aquests principis ens faciliten la comprensió de les implementacions particulars que proporcionen els SGBD del mercat.

Observació

Fixeu-vos que els índexs també es poden fer servir per a implementar els accessos que requereixen els fitxers per valor.

L'estimació del cost d'execució dels accessos a les dades mitjançant la utilització d'índexs és un aspecte que ens caldrà considerar d'ara endavant per tal d'avaluar la bondat dels diferents tipus d'índex. Per a aquesta estimació prenem les convencions següents:

1) Considerem només el cost de les E/S i no altres components del cost, com ara el que correspon als càlculs de la UCP[16]. El motiu és que el cost de les E/S és normalment el component dominant en el cost de les operacions que es fan en les BD i ens proporciona una bona aproximació als costos reals.

(16)UCP és la sigla del terme *unitat central de processament*.

2) Per a comptabilitzar el cost de les E/S adoptem la simplificació de comptar el nombre de pàgines que es llegeixen o que es graven en el disc. Aquesta simplificació ens serveix perquè suposem que totes les E/S transporten únicament una pàgina (és el cas més habitual) i que per tal d'accedir a una pàgina sempre cal fer una E/S, tot i que en alguns casos concrets podria no caldre (per exemple, si ja s'ha accedit a la pàgina amb anterioritat i encara se'n té una còpia en la memòria interna).

6.3.3. Arbres B+

En els subapartats següents presentem un tipus d'índex que serveix per a facilitar els accessos directe i seqüencial per valor d'un atribut: els arbres B^+.

Els **arbres B^+** són un tipus particular d'arbre de cerca. L'objectiu primordial de l'estructura dels arbres B^+ és el d'intentar aconseguir que les cerques s'efectuïn amb un nombre petit d'E/S.

És important tenir en compte que, a diferència d'altres tipus d'arbres, els arbres B^+ que estudiem aquí estan orientats a fer cerques en el disc.

L'estudi dels arbres B^+ és molt interessant, perquè la majoria de sistemes comercials (com ara Oracle, PostgreSQL, Informix, DB2, etc.) l'implementen.

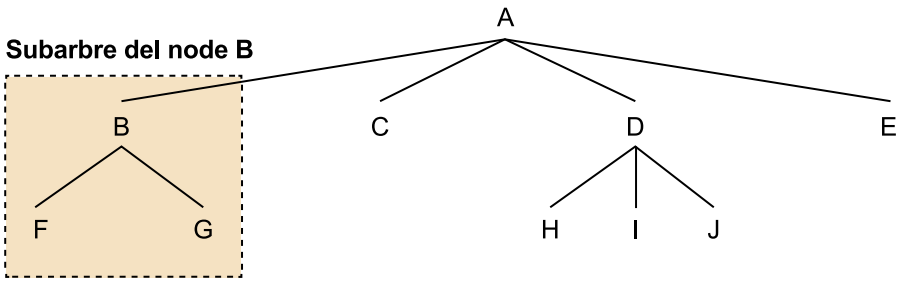
Terminologia de les estructures de dades d'arbre

Comencem presentant breument la terminologia que s'empra en parlar de les estructures de dades en forma d'arbre.

Un **arbre** es compon de nodes. Cada **node de l'arbre**, excepte un node especial anomenat *arrel*, té un node pare i diversos (zero o més) nodes fills. El **node arrel** no té pare. Els nodes que no tenen fills s'anomenen **nodes fulla** i els nodes que no són fulles s'anomenen **nodes interns**. El nivell d'un node és sempre el nivell del seu pare més un, i el nivell del node arrel és un. Un **subarbre d'un node** consisteix en el node i tots els seus nodes descendents: els seus nodes fills, els nodes fills dels seus fills, etc.

La figura 17 il·lustra l'estructura de dades d'arbre que acabem de presentar.

Figura 17. Estructura d'un arbre



Exemple

En la figura 17 podem veure els següents elements de l'arbre:

- El node arrel de la figura és A.
- Els nodes fills de A són B, C, D i E.
- Els nodes fulla són F, G, C, H, I, J i E.
- El subarbre del node B és el marcat en la figura.

Estructura dels nodes

En primer lloc, cal considerar que tot arbre B^+ té un número d'ordre d que indica la capacitat dels seus nodes. Més concretament, si un arbre B^+ té ordre d, aleshores els seus nodes contenen com a màxim $2d$ valors.

En un arbre B^+ els nodes interns i els nodes fulla tenen una estructura diferent. Les causes d'aquesta diferència estructural són tres:

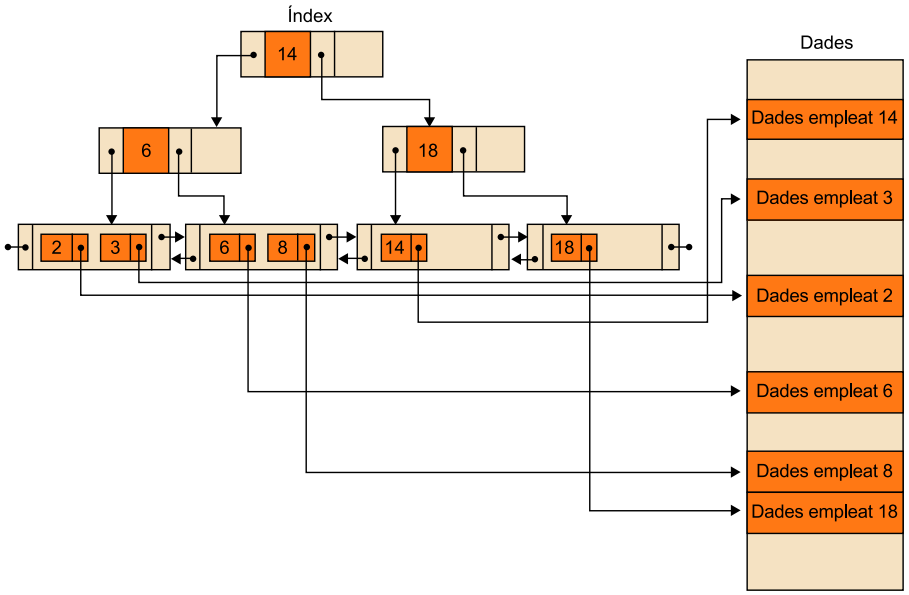
1) Els nodes fulla són els que contenen totes les entrades de l'índex (o sigui, les parelles de valor i RID).

2) Els nodes interns tenen com a objectiu dirigir la cerca de la fulla que té l'entrada corresponent a un valor determinat. L'accés directe per valor consistirà, doncs, a fer un recorregut de l'arbre que començarà en l'arrel i anirà baixant pels nodes de l'arbre fins a arribar a la fulla adequada (la que conté, si existeix, l'entrada corresponent al valor buscat).

3) Els nodes fulla estan connectats per apuntadors, que serveixen per a facilitar l'accés seqüencial per valor (el recorregut de les fulles seguint els apuntadors proporciona les entrades ordenades per valor).

La figura 18 il·lustra el que acabem d'explicar.

Figura 18. Índex de l'arbre B^+



Exemple d'estructura d'un arbre B^+

La figura 18 mostra un índex d'arbre B^+ d'ordre $d = 1$ que serveix per a accedir a dades d'empleats segons el valor de l'atribut "número d'empleat". Fixeu-vos que els RID que apunten a les dades són només en els nodes fulla, que els nodes interns serveixen per a buscar els valors de les fulles i que el fet que les fulles estiguin connectades entre si facilita l'accés seqüencial per valor.

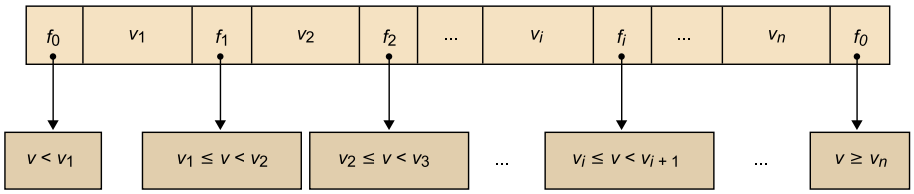
A continuació descriurem l'estructura dels nodes interns i, després, l'estructura dels nodes fulla dels arbres B^+.

1) Estructura d'un node intern

Un **node intern** conté valors i apuntadors cap als seus nodes fills (una manera d'implementar un arbre és tenir en cada node apuntadors cap a tots els seus nodes fills).

L'estructura d'un node intern que conté n valors (on n és menor o igual que $2d$) és la que mostra la figura 19.

Figura 19. Estructura d'un node intern



Cada v_i indica un valor i cada f_i indica un apuntador a un node fill de l'arbre. Si un node conté n valors, aleshores té $n + 1$ apuntadors a altres nodes de l'índex.

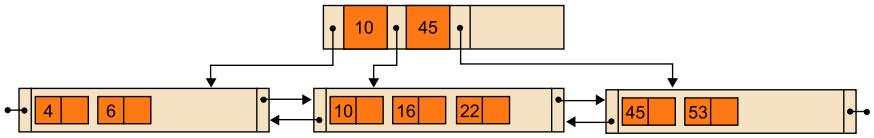
Tal com indica la figura 19, es compleix que el subarbre del node apuntat per f_i conté valors v tals que:

- $v_i \leq v < v_{i+1}$, si $i > 0$ i $i < n$.
- $v < v_1$, si $i = 0$.
- $v \geq v_n$, si $i = n$.

Exemple d'estructura d'un node intern

La figura 20 mostra diversos nodes d'un índex d'arbre B^+ d'ordre 2 per l'atribut número d'empleat.

Figura 20. Exemple d'arbre B^+ d'ordre 2



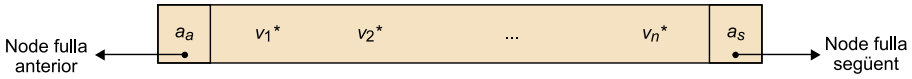
Considereu el node arrel: té tres apuntadors que apunten als seus nodes fills. Fixeu-vos que el subarbre apuntat pel primer d'aquests apuntadors conté valors menors que 10. El subarbre apuntat pel segon conté valors més grans o iguals que 10 i més petits que 45. Finalment, el subarbre apuntat pel tercer conté valors més grans o iguals que 45.

2) Estructura d'un node fulla

Els **nodes fulla** dels arbres B^+ contenen entrades formades pels valors als quals s'ha d'accedir i el RID que apunta a les dades, un apuntador al node fulla anterior i un apuntador al node fulla següent.

L'estructura d'un node fulla que conté n valors (on n és menor o igual que $2d$) és la que mostra la figura 21. En la figura, cada v_i correspon a l'entrada del valor v_i (per tant, v_i i el seu RID), a_a indica l'apuntador al node fulla anterior i a_s indica l'apuntador al següent.

Figura 21. Estructura d'un node fulla



Addicionalment, cal que els nodes fulla compleixin les condicions següents:

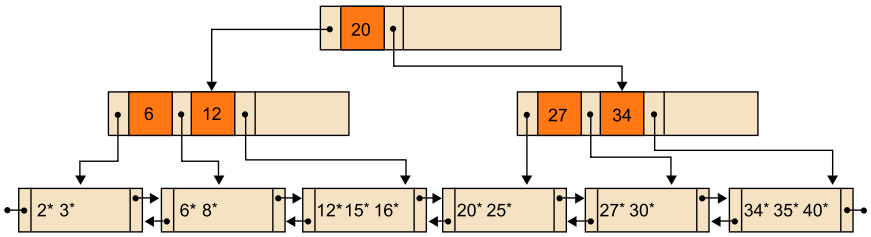
- Tots els valors d'un node fulla són més petits que els valors del node fulla següent.

- Tots els valors d'algun node intern de l'arbre estan repetits en alguna fulla de l'arbre (així, les fulles contenen tots els valors de l'arbre).

Exemple d'estructura d'un node fulla

La figura 22 mostra diversos nodes d'un índex d'arbre B^+ d'ordre 2 per l'atribut número d'empleat.

Figura 22. Exemple d'estructura d'un node fulla



Considereu, per exemple, el node fulla que està situat més a l'esquerra. Conté les entrades de l'empleat número 2 i de l'empleat número 3. També té els apuntadors a les fulles anterior i següent de l'arbre (en aquest cas concret, no existeix fulla anterior). Fixeu-vos que tots els valors del node considerat (els valors 2 i 3) són menors que els valors de la fulla següent (6 i 8). Finalment, observeu que el valor 6 de la segona fulla està repetit en un node intern. El motiu és que en un arbre B^+ les fulles han de contenir tots els valors de l'arbre.

Accés directe per valor

Per a fer un **accés directe per valor** amb un índex del tipus arbre B^+ cal, primer, localitzar la fulla que té l'entrada del valor buscat i, després, fer servir el RID de l'entrada per a trobar les dades a les quals es vol accedir.

Exemple de localització de la fulla que té l'entrada del valor buscat

Mostrarem com es pot fer la localització d'un valor v a l'arbre amb un exemple de la figura 22. Suposem que volem trobar el valor 8 a l'arbre de l'exemple anterior.

Accedim primer al node arrel. Després seguim el primer apuntador perquè $8 < 20$. A continuació, seguim el segon apuntador, atès que $6 \leq 8 < 12$, i localitzem el node fulla que conté l'entrada del valor desitjat.

Suposem ara que busquem el valor 26 en el mateix arbre. Accedirem igualment al node arrel. Seguirem l'apuntador al fill de la dreta, perquè $26 \geq 20$. Després seguirem el primer apuntador perquè $26 < 27$. El node fulla que obtenim aquesta vegada no conté l'entrada del valor 26, la qual cosa indica que el 26 no és a les nostres dades.

Accés seqüencial per valor

Per tal de fer un accés seqüencial per valor amb un índex del tipus arbre B^+ cal, primer, localitzar ordenadament totes les entrades dels valors als quals es vol accedir i, per a cadascuna, fer servir el RID que apunta a les dades per a trobar el valor.

La localització ordenada de les entrades de l'arbre és senzilla. Convé recordar que les fulles contenen tots els valors de l'arbre, que cada fulla té un apuntador a la fulla següent i que els valors d'un node fulla són menors que els valors del node fulla següent. Aleshores, només cal accedir primer a la fulla situada més a l'esquerra i, després, anar accedint cada vegada a la fulla següent fins que s'acabi la cadena de fulles.

Observació

Si seguim el procediment d'accés seqüencial per valor explicat en aquest subapartat amb l'arbre de la figura 21, anirem obtenint totes les entrades de l'arbre de manera ordenada.

Propietats destinades a millorar el rendiment

Considerem un accés directe per valor que s'implementa amb un arbre B^+. Per tal de localitzar la fulla que conté l'entrada del valor, el nombre de nodes que cal recórrer és igual al del nivell de la fulla esmentada. En conseqüència, si reduïm el nivell de les fulles d'un arbre B^+, aconseguirem reduir el nombre de nodes que cal recórrer quan es fa un accés directe per valor.

Una propietat dels arbres B^+, destinada a reduir el nivell de les fulles, és que tots els nodes de l'arbre (excepte l'arrel) han d'estar plens, com a mínim, fins a un 50% de la seva capacitat. La norma que acabem d'enunciar equival a exigir que, en un arbre d'ordre *d*, tots els nodes excepte l'arrel continguin, almenys, *d* valors.

Aquesta norma evita tenir arbres B^+ on molts dels nodes estan pràcticament buits. Si aconseguim que els nodes d'un arbre no estiguin gaire buits, l'arbre tindrà menys nodes i també menys nivells.

Amb vista al bon rendiment d'un índex, habitualment és desitjable que el nombre de nodes que cal recórrer sigui sempre més o menys el mateix, independentment de quin sigui el valor al qual es vol accedir.

Pel motiu anterior, els arbres B^+ compleixen una segona propietat: ser equilibrats. Un arbre és equilibrat si totes les seves fulles estan al mateix nivell.

En un arbre equilibrat, el nivell de les fulles és el que s'anomena **alçada de l'arbre**. La localització de qualsevol fulla sempre requerirà recórrer un nombre de nodes que coincidirà amb l'alçada de l'arbre.

Observació

Fixeu-vos que l'arbre de la figura 22 és un arbre equilibrat i la seva alçada és 3.

Emmagatzematge de l'arbre i cost de localització d'una entrada

Tal com ja hem vist, les peculiaritats dels índexs fan que sigui aconsellable gestionar-los separadament de les dades de les taules. Per aquest motiu, hi ha un tipus d'espai virtual per a contenir els índexs, anomenat *espai d'índexs*.

Una qüestió important de l'emmagatzematge d'un arbre és l'elecció de la mida adequada dels seus nodes (que depèn, almenys, de l'ordre de l'arbre). Ja hem explicat que interessa que un arbre B^+ tingui una alçada petita; per tal de facilitar-ho, cal que els nodes de l'arbre siguin grans però sense sobrepassar la mida d'una pàgina, perquè altrament no es podrien consultar amb una única E/S.

En conseqüència, la mida habitual dels nodes d'un arbre B^+ coincideix amb la mida de les pàgines i cada node de l'arbre s'emmagatzema en una pàgina virtual diferent. Observeu que, segons aquesta forma d'emmagatzematge, en un arbre B^+ d'alçada h calen h E/S per a localitzar una entrada de l'arbre (per exemple, si $h = 3$, caldran tres E/S).

El fet que els nodes siguin de la mida d'una pàgina (que emmagatzema normalment 4 kB) permet tenir habitualment arbres d'ordres entre 50 i 100.

Reflexió

Si pensem que l'arbre B^+ es construeix sobre el número d'empleat, i suposant que aquest número és la clau primària de la relació d'empleats, la nostra relació podrà contenir fins a 338.100 empleats diferents.

Ordre habitual d'un arbre i volum de dades indexades

a) Suposem que disposem de pàgines de 4 kB i els valors que cal indexar ocupen 20 bytes. Considerem també que la mida del RID és de 4 bytes i els apuntadors a pàgines de l'arbre ocupen 3 bytes. Segons aquestes dades, en els nodes interns hi caben 177 valors i 178 apuntadors a altres nodes de l'arbre; en els nodes fulla hi caben 170 entrades (parelles de valor i RID) i els 2 apuntadors a les fulles anterior i següent. Com que la capacitat en valors ha de ser la mateixa en tots els nodes, la restringirem a 170 (cas pitjor). L'ordre de l'arbre serà, doncs, la meitat de 170, és a dir, $d = 85$. Fixeu-vos que hem obtingut un ordre que es troba entre 50 i 100.

b) Considerem ara un arbre d'ordre $d = 50$, d'alçada $h = 3$ i on tots els nodes tenen una ocupació del 69%. Si $d = 50$, la capacitat màxima dels nodes és de 100 i una ocupació del 69% suposa tenir 69 valors a cada node. En aquest arbre tindrem el nombre de valors següent a cada nivell:

- Nivell 1: 1 node amb 69 valors i 70 apuntadors.
- Nivell 2: 70 nodes amb 69 valors cadascun i 70 apuntadors cadascun.
- Nivell 3: 4.900 nodes amb 69 entrades cadascun.

És a dir, 338.100 entrades en total. Segons això, l'arbre B^+ anterior, que té una alçada de només 3 nivells, ens permet indexar un volum de dades de 338.100 files.

D'aquest exemple es desprèn que amb ordres $d = 50$ i una alçada de $h = 3$ es pot indexar una quantitat de dades força considerable.

Insercions i supressions

Les insercions i supressions s'han de fer de manera que l'arbre resultant satisfaci totes les propietats dels arbres B^+. Això implica fer, en alguns casos, una reestructuració de l'arbre.

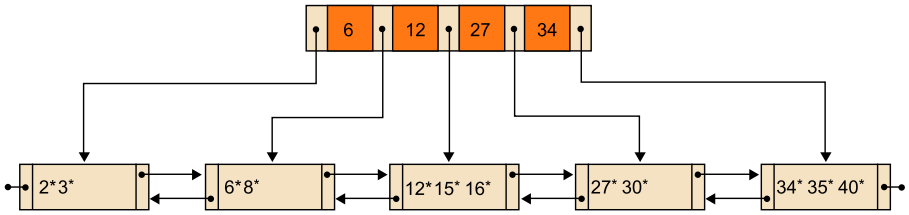
A) Insercions

Descrivim un algorisme que, donada una entrada, localitza el node fulla que li correspon i, si aquest node té espai lliure, la hi insereix. En el cas que aquest node fulla no tingui espai lliure per a la nova entrada, reestructura l'arbre per a fer-li lloc.

Comentarem alguns exemples que ens ajudaran a entendre amb més exactitud què ha de fer aquest algorisme d'inserció, la descripció detallada del qual queda fora de l'abast d'aquest mòdul.

Considerem l'arbre B^+ d'ordre 2 de la figura 23.

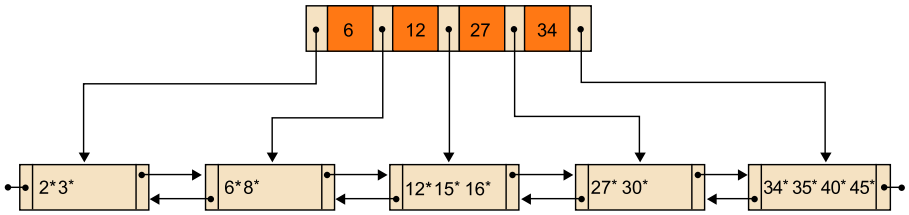
Figura 23. Arbre de partida I



A partir d'aquest arbre considerarem els casos d'inserció que exposem tot seguit:

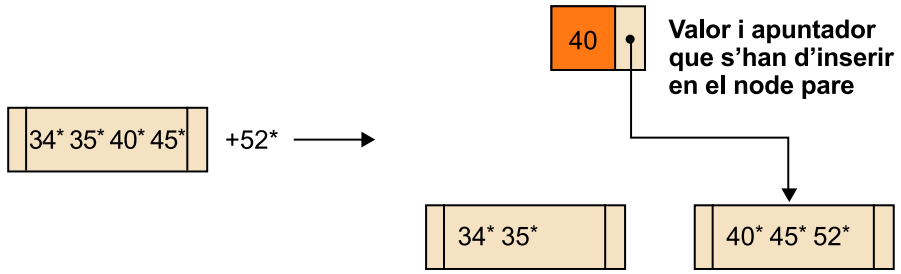
a) Suposem que es vol inserir en aquest arbre l'entrada 45*. El 45 ha de pertànyer a la fulla situada més a la dreta. Atès que aquesta fulla té espai lliure, l'entrada 45* es pot col·locar en la fulla que li correspon i no cal fer cap reestructuració de l'arbre. S'obté l'arbre que mostra la figura 24.

Figura 24. Inserció sense necessitat de reestructuració



b) Ara considerem que es vol inserir l'entrada 52* a l'arbre que acabem d'obtenir. Li correspon la fulla situada més a la dreta, però no té espai lliure. Per a poder inserir la nova entrada cal dividir la fulla en dues. Les dues fulles resultants de la divisió es mostren en la figura 25.

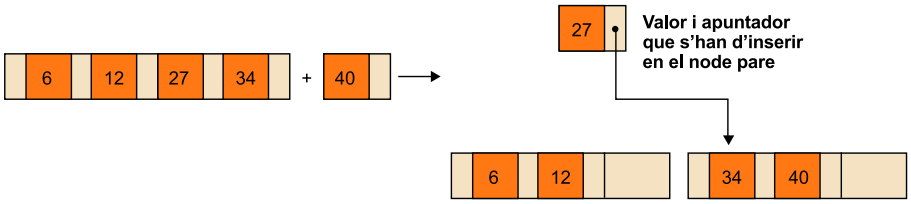
Figura 25. Divisió d'un node fulla



Quan es divideix un node, cal inserir en el seu node pare un valor i un apuntador al nou fill. En el nostre exemple, aquest valor és el 40, perquè és el valor més petit del segon node fulla que ha resultat de la divisió. Cal, doncs, inserir el 40 i l'apuntador al node pare. El node pare no té espai i caldrà dividir-lo.

Quan es divideix un node no-fulla, els seus primers *d* valors es deixen en el node (el 6 i el 12), els *d* valors més grans es col·loquen en un nou node (el 34 i el 40) i el valor del mig s'insereix en el node pare (el 27), juntament amb un apuntador cap al nou node. La figura 26 ens mostra aquesta divisió d'un node no-fulla.

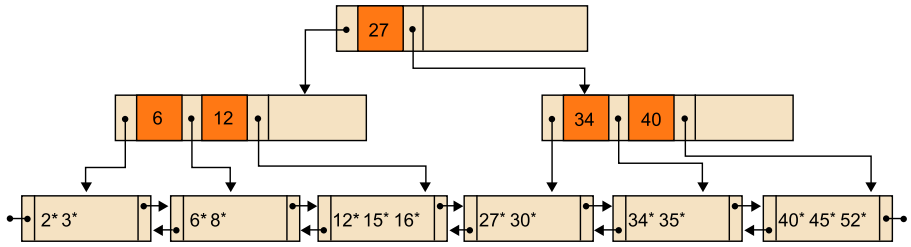
Figura 26. Divisió d'un node no-fulla



Observeu les diferències que hi ha entre aquesta divisió i la divisió d'un node fulla.

Com que acabem de dividir el node arrel de l'arbre, cal crear un nou node arrel, en el qual es col·loquen el valor 27, l'apuntador al nou node i també un apuntador a l'antiga arrel. L'arbre que s'obté finalment és el que podeu veure en la figura 27.

Figura 27. Inserció amb reestructuració



B) Supressions

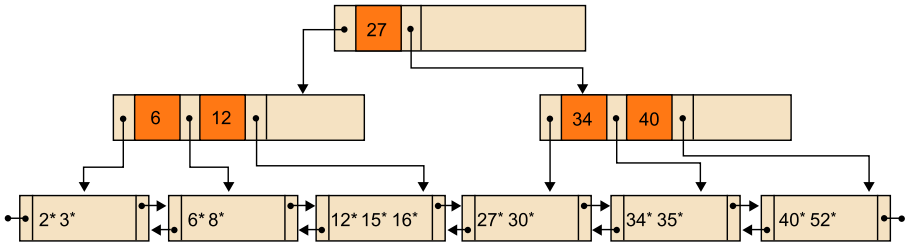
Descriurem un algorisme que, partint d'una entrada, localitza el node fulla que li correspon, l'esborra i, si cal, reestructura l'arbre perquè tots els nodes de l'arbre (excepte l'arrel) tinguin, almenys, *d* valors.

Comentarem alguns exemples per a entendre de manera més precisa com funciona l'algorisme, la descripció detallada del qual queda fora de l'abast d'aquest mòdul.

Considerem l'arbre de la figura 27 i suposem els casos de supressió següents:

a) Suposem que volem esborrar el valor 45 de l'arbre. La fulla que conté l'entrada del valor 45 és la que està situada més a la dreta. Després d'esborrar 45*, la fulla encara conté dos valors i, per tant, no cal fer cap reestructuració. L'arbre obtingut és el que mostra la figura 28.

Figura 28. Supressió sense necessitat de reestructuració



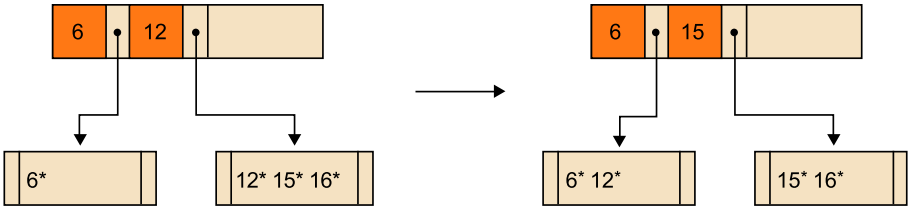
b) Ara suposarem que volem esborrar el valor 8 de l'arbre que acabem d'obtenir. La seva entrada està situada en la segona fulla de l'arbre. Després d'esborrar-la, la fulla es queda amb menys de dues entrades, cosa que cal corregir. Per a garantir que l'ocupació d'un node sigui correcta, és a dir, que l'estructura resultant després de la supressió segueixi essent un arbre B^+, sempre utilitzem el seu node germà de la dreta; només si no té germà a la dreta farem servir el de l'esquerra.

Nodes germans

Dos nodes són germans si comparteixen el mateix node pare.

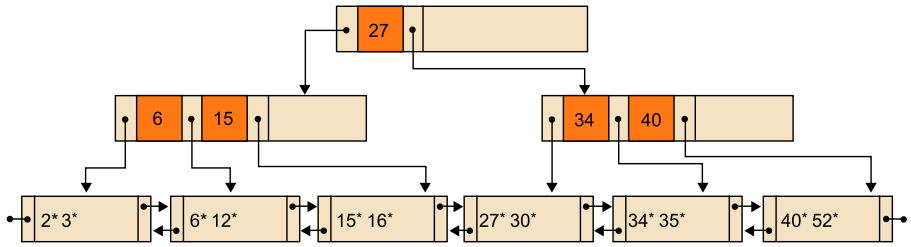
En el nostre exemple, com que el node germà de la dreta té tres entrades, es fa una redistribució d'entrades entre els dos nodes fulla. La figura 29 il·lustra aquesta redistribució.

Figura 29. Redistribució amb nodes fulla



Observeu que la redistribució afecta el contingut del node pare dels dos nodes que hi participen. El valor 12 del node pare es canvia pel 15. L'arbre que s'obté finalment és el que es mostra en la figura 30.

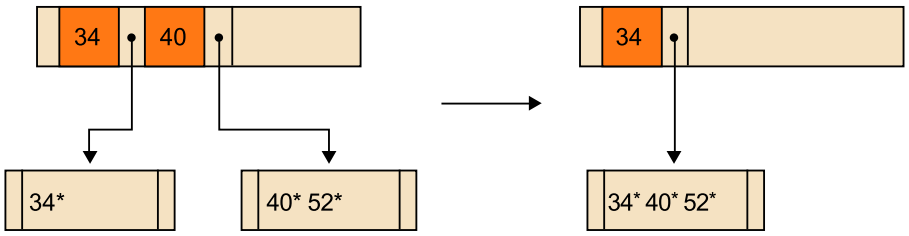
Figura 30. Supressió amb reestructuració



c) Considerem ara el cas de l'esborrament del 35 de l'arbre que acabem d'obtenir. L'entrada del 35 està situada en la penúltima fulla. Després d'esborrar-la, la fulla es queda amb menys de dues entrades, de manera que cal fer una reestructuració. En aquest cas, el seu node germà de la dreta té només dues entrades i, per tant, no es farà una redistribució com en el cas anterior (el node germà no té entrades sobreres per a redistribuir), sinó que es farà una fusió dels dos nodes fulla en un únic node.

La figura 31 ens mostra aquesta fusió dels nodes fulla.

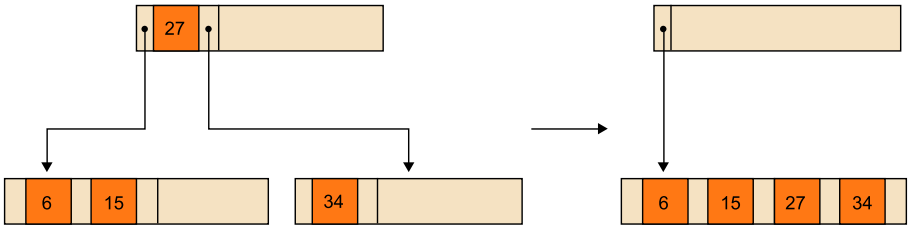
Figura 31. Fusió de nodes fulla



Fixeu-vos que quan es fusionen dos nodes, el seu node pare perd un valor. A causa d'aquesta pèrdua, el node pare del nostre exemple ens ha quedat amb menys de dos valors. En aquest cas, caldrà fer encara una altra reestructuració de l'arbre per tal de corregir-ho. El node pare no té cap germà a la dreta; per tant, farem servir el de l'esquerra per a reestructurar l'arbre.

El germà de l'esquerra té només dos valors i, en conseqüència, es farà una fusió dels dos nodes. La figura 32 ens il·lustra aquesta fusió de dos nodes no-fulla.

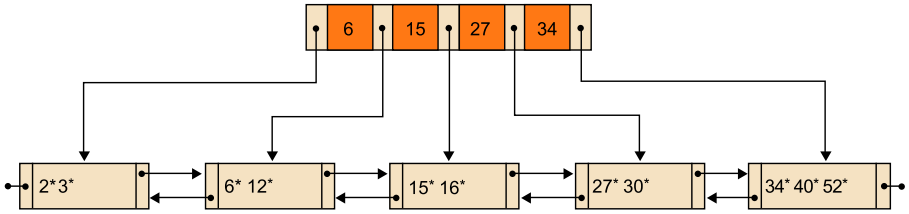
Figura 32. Fusió de nodes no-fulla



Convé observar les diferències respecte de la fusió anterior de dues fulles.

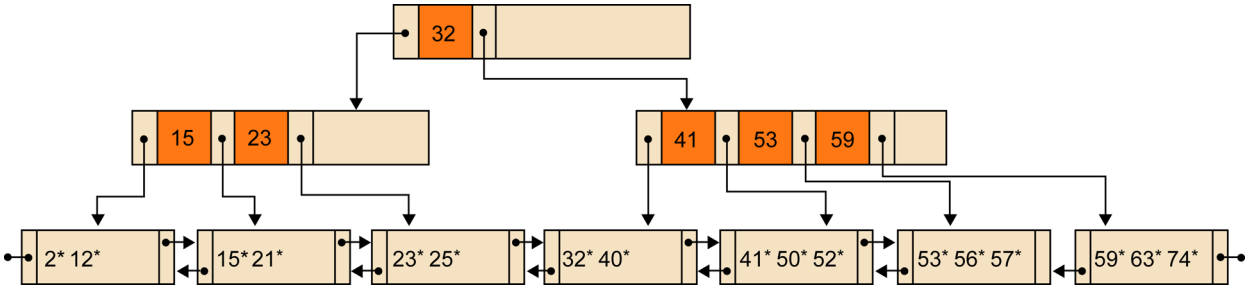
Després d'aquesta darrera fusió, ens ha quedat un arbre amb l'arrel buida. En aquests casos cal descartar l'arrel antiga i fer que l'arrel nova sigui el node resultant de la fusió. Finalment, s'obté l'arbre que mostra la figura 33.

Figura 33. Supressió amb reestructuració i pèrdua d'un nivell



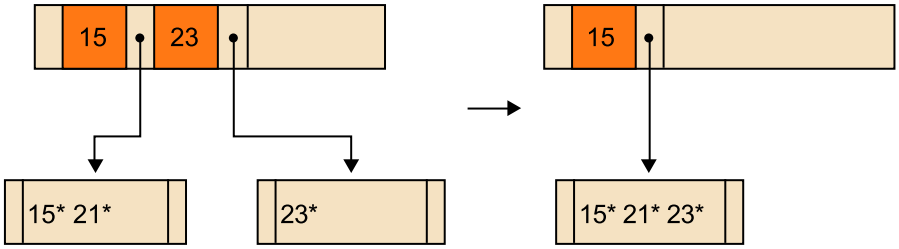
d) Per al darrer exemple de supressió, escollim l'arbre B^+ de partida que mostra la figura 34.

Figura 34. Arbre de partida II



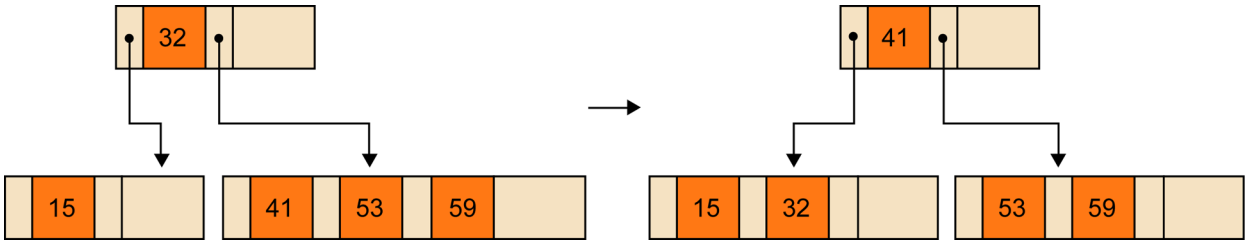
Considerem l'esborrament del valor 25 de l'arbre. L'entrada del 25 és a la tercera fulla. Després d'esborrar-lo, la fulla es queda amb menys de dues entrades. Aquesta fulla no té cap fulla germana a la dreta i cal fusionar-la amb la de l'esquerra, com mostra la figura 35.

Figura 35. Fusió de nodes fulla



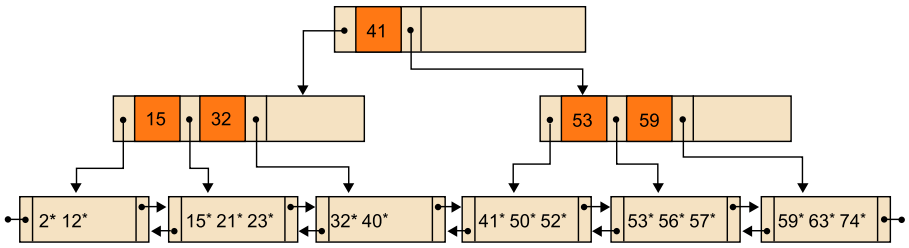
Després de la fusió, el node pare s'ha quedat amb menys de dos valors. Caldrà fer una redistribució amb el seu node germà de la dreta (que té més de dos valors). La figura 36 mostra aquesta redistribució de valors entre nodes no-fulla.

Figura 36. Redistribució amb nodes no-fulla



Aleshores, l'arbre B^+ que s'obté finalment és el que mostra la figura 37.

Figura 37. Supressió amb redistribució de nodes

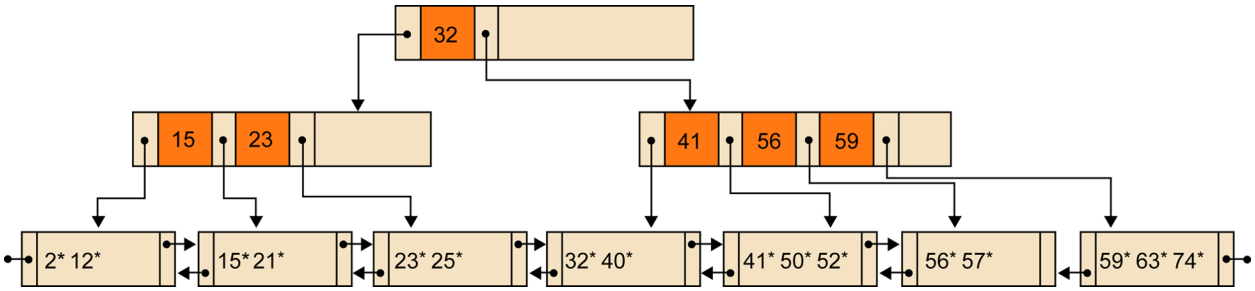


C) Supressions de valors que tenen una còpia en nodes interns

Hem analitzat un procediment d'esborrat dels valors d'un arbre B^+ que només apareixen en un node fulla de l'arbre. Com ja sabem, un arbre B^+ pot tenir valors que figuren en dos llocs: en un node fulla i en un node no-fulla (node intern). Per a esborrar els valors que tenen una còpia en nodes interns, podem utilitzar el mateix procediment si fem prèviament una substitució del valor copiat en el node intern pel valor de l'arbre que el segueix segons l'ordre dels valors. Un cop fet això, podrem emprar el procediment de supressió anterior per a eliminar el valor de la fulla.

Esborrem el valor 53 de la figura 34, que té l'entrada en la penúltima fulla de l'arbre i una còpia en el node pare de la fulla esmentada. Substituïm la còpia del 53 pel valor que el segueix, que és el 56. Un cop fet això esborrem el 53 de la fulla. L'arbre que ens queda es mostra en la figura 38.

Figura 38. Supressió de valors que tenen una còpia en nodes interns



Valors repetits

En les cerques, insercions i supressions dels arbres B^+ que acabem d'explicar no hem considerat la possibilitat que hi pogués haver valors repetits en les dades; hem considerat que indexàvem els valors d'atributs identificadors.

Hi ha diverses solucions per al tractament dels valors repetits en els arbres B^+; en comentarem dues breument:

1) Una possibilitat és permetre l'existència d'entrades diferents amb el mateix valor en l'arbre. Aleshores, fulles diferents poden tenir entrades corresponents a un mateix valor. L'algorisme d'accés directe per valor haurà de localitzar primer l'entrada situada més a l'esquerra del valor i, després, totes les possibles entrades addicionals del mateix valor. Aquestes entrades addicionals es poden trobar en altres fulles i es localitzen seguint els apuntadors entre fulles de l'arbre. Les insercions i supressions també s'han d'adaptar.

2) Una altra possibilitat és tenir una sola entrada que contingui diversos RID (un per a cada aparició del valor). Aquesta possibilitat fa que la mida de les entrades sigui variable i que la seva gestió sigui més complexa. Per contra, com que el valor en entrades diferents no es repeteix, s'estalvia espai, i, conseqüentment, operacions d'E/S.

6.3.4. Dispersió

Hi ha un altre tipus d'índexs que serveixen per a facilitar l'accés directe per valor, però no l'accés seqüencial per valor: són els índexs basats en la dispersió[17].

[17] en anglès, *hashing*.

A continuació, analitzarem la utilització de la dispersió per tal de fer cerques en el disc, on és fonamental la minimització del nombre d'E/S.

Els **índexs basats en la dispersió** aconsegueixen normalment un rendiment una mica millor que els índexs d'arbre B^+ quan es tracta d'implementar els accessos directes per valor. Per contra, no permeten implementar els accessos seqüencials per valor, mentre que els arbres B^+ sí. En conseqüència, alguns sistemes comercials proporcionen només implementacions basades en els arbres B^+.

Els sistemes comercials

Alguns sistemes comercials, com ara Oracle, proporcionen només implementacions basades en els arbres B^+. D'altres, com per exemple el sistema PostgreSQL, proporcionen tots dos tipus d'índexs.

Introducció a la dispersió

La característica fonamental dels índexs basats en la dispersió és que les entrades es col·loquen en les pàgines segons una funció de dispersió *h*.

La **funció de dispersió *h*** és una funció que a partir d'un valor *v* ens retorna un número de pàgina *p*, és a dir, $p = h(v)$. El **número de pàgina** *p* serà un enter de l'interval [1, ..., *N*]. Aleshores, l'entrada corresponent al valor *v*, que representem per *v**, se situarà en la pàgina de número *p* de l'interval.

Quan es vulgui accedir al valor *v*, podrem accedir a la seva entrada *v** de la manera següent:

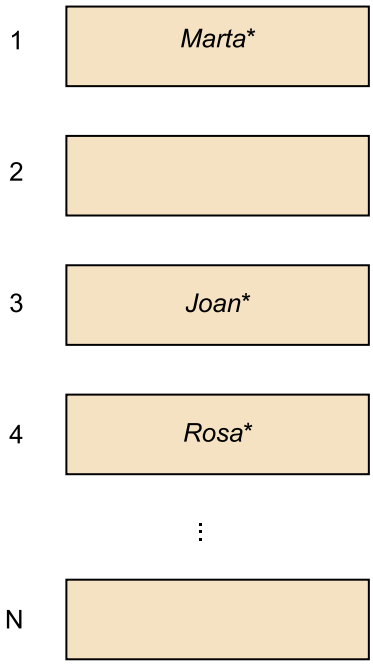
- Es calcula el número de pàgina *p*, tal que $p = h(v)$.
- S'accedeix a la pàgina *p* per a localitzar l'entrada del valor *v*.

Podem utilitzar diferents funcions *h* de dispersió que transformen un valor numèric o alfanumèric en un número enter de l'interval [0, ..., $N - 1$]. Podrem fer servir qualsevol d'aquestes funcions sempre que sumem 1 al resultat que ens donin, perquè ens interessa obtenir números de pàgina que estiguin en l'interval [1, ..., *N*].

Localització d'una entrada amb un índex basat en la dispersió

Volem indexar unes dades pel valor d'un atribut que representa noms de pila. Suposem que $h(Joan) = 3$, $h(Marta) = 1$ i $h(Rosa) = 4$. Aleshores, les entrades d'aquests valors quedaran col·locades en les pàgines de l'índex tal com es mostra en la figura 39.

Figura 39. Exemple de pàgina d'índex



Per tal de localitzar l'entrada del valor *Rosa* en l'índex anterior, calcularem h(*Rosa*). Aquest càlcul ens donarà el número de pàgina 4. Aleshores accedirem a la pàgina 4, on trobarem l'entrada corresponent a *Rosa*.

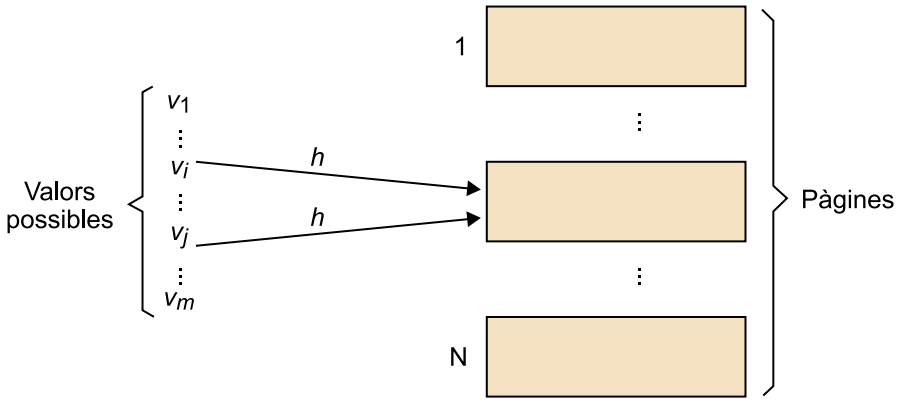
Exemple

Penseu en valors d'un atribut que representa un DNI. Els valors possibles de l'atribut es troben entre 0 i 99.999.999. En canvi, el nombre de pàgines que pot ocupar l'índex en el disc difícilment arribarà als cent milions.

Observeu que el nombre de valors possibles pot ser molt gran, mentre que el nombre de pàgines que pot ocupar l'índex en el disc sol ser molt més limitat. Això fa que el nombre de valors v possibles sigui molt més gran que el nombre N de pàgines possibles. La funció de dispersió h transforma valors d'un interval molt gran en posicions d'un interval més petit (l'interval [1, ..., N]).

Per aquest motiu, és possible que per a dos valors diferents, v_i i v_j, passi que $h(v_i) = h(v_j)$, és a dir, que la funció ens retorni la mateixa pàgina per als dos valors, com es veu en la figura 40.

Figura 40. Valors sinònims

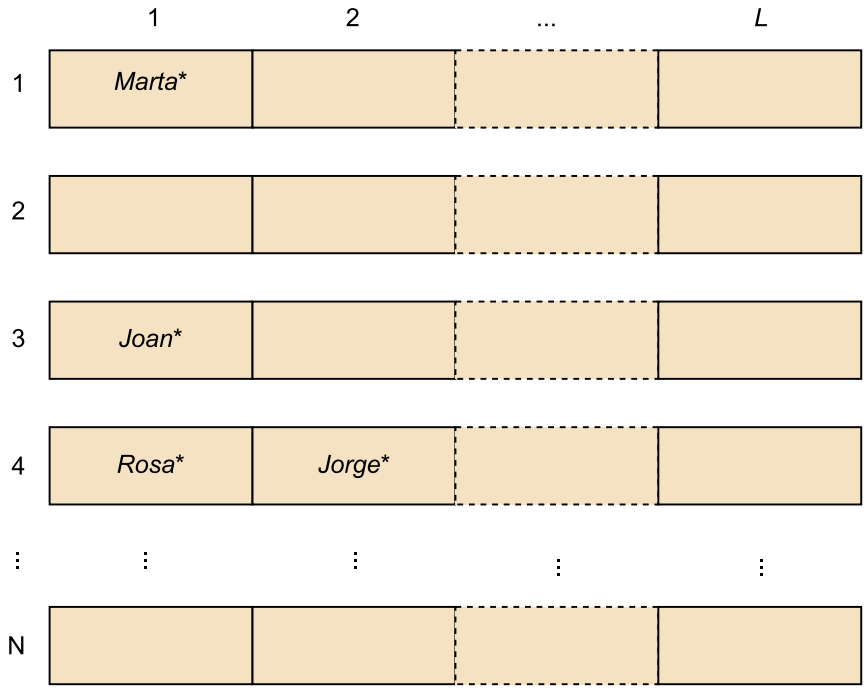


En aquest cas, els valors v_i i v_j s'anomenen *sinònims*. Les entrades corresponents als valors sinònims han d'estar en la mateixa pàgina. Les pàgines, per tant, poden haver d'emmagatzemar diverses entrades. Anomenem L el nombre d'entrades que caben en una pàgina.

Exemple de valors sinònims

Si a l'índex per l'atribut "nom de pila" anterior hi afegim el valor *Jorge* i h(*Jorge*) = 4, aleshores tindrem la situació que es mostra en la figura 41.

Figura 41. Exemple de pàgina d'índex



De vegades, una pàgina pot ser insuficient per a contenir tots els sinònims que s'hi haurien de col·locar. Això passarà quan a una pàgina li corresponguin més de *L* sinònims.

Quan un sinònim no té espai en la pàgina que li correspon, s'anomena *excedent* i es col·loca en alguna altra pàgina, que s'escull segons una determinada política de gestió d'excedents. Hi ha moltes maneres alternatives de gestionar els excedents; a continuació n'explicarem una.

Gestió d'excedents

La política de gestió d'excedents que estudiem es basa en el fet que l'índex el formen dos tipus disjunts de pàgines:

- **Pàgines primàries**: pàgines on col·locarem totes les entrades que no són excedents (n'hi haurà *N*).
- **Pàgines d'excedents**: pàgines destinades a desar les entrades excedents.

Quan una pàgina primària *p* s'omple i s'hi insereix un nou valor que segons la funció de dispersió hauria d'anar a la pàgina *p*, l'entrada del valor es col·loca en una pàgina d'excedents *e* i a la pàgina primària *p* s'afegeix un apuntador cap a la pàgina *e*.

Aquesta pàgina d'excedents *e* s'utilitzarà només per a encabir-hi els possibles excedents de la pàgina primària *p* que s'insereixin posteriorment. Si la pàgina *e* també s'omple, se li encadenarà una nova pàgina d'excedents *e'* on es podran col·locar més excedents de la pàgina primària *p*, i així successivament.

D'aquesta manera, per a cada pàgina primària que té excedents, es construeix una cadena de pàgines d'excedents. El primer apuntador de la cadena està situat en la pàgina primària que l'ha originat.

Com que hi ha una cadena separada de pàgines d'excedents per a cada pàgina primària, aquesta política de gestió d'excedents de vegades s'anomena **encadenament separat**[18].

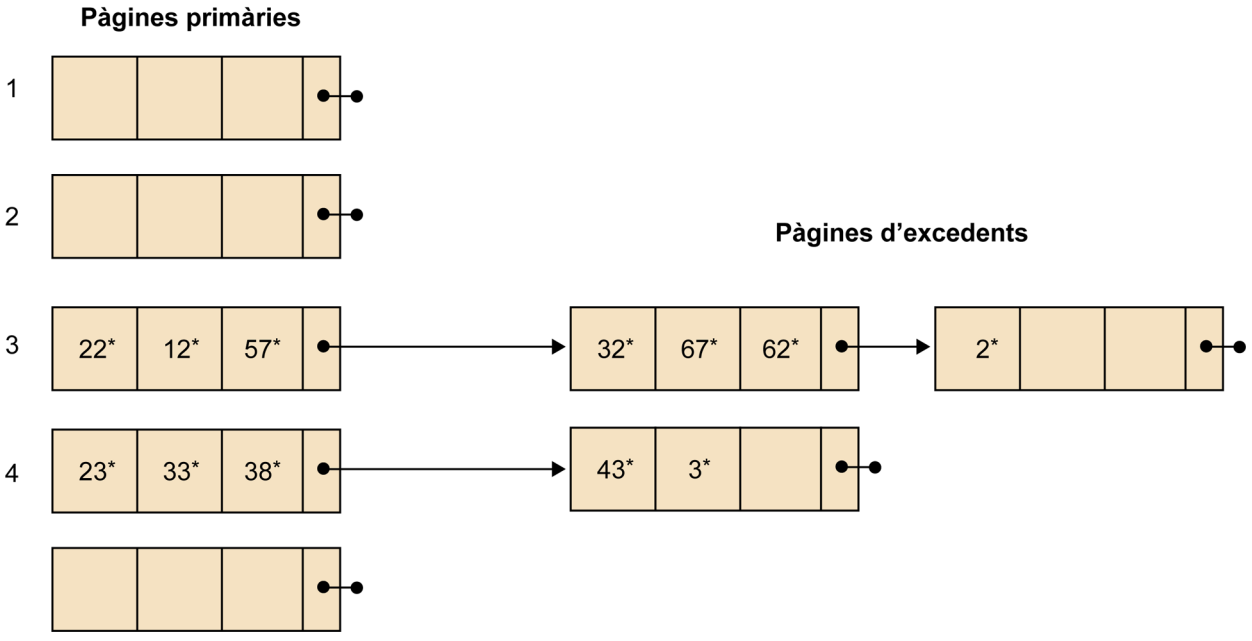
[18] En anglès, *separate chaining*.

Les supressions dels valors de l'índex es poden fer de diverses maneres. Una possibilitat és marcar el valor amb un senyal especial que indiqui que està esborrat, sense reorganitzar res. Una altra possibilitat diferent és aprofitar aquestes supressions de valors per a reduir la longitud de les cadenes d'excedents.

Exemple de gestió d'excedents amb encadenament separat

Suposem que tenim un índex amb 5 pàgines primàries (N = 5), que totes les pàgines tenen una capacitat de 3 entrades (L = 3) i que la funció de dispersió és $h(x) = (x \bmod 5) + 1$. Si inserim els valors: 22, 23, 12, 57, 33, 38, 32, 67, 62, 43, 2, 3, l'índex quedarà com mostra la figura 42.

Figura 42. Exemple de pàgina d'excedents



Emmagatzematge i cost de localització d'una entrada de l'índex

Els índexs basats en la dispersió s'emmagatzemen en un tipus d'espai virtual anomenat *espai d'índexs* (igual que els índexs d'arbre B^+).

És fàcil adonar-se que el cost de localitzar una entrada en l'índex emmagatzemat en les pàgines d'un espai d'índexs varia força segons si és excedent o no:

- Si una entrada no és excedent i, per tant, es troba en la seva pàgina primària, només caldrà fer una E/S per tal d'obtenir-la.
- Per a les entrades excedents, en canvi, sempre caldrà fer més d'una E/S.

En conseqüència, per a aconseguir un bon rendiment de l'índex interessa que hi hagi poques entrades excedents. Una manera de reduir el nombre d'entrades excedents és tenir més espai del que cal en les pàgines primàries; és a dir, aconseguir que les pàgines primàries estiguin poc carregades d'entrades.

S'anomena **factor de càrrega** el nombre d'entrades que s'espera tenir dividit pel nombre d'entrades que caben en les pàgines primàries:

$$C = M / (N \text{ x } L)$$

on M representa el nombre d'entrades que esperem tenir i N x L és el nombre d'entrades que caben en les pàgines primàries.

Com més baix sigui el factor de càrrega, menys excedents hi haurà i menys E/S caldran. En contrapartida, si el factor de càrrega és molt baix, es gasta més espai.

Exemple de càlcul de *L*

Si disposem de pàgines de 4 kB, els valors indexats ocupen 40 bytes, els RID, 4 i els apuntadors a pàgines, 3. Aleshores, el nombre L d'entrades serà de 93, atès que s'ha de complir que 4 kB = $(40 + 4)L + 3$ (la pàgina ha de contenir fins a L entrades formades pel valor i un RID cadascuna, i també l'apuntador a la primera pàgina d'excedents).

Introducció a la dispersió dinàmica

La dispersió que acabem d'explicar és una dispersió estàtica, en el sentit que el nombre de pàgines primàries N és fix. Un problema que té aquest tipus de dispersió és que, si el nombre de dades indexades creix més del previst, pot passar que el factor de càrrega C augmenti excessivament i el rendiment de l'índex empitjori.

Una solució és crear un nou índex amb un nombre N' de pàgines primàries més gran que N, canviar de funció de dispersió per tal que retorni valors de $[0, ..., N' - 1]$ i no pas valors de $[0, ..., N - 1]$ i reinserir totes les entrades en el nou índex emprant la nova funció de dispersió. Però aquesta solució és molt costosa.

La dispersió dinàmica té l'objectiu d'admetre l'increment dinàmic del nombre de pàgines primàries sense que calgui la reinserció de totes les entrades.

Algunes tècniques de dispersió dinàmica permeten modificar la funció de dispersió dinàmicament per tal d'acomodar-se a l'augment o la disminució de les dades indexades. En aquest mòdul, però, no descrivim les tècniques de dispersió dinàmica, dues de les quals, la dispersió extensible i la dispersió lineal, les podeu trobar explicades amb tots els detalls a l'obra de Ramakrishnan (2002).

Lectura complementària

Trobareu l'explicació detallada de la dispersió extensible i la lineal en l'obra següent:

Ramakrishnan, R.; Gehrke, J. (2002). *Database management systems*. Boston: McGraw-Hill.

6.3.5. Índexs agrupats

Els índexs que permeten implementar els accessos seqüencials per valor (com ara els arbres B^+) poden ser índexs agrupats[19] o índexs no agrupats[20].

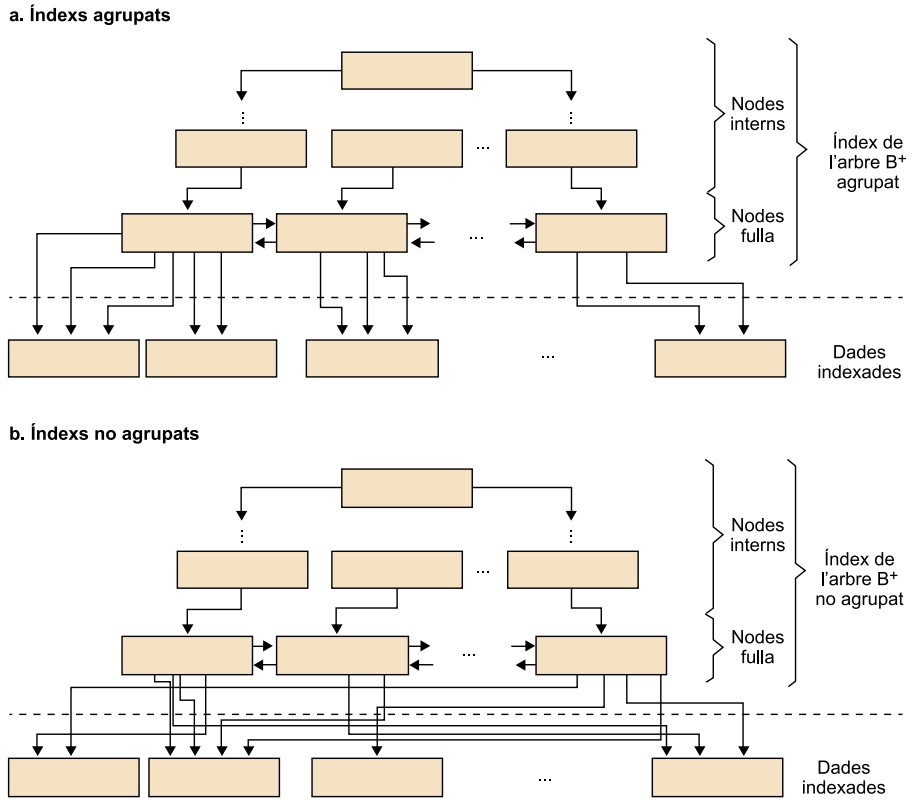
[19] En anglès, *clustered*.

[20] En anglès, *unclustered*.

Un **índex agrupat** és aquell en què les dades que indexa estan ordenades físicament segons l'accés seqüencial per valor que proporciona l'índex. En canvi, un **índex no agrupat** és un índex en què les dades indexades estan col·locades de manera aleatòria.

La figura 43 il·lustra les diferències entre els índexs agrupats i els no agrupats.

Figura 43. Diferències entre índexs agrupats i no agrupats



Reflexió

Tot i que podem tenir diversos índexs sobre unes dades, només un d'ells pot ser agrupat, perquè les dades poden tenir una única ordenació física.

El **cost d'un accés seqüencial per valor** varia molt segons si l'índex que ens proporciona l'accés és agrupat o no:

- Si l'índex és agrupat, els RID que s'obtinguin consecutivament apuntaran a files contigües. Llavors, caldrà fer molts accessos seguits a files de la mateixa pàgina (i es podran portar conjuntament amb una única E/S).

- En índexs no agrupats pot passar que la majoria de RID consecutius apuntin a files de pàgines diferents. Aquesta situació pot arribar a comportar la realització de tantes E/S com files a les quals calgui accedir.

Un aspecte que cal tenir en compte dels índexs agrupats és que l'ordenació física de les seves dades és difícil de mantenir quan es produeixen insercions i supressions. En la pràctica, aquesta dificultat es resol de la manera següent:

1) Inicialment, es deixa espai lliure en les pàgines que contenen les dades per tal de poder absorbir insercions futures.

2) Si l'espai lliure d'una pàgina s'esgota, les insercions futures a la pàgina es fan en pàgines d'excedents que s'hi encadenen.

3) Si hi ha moltes pàgines d'excedents, les dades es reorganitzen per a millorar el rendiment.

6.4. Implementació dels accessos per diversos valors

La implementació dels accessos per diversos valors és, en alguns casos, molt similar a la implementació dels accessos per un sol valor que ja hem estudiat en el subapartat anterior. En altres casos, però, presenten algunes dificultats addicionals. Explicarem en primer lloc la implementació dels accessos directes per diversos valors i, a continuació, analitzarem els accessos seqüencials i mixtos per diversos valors.

6.4.1. Implementació dels accessos directes

Considereu la relació *Employees*(*emplId*, *emplName*, *officeNum*, *salary*), que té un índex d'arbre B^+ definit per tal de facilitar els accessos per valor de l'atribut *officeNum*, i un altre índex d'arbre B^+ per a facilitar els accessos per valor de l'atribut *salary*.

Suposem que es vol executar la sentència següent, que requereix un accés directe per diversos valors, concretament pels valors de *salary* i d'*officeNum*:

```
SELECT *
FROM Employees
```

```
WHERE officeNum = 150 AND salary = 1200;
```

Una bona estratègia per a processar la consulta anterior és la que presentem tot seguit:

1) Fer servir l'índex de l'atribut *officeNum* per a obtenir tots els RID de les files d'empleats que tenen el despatx 150.

2) Fer servir l'índex de l'atribut *salary* per tal de trobar tots els RID de files d'empleats que tenen el sou 1.200.

3) Fer la intersecció entre els dos conjunts de RID. Els RID de la intersecció corresponen a empleats que tenen, alhora, el despatx 150 i un sou de 1.200.

Aquesta és una bona estratègia perquè aprofita el fet de tenir dos índexs per a les dades dels empleats. Tot i això, en alguns casos concrets pot tenir un mal rendiment.

Per exemple, si hi ha molts empleats al despatx 150, molts empleats amb un sou de 1.200 i molt pocs empleats que compleixin les dues condicions alhora, caldrà examinar molts RID per a localitzar pocs empleats. Una solució més eficient per a aquest cas és definir un únic índex, en lloc de dos com en el cas anterior, que utilitzi valors compostos dels atributs *officeNum* i *salary*.

Un **índex de valors compostos pels atributs** $[A_1, ..., A_i, ..., A_n]$ té la mateixa estructura que els altres índexs. L'única diferència és que els valors de l'índex seran, de fet, llistes d'elements $[v_1, ..., v_i, ..., v_n]$ on v_i és un valor de l'atribut A_i.

La gestió de l'índex fa necessari poder establir una relació d'ordre entre els valors compostos de l'índex. S'ordenen segons el primer element de la llista i, si el primer element coincideix, es fa d'acord amb el segon; si aquest també coincideix, s'ordena segons el tercer, etc.

Exemple d'índex de valors compostos

L'índex de valors compostos pels atributs [*officeNum*, *salary*] té valors com ara [150, 1.200], [150, 1.500], etc.

Per a establir una relació d'ordre entre els valors compostos de l'índex cal poder deduir per als dos valors anteriors [150, 1.200], [150, 1.500] quin és el més gran. Aquí, com que el primer element de la llista coincideix, s'utilitza el segon per a ordenar. S'obté que [150, 1.200] < [150, 1.500].

En general, l'ordre que hi ha entre dos valors compostos $v = [v_1, ..., v_i, ..., v_n]$ i $w = [w_1, ..., w_i, ..., w_n]$ s'estableix de la manera següent:

- Si $v_1 > w_1$, aleshores $v > w$; i si $v_1 < w_1$, aleshores $v < w$.

- Si $v_1 = w_1$ i $v_2 > w_2$, aleshores $v > w$; i si $v_1 = w_1$ i $v_2 < w_2$, aleshores $v < w$.

- Si $v_1 = w_1$, ..., $v_{i-1} = w_{i-1}$ i $v_i > w_i$, aleshores $v > w$; i si $v_1 = w_1$, ..., $v_{i-1} = w_{i-1}$ i $v_i < w_i$, aleshores $v < w$.

- Si $v_1 = w_1$, ..., $v_i = w_i$, ..., $v_n = w_n$, aleshores $v = w$.

6.4.2. Implementació dels accessos seqüencials i mixtos

Hem vist que els índexs de valors compostos pels atributs $[A_1, ..., A_i, ..., A_n]$ són una bona implementació dels accessos directes per diversos valors. En aquest subapartat comentarem la seva aplicació per tal d'implementar accessos seqüencials i mixtos per diversos valors i veurem que presenta algunes deficiències.

Considereu una altra vegada la relació *Employees*(*emplId*, *emplName*, *officeNum*, *salary*) i recordeu que té un índex d'arbre B^+ de valors compostos pels atributs [*officeNum*, *salary*].

Suposem que es volen executar les sentències següents:

- Sentència 1:

```
SELECT *
FROM Employees
ORDER BY officeNum, salary;
```

Aquesta sentència correspon a un accés seqüencial per diversos valors. L'ordre de presentació dels empleats que requereix aquesta sentència coincideix amb l'ordre dels valors compostos de l'índex per [*officeNum*, *salary*]. Així, doncs, l'índex ens permetrà processar la sentència de manera eficient.

- Sentència 2:

```
SELECT *
FROM Employees
WHERE officeNum = 100 AND salary > 960;
```

Aquesta sentència correspon a un accés mixt per diversos valors. Observeu que l'ordre dels valors compostos de l'índex per [*officeNum*, *salary*] concorda amb l'ordre en què ens interessa obtenir els empleats per a processar la consulta. Així, l'índex ens permet també processar la consulta de manera eficient.

Malauradament, no passarà el mateix amb altres exemples d'accessos seqüencials o mixtos per diversos valors, perquè no hi haurà concordança entre l'ordre de l'índex per [*officeNum*, *salary*] i l'ordre que requereixi la consulta. Aquest és el cas de les sentències següents:

- Sentència 3:

```
SELECT *
FROM Employees
ORDER BY salary, officeNum;
```

- Sentència 4:

```
SELECT *
FROM Employees
WHERE salary = 1200 AND officeNum > 100;
```

Per a les sentències 3 i 4 necessitaríem un índex [*salary*, *officeNum*] i no pas l'índex [*officeNum*, *salary*].

Hi ha altres tipus d'índexs, anomenats **índexs multidimensionals per diversos atributs [A_1, ..., A_i, ..., A_n]**, que no imposen un ordre lineal en el conjunt de valors indexats. El que fan és organitzar les dades segons una cert lligam espacial: cada valor es considera un punt d'un espai de dimensió *n*, on *n* és el nombre d'atributs de l'índex.

Utilitat dun índex multidimensional

Amb un sol índex multidimensional pels atributs *salary* i *officeNum* podríem processar totes les sentències de consulta 1, 2, 3 i 4 presentades en aquest subapartat.

S'han proposat diversos tipus d'índexs multidimensionals: els arbres R, els arxius en retícula, etc. Es fan servir sobretot en SGBD destinats a emmagatzemar informació geogràfica, però pocs sistemes relacionals els incorporen.

Lectura complementària

Podeu trobar la descripció dels arbres R i dels arxius en retícula a l'obra següent: **A. Silberschatz; H. F. Korth; S. Sudarshan** (2010). *Fundamentos de bases de datos* (3a ed.). Madrid: McGraw-Hill.

6.5. Índexs del sistema Oracle

El sistema Oracle posa a l'abast del dissenyador alguns dels índexs que hem estudiat en aquest mòdul.

Documentació Oracle

Per a obtenir els detalls de la implementació, així com una guia de la sintaxi i exemples d'ús, es pot consultar la documentació en línia de l'SGBD Oracle.

6.5.1. Accessos per valor

Per a implementar accessos per valor, directes o seqüencials, el sistema Oracle proporciona índexs d'arbre B^+ per un atribut, que poden ser agrupats o no.

Per exemple, si tenim una relació *Employees*(*emplId*, *emplName*, *officeNum*, *salary*), podem declarar un índex per a accedir per valor de l'atribut anomenat *officeNum* així:

```
CREATE INDEX indexOfficeNum ON Employees (officeNum);
```

Si es desitja que l'accés seqüencial sigui per ordre descendent, cal declarar-ho explícitament (perquè per defecte l'ordre es considera ascendent):

```
CREATE INDEX indexOfficeNum ON Employees (officeNum DESC);
```

Podem aconseguir que l'índex sigui agrupat si declarem:

```
CREATE INDEX CLUSTER indexOfficeNum ON Employees (officeNum);
```

Finalment, si volem que l'índex no admeti valors repetits, la sentència serà:

```
CREATE UNIQUE INDEX indexOfficeNum ON Employees (officeNum);
```

6.5.2. Accessos per diversos valors

Per a implementar accessos per diversos valors, el sistema Oracle proporciona índexs d'arbre B^+ de valors compostos, que també poden ser agrupats o no.

Per exemple, si en la relació *Employees*(*emplId*, *emplName*, *officeNum*, *salary*) volem declarar un índex de valors compostos pels atributs [*officeNum*, *salary*], farem:

```
CREATE INDEX indexOfficeSalary ON Employees (officeNum, salary);
```

Si volem que l'accés seqüencial per algun dels atributs sigui descendent, cal declarar-ho explícitament. Per exemple, si volem que l'ordenació dels sous sigui descendent:

```
CREATE INDEX indexOfficeSalary ON Employees (officeNum, salary DESC);
```

També podem aconseguir que l'índex sigui agrupat:

```
CREATE INDEX CLUSTER indexOfficeSalary ON Employees (officeNum, salary);
```

Finalment, si es desitja que l'índex no admeti valors repetits, caldrà declarar:

```
CREATE UNIQUE INDEX indexOfficeSalary ON Employees (officeNum, salary);
```

6.5.3. Consideracions del planificador d'execució per a l'ús dels índexs

Els índexs es poden crear per a qualsevol columna de qualsevol taula, però no sempre és beneficiós ni eficient, atès que cal mantenir-los actualitzats i això implica consum de recursos de l'SGBD. El mateix Oracle crea índexs automàticament en determinats casos, com per exemple per a les columnes que defineixen la clau primària (amb restricció *PRIMARY KEY*) o alternativa (amb restricció *UNIQUE*).

Per tal de construir consultes òptimes que facin servir el mínim de recursos, cal seguir algunes bones pràctiques. Una d'elles és disposar dels índexs adequats per a cada taula, ja que l'existència d'índexs inadequats o no utilitzats provoca ineficiències en el moment d'inserir, esborrar o modificar dades a/de les taules, especialment si hi ha diversos índexs i les taules són grans. Això és perquè, en el cas d'inserció o esborrat de dades en una taula, l'SGBD ha d'actualitzar tots els índexs existents sobre les columnes d'aquesta taula, mentre que, en el cas de modificació de dades, l'SGBD només ha d'actualitzar els índexs de la columna o les columnes implicada/es.

Així, com a regla general, les taules sobre les quals es preveu un elevat nombre d'insercions, esborrats o modificacions, es recomana que tinguin un número reduït d'índexs, mentre que les taules sobre les quals principalment es fan consultes (poques actualitzacions) no hi ha problema si tenen més índexs.

En el cas d'Oracle, si busquem el rendiment òptim de l'SGBD, hem de tenir en compte les comprovacions que fa el planificador d'execucions per determinar els índexs que convé crear:

- Si la consulta feta filtrant els valors de la columna amb índexs, recupera menys del 15% (aproximadament) de les files d'una taula, es recomanarà utilitzar l'índex.

- En les consultes sobre columnes amb valors únics o amb pocs duplicats, s'utilitzarà gairebé sempre l'índex existent.

- Els valors de tipus *null* existents a les columnes no s'indexen, però en el cas de consultes sobre la resta de valors d'una columna, també serà habitual utilitzar l'índex.

- Per al cas de consultes sobre taules petites, que caben en un bloc, no s'acostuma a fer servir cap índex, ja que s'optarà per dur tota la taula a memòria per fer un recorregut sobre ella.

- L'Oracle no crea índexs automàticament per a les columnes que són clau forana a altres taules, i com que s'utilitzaran per fer combinacions, és recomanable crear índexs per a aquestes columnes, ja que s'utilitzaran habitualment.

Si ens trobem amb els casos descrits, cal analitzar l'existència d'índex i, si fos necessari, crear-lo. Tenim els diferents tipus d'índexs per triar:

- Simples: són els creats per omissió per l'Oracle, per a una sola columna i de tipus ascendent.

- Compostos: són aquells en els quals intervé més d'una columna d'una taula. És important situar com a primera columna la que tingui més valors diferents. En determinats casos, també poden ser utilitzats per l'SGBD en fer-se cerques per a només una de les columnes de l'índex.

- Ascendents/descendents: un índex ascendent (creat per omissió) no pot ser utilitzat quan es demanen resultats ordenats de forma descendent. En aquests casos, s'haurà de crear un índex específic de tipus *DESC*.

- Calculats: són els creats utilitzant funcions, expressions aritmètiques, crides a C, funcions SQL, entre d'altres possibilitats. Tot seguit s'ofereixen un parell d'exemples:

```
CREATE INDEX IDX_UPP_Nom ON UPPER(nom);
```

o

```
CREATE INDEX IDX_result ON (saldo/numMoviments);
```

Vegem més recomanacions per a l'ús dels índexs.

En el cas dels índexs calculats, cal assegurar que les consultes faran servir la mateixa expressió que es va utilitzar en crear l'índex. Així ens assegurarem que l'Oracle trobarà un índex compatible per a la consulta. A més, no es poden utilitzar índex calculats que no siguin deterministes, es a dir, que no retornin sempre el mateix valor.

El índexs han de ser deterministes, el seu valor no pot variar en funció del moment en què es calculi. Per exemple, si tinguéssim una camp anomenat *dataNaixement*, no hauríem de crear un índex per al camp *edat* que es calculés en funció de la diferència entre l'edat actual i la data de naixement, ja que Oracle faria servir l'expressió del moment de la seva creació i al dia següent ja deixaria de ser correcte.

Un altre tema a considerar és si pot ser millor utilitzar un índex tipus arbre B+ o tipus *bitmap* (aquests últims no disponibles a l'Oracle Express Edition). Podem adoptar aquesta norma: si existeixen molts valors diferents en una columna, el tipus d'índex més adequat és l'arbre B+, mentre que, si existeixen pocs valors diferents en una columna, el més recomanable és fer servir l'índex de tipus *bitmap*.

El moment de crear els índexs també és important ja que, per exemple i de forma general, és millor crear els índexs d'una taula després de fer els processos de càrrega de dades (SQL Loader), modificacions massives o similars, on el contingut dels camps i fins i tot la quantitat de registres variï substancialment, que no tenir-los creats abans i provocar l'actualització continuada durant aquests processos.

Finalment, algunes restriccions a tenir en compte en utilitzar/crear índexs:

- No es poden fer servir índexs sobre columnes de tipus *LONG* o *LONG RAW*.

- La mida de l'entrada d'un índex no pot excedir (aproximadament) la meitat de l'espai d'un bloc, per la qual cosa no es podran crear índexs sobre columnes amb valors molt grans.

En els plans d'execució de les consultes és on es mostra la utilització dels índexs. Així doncs, la manera més senzilla de comprovar quin és el pla d'execució previst per a una consulta en Oracle, per exemple, és prement F10 en l'SQL Developer. Tot seguit es mostra un pla d'execució de consulta on es veu que l'optimitzador d'Oracle, per accedir a les dades, farà servir l'índex *IDX_cityName*.

Figura 44. Pla d'execució que mostra la recomanació de fer servir l'índex anomenat *IDX_cityName* donada una consulta

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	2
TABLE ACCESS	CITY	BY INDEX ROWID	1	2
INDEX	IDX_CITYNAME	RANGE SCAN	1	1
Access Predicates				
CITYNAME='Brasilia'				

6.6. Claus sintètiques en el model físic

En finalitzar el procés de disseny d'una base de dades cal assegurar-se que el resultat obtingut és òptim tant pel que fa a estructura lògica (la definició de les columnes és coherent amb el tipus de dades que emmagatzemaran) com de recursos utilitzats (mínim de recursos utilitzats). A més, també és important assegurar que el disseny desenvolupat serà fàcil de mantenir. Aquestes comprovacions impliquen fer algunes reflexions que poden comportar realitzar adaptacions i refinaments del model físic obtingut.

Un dels temes que s'acostuma a revisar són els valors que contenen les claus primàries com a identificadors unívocs de les dades.

Per definició, una clau primària ha de contenir un valor que sigui únic en totes les files de la taula, ja que ha de sevir per a identificar de forma unívoca cadascuna d'elles i, idealment, aquest valor ha de ser constant al llarg del temps. No obstant, ens trobem que en el món quotidià poques dades hi ha que siguin identificadores, inamovibles i/o immutables, com per a ser considerades bones candidates a clau primària.

6.6.1. Necessitat d'ús de claus sintètiques

Hi ha casos en què l'ús de claus primàries pot comportar ineficiències que es poden solucionar amb claus sintètiques.

Suposem que volem identificar un vehicle:

- Si triem la matrícula per identificar-lo, pot donar-se el cas que el vehicle es torni a matricular, per exemple, passant de matrícula no europea a matrícula europea, o en el cas d'una operació de compra/venda, quan es canvia de país.

- Si escollim el número de bastidor del vehicle, pot passar que el vehicle pateixi un accident d'importància i se li hagi de canviar el xassís (on porta estampat el número de bastidor).

- Podríem pensar a fer servir la combinació de número de matrícula + número de bastidor...

O potser ens interessa identificar una persona:

- Si prenem el DNI com a identificador, tenim que els menors d'edat sense DNI no es podran identificar. També ens trobaríem que les persones d'un altre país no tenen DNI (tenen número de passaport) i que, en tot cas, una persona que és immigrant i es nacionalitza pot tenir primer NIE (número d'identitat d'estranger) i posteriorment, en obtenir la nacionalitat, li poden canviar aquest número per un altre corresponent al DNI.

- Es podria pensar a utilitzar la combinació de nom + cognoms + data de naixement. Tot i així, en els casos de noms molt comuns, pot passar que aquests valors coincideixin amb els d'altres persones i, per tant, que no siguin únics.

Problemes per homonímia

La coincidència de noms i cognoms s'anomena *homonímia*, i pot provocar problemes molt importants, ja siguin legals, mèdics...

En els casos d'exemple, i si es produïssin les situacions descrites, caldria canviar els valors triats com a clau primària, amb la dificultat que això pot comportar en un sistema complex, on poden haver desenes de taules relacionades. Canviar el valor d'una clau primària per variar l'identificador d'una persona,

per exemple, pot significar haver d'actualitzar desenes de taules, ja que el valor modificat com a clau primària ha de constar com a clau forana en les taules relacionades. Una manera de no trobar-se amb aquesta problemàtica consisteix a fer servir el que s'anomena *clau sintètica*.

Una clau primària sintètica no és més que un camp declarat com a clau primària, que conté un valor no repetit que es genera de forma seqüencial. Per exemple, si considerem la taula *Persona* i tenim una clau sintètica, aquesta prendrà valors consecutius des de '1', per a la primera fila que s'insereixi amb dades de persona, '2' per a la segona i així successivament.

Normalment, per a crear una clau sintètica es fa servir un disparador (*trigger*) i una seqüència definida a la base de dades. Amb cada nova inserció, el disparador recuperarà de la seqüència un nou valor i l'utilitzarà per omplir la columna, independentment del possible valor inicial que s'hagi donat en la instrucció d'inserció.

No obstant, una vegada determinada la necessitat d'utilitzar un valor sintètic com a clau primària, cal triar quin/s altre/s valor/s poden servir per identificar unívocament cada registre, ja que habitualment no es faran les consultes per clau sintètica, sinó per alguna altra clau que aparentment sembli més lògica, si més no pel propi nom i significat.

Per exemple, tornant al cas de la taula *Persona* considerat anteriorment, si definíssim una clau primària sintètica caldria definir també una clau alternativa amb la clàusula *UNIQUE*, com ara el NIF. La primera clau permetria relacionar les files d'aquesta taula amb les d'altres taules, i la segona serviria per realitzar cerques dins la taula *Persona* i per evitar repeticions de valors.

Els avantatges d'utilitzar aquesta tècnica són diversos:

- Les files de les taules s'identifiquen unívocament des de la seva inserció fins al seu esborrat amb un valor identificatiu únic i immutable.

- És possible disposar d'una clau alternativa, que també permet identificar cada fila (si més no, evitar duplicitats) el valor de la qual és fàcilment actualitzable, atès que només consta en una taula.

- L'ocupació d'espai als índexs de la base de dades per l'atribut que és clau primària, i que serà el que més s'utilitzarà en fer consultes, serà molt compacte, ja que amb tota probabilitat serà un camp de tipus *INTEGER*. Si féssim servir els altres identificadors descrits, com ara el NIF, la matrícula del vehicle o l'adreça, el nombre de bytes que ocuparia la clau primària seria un ordre de magnitud superior.

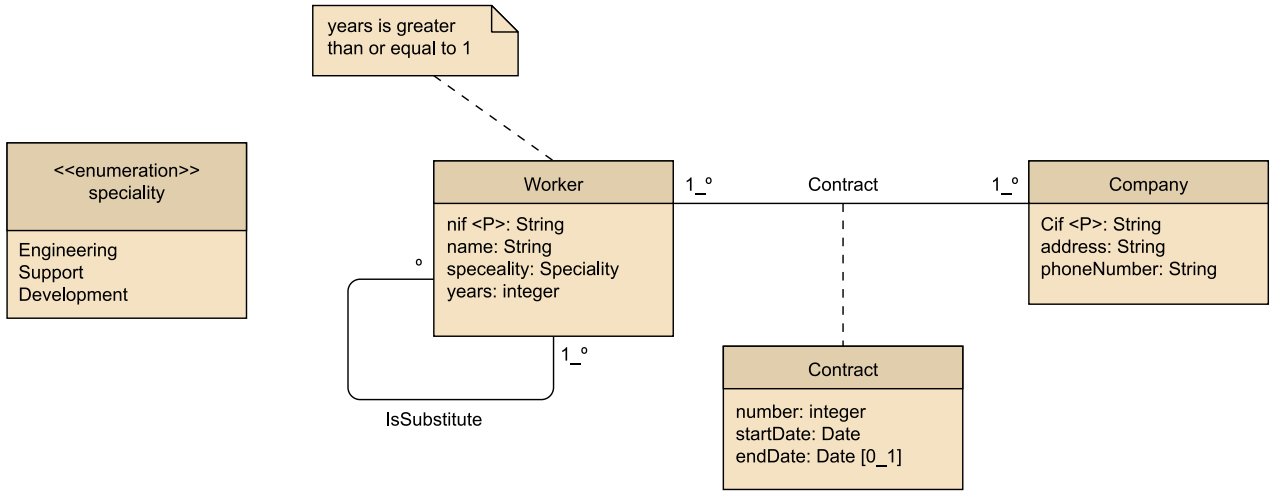
La determinació d'ús d'una clau primària sintètica es fa en l'última etapa de disseny, en el pas de transformació del disseny lògic relacional a físic, i de forma general, s'utilitzarà sempre una clau primària sintètica per a les taules

que identifiquen objectes del món físic, utilitzant-se com a clau alternativa els atributs que en l'etapa del disseny conceptual s'havien detectat com a claus primàries.

6.6.2. Cas pràctic

Ens diuen que tenim el següent model conceptual:

Figura 45. Exemple de model conceptual per a modelar la contractació de treballadors en una empresa



I la seva transformació a model lògic relacional és:

Company (cif, **address, phoneNumber**)

Worker (nif, **name, specialty, years**)

Contract (cifCompany, nifWorker, **number, startDate,** endDate)
{cifCompany} is foreign key to Company
{nifWorker} is foreign key to Worker

IsSubstitute (nifWorker, nifSubstitute)
{nifWorker} is foreign key to Worker
{nifSubstitute} is foreign key to Worker

Essent el codi SQL per a la transformació a físic en l'SGBD Oracle, el següent:

```
CREATE TABLE Company (
  cif VARCHAR2(10 CHAR) CONSTRAINT PK_Company PRIMARY KEY,
  address VARCHAR2(100 CHAR) CONSTRAINT NN_Address NOT NULL,
  phoneNumber VARCHAR2(50 CHAR) CONSTRAINT NN_PhoneNumber NOT NULL
);
```

```
CREATE TABLE Worker (
```

```
  nif VARCHAR2(10 CHAR) CONSTRAINT PK_Worker PRIMARY KEY,
  name VARCHAR2(100 CHAR) CONSTRAINT NN_WorkerName NOT NULL,
  specialty VARCHAR2(11 CHAR) CONSTRAINT NN_WorkerSpecialty NOT NULL
  CONSTRAINT CH_WorkerSpecialty CHECK (specialty = 'ENGINEERING' OR
              specialty = 'SUPPORT' OR
              specialty = 'DEVELOPMENT'),
  years INTEGER CONSTRAINT NN_WorkerYears NOT NULL
     CONSTRAINT CH_WorkerYears CHECK (years >= 1)
);

CREATE TABLE Contract (
  cifCompany VARCHAR2(10 CHAR),
  nifWorker VARCHAR2(10 CHAR),
  "number" INTEGER CONSTRAINT NN_ContractNumber NOT NULL,
  startDate DATE CONSTRAINT NN_ContractStartDate NOT NULL,
  endDate DATE,
  CONSTRAINT PK_Contract PRIMARY KEY (cifCompany, nifWorker),
  CONSTRAINT FK_CompCont FOREIGN KEY (cifCompany) REFERENCES Company (cif),
  CONSTRAINT FK_WorkerCont FOREIGN KEY (nifWorker) REFERENCES Worker (nif),
  CONSTRAINT CH_ContractDates CHECK (endDate IS NULL OR endDate >= startDate)
);

CREATE TABLE IsSubstitute (
  nifWorker VARCHAR2(10 CHAR),
  nifSubstitute VARCHAR2(10 CHAR),
  CONSTRAINT PK_Substitutes PRIMARY KEY (nifWorker, nifSubstitute),
  CONSTRAINT FK_SubstWorker1 FOREIGN KEY (nifWorker) REFERENCES Worker (nif),
  CONSTRAINT FK_SubstWorker2 FOREIGN KEY (nifSubstitute) REFERENCES Worker (nif),
  CONSTRAINT CH_Substitutes CHECK (nifWorker <> nifSubstitute)
);
```

S'observa que tant la taula *Worker* com *Company* són susceptibles de fer servir una clau sintètica. Ens centrarem en *Worker*.

Tal com s'ha transformat el model lògic relacional a disseny físic, ens trobem que, si un treballador varia el seu NIF, perquè abans de nacionalitzar-se té un NIE (número d'identitat d'estranger) i després un DNI, la modificació hauria de fer-se a les taules *Worker*, *Contract* i *isSubstitute* pel fet d'estar relacionades. Per tant, cal crear una transacció, amb la feina que això comporta, per assegurar que la base de dades quedi íntegra després dels canvis.

Si volguéssim introduir una clau sintètica a la taula *Worker* en el moment de fer la transformació del model lògic relacional a model físic, ho faríem així:

```
CREATE TABLE Company (
  cif VARCHAR2(10 CHAR) CONSTRAINT PK_Company PRIMARY KEY,
  address VARCHAR2(100 CHAR) CONSTRAINT NN_Address NOT NULL,
```

```
  phoneNumber VARCHAR2(50 CHAR) CONSTRAINT NN_PhoneNumber NOT NULL
);

CREATE TABLE Worker (
  idWorker INTEGER CONSTRAINT PK_Worker PRIMARY KEY,
  nif VARCHAR2(10 CHAR) CONSTRAINT AK_Worker UNIQUE,
  name VARCHAR2(100 CHAR) CONSTRAINT NN_WorkerName NOT NULL,
  specialty VARCHAR2(11 CHAR) CONSTRAINT NN_WorkerSpecialty NOT NULL
CONSTRAINT CH_WorkerSpecialty CHECK (specialty = 'ENGINEERING' OR
           specialty = 'SUPPORT' OR
           specialty = 'DEVELOPMENT'),
  years INTEGER CONSTRAINT NN_WorkerYears NOT NULL
     CONSTRAINT CH_WorkerYears CHECK (years >= 1)
);
```

```
CREATE TABLE Contract (
  cifCompany VARCHAR2(10 CHAR),
  idWorker INTEGER,
  "number" INTEGER CONSTRAINT NN_ContractNumber NOT NULL,
  startDate DATE CONSTRAINT NN_ContractStartDate NOT NULL,
  endDate DATE,
  CONSTRAINT PK_Contract PRIMARY KEY (cifCompany, idWorker),
  CONSTRAINT FK_CompCont FOREIGN KEY (cifCompany) REFERENCES Company (cif),
  CONSTRAINT FK_WorkerCont FOREIGN KEY (idWorker) REFERENCES Worker (idWorker),
  CONSTRAINT CH_ContractDates CHECK (endDate IS NULL OR endDate >= startDate)
);

CREATE TABLE IsSubstitute (
  idWorker INTEGER,
  idSubstitute INTEGER,
  CONSTRAINT PK_Substitutes PRIMARY KEY (idWorker, idSubstitute),
  CONSTRAINT FK_SubstWorker1 FOREIGN KEY (idWorker) REFERENCES Worker (idWorker),
  CONSTRAINT FK_SubstWorker2 FOREIGN KEY (idSubstitute) REFERENCES Worker (idWorker),
  CONSTRAINT CH_Substitutes CHECK (idWorker <> idSubstitute)
);

-- Creació de seqüencia per a obtenir el valor en curs del treballador a inserir
CREATE SEQUENCE s_Worker
  INCREMENT BY 1
  START WITH 1;

-- Creació del disparador que inserirà automàticament el valor a idWorker
CREATE OR REPLACE TRIGGER t_sintetic_idWorker
  BEFORE INSERT ON Worker
  FOR EACH ROW
BEGIN
  SELECT s_Worker.NEXTVAL INTO :NEW.idWorker
```

```
    FROM DUAL;
END t_sintetic_idWorker;
```

Després, comprovaríem que:

- Les consultes que involucrin diferents taules serien més eficients, atès que els índexs de les claus primàries ocuparien menys espai.

- Si es vol variar el NIF de *Worker* es podria fer amb úna única operació de modificació (*UPDATE*), de forma molt ràpida i senzilla.

- Es possible realitzar cerques de persones per NIF atès que aquest camp és clau alternativa.

- Cal fer notar que en fer insercions a la taula *Worker* no haurem d'inicialitzar el valor del camp *idWorker*, ja que el disparador generarà automàticament el valor per informar-lo d'acord amb una seqüència donada.

Imaginem que volem inserir les dades d'un treballador amb aquesta sentència SQL:

```
INSERT INTO Worker (nif, name, specialty, years, cifCompany)
  VALUES ('39893123U', 'John Smith', 'ENGINEERING', 7, ' G12343212'');
```

Observeu que no es dona cap valor explícit a l'*idWorker*. Aquest s'inicialitzarà en el moment de la inserció a través del disparador corresponent.

Resum

En la primera part d'aquest mòdul didàctic hem presentat els components que fan servir els SGBD en l'emmagatzematge de les dades que gestionen. Hem vist el nivell físic de l'arquitectura d'una base de dades, en el qual els SGBD usen fitxers, i el nivell virtual, que és un nivell propi dels SGBD que permet simplificar la visió que tenen de les dades emmagatzemades.

En la segona part d'aquest mòdul hem tractat la forma d'implementar el disseny físic de la base de dades.

Finalment, en la tercera i última part d'aquest mòdul didàctic hem identificat mètodes d'accés que calen per a poder fer diferents tipus de consultes i actualitzacions en les BD. Aquests mètodes d'accés són els accessos directes i seqüencials per posició, els accessos directes i seqüencials per valor i, finalment, els accessos per diversos valors, que poden ser directes, seqüencials o mixtos.

Hem explicat que la implementació dels accessos per posició es fonamenta, gairebé completament, en els serveis proporcionats per l'SO i que, en canvi, la implementació eficient dels accessos per un o diversos valors és més complexa i requereix que l'SGBD disposi d'estructures pròpies especialitzades.

Hem descrit algunes de les estructures que els SGBD fan servir per a implementar els accessos per un o diversos valors: índexs del tipus arbre B^+, índexs estructurats segons funcions de dispersió, índexs agrupats i índexs de valors compostos. Hem mostrat l'impacte que poden tenir aquestes estructures en el nombre d'E/S necessari per a implementar els accessos a les dades i hem vist com l'optimitzador de l'SGBD pot ajudar a construir consultes que optimitzin els recursos.

Glossari

accés directe per posició *m* Mètode d'accés que consisteix a obtenir una pàgina que té un número de pàgina determinat dins d'un espai.

accés directe per valor *m* Mètode d'accés que consisteix a obtenir totes les files que contenen un determinat valor per un atribut.

accés per diversos valors *m* Mètode d'accés que consisteix a obtenir diverses files segons els valors de diversos atributs. Pot ser directe, seqüencial o mixt.

accés seqüencial per posició *m* Mètode d'accés que consisteix a anar obtenint les pàgines d'un espai seguint l'ordre definit pels seus números de pàgina.

accés seqüencial per valor *m* Mètode d'accés que consisteix a obtenir diverses files per ordre dels valors d'un atribut.

arbre B^+ *m* Estructura de dades que s'empra per a 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 classifiquem i descrivim cada component dels SGBD, especialment aquells que estan relacionats amb l'emmagatzematge de les dades.

bloc *m* Unitat de transferència de dades entre la memòria de l'ordinador i els dispositius o fitxers externs. El sistema operatiu de la màquina, concretament la part especialitzada en l'E/S, és qui du a terme aquesta transferència.

catàleg *m* Conjunt de taules que contenen les metadades de la base de dades.

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

entrada *f* Element d'un índex que consisteix en una parella formada per un valor i un RID.

espai virtual *m* Seqüència de pàgines virtuals. Proporciona una visió ordenada i contigua de les pàgines físiques. Segons la seva funcionalitat específica té característiques lleugerament diferents, en funció de les quals pot ser un espai per a taules, fragmentat, d'agrupació, d'objectes grans, d'índexs, temporal, etc.
sigla **EV**

EV *m* Vegeu **espai virtual**

extensió *f* Unitat d'assignació d'espai en un dispositiu perifèric. Cada extensió és un nombre enter de pàgines consecutives i està continguda dins d'un fitxer. Normalment hi ha una extensió primària, que s'adquireix la primera vegada que el fitxer s'estén, i una extensió secundària, que correspon a les extensions següents.

fitxer *m* Unitat de gestió de l'espai en els dispositius perifèrics. Normalment el sistema operatiu gestiona els fitxers en lloc de l'SGBD.

identificador de fila *m* Adreça uniforme que fan servir els SGBD per a enregistrar les referències internes de les seves estructures de dades. Altres noms equivalents són ROWID i, darrerament, OID (en la mesura que els SGBD relacionals es converteixen en els anomenats *object-relational*).
sigla **RID**

índex *m* Estructura de dades auxiliar que els SGBD fan servir per a facilitar les cerques necessàries per tal d'implementar els accessos per un o diversos valors.

índex agrupat *m* Índex que proporciona l'accés seqüencial per valor (i també l'accés directe per valor) i que indexa dades que estan ordenades físicament segons l'ordre de l'accés seqüencial per valor proporcionat.

índex de valors compostos *m* Índex de valors compostos pels atributs $[A_1, ..., A_i, ..., A_n]$. Té la mateixa estructura que els altres índexs, però amb la diferència que els valors de l'índex són, de fet, llistes de valors $[v_1, ..., v_i, ..., v_n]$ en les quals cada v_i és un valor de l'atribut A_i.

memòria intermèdia *f* Espai de la memòria principal de l'ordinador, normalment força gran, dedicat a contenir totes les unitats de memòria intermèdia que gestiona l'SGBD.

mètode d'accés *m* Tipus d'accés a les dades emmagatzemades per un SGBD.

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 (base de dades, taules, vistes, restriccions, etc.) dins de l'arquitectura de components d'emmagatzematge.

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

pàgina *f* Unitat de transferència de dades entre la memòria de l'ordinador i els fitxers d'una BD. L'SO de la màquina és qui du a terme aquesta transferència i passa la pàgina a l'SGBD perquè aquest en gestioni la informació que conté, ja que l'SGBD és qui entén l'estructura interior de la pàgina. La pàgina també és la unitat principal d'enregistrament de les dades d'una base de dades en els dispositius perifèrics. En aquest mòdul de vegades l'anomenem *pàgina física* o *pàgina real* per tal de distingir-la de la pàgina virtual.

pàgina virtual *f* Imatge de la pàgina real o visió que des del nivell virtual es té de la pàgina real sense materialitzar-la físicament. Hi ha una relació biunívoca entre pàgina real i pàgina virtual.

RID *m* Vegeu **identificador de fila**

SGBD *m* Vegeu **sistema de gestió de bases de dades**

SO *m* Vegeu **sistema operatiu**

vector d'adreces de fila *m* Estructura que ocupa les posicions més altes dins d'una pàgina. És un vector amb tants elements com files hi ha a la pàgina. Cada element apunta a una d'aquestes files. L'últim element apunta a la primera fila, el penúltim a la segona, i així successivament.
sigla **VAF**

VAF *m* Vegeu **vector d'adreces de fila**

Bibliografia

Elmasri, Ramez; Navathe, Shamkant, B. (2007). *Fundamentos de sistemas de bases de datos* (5a ed.). Madrid: Pearson Educación.

Hansen, G. W.; Hansen, J. V. (1997). *Diseño y administración de bases de datos* (2a ed.). Prentice Hall.

Ramakrishnan, Raghu; Gehrke, Johannes (2003). *Database management systems* (3a ed.). Boston: McGraw-Hill Higher Education.

Silberschatz, A.; Korth, H. F.; Sudarshan, S. (2006). *Fundamentos de bases de datos* (5a ed.). Madrid: McGraw-Hill. Edició de 2011 en eBook.

Teorey, T. J. (2011). *Database Modeling & Design* (5a ed.). San Francisco: Morgan Kaufmann Publishers, Inc.