Baze de date - Fundamente

O carte a comunităţii pentru comunitate

Neeraj Sharma, Liviu Perniu, Raul F. Chong, Abhishek Iyer, Adi-Cristina Mitea,
Chaitali Nandan, Mallarswami Nonvinkere, Mirela Danubianu

Ediţie tradusă în limba română de către

conf. dr. ing. Liviu Perniu

Prima editie

Prima ediţie (November 2010)

IBM Canada
8200 Warden Avenue
Markham, ON
L6G 1C7
Canada

Această ediţie face referire la produsul software IBM® DB2® Express-C Version 9.7 for Linux®, UNIX® and Windows®.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
3-2-12, Roppongi, Minato-ku, Tokyo 106-8711

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

The licensed program described in this document and all licensed material available for it are

provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Observaţii

Aceste informaţii se referă la produse şi servicii oferite în Statele Unite ale Americii.

S-ar putea ca IBM să nu ofere produsele, serviciile sau caracteristicile prezentate în acest document în alte ţări. Consultaţi reprezentanţa locală IBM pentru informaţii referitoare la produsele şi serviciile disponibile la această oră în zona dumneavoastră. Prin referire la un produs, program, sau serviciu

IBM nu înseamnă neapărat faptul că doar acel produs, program sau serviciu poate fi folosit. Se poate folosi orice alt produs funcţional echivalent, program sau serviciu care nu încalcă nici unul dintre drepturile de proprietate intelectuală IBM. Rămâne în responsabilitatea utilizatorului să evalueze şi să verifice operarea cu orice alt produs, program sau serviciu ce nu este în proprietatea IBM.

IBM poate avea patente sau cereri de patente care să acopere subiectele discutate în cadrul acestui document. Prin furnizarea acestui document nu vi se oferă licenţe în ceea ce priveşte patentele. Puteţi trimite cereri de licenţiere, în scris, la adresa:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Pentru cereri de licenţiere referitor la setul de caractere pentru reprezentare pe doi bytes (DBCS), contactaţi departamentul de proprietate intelectuală de la IBM (IBM Intellectual Property Department) din ţara în care locuiţi sau trimiteţi cererile în scris la adresa:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
3-2-12, Roppongi, Minato-ku, Tokyo 106-8711

Următorul paragraf nu se aplică în Marea Britanie sau în orice altă ţară în care aceste cereri contravin legii locale: COMPANIA INTERNATIONAL BUSINESS MACHINES OFERĂ ACEASTĂ PUBLICAŢIE “CA ATARE” FĂRĂ A OFERI NICI UN FEL DE GARANŢII, NICI IMPLICITE ŞI NICI EXPLICITE, INCLUZÂND, DAR FĂRĂ A SE LIMITA LA, GARANŢIILE IMPLICITE DE NEÎNCĂLCARE, COMERCIALIZARE SAU UTILIZARE ÎN ALTE SCOPURI. Anumite state nu permit acordarea de garanţii nici implicite şi nici explicite la efectuarea unor tranzacţii, motiv pentru care această formulare s-ar putea să nu se aplice în situaţia respectivă.

Informaţiile prezente în acest document s-ar puteas să conţină imprecizii tehnice sau erori de dactilografiere. Din acest motiv aceste informaţii s-ar putea modifica periodic; astfel de modificări vor apare în ediţiile următoare ale acestei publicaţii. IBM poate aduce îmbunătăţiri şi/sau face modificări în cadrul produsului (produselor) şi/sau programului (programelor) ce apar în această publicaţie în orice moment fără a face notificări prelabile.

Orice referire făcută în materialul de faţă la informaţii ce provin de la site-uri non-IBM este introdusă doar din motive de conformitate şi nu are, sub nici o formă, aprobarea acestora. Materialele provenite din cadrul acelor site-uri nu fac parte din materialele acestui produs IBM, iar utilizarea site-urilor respective se va face pe propria răspundere.

IBM poate utiliza sau distribui oricare dintre informaţiile furnizate aşa cum crede de cuvinţă fără a necesita vreo obligaţie din partea dumneavoastră.

Programele licenţiate prezentate în acest document, precum şi toate materialele asociate acestora sunt furnizate de către IBM pe baza termenilor din acordul IBM Customer Agreement, IBM International Program License Agreement sau orice alt acord echivalent existent între părţi.

Toate datele performante prezentate aici au fost determinate într-un mediu controlat. De aceea, rezultatele obţinute în alte medii de operare s-ar putea să difere în mod semnificativ. Anumite măsurători s-au făcut pe medii de dezvoltare, motiv pentru care este posibil să nu existe nici o garanţie a faptului că aceste măsurători vor fi identice pe medii obişnuite. Mai mult decât atât, anumite măsurători s-au făcut prin extrapolare, iar rezultatele obţinute s-ar putea să difere. Utilizatorii acestui document trebuie să-şi verifice datele folosite în propriile medii.

Informaţiile despre produsele non-IBM au fost obţinute de la furnizorii acestor produse, anunţurile făcute de către aceştia, sau din alte surse disponibile. IBM nu a testat aceste produse şi nu poate confirma acurateţea performanţelor, compatibilitatea sau alte probleme ale produselor non-IBM. Eventuale întrebări referitoare la produsele non-IBM trebuie adresate furnizorilor acestora.

Toate afirmaţiile referitoare la direcţiile viitoare sau tendinţele IBM se pot modifica sau se pot retrage, fără notificare, acestea nereprezentând altceva decât obiective sau scopuri propuse.

O serie de informaţii conţin exemple de date şi rapoarte folosite în activitatea de zi cu zi. Pentru a le prezenta cât mai complet posibil, exemplele conţin nume de persoane, companii, branduri şi produse. Toate acestea sunt fictive şi orice asemănare cu nume sau adrese din realitate este pur întâmplătoare.

Referirile făcute în această publicaţie la produse sau servicii IBM nu înseamnă automat şi faptul că IBM intenţionează să le promoveze în toate ţările în care operează.

Dacă citiţi acest document în format electronic este posibil să nu vedeţi fotografiile sau ilustraţiile color.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Mărci

IBM, logo-ul IBM şi site-ul ibm.com sunt mărci sau mărci înregistrate de compania International Business Machines Corp., înregistrate în multe jurisdicţii din întreaga lume. Alte nume de produse şi servicii pot fi mărci ce aparţin IBM sau altor companii. Lista curentă a mărcilor IBM o găsiţi pe Web, sub denumirea "Copyright and trademark information" la adresa www.ibm.com/legal/copytrade.shtml.

Java şi toate mărcile bazate pe Java aparţin Sun Microsystems, Inc. din Statele Unite ale Americii, din alte ţări, sau ambele.

Microsoft şi Windows sunt mărci ale Microsoft Corporation din Statele Unite ale Americii, din alte ţări, sau ambele.

Linux este marcă înregistrată a Linus Torvalds din Statele Unite ale Americii, din alte ţări, sau ambele.

UNIX este marcă înregistrată a The Open Group din Statele Unite ale Americii, din alte ţări, sau ambele.

Alte companii, nume de produse sau servicii pot fi mărci sau servicii ce aparţin unor terţi.

Cuprins

Prefaţa

Păstrarea actualizată a competenţelor reprezintă o provocare tot mai mare în lumea de astăzi, deoarece pe zi ce trece apar din ce în ce multe tehnologii noi, iar timpul avut la dispoziţie pentru a le stăpâni pe toate este din ce în ce mai scurt. Din acest motiv, seria de cărţi DB2® on Campus a fost realizată tocmai în scopul reducerii timpului şi eforturilor necesare în vederea familiarizării dumneavoastră cu cât mai multe dintre aceste noi tehnologii.

Cartea de faţă îşi propune să sprijine profesioniştii nou veniţi în domeniul bazelor de date astfel încât aceştia să înţeleagă conceptele fundamentale ale bazelor de date in contextul amplorii şi profunzimii informaţiilor.

Cui se adresează această carte?

Această carte se adresează entuziaştilor din domeniul bazelor de date, programatorilor, administratorilor de baze de date, dar şi tuturor celor care au un interes în această direcţie, cum ar fi studenţii sau absolvenţii instituţiilor de învăţământ superior.

Cum este structurată acestă carte?

Cartea este structurată pe capitole şi porneşte de la conceptele fundamentale ale bazelor de date, dar şi a modelelor de informaţii care sunt prezentate în capitolul I. În capitolul al II-lea se discută despre modelele relaţionale de date, iar capitolele al III-lea şi al IV-lea explică modelarea conceptuală şi proiectarea bazelor de date relaţionale. În capitolele al V-lea, al VI-lea şi al VII-lea se pune accentul pe limbajul SQL, iar capitolul al VIII-lea abordează aspecte referitoare la stocarea şi extragerea datelor în format XML folosind limbajele SQL şi XQuery. În capitolul al IX-lea se prezintă o serie de aspecte legate de securitatea bazelor de date. Cartea se încheie prin prezentarea altor tehnologii importante şi aplicaţii relevante care prezintă un interes din ce în ce mai mare în industria actuală.

În cadrul celor mai multe dintre capitole veţi avea şi o parte de exerciţii şi întrebări recapitulative ale căror răspunsuri le puteţi regăsi în anexa A.

O carte pentru comunitate

Această carte a apărut având la început un nucleu de cadre didactice universitare, studenţi şi profesionişti (inclusiv angajaţi IBM), dar, pe parcurs, o serie de alte persoane din întreaga lume şi-au adus contribuţia la forma finală a cărţii. Versiunea online a acestei cărţi este gratuită pentru întreaga comunitate. Pentru cei care doresc să ofere feedback sau să contribuie cu material nou în scopul îmbunătăţirii conţinuturilor existente, sau pentru cei care doresc să contribuie la traducerea cărţii în limba natală se pune la dispoziţie un e-mail la adresa db2univ@ca.ibm.com la care puteţi trimite un mesaj cu titlul ““Database fundamentals book feedback”.

Convenţii

Pe parcursul acestei cărţi se vor folosi o serie de exemple de comenzi, clauze SQL, dar şi diverse alte coduri, ceea ce a impus utilizarea unor convenţii pentru o mai bună înţelegere. Astfel, cuvintele cheie specifice sunt scrise cu litere mari, îngroşate. De exemplu: **`NULL`** care reprezintă o stare necunoscută.

Pentru comenzi se folosesc caractere îngroşate, scrise cu litere mici. De exemplu: comanda **`dir`** care afişează toate fişierele şi subdirectoarele din Windows.

Clauzele SQL sunt prezentate cu caractere îngroşate, scrise cu litere mari. De exemplu, clauza **`SELECT`** care se foloseşte pentru a regăsi datele inserate în cadrul unui tabel.

Numele obiectelor folosite în cadrul exemplelor sunt prezentate cu litere îngroşate, înclinate. De exemplu, tabelul ***`flights`*** care are cinci coloane.

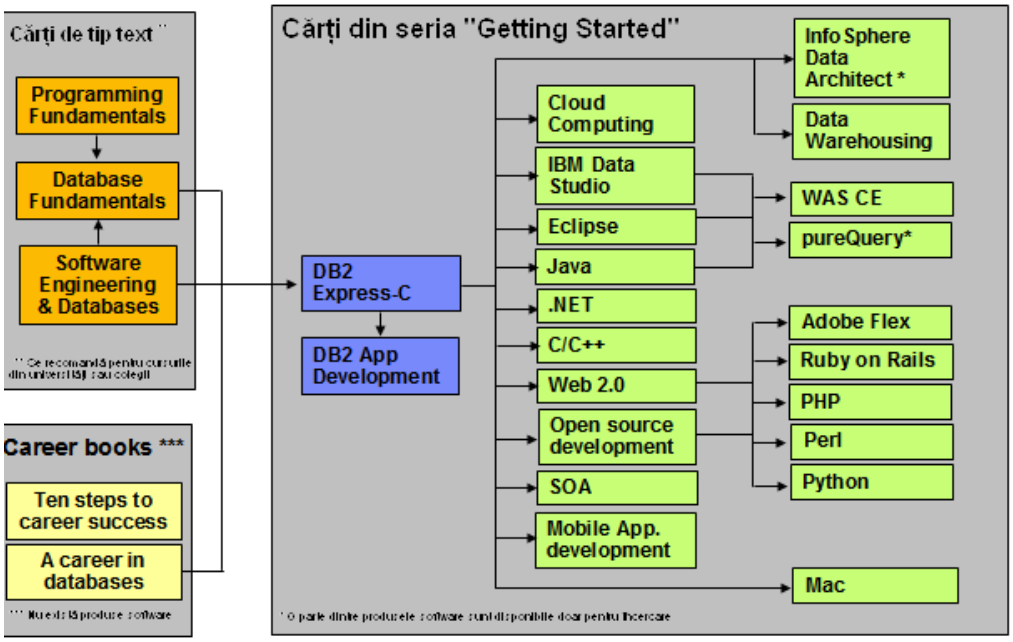
Caracterele înclinate se mai folosesc şi în cazul numelor de variabile din sintaxa unei comenzi sau clauze. Dacă numele variabilei este alcătuit din mai multe cuvinte, se foloseşte caracterul de subliniere pentru a înlocui spaţiile libere. De exemplu, **`CREATE TABLE`** ***`nume_tabel`***

Ce urmează?

Pentru a obţine mai multe informaţii despre subiectele discutate în cadrul acestei cărţi vă recomandăm să parcurgeţi şi alte cărţi din această serie, cum ar fi:

- *Getting started with* DB2 Express-C
- *Getting started with InfoSphere Data Architect*
- *Getting started with data warehousing*
- *Getting started with DB2 application development*

În figura următoare este prezentată o listă de cărţi în format electronic din cadrul seriei DB2 on Campus disponibile în mod gratuit la dresa **db2university.com**



Cărţi din seria DB2 on Campus

Autorii

Neeraj Sharma este specialist senior IT la Dynamic Warehousing Center of Competency, IBM India Software Labs. Rolul său principal este proiectarea, configurarea şi implementarea depozitelor mari de date în diverse domenii industriale; implementarea de soluţii personalizate şi execuţia de teste de performanţă la solicitarea clienţilor. Are diplomă de licenţă în Electronică şi Ingineria Comunicaţiilor şi diplomă de master în Sisteme software.

Liviu Perniu este conferenţiar la Catedra de Automatică a Universităţii Transilvania din Braşov, România şi predă cursuri în domeniul analizei cerinţelor şi modelării datelor. Este posesorul unui premiu IBM în 2006 în cadrul programului Faculty Award la secţiunea Eclipse Innovation Awards şi a titlului IBM Champion conferit în anul 2011.

Raul F. Chong este conducătorul programului *DB2 on Campus* pornit la IBM Toronto Laboratory fiind promotor al aspectelor tehnice din cadrul sistemului DB2. Principala responsabilitate pe care o are este creşterea comunităţii DB2 în întreaga lume. Raul s-a alăturat IBM din anul 1997, deţinând numeroase poziţii în cadrul companiei. Din poziţia de consultant, Raul a ajutat partenerii IBM să migreze de la alte sisteme de baze de date relaţionale la DB2, dar şi să rezolve problemele legate de performanţa bazelor de date şi de proiectarea aplicaţiilor. Din poziţia de consultant tehnic DB2, Raul a ajutat la rezolvarea problemelor DB2 pe platformele OS/390®, z/OS®, Linux®, UNIX® şi Windows. Raul a participat la numeroase laboratoare de lucru DB2, a publicat numeroase articole şi a contribuit la realizarea tutorialelor pentru examenele DB2 Certification. Raul şi-a adunat cea mai mare parte a experienţei căpătate de-a lungul anilor în DB2 în cartea *Understanding DB2 - Learning Visually with Examples 2nd Edition (ISBN-10: 0131580183*) la care este autorul coordonator. Este, de asemenea, coautor al cărţii *DB2 SQL PL Essential Guide for DB2 UDB on Linux, UNIX, Windows, i5/OS, and z/OS (ISBN* 0131477005), fiind coordonatorul proiectului şi coautor al multor cărţi din seria *DB2 on Campus.*

Abhishek Iyer este inginer la Warehousing Center of Competency, IBM India Software Laboratory. Rolul său principal este acela de a furniza soluţii viabile şi de a executa teste de performanţă la solicitările clienţilor. Expertiza sa acoperă implementări ale volumelor mari de date şi în regăsiri ale datelor in depozite de date de mari dimensiuni. Are diplomă de licenţă în Ştiinţa Calculatoarelor.

Adi-Cristina Mitea este conferenţiar la Catedra de Ştiinţa Calculatoarelor, de la Facultatea de inginerie a universităţii “Lucian Blaga” din Sibiu. Predă cursuri în domeniul bazelor de date, a sistemelor distribuite, a algoritmilor paraleli şi distribuiţi, a sistemelor tolerante la defectări şi altele. Activităţile sale de cercetare fac parte din aceste domenii. Are diplomă de licenţă şi doctorat in Ştiinţa Calculatoarelor.

Chaitali Nandan este inginer software în echipa *DB2 Advanced Technical Support* a laboratorului *IBM India Software Laboratory*. Rolul său principal în cadrul acestui colectiv este acela de a oferi primul ajutor şi sprijin în producţie utilizatorilor de sisteme de baze de date DB2 Enterprise. Este specializată în rezolvarea de probleme critice apărute la

sistemele de baze de date DB2 şi are diplomă de licenţă în Tehnologia informaţiei.

Mallarswami Nonvinkere este specialist în tehnologia pureXML® la IBM's India Software Laboratory şi lucrează pentru echipa de activare a caracteristicilor DB2 pureXML în India. Lucrează cu clienţi IBM şi ISV pentru a-i ajuta să înţeleagă folosirea tehnologiei pureXML şi să elaboreze aplicaţii de mare performanţă folosind XML. Mallarswami oferă clienţilor recomandări practice, fiind activ implicat în oferirea de soluţii legate de tehnologiile DB2. A luat cuvântul la numeroase conferinţe internaţionale, printre care cele de la IDUG Australasia, IDUG India şi IMTC şi este activ în numeroase forumuri din cadrul developerWorks®

Mirela Danubianu este şef de lucrări la Universitatea Ştefan cel Mare din Suceava, Facultatea de Inginerie Electrică şi Ştiinţa Calculatoarelor. Posedă o diplomă de master în Ştiinţa Calculatoarelor obţinută la Universitatea din Craiova (1985 – Automatizări şi Calculatoare) şi o alta în Economie la Universitatea Ştefan cel Mare din Suceava, (2009 - Management). De asemenea, deţine doctoratul în Ştiinţa Calculatoarelor la Universitatea Ştefan cel Mare din Suceava (2006 – Contribuţii la dezvoltarea metodelor şi tehnicilor de exploatare a datelor şi managementul cunoştinţelor). Cercetările actuale se încadrează în domeniul teoriei şi implementării bazelor de date, data mining, data warehouse, aplicaţiilor avansate ale tehnologiei informaţiei în economie şi sănătate. Mirela este coautoare la 7 cărţi şi la peste 25 de articole. A participat la peste 15 conferinţe şi este membru în conferinţe internaţionale cu comitet de program.

Au contribuit

La elaborarea acestei cărţi, au avut o contribuţie semnificativă în scopul editării, revizuirii, sau oferirii de conţinuturi, următoarele persoane:

Contribuabil	Companie/Universitate	Poziţie/Ocupaţie	Contribuţie
Agatha Colangelo	ION Designs, Inc	Modelator de date	A elaborat cuprinsul cărţii
Cuneyt Goksu	VBT Vizyon Bilgi Teknolojileri	DB2 SME şi IBM Gold Consultant	Recenzie tehnică
Marcus Graham	IBM US	Programator	Recenzie tehnică şi a limbii engleze pentru capitolul 10
Amna Iqbal	IBM Toronto Lab	Asigurarea calităţii - Lotus Foundations	Recenzia limbii engleze pentru toată cartea fără capitolele 5 şi 7
Leon Katsnelson	IBM Toronto Lab	Director de program, IBM Data Servers	Recenzie tehnică şi controbuţii la capitolul 10
Jeff (J.Y.) Luo	IBM Toronto Lab	Technical Enablement Specialist	Recenzia limbii engleze pentru capitolul 7
Fraser McArthur	IBM Toronto Lab	Promotor în domeniul Managementul informaţiei	Recenzie tehnică
Danna Nicholson	IBM US	STG ISV Enablement, Web Services	Recenzia limbii engleze pentru întreaga carte
Rulesh Rebello	IBM India	Advisory Manager - IBM Software Group Client Support	Recenzie tehnică
Suresh Sane	DST Systems, Inc	Arhitect baze de date	Recenzia diverselor capitole, în special a celor legate de SQL

Nadim Sayed	IBM Toronto Lab	Specialist în proiectarea centrată pe utilizator	Recenzia limbii engleze pentru capitolul 1
Ramona Truta	University of Toronto	Şef de lucrări	A elaborat cuprinsul întregii cărţi

Mulţumiri

Mulţumim următoarelor persoane pentru ajutorul acordat la elaborarea materialelor din această carte.

Natasha Tolub pentru realizarea coperţii cărţii.

Susan Visser pentru ajutorul acordat la publicarea cărţii.

1

Capitolul 1 – Baze de date şi modele de informaţie

Datele reprezintă unul dintre cele mai importante atuuri ale oricărei afaceri. Acestea sunt utilizate şi colectate practic de oriunde, de la companiile care îşi propun să identifice şabloane ce pot fi aplicate consumatorilor în funcţie de utilizarea cărţilor de credit, până la agenţiile spaţiale care urmăresc să obţină date despre alte planete. Datele, pe măsura importanţei lor, au nevoie de produse software robuste şi sigure, cu un grad ridicat de disponibilitate care să le poată stoca şi procesa cu rapiditate. Aceste cerinţe sunt îndeplinite cu ajutorul unor produse solide şi fiabile numite baze de date.

Utilizarea produselor software de baze de date este omniprezentă, acest lucru fiind obligatoriu pentru a permite accesul zilnic la date al utilizatorilor din întreaga lume. Prezenţa acestor baze de date o regăsim pretutindeni, de la preluarea banilor drintr-un bancomat până la accesul securizat la birou cu ajutorul cartelelor magnetice.

Capitolul de faţă oferă o trecere în revistă a elementelor fundamentale ale sistemelor de gestiune a bazelor de date şi a modelelor de informaţii.

1.1 Ce este o bază de date?

Încă de la apariţia lor, bazele de date au reprezentat unul dintre domeniile cele mai cercetate în informatică. O ***bază de date*** este un depozit de date destinat să sprijine stocarea eficientă a datelor, dar şi regăsirea şi întreţinerea acestora. Pentru a răspunde cerinţelor diverse care există în industrie s-au creat în timp mai multe tipuri de baze de date. O bază de date poate fi folosită pentru a stoca fisiere binare, documente, imagini grafice sau video, date relaţionale, date multidimensionale, date tranzactionale, date analitice, date geografice, pentru a numi doar câteva dintre tipurile cele mai uzuale.

Datele pot fi stocate în diferite formate, cum ar fi: tabelar, ierarhic sau graf. În cazul în care datele sunt stocate în format tabelar, atunci avem de a face cu o ***bază de date relaţională***. Atunci când datele sunt organizate într-o structură arborescenta, avem de a face cu o ***bază de date ierarhică***. Datele stocate sub forma unui graf care reprezintă relaţiile dintre obiecte alcătuiesc o ***bază de date de tip reţea***. În această carte, ne vom concentra atenţia asupra bazelor de date relaţionale.

1.2 Ce este un sistem de gestiune al bazelor de date?

În timp ce o bază de date reprezintă doar un simplu depozit de date, ***sistemele de gestiune ale bazelor de date***, pe scurt SGBD, reprezintă un grup de instrumente software cu ajutorul cărora se controlează accesul, se organizează, înmagazinează, gestionează, se extrag şi se întreţin datele din cadrul unei baze de date. În practică, termenii de bază de date, server de bază de date, sistem de baze de date, server de date şi sistem de gestiune al bazelor de date se folosesc de cele mai multe ori în mod alternativ.

De ce avem nevoie de un produs software pentru bazele de date? Oare nu s-ar putea păstra datele în simple fişiere text, de exemplu? Răspunsul se află în felul în care utilizatorii accesează datele şi rezolvă diversele probleme care apar la fiecare situaţie în parte.

În primul rând, apare necesitatea de a avea mai mulţi utilizatori care să insereze, să actualizeze şi să şteargă datele în cadrul aceluiaşi fişier de date fără a se incomoda unul pe celălalt. Acest lucru presupune faptul că utilizatorii nu vor putea întreprinde acţiuni în urma cărora datele să devină inconsistente şi nici nu se vor pierde date în urma efectuării anumitor operaţii. De asemenea, este nevoie de o interfaţă standard folosită la accesul datelor, de instrumente pentru realizarea de copii de rezervă, de restaurare şi recuperare a datelor, precum şi de o modalitate de a rezolva şi alte aspecte, cum ar fi capacitatea de a lucra cu volume mari de date şi utilizatori. Produsele software destinate lucrului cu baze de date au fost proiectate tocmai pentru a rezolva toate aceste aspecte.

Cele mai mature sisteme de baze de date aflate în producţie sunt sistemele relaţionale de gestiune a bazelor de date (SGBDR). Sistemele SGBDR reprezintă coloana vertebrală a celor mai multe aplicaţii folosite în industrie, cum ar fi cele din domeniul bancar, transporturi, sănătate şi altele.

Apariţia interfeţelor ce folosesc Web a făcut să crească foarte mult volumul şi domeniul de utilizare al SGBDR-urilor, care servesc păstrării datelor aflate în spatele celor mai importante aplicaţii de comerţ electronic.

1.2.1. Evoluţia sistemelor de gestiune a bazelor de date

În anii ’60, sistemele ierarhice şi în reţea cum ar fi CODASYL sau IMSTM reprezentau tehnologiile cele mai folosite pentru sistemele bancare, de contabilitate şi de prelucrare a comenzilor odată cu introducerea sistemelor de tip mainframe în domeniul comercial. Deşi aceste sisteme ofereau o bază solidă pentru sistemele incipiente, arhitectura acestora făcea să se amestece manipularea fizică a datelor cu manipularea logică a acestora. Atunci când se modifica locaţia fizică a datelor dintr-o zonă a discului în alta, aplicaţiile trebuiau să fie actualizate pentru a face referire la noua locaţie.

Un articol revoluţionar din 1970 scris de către E.F. Codd, angajat al IBM San Jose Research Laboratory, a făcut să se schimbe toate acestea. Lucrarea, intitulată “*Modelul relaţional al datelor pentru bănci de date distribuite de mari dimensiuni*” [1.1] a introdus noţiunea de independenţă a datelor, care separa reprezentarea fizică a acestora de reprezentarea logică folosită în cadrul aplicaţiilor. Datele puteau fi mutate dintr-o parte în

alta a discului sau păstrate în formate diferite fără a necesita rescrierea aplicaţiilor. Programatorii de aplicaţii erau acum eliberaţi de detaliile fizice plictisitoare de manipulare a datelor putându-se concentra mai mult pe aspectele logice de manipulare a acestora în contextul diverselor aplicaţii specifice la care lucrau.

Figura 1.1. ilustrează evoluţia sistemelor de gestiune a bazelor de date

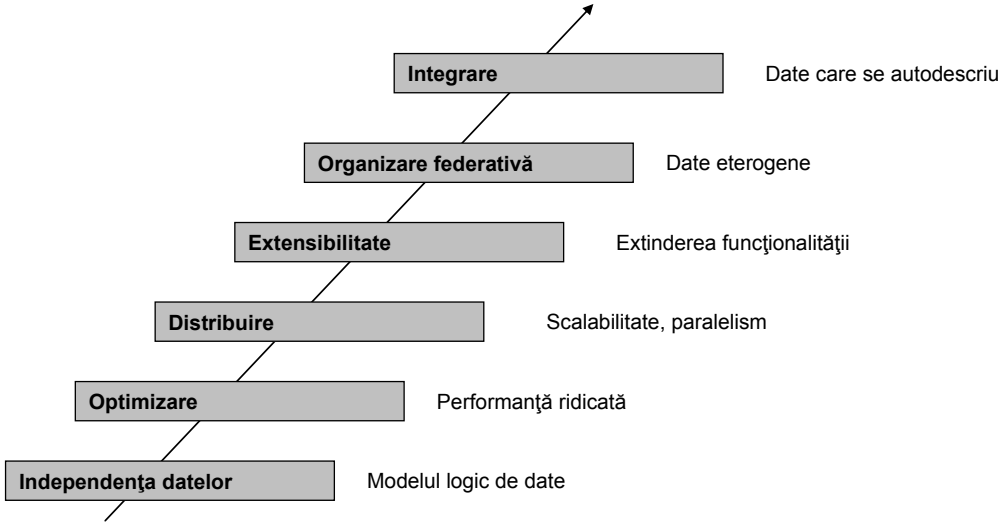


Figura 1.1 Evoluţia sistemelor de gestiune a bazelor de date

Figura de mai sus descrie evoluţia sistemelor de gestiune a bazelor de date cu model relaţional care asigură independenţa datelor. Produsul System R de la IBM a fost primul sistem care a implementat ideile lui Codd punând bazele SQL/DS care va deveni ulterior DB2. Prin intermediul acestuia s-a introdus pentru prima dată limbajul SQL, limbaj destinat lucrului cu date în sistemele relaţionale, care a devenit ulterior standard şi a deschis porţile sistemelor comerciale de gestiune a bazelor de date.

Astăzi, sistemele relaţionale de gestiune a bazelor de date reprezintă cele mai utilizate sisteme de gestiune a bazelor de date şi sunt furnizate de câteva companii producătoare de software. Unul dintre liderii de piaţă în acest domeniu este IBM cu produsul server de baze de date numit DB2. Printre alte sistemele de gestiune a bazelor de date relaţionale putem enumera: Oracle, Microsoft SQL Server, INGRES, PostgreSQL, MySQL, şi dBASE.

Deoarece bazele de date relaţionale au devenit din ce în ce mai populare, a apărut necesitatea de a folosi interogări de înaltă performanţă. Din acest motiv, optimizatorul DB2 este una dintre cele mai sofisticate componente ale produsului. Din punct de vedere al utilizatorului, optimizatorul nu reprezintă altceva decât o cutie neagră spre care este transmisă orice interogare SQL. Optimizatorul DB2 va calcula apoi care este cea mai rapidă modalitate de a regăsire a datelor, luând în considerare mai mulţi factori, cum ar fi viteza procesorului şi a discului, cantitatea de date disponibile, localizarea datelor, tipul de date, existenţa unor indecşi, şi aşa mai departe. Optimizatorul DB2 se bazează pe estimarea costurilor.

Datorită creşterii cantităţii de date care sunt colectate şi stocate în bazele de date, sistemele de gestiune a bazelor de date trebuiau să permită scalabilitate. De exemplu, DB2 pentru Linux, UNIX şi Windows prezintă o caracteristică numită *Database Partitioning Feature (DPF)* ce permite bazelor de date să se extindă şi pe alte maşini folosind o arhitectură specială. Fiecare maşină adăugată vine cu propriul procesor (procesoare) şi disc (discuri), ceea ce permite o scalabilitate aproape liniară. Fiecare interogare dintr-un astfel de mediu este paralelizată astfel încât fiecare maşină poate extrage părţile de care are nevoie din rezultatul final.

Ulterior, în evoluţia sistemelor de gestiune a bazelor de date relaţionale a apărut conceptul de extensibilitate. Limbajul structurat de interogare (SQL) descoperit de către IBM la începutul anilor ’70 a fost constant îmbunătăţit pe parcurs. Chiar dacă acest limbaj este unul foarte puternic, utilizatorii sunt totuşi încurajaţi să-şi dezvolte propriul cod pe care să-l extindă. De exemplu, în DB2 se pot crea funcţii definite de către utilizator şi proceduri stocate care permit extinderea limbajului SQL pe logica specifică fiecăruia.

În continuare, sistemele de gestiune a bazelor de date au început să abordeze problema manipulării diferitelor tipuri de date provenite din surse diverse. La un moment dat, serverul de date DB2 a fost redenumit pentru a include termenul "universal" aşa cum apare în "DB2 Universal Database" (DB2 UDB). Deşi acest termen a fost abandonat ulterior din motive de simplitate, acest lucru a fost important deoarece s-a evidenţiat faptul că serverul de date DB2 poate stoca toate tipurile de informaţii, inclusiv video, date audio, binare, şi aşa mai departe. Mai mult decât atât, prin conceptul de federalizare, o interogare poate fi folosită în DB2 pentru a accesa datele de la alte produse ale IBM, sau chiar de la produse non-IBM.

Urmatorul pas evolutiv ce este pus în evidenţă la finalul figurii de mai sus este acela de integrare. Astăzi multe organizaţii au nevoie să facă schimb de informaţii, iar Extensible Markup Language (XML) este tehnologia de bază, care este utilizată în acest scop şi care foloseşte un limbaj de autodescriere extensibil. Utilizarea acestuia a marcat o creştere exponenţială datorită apariţiei Web 2.0, şi a arhitecturii orientate spre servicii (SOA). IBM a recunoscut din vreme importanţa XML şi, prin urmare, a dezvoltat o tehnologie numita pureXML®, care este disponibilă în cadrul serverelor de baze de date DB2. Folosind această tehnologie, documentele XML se pot stoca într-o bază de date DB2 în format ierarhic (care este formatul nativ XML). În plus, motorul DB2 a fost îmbunătăţit, astfel încât să poată lucra cu XQuery, care este limbajul folosit pentru parcurgerea documentelor XML. Prin folosirea pureXML, DB2 aduce cea mai bună performanţă în lucrul cu tehnologia XML, oferind în acelaşi timp securitate, robusteţe si scalabilitate, obiectiv urmărit încă de la începuturi la folosirea datelor relaţionale.

Acum, la momentul scrierii acestei cărţi, subiectul “fierbinte” îl constituie conceptul de Cloud Computing, iar din acest punct de vedere, DB2 este un produs bine poziţionat pentru a face faţă noilor provocări de lucru în Cloud. De fapt, există deja imagini DB2 disponibile pe cloud-ul Amazon EC2, şi pe dezvoltarea şi testarea mediului inteligent de afaceri de pe IBM Cloud (cunoscut, de asemenea, sub numele de dezvoltare şi testare în mediul IBM Cloud). Caracteristica de partiţionare a bazei de date existentă în DB2 amintită anterior se potriveşte perfect într-un mediu cloud în care se pot cere la comandă noduri standard sau servere ce pot fi adăugate unui cluster existent. Rebalansarea datelor este efectuată în

mod automat de către DB2 în timpul lucrului. Acest aspect poate fi extrem de util pe parcurs, de exemplu, în momentul în care se solicită mai multă putere ce trebuie acordată serverului de baze de date pentru a putea finaliza tranzacţiile de la sfârşitul lunii sau de la sfârşitul anului.

1.3 Introducere în modelele de informaţii şi modelele de date

Un model al informaţiilor este o reprezentare abstractă, formală a entităţilor cu ajutorul căreia se precizează proprietăţile, relaţiile şi operaţiile ce pot fi efectuate asupra acestora. Entităţile care se modelează pot proveni din lumea reală, cum ar fi dispozitivele existente în cadrul unei reţele, sau pot să fie le însele abstracte, cum ar fi entităţile utilizate în cadrul sistemelor de facturare.

Scopul principal care se ascunde în spatele acestui concept este acela de a formaliza descrierea domeniului unei probleme fără a avea în vedere constrângerile prin care descrierea respectivă este afectată la momentul transformării acesteia în cadrul implementării software. În practică pot exista mai multe corespondenţe ale unui model al informaţiilor. Corespondenţele sunt numite modele de date, indiferent dacă acestea sunt modele obiect (de exemplu, folosirea un limbaj unificat de modelare - UML), modele entitate-relaţie, sau scheme XML.

Modelarea este un aspect deosebit de important, deoarece trebuie luate în considerare aspecte ce trebuie avute în vedere în cazul eventualelor modificări ulterioare, fără a afecta în mod semnificativ utilizarea produsului. Modelarea permite compatibilizarea cu modelele anterioare şi prevede soluţii pentru extensiile viitoare.

Modelele de informaţii şi modelele de date sunt distincte deoarece au scopuri diferite. Scopul principal al unui model al informaţiilor este acela de modelare a obiectelor gestionate la nivel conceptual, independent de orice implementare specifică sau protocoale folosite la transportul datelor. Nivelul de detaliere al abstractizării efectuate în cadrul unui model al informaţiilor depinde de necesităţile de modelare ale proiectanţilor. Pentru a face proiectarea de ansamblu cât mai uşor de realizat, un model al informaţiilor trebuie să ascundă toate protocoalele de comunicare şi detaliile de implementare. O altă caracteristică importantă a unui model al informaţiilor o reprezintă stabilirea relaţiilor dintre obiectele avute în vedere.

Pe de altă parte, modelele de date sunt realizate la un nivel mult mai concret şi conţin mai multe detalii. Deoarece sunt destinate programatorilor, conţin construcţii specifice referitoare la protocoalele folosite. Un model de date reprezintă planul unui sistem de baze de date. *Figura 1.1* ilustrează relaţia dintre un model al informaţiilor şi un model de date.

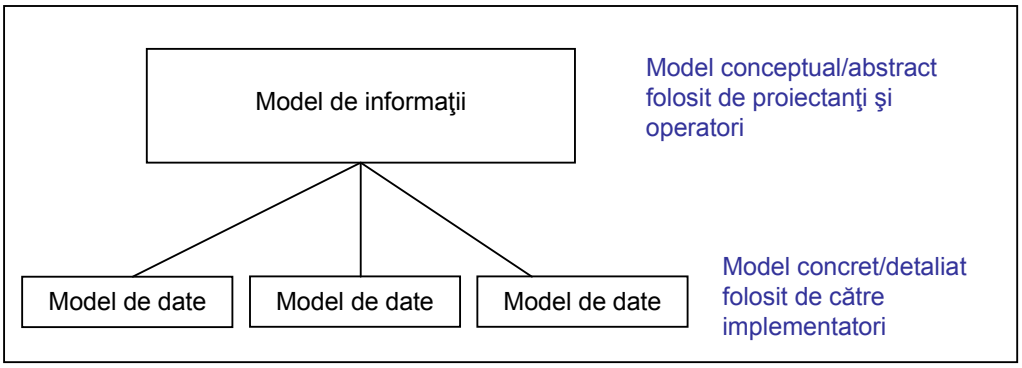


Figura 1.1 – Legătura dintre modelul de informaţii şi modelul de date

Deoarece modelele conceptuale pot fi implementate în diverse moduri, se pot obţine mai multe modele de date dintr-un singur model de informaţii.

1.4 Tipuri de modele de informaţii

Propunerile de modele de informaţii pot fi clasificate în 9 perioade istorice:

- Reţea (CODASYL): în anii ‘70
- Ierarhic (IMS): la sfârşitul anilor ‘60 şi începutul anilor ‘70
- Relaţional: în anii ‘70 şi începutul anilor ‘80
- Entitate-relaţie: în anii ‘70
- Relaţional extins: în anii ‘80
- Semantic: la sfârşitul anilor ‘70 şi începutul anilor ’80
- Orientat pe obiect: la sfârşitul anilor ‘80 şi începutul anilor ’90
- Relaţional-obiectual: la sfârşitul anilor ‘80 şi începutul anilor ’90
- Semi-structurat (XML): de la sfârşitul anilor ’90 până în prezent

Secţiunile următoare se vor referi mai în detaliu la unele dintre aceste modele.

1.4.1 Modelul reţea

În 1969 CODASYL (comitetul care se ocupă cu limbajele sistemelor de date) a lansat prima specificaţie referitoare la modelul reţea. Aceasta a fost urmată în 1971 şi 1973 de specificaţiile referitoare la limbajul de manipulare a datelor de tip înregistrare în timp. Un exemplu de model de date de tip reţea este prezentat în *Figura 1.2*.

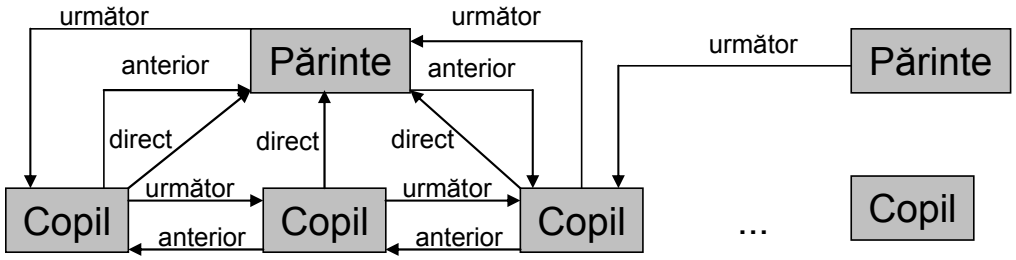


Figura 1.2 – Modelul reţea

Figura arată tipurile înregistrare reprezentate prin dreptunghiuri. Aceste tipuri pot folosi şi chei pentru a identifica o înregistrare. O colecţie de tipuri de înregistrări şi chei alcătuiesc o reţea CODASYL sau o bază de date CODASYL. Se observă faptul că un copil poate avea mai mulţi părinţi şi fiecare tip de înregistrare se poate lega de altul folosind pointerii următori, anteriori sau direcţi.

1.4.2 Modelul ierarhic

Modelul ierarhic îşi organizează datele pe baza unei structuri arborescente. Rădăcina arborelui este părintele, urmat de nodurile copil. Un nod copil nu poate avea mai multe noduri părinte, dar un nod părinte poate avea mai multe noduri copil. Acest lucru este prezentat în *Figura 1.3*.

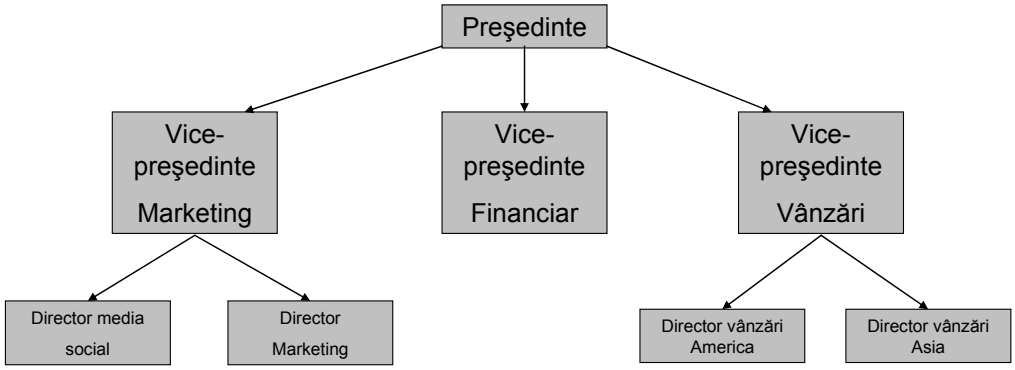


Figura 1.3 – Model ierarhic

Într-un model ierarhic, colecţia de câmpuri împreună cu tipurile de date asociate este denumită tip înregistrare. Fiecare instanţă a unui tip înregistrare este obligat să se supună descrierii datelor care se află în definiţia acelui tip înregistrare. Unele dintre câmpurile unui tip înregistrare sunt chei.

Primul sistem de gestiune al bazelor de date de tip ierarhic a fost IMS (sistem de management al informaţiilor) lansat de către IBM în 1968. Iniţial, a fost construit pentru a servi ca bază de date a programului spaţial Apollo care a lansat primul om pe lună. IMS este o bază de date foarte robustă care se mai află încă în funcţiune la mai multe companii din întreaga lume.

1.4.3 Modelul relaţional

Modelul relaţional de date este simplu şi elegant având un suport matematic solid bazat pe teoria mulţimilor şi calcul predicativ, fiind cel mai folosit model de date pentru bazele de date actuale.

Unul dintre punctele de pornire ale cercetării lui Codd a fost acela că programatorii care foloseau sistemul IMS pierdeau o mare parte din timp pentru mentenenţa aplicaţiilor IMS în care apăreau modificări logice sau fizice. Din acest motiv, scopul a fost acela de a introduce un model care să permită o mai bună independenţă a datelor. Propunerea sa a fost:

- Păstrarea datelor într-o structură simplă de date (tabele)
- Accesarea datelor la un nivel înalt folosind limbajul de manipulare a datelor (DML)
- Independenţa datelor din punct de vedere al stocării fizice

Având de a face cu o structură de date simplă, există o şansă mai bună de a oferi independenţa logică a datelor. De asemenea, având la dispoziţie un limbaj de nivel înalt, se poate oferi o independenţă mare a datelor din punct de vedere fizic. Din acest motiv, modelul relaţional permite independenţa faţă de dispozitivul fizic de stocare a datelor, ceea ce nu era posibil nici cu IMS, nici cu CODASYL. *Figura 1.4* arată un exemplu de diagramă entitate-relaţie care prezintă tipurile de entităţi (tabelele) împreună cu relaţiile dintre acestea în cadrul unui exemplu de model relaţional de date. În secţiunea următoare se va discuta mai în detaliu despre diagramele entitate-relaţie.

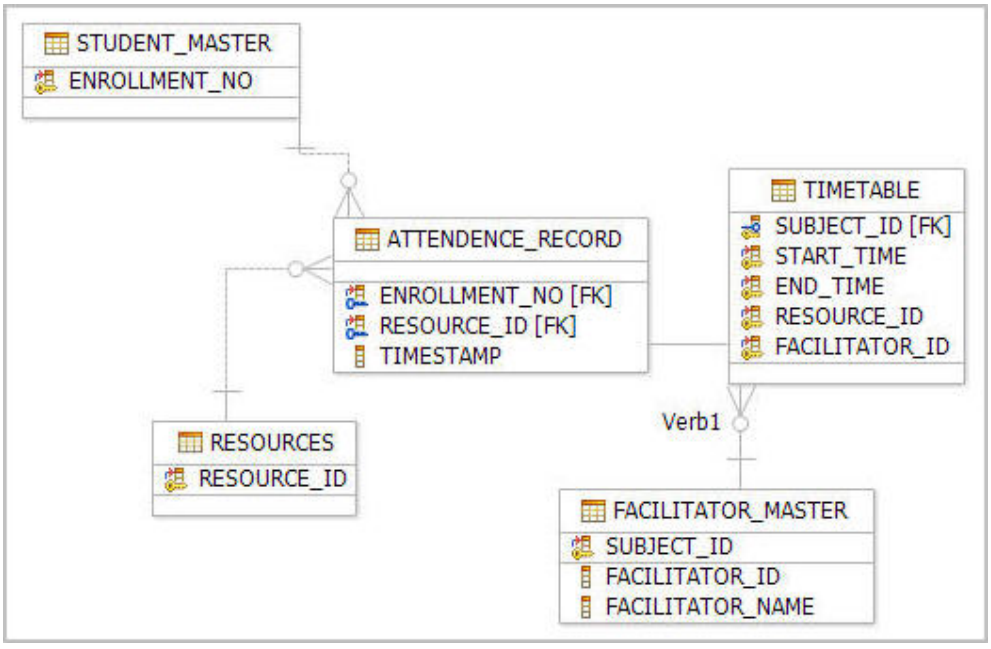


Figura 1.4 – Diagrama entitate-relaţie ce prezintă un model relaţional de date

1.4.4 Modelul entitate-relaţie

La mijlocul anilor ’70, Peter Chen a propus modelul de date entitate-relaţie (E-R). Acesta a reprezentat atunci o alternativă la modelele relaţionale, CODASYL şi ierarhice de date. El a propus un mod de gândire asupra bazelor de date sub forma unei colecţii de instanţe de entităţi. Entităţile sunt obiecte care au o existenţă independentă de existenţa oricărei alte entităţi dintr-o bază de date. Entităţile au atribute, ce reprezintă elementele de date ce caracterizează entitatea respectivă. Unul sau mai multe dintre aceste atribute pot fi desemnate ca fiind chei. Între entităţi pot exista relaţii, care pot fi 1-la-1, 1-la-mulţi, mulţi-la-1, sau mulţi-la-mulţi, în funcţie de modul de participare a acestora la relaţie. Relaţiile pot avea la rândul lor atribute care descriu relaţia repectivă. *Figura 1.5* oferă un exemplu de diagramă E-R.

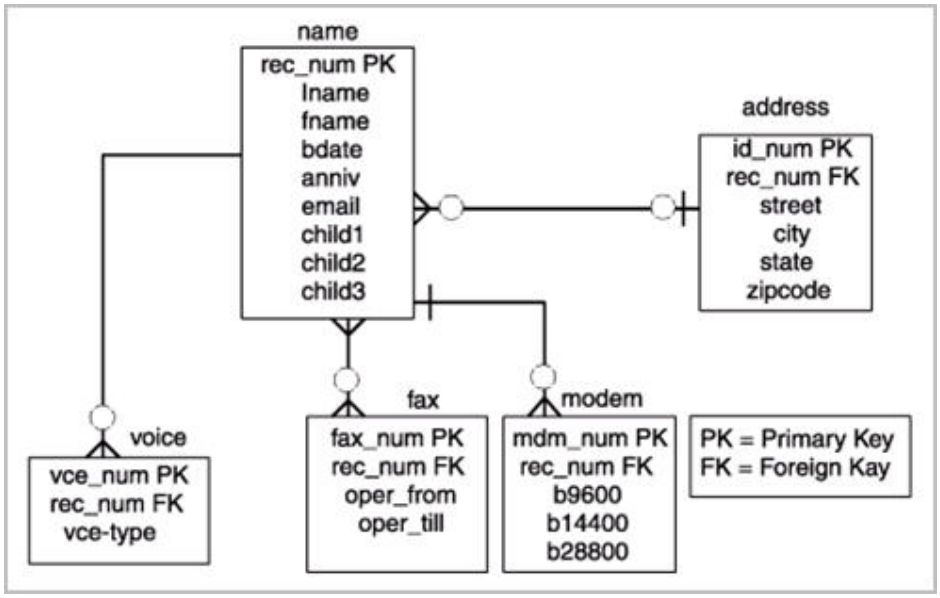


Figura 1.5 - Diagrama E-R pentru un model de date

În figură, tipurile de entităţi sunt reprezentate sub forma unor dreptunghiuri şi sunt denumite ***name***, ***address***, ***voice***, ***fax*** şi ***modem***. În interiorul fiecărui tip de entitate sunt afişate atributele corespunzătoare. De exemplu, tipul de entitate ***voice*** are atributele ***vce_num, rec_num*** şi ***vce-type.*** PK reprezintă cheia primară, iar FK reprezintă cheia externă. Conceptul de cheie va fi discutat mai în detaliu într-o altă secţiune a acestei cărţi.

Modelul E-R a avut mare succes fiind folosit ca instrument de proiectare al bazelor de date relaţionale. Articolul lui Chen oferă o metodologie de creare a unei diagrame iniţiale E-R, prin intermediul căreia se pune la dispoziţie un proces simplu de convertire a unei diagrame E-R într-o colecţie de tabele aflate în forma normală 3. Despre forma normală 3 şi teoria normalizării aflaţi mai multe într-o altă secţiune a acestei cărţi.

Astăzi, crearea de diagrame E-R este facilitată prin intermediul unor instrumente de modelare a datelor, cum ar fi IBM InfoSphere™ Data Architect. Pentru a afla mai multe

despre acest instrument ajutător citiţi cartea în format electronic, *Getting started with InfoSphere Data Architect*, care este parte a seriei de cărţi din cadrul proiectului DB2 on Campus.

1.4.5 Modelul relaţional-obiectual

Modelul obiectual-relaţional (OR) este foarte asemănător cu modelul relaţional, cu deosebirea că tratează fiecare entitate sub forma unui obiect (instanţă a unei clase) şi o relaţie sub forma unei moşteniri. Dintre caracteristicile şi avantajele modelului OR amintim:

- Suport pentru tipuri complexe, definite de utilizator
- Moştenirea obiectelor
- Obiecte extensibile

Bazele de date obiectual-relaţionale au capacitatea de a păstra relaţiile dintre obiecte sub o formă relaţională.

1.4.6 Alte modele de date

Ultima decadă s-a remarcat prin concentrarea eforturilor în domeniul modelelor de date semi-structurate, semantice şi orientate pe obiecte.

XML reprezintă modalitatea ideală de stocare a datelor semi-structurate. Modelele bazate pe XML şi-au câştigat o mare popularitate în industrie datorită Web 2.0 şi a arhitecturilor orientate pe servicii (SOA).

Modelele de date orientate pe obiecte sunt foarte populare în universităţi, dar nu au fost acceptate pe scară largă în industrie; instrumentele existente de realizare a corespondenţelor obiectual-relaţionale (ORM) permit o integrare perfectă între aplicaţiile orientate pe obiecte şi bazele de date relaţionale.

1.5 Roluri tipice şi direcţii în carieră pentru profesioniştii bazelor de date

La fel ca orice alt domeniu de activitate, şi domeniul bazelor de date are câteva roluri şi direcţii în carieră asociate. În continuare se va face o prezentare a acestor roluri.

1.5.1 Arhitectul de date

Arhitectul de date este responsabil de proiectarea unei arhitecturi care să sprijine cerinţele existente şi viitoare ale unei organizaţii în ceea ce priveşte managementul datelor. Arhitectura trebuie să acopere partea de baze de date, partea de integrare a datelor, precum şi modalitatea de obţinere a datelor. De obicei, arhitectul de date atinge aceste obiective prin stabilirea unor standarde în cea ce priveşte datele organizaţiei. Arhitectul de date mai este cunoscut şi sub denumirea de modelator de date, deşi un astfel de rol implică mult mai multe decât simpla creare de modele de date.

O parte din deprinderile fundamentale ale unui arhitect de date trebuie să se afle în zona:

- Modelării logice a datelor
- Modelării fizice a datelor
- Elaborării de strategii a datelor precum şi a politicilor asociate
- Alegerii caracteristicilor şi sistemelor care să îndeplinească cerinţele informaţionale ale unei organizaţii

1.5.2 Arhitectul bazei de date

Acest rol este asemănător arhitectului de date, fiind însă mai mult orientat către o soluţie ce conţine o bază de date. Arhitectul bazei de date este responsabil de îndeplinirea următoarelor activităţi:

- Achiziţionarea şi documentarea cerinţelor de la utilizatori şi de la managementul organizaţiei şi introducerea acestora în cadrul unei soluţii arhitecturale.
- Partajarea arhitecturii între utilizatori şi managementul organizaţiei
- Crearea şi implementarea bazei de date, precum şi elaborarea de standarde şi procese pentru dezvoltarea aplicaţiilor.
- Crearea şi punerea în aplicare a înţelegerilor la nivel de serviciu (SLA) în cadrul organizaţiei, în special în ceea ce priveşte disponibilitatea, realizarea de copii de siguranţă şi restaurare, precum şi securitatea.
- Studierea produselor noi, a compatibilităţii versiunilor şi a fezabilităţii introducerii în mediul real de funcţionare, cu asigurarea de recomandări pentru echipele de programatori şi de management.
- Înţelegerea suportului hardware, a sistemului de operare, a arhitecturii pe bază de componente la mai multe nivele şi a interacţiunii dintre aceste componente.
- Pregătirea documentaţiei de nivel înalt corespunzătoare cerinţelor.
- Revizuirea proiectelor detaliate şi a detaliilor de implementare.

Esenţial pentru un arhitect de baze de date este de a ţine pasul cu noile instrumente, produse de baze de date, platforme hardware si sisteme de operare provenite de la diverşi producători pe măsură ce acestea evoluează şi sunt îmbunătăţite.

1.5.3 Administratorul bazei de date (DBA)

Administratorul bazei de date este răspunzător de mentenenţa, performanţele, integritatea şi securitatea acesteia. Mai pot exista cerinţe suplimentare, cum ar fi planificarea, dezvoltarea şi depanarea.

Activitatea administratorului bazei de date poate să difere în funcţie de tipul organizaţiei şi de nivelul de responsabilitate asociat postului. Astfel, se poate avea în vedere doar simpla mentenanţă sau se poate implica administratorul bazei de date în activităţi de dezvoltare a bazei de date.

Responsabilităţile obişnuite includ printre altele:

- Determinarea cerinţelor utilizatorilor şi monitorizarea securităţii şi accesului acestora la baza de date;
- Monitorizarea performanţelor şi administrarea parametrilor în scopul creşterii vitezei de răspuns către utilizatori;
- Realizarea modelului conceptual al bazei de date;
- Luarea în considerare atât a organizării datelor pe partea de back-end cât şi a accesibilităţii pe partea de front-end din punctul de vedere al utilizatorilor;
- Rafinarea modelului logic astfel încât acesta să poată fi uşor adaptat unui model specific de date;
- Rafinarea în continuare a modelului fizic pentru a îndeplini cerinţele legate de partea de stocare a sistemului;
- Instalarea şi testarea noilor versiuni de sisteme de gestiune a bazelor de date (SGBD);
- Asigurarea respectării standardelor, inclusiv aderarea la regulile de protecţie a datelor personale prevăzute în legislaţia Data Protection Act;
- Scrierea documentaţiei bazei de date, inclusiv standardele datelor, procedurilor şi definiţiilor ce apar în dicţionarul de date (metadatele)
- Controlul drepturilor de acces şi a privilegiilor;
- Elaborarea, gestionarea şi testarea planurilor pentru realizarea copiilor de siguranţă şi a recuperării datelor;
- Asigurarea faptului că sunt respectate şi folosite corect procedurile de stocare, arhivare, realizarea de copii de siguranţă şi de recuperare a datelor;
- Capacitatea de planificare;
- Lucrul împreună cu managerii proiectului IT, programatorii bazei de date şi programatorii de aplicaţii Web;
- Comunicarea permanentă cu personalul tehnic, operaţional şi de realizare a aplicaţiilor în scopul asigurării integrităţii şi securităţii bazei de date;
- Punerea în funcţiune şi instalarea de noi aplicaţii

Datorită creşterii pericolului de furt al datelor, dar şi a naturii sensibile a datelor stocate, securitatea şi recuperarea acestora sau restaurarea datelor în cazul apariţiei unui dezastru au devenit aspecte din ce în ce mai importante.

1.5.4 Programatorii de aplicaţii

Programatorii de aplicaţii cu baze de date sunt acele persoane care elaborează aplicaţii ce accesează baze de date. Un astfel de programator are nevoie de următoarele cunoştinţe:

- Folosirea de medii de dezvoltare integrate ce folosesc baze de date (IDEs).
- Folosirea de plug-in-uri pentru IDE-uri.
- Instrumente de programare ce folosesc SQL.
- Monitorizarea şi păstrarea performanţelor bazei de date.
- Medii pentru servere de aplicaţii, punerea în mediul real de funcţionare a aplicaţiilor, depanarea şi monitorizarea performanţelor aplicaţiilor.

Un exemplu de mediu de dezvoltare integrat este IBM Data Studio, un mediu de dezvoltare bazat pe Eclipse care permite programatorilor să lucreze cu obiectele DB2, cum ar fi tabele, vederi, indecşi, proceduri stocate, funcţii definite de utilizatori şi servicii Web. Cu ajutorul acestui mediu se asigură facilităţi de depanare, de folosire a limbajelor SQL şi XQuery, dar şi integrarea cu alte servere de aplicaţii cum ar fi WebSphere® Application Server.

DB2 mai conţine şi module de integrare cu mediul de dezvoltare Microsoft® Visual Studio, punând la dispoziţie o serie de instrumente de lucru cu obiectele DB2 (tabele, vederi, proceduri stocate, funcţii definite de către utilizatori etc.). În acest fel programatorii .NET nu trebuie să treacă alternativ de la instrumentele DB2 la cele din Microsoft Visual Studio.

Rolurile şi responsabilităţile discutate până acum sunt prezentate în sens foarte larg deoarece organizaţiile au propria lor viziune asupra acestora în funcţie de contextul activităţii lor. Trecerea în revistă a acestor roluri s-a făcut cu scopul a oferi o imagine de ansamblu referitoare la diversele aspecte din jurul administrării bazelor de date, a dezvoltării de aplicatii si a utilizării acestora.

1.6 Rezumat

În cadrul acestui capitol s-au discutat câteva dintre conceptele fundamentale referitoare la lucrul cu bazele de date, pornind de la simpla definiţie a unei baze de date şi ajungând până la sistemele de gestiune a bazelor de date. Apoi s-au trecut în revistă modelele de date şi de informaţii cum ar fi, modelele reţea, ierarhic şi relaţional. La sfârşitul acestui capitol s-au prezentat o serie de roluri asociate domeniului bazelor de date. În capitolele următoare se vor prezenta mai detaliat conceptele din domeniul bazelor de date.

1.7 Exerciţii

1. Învăţaţi mai multe despre bazele de date lucrând cu DB2 Express-C, versiunea gratuită a serverului de baze de date DB2. Acest produs se poate descărca de la adresa: ibm.com/db2/express
2. Învăţaţi mai multe despre mediile integrate de dezvoltarea (IDE) lucrând cu produsul gratuit IBM Data Studio. Acest produs se poate descărca de asemenea de la adresa ibm.com/db2/express

1.8 Întrebări recapitulative

1. Ce este o bază de date?

2. Ce este un sistem de gestiune al bazelor de date?

3. Care este diferenţa dintre un model al informaţiilor şi un model de date?

4. Care este principalul avantaj al modelului relaţional comparativ cu alte modele?

5. Prezentaţi două dintre activităţile ce trebuie desfăşurate de către un administrator de baze de date.

6. Care dintre următoarele nu este un model al informaţiilor:

   A. modelul pureXML

   B. modelul relaţional

   C. modelul ierarhic

   D. modelul reţea

   E. nici unul dintre cele enumerate

7. În evoluţia sistemelor de gestiune a bazelor de date, la ce se referă noţiunea de optimizare?

   A. Disponibilitate ridicată

   B. Securitate

   C. Performanţă

   D. Scalabilitate

   E. Nici una dintre cele enumerate

8. Care dintre următoarele nu este menţionată în evoluţia sistemelor de gestiune a bazelor de date:

   A. Distribuirea

   B. Independenţa datelor

   C. Integrarea

   D. Federalizarea

   E. Nici una dintre cele enumerate

9. Din punct de vedere al evoluţiei sistemelor de gestiune al bazelor de date, în care stadiu s-ar afla pureXML?

   A. Independenţa datelor

   B. Extensibilitate

   C. Optimizare

D. Integrare

E. Nici una dintre cele enumerate

10. Prin ce este diferit DB2 atunci când operează în Cloud?

A. Are o extensie spaţială

B. Are o caracteristică de partiţionare a bazei de date

C. Are o tehnologie pureXML

D. Toate cele enumerate

E. Nici una dintre cele enumerate

2

Capitolul 2 – Modelul relaţional de date

În cadrul acestui capitol se vor lua în considerare aspectele fundamentale ale modelului relaţional de date, introducând conceptele de atribut, tuplu, relaţie, domeniu, schemă şi cheie. Se vor mai prezenta diverse tipuri de constrângeri ale modelului relaţional precum şi o serie de informaţii referitoare la algebra relatională şi calculul relaţional. Capitolul de faţă este puternic legat de *Capitolul 3, Modelul conceptual de date*, în care se va prezenta cum se face trecerea de la modelul conceptual de date la schema relaţională a bazei de date şi de *Capitolul 4, Proiectarea bazelor de date relaţionale*, în care se prezintă aspecte importante referitoare la proiectarea unei baze de date relaţionale. Acest capitol este foarte important pentru înţelegerea limbajului SQL. În cadrul acestui capitol se vor aborda aspectele:

- Vedere de ansamblu asupra modelului relaţional de date
- Definiţiile atributelor, tuplurilor, relaţiilor, domeniilor, schemelor şi cheilor
- Constrângerile modelului relaţional
- Operatorii folosiţi în algebra relaţională
- Calculul relaţional

2.1 Modelul relaţional de date: Vedere de ansamblu

Modelele de informaţii urmăresc să pună complexitatea informaţiilor din lumea reală într-un şablon care să poată fi mai bine folosit pentru înţelegere. Modelele de date trebuie să surprindă structura şi caracteristicile datelor, relaţiile dintre date, regulile de validare şi constrângerile aplicate datelor, precum şi toate transformările ce trebuie suportate de către acestea. Un astfel de model poate fi privit ca un instrument de comunicare dintre proiectanţii, programatorii şi utilizatorii unei baze de date. Pe piaţa de astăzi există mai multe tipuri de modele de date, fiecare având propriile caracteristici, dar în cadrul acestui capitol ne vom îndrepta atenţia către modelul relaţional de date care este cel mai folosit la ora actuală.

Principalele aspecte ale modelului relaţional de date sunt prezentate în *Figura 2.1*.

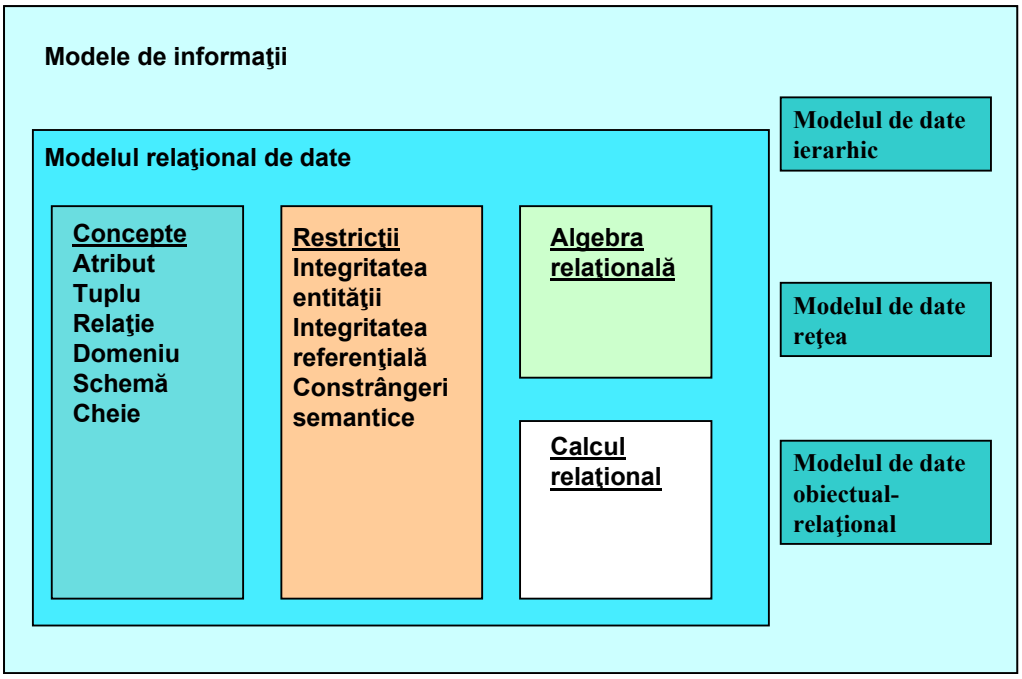


Figura 2.1 – Modelul relaţional de date în cadrul modelelor de informaţii: vedere de ansamblu

Figura 2.1 prezintă principalele aspecte ale modelului relaţional de date:

- ***Concepte*** specifice modelului relaţional de date: atribute, tupluri, domenii, relaţii, scheme, chei.
- ***Restricţiile (constrângerile)*** modelului relaţional de date: integritatea entităţii, integritatea referenţială, precum şi constrângerile de natură semantică care sunt folosite pentru a impune reguli în cadrul unei baze de date relaţionale.
- Operaţii din ***algebra relaţională*** precum reuniunea, intersecţia, diferenţa, produsul cartezian, selecţia, proiecţia, joncţiunea şi împărţirea care se folosesc pentru manipularea relaţiilor din cadrul modelului relaţional de date.
- ***Calculul relaţional*** care reprezintă o alternativă la algebra relaţională în ceea ce priveşte partea de manipulare a modelului.

2.2 Concepte de bază

Modelul relaţional de date foloseşte termeni formali pentru definirea conceptelor sale. În capitolele următoare se va folosi terminologia formală relaţională ce conţine concepte cum ar fi: atribut, domeniu, tuplu, relaţie, schemă, cheie candidat, **cheie primară** şi cheie externă. Să vedem în continuare definiţiile acestor termeni.

2.2.1 Atribute

Un ***atribut*** este o caracteristică a unei date. Fiecare caracteristică a unei date din lumea reală va fi modelată în baza de date sub forma unui atribut. Fiecare atribut trebuie să aibe un nume astfel încât să se poată face referire la acea caracteristică în orice alt moment este necesar, iar numele trebuie să fie cât mai relevant posibil. De exemplu, atributele unei persoane pot fi: **Nume, Sex, DataNasterii**. Termenii informali folosiţi pentru definirea unui atribut sunt: ***coloana*** în cazul unui tabel, sau ***câmp*** în cazul unui fişier.

În *figura 2.2* se pot observa caracteristicile unei maşini: **Tip, Producator, Model, AnFabricatie, Culoare, Combustibil**. Celelalte elemente care apar în figură vor fi discutate în secţiunile următoare.

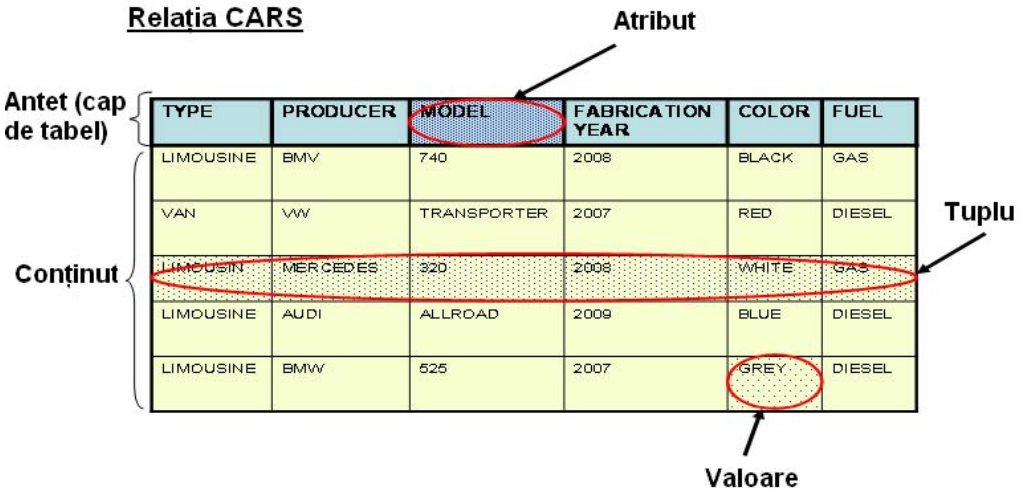


Figura 2.2 – Relaţia CARS – Antetul conţine atribute, iar conţinutul este alcătuit din tupluri

2.2.2 Domenii

Un ***domeniu*** este un grup de valori la nivel atomic care sunt toate de acelaşi tip. O ***valoare*** reprezintă cea mai mică unitate de date din modelul relaţional. De exemplu, **`BMW`**, **`Mercedes`**, **`Audi`**, şi **`VW`** sunt valori ale atributului **`Producer`**. Acele valori care nu mai pot fi descompuse în continuare sunt considerate a fi la nivel atomic. Domeniul atributului **`Producer`** este constituit din mulţimea numelor de producători de maşini. Un atribut are întotdeauna asociat un domeniu de valori. De exemplu, atributul de mai sus, conţine toate valorile posibile pentru numele de producători de maşini. Pe acelaşi domeniu se pot defini 2 sau mai multe atribute.

Orice domeniu trebuie să posede un nume, astfel încât să se poată face referire la el şi o mărime care este dată de numărul de valori pe care îl are domeniul. De exemplu, domeniul atributului **`Fuel`** are doar 2 valori (**`GAS, DIESEL`**). Un domeniu poate fi văzut ca un **`depozit de valori`**, din care sunt extrase valorile care apar în cadrul atributului

respectiv. De reţinut este faptul că în orice moment pot exista valori într-un anumit domeniu care nu sunt folosite de către nici unul dintre atributele ce aparţin domeniului respectiv.

Domeniile au o anumită semnificaţie operaţională. Dacă două atribute îşi preiau valorile din cadrul aceluiaşi domeniu, atunci operatorii de comparare folosiţi sunt operaţionali deoarece compară valori compatibile. În schimb, dacă două atribute îşi preiau valorile din domenii diferite şi se fac operaţii de comparare între valorile respective, compararea celor două atribute nu are sens.

Domeniile sunt în primul rând de natură conceptuală. În cele mai multe cazuri, acestea nu sunt în mod explicit stocate în baza de date. Domeniile se specifică ca parte a definiţiei bazei de date după care fiecare definiţie a unui atribut include o referinţă la domeniul corespunzător, astfel încât sistemul poate să identifice atributele care se pot compara unele cu altele.

Un atribut poate avea acelaşi nume ca şi domeniul său corespunzător, dar această situaţie trebuie evitată, deoarece se poate genera confuzie. Cu toate acestea, este posibil să se includă numele domeniului ca parte finală a numelui atributului în scopul de a oferi mai multe informaţii în numele atributului

2.2.3 Tupluri

Un ***tuplu*** este un grup ordonat de valori care descriu caracteristicile datelor în orice moment. În *Figura 2.2* de mai sus, se poate observa un exemplu de tuplu. Un alt termen formal folosit la definirea unui tuplu este ***n-tuplul***. Termenii informali folosiţi la denumirea tuplurilor sunt ***rând*** în cadrul unui tabel sau ***înregistrare*** în cadrul unui fişier ce conţine date.

2.2.4 Relaţii

Relaţia reprezintă partea centrală a modelului relaţional de date. În conformitate cu *introducerea în sistemele de baze de date* [2.1] o ***relaţie*** pe domeniile **`D1, D2, …, Dn`** (nu neapărat distincte) este alcătuită dintr-un antet şi un conţinut.

Antetul este alcătuit dintr-un grup fix de atribute **`A1, A2, …, An`**, astfel încât fiecare atribut **`Ai`** are exact un corespondent în domeniile respective **`Di (i=1, 2, …, n)`**.

Conţinutul este alcătuit dintr-un grup de tupluri variabil în timp, în care fiecrare tuplu conţine un număr de perechi atribut-valoare **`(Ai:vi) (i=1, 2, …, n)`**, câte o pereche pentru fiecare atribut **`Ai`** din antet. Pentru fiecare pereche atribut-valoare **`(Ai:vi)`**, **`vi`** există o valoare din domeniul unic **`Di`** care este asociată atributului **`Ai`**.

În *Figura 2.2* se poate observa relaţia **`CARS`**. Antetul acestei relaţii are un grup fix de 6 atribute: Type, Producer, Model, FabricationYear, Color, Fuel. Fiecare dintre aceste atribute are un domeniu de valori corespunzător. Conţinutul relaţiei este alcătuit dintr-un număr de tupluri (în figură sunt prezentate 5 tupluri, dar acest număr este variabil), fiecare dintre tupluri fiind alcătuit din 6 perechi atribut-valoare, câte unul pentru fiecare dintre cele 6 atribute din antet.

Gradul unei relaţii reprezintă numărul de atribute al relaţiei. Relaţia din F*igura 2.2* are gradul 6. O relaţie de gradul I este numită **`unară,`** o relaţie de gradul II este numită **`binară,`** o relaţie de gradul III **`ternară`** şi aşa mai departe. O relaţie de gradul n este numită **`n-ară.`**

Cardinalitatea unei relaţii reprezintă numărul de tupluri din relaţia respectivă. Relaţia din *Figura 2.2* are cardinalitatea egală cu 5. Cardinalitatea unei relaţii se modifică frecvent, în timp ce gradul relaţiei nu se modfică la fel de des.

Aşa cum se vede în relaţia respectivă, conţinutul este alcătuit dintr-un grup de tupluri al cărui număr se modifică în timp. La un moment dat, cardinalitatea relaţiei poate fi *m*, în timp ce după un timp cardinalitatea poate deveni *n*. Starea în care se află o relaţie la un moment dat este numită ***instanţa relaţiei***, ceea ce înseamnă faptul că pe parcursul existenţei unei relaţii pot exista foarte multe instanţe ale acesteia.

Relaţiile pot avea anumite proprietăţi importante. Aceste proprietăţi sunt consecinţe ale definiţiei relaţiei dată anterior. Cele 4 proprietăţi sunt:

- Într-o relaţie nu pot exista tupluri duplicat
- Într-o relaţie tuplurile nu sunt ordonate (de sus în jos)
- Într-o relaţie atributele nu sunt ordonate (de la stânga la dreapta)
- Toate valorile atributelor se află la nivel atomic

Denumirile informale folosite pentru termenul de relaţie sunt ***tabel*** sau ***fişier de date***.

2.2.5 Scheme

Schema unei baze de date este o descriere formală a tuturor relaţiilor existente în baza de date precum şi a tuturor asocierilor dintre acestea. În *Capitolul 3, Modelarea conceptuală a datelor*, şi în *Capitolul 4, Proiectul relaţional al unei baze de date*, se vor oferi mai multe informaţii despre schemele bazelor de date relaţionale.

2.2.6 Chei

Modelul relaţional de date apelează la chei pentru a defini identificatori asociaţi tuplurilor unei relaţii. Cheile sunt utilizate pentru a impune reguli şi/sau constrângeri pe datele bazei de date. Aceste constrângeri sunt esenţiale pentru păstrarea consistenţei şi corectitudinii datelor. Sistemele de gestiune a bazelor de date relaţionale permit definirea cheilor, iar sistemele de gestiune ale bazelor de date relaţionale trebuie să verifice şi să păstreze corectitudinea şi consistenţa datelor din baza de date. În continuare se va defini fiecare tip de cheie.

2.2.6.1 Cheie candidat

O ***cheie candidat*** reprezintă un identificator unic al tuplurilor unei relaţii. Prin definiţie, fiecare relaţie are cel puţin o cheie candidat (prima proprietate a unei relaţii). În practică, cele mai multe dintre relaţii au mai multe chei candidat.

C. J. Date [2.2] dă următoarea definiţie unei chei candidat:

Fie **R** o relaţie ce are atributele **A1, A2, …, An**. Mulţimea **K=(Ai, Aj, …, Ak)** a relaţiei **R** se numeşte cheie candidat a acesteia dacă şi numai dacă satisface următoarele două proprietăţi independente de timp:

- **Unicitatea**

  La un anumit moment, nu pot exista două tupluri distincte ale relaţiei **R** care să aibe aceeaşi valoare atât pentru atributele **Ai**, cât şi pentru atributele **Aj**, …, sau pentru atributele **Ak**.

- **Minimalitatea**

  Nici unul dintre atributele **Ai, Aj, …, Ak** nu pot fi eliminate din **K** fără a distruge proprietatea de unicitate.

Fiecare relaţie are cel puţin o cheie candidat, deoarece cel puţin combinaţia tuturor atributelor sale are proprietatea de unicitate (prima proprietate a unei relaţii), dar de obicei există cel puţin o altă cheie candidat alcătuită din mai puţine atribute ale relaţiei respective. De exemlu, relaţia **CARS** prezentată anterior în *Figura 2.2* are doar o singură cheie candidat **K=(Type, Producer, Model, FabricationYear, Color, Fuel)** având în vedere faptul că putem avea mai multe maşini care au aceleaşi caracteristici în cadrul relaţiei. Însă, dacă vom crea o altă relaţie **CARS** aşa cum se vede în *Figura 2.3* prin adăugarea altor două atribute **SerialNumber** (numărul serial al motorului) şi **IdentificationNumber** (numărul de identificare al maşinii) vom avea 3 chei candidat în cadrul relaţiei.

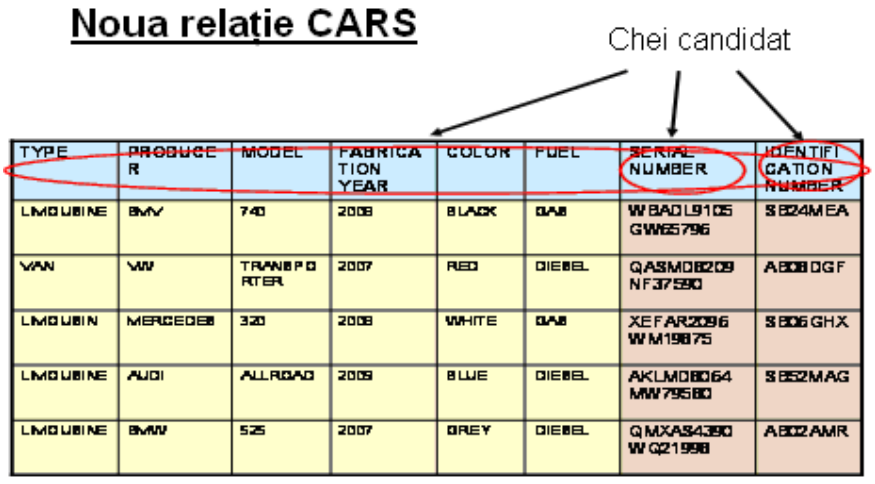


Figura 2.3 – Noua relaţie CARS şi cheile sale candidat

O cheie candidat mai este numită uneori şi ***cheie unică***. O cheie unică poate fi definită la nivelul limbajului de definire a datelor (DDL) folosind parametrul **UNIQUE** la definirea atributului respectiv. Dacă o relaţie are mai multe chei candidat, cheia aleasă să reprezinte relaţia este numită ***cheie primară,*** iar cheile candidat rămase sunt numite ***chei alternative***.

Observaţie:

Pentru a defini corect o cheie candidat trebuie luate în considerare toate instanţele relaţiei pentru a înţelege semnificaţia atributelor astfel încât să se poată stabili dacă este posibilă apariţia de duplicate pe parcursul existenţei relaţiei.

2.2.6.2 Chei primare

O ***cheie primară*** este un identificator unic pentru tuplurile din cadrul unei relaţii. Aşa cum s-a arătat anterior, cheia primară este o cheie candidat aleasă să reprezinte o relaţie din cadrul unei baze de date şi să ofere o modalitate de identificare unică a fiecărui tuplu al relaţiei. O relaţie din cadrul unei baze de date are întotdeauna o **cheie primară.**

Sistemele relaţionale de gestiune a bazelor de date permit specificarea unei **chei primare** din momentul creării relaţiei (tabelului). Sublimbajul de definire a datelor are o construcţie ajutătoare în acest sens, numită **`PRIMARY KEY.`** De exemplu, în cazul relaţiei **`CARS`** din *Figura 2.3* **cheia primară** este cheia candidat **`IdentificationNumber`**. Valorile acestui atribut trebuie să posede constrângerile "UNIQUE" şi "NOT NULL" pentru toate tuplurile tuturor instanţelor relaţiei.

Există situaţii în care caracteristici preluate din lumea reală a datelor, modelate pentru relaţia respectivă, să nu posede valori unice. De exemplu, prima relaţie **`CARS`** din *Figura 2.2* se află în această situaţie. Pentru a rezolva această problemă va trebui să se formeze **cheia primară** prin combinaţia tuturor atributelor relaţiei. O astfel de **cheie primară** nu este convenabilă din motive practice, deoarece ar necesita prea mult spaţiu fizic pentru păstrare, iar mentenanţa asocierilor dintre relaţiile bazei de date ar fi prea dificilă. În astfel de situaţii, soluţia adoptată este aceeea de a introduce un alt atribut, cum ar fi **`ID`**, care nu are nici o semnificaţie în lumea reală, dar care va avea valori unice şi poate fi folosit drept **cheie primară**. Acest atribut este de obicei denumit ***cheie surogat.*** Uneori, în literatura de specialitate, un astfel de atribut mai poate fi întâlnit şi sub denumirea de ***cheie artificială***.

Cheile surogat au de obicei valori numerice cu caracter de unicat care cresc sau descresc automat prin aplicarea unui increment (de obicei, cu pasul de 1).

2.2.6.3 Chei externe

O ***cheie externă*** este un atribut (sau o combinaţie de atribute) din cadrul unei relaţii R2 ale cărei valori trebuie să corespundă celor ale **cheii primare** din relaţia R1 (relaţiile R1 şi R2 nu trebuie să fie neapărat distincte). De notat este faptul că atât ***cheia externă*** cât şi **cheia primară** corespunzătoare trebuie să fie definite pe acelaşi domeniu de bază.

De exemplu, în *Figura 2.4* avem o altă relaţie numită **`OWNERS`** care conţine date despre proprietarii maşinilor din relaţia **`CARS`**.

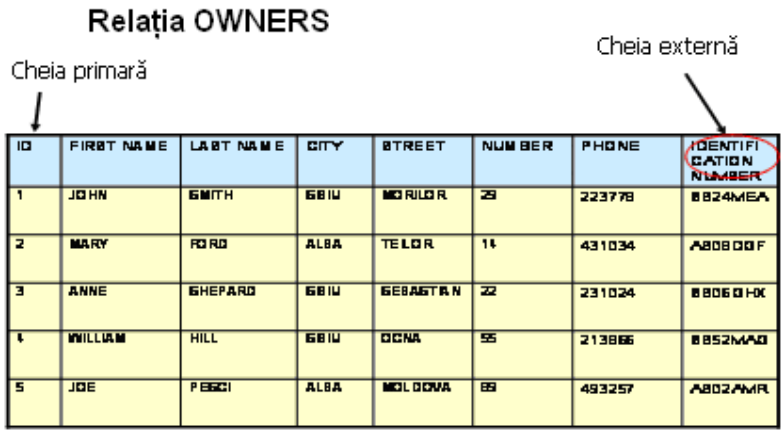


Figura 2.4 – Relaţia OWNERS şi cheile sale primară şi externă

Cheia externă `IdentificationNumber` din cadrul relaţiei `OWNERS` face referire la **cheia primară** `IdentificationNumber` din cadrul relaţiei `CARS`. În acest fel, putem afla persoana căreia îi aparţine maşina.

Corespondenţele dintre cheia primară şi cheia externă reprezintă asocierile stabilite între două relaţii şi alcătuiesc "`liantul`" cu care se păstrează integritatea bazei de date. Corespondenţele dintre cheia primară şi cheia externă reprezintă asocierile dintre tupluri, ceea ce înseamnă un alt fel de a spune acelaşi lucru. Trebuie precizat totuşi că nu toate asocierile de acest fel sunt reprezentate prin corespondenţe de tip cheie primară-cheie externă.

Sublimbajul de definire a datelor foloseşte de obicei o construcţie `FOREIGN KEY` pentru definirea cheilor externe. Pentru fiecare cheie externă trebuie precizată **cheia primară** şi relaţia căreia îi aparţine aceasta.

2.3 Constrângerile modelului relaţional de date

În cadrul unui model relaţional de date, integritatea datelor se poate păstra cu ajutorul regulilor de integritate sau a constrângerilor. Astfel de reguli au caracter general, sunt specificate la nivelul schemei bazei de date şi trebuie respectate de către fiecare instanţă a schemei. Dacă se doreşte obţinerea unei definiţii corecte a unei baze de date relaţionale, trebuie să se declare astfel de restricţii [2.2]. Dacă un utilizator încearcă să execute o operaţie care încalcă restricţia respectivă, atunci sistemul trebuie să respingă operaţia sau în cazuri mai complexe, să efectueze anumite operaţii de compensare într-o altă parte a bazei de date.

În acest fel se obţine asigurarea că rezultatul final se află în continuare într-o stare corectă. Să vedem în continuare care sunt constrângerile modelului relaţional de date.

2.3.1 Integritatea entităţii

Constrângerea ***integritatea entităţii*** afirmă faptul că nici un atribut care aparţine cheii

primare a unei relaţii nu poate să aibe valori nule.

O valoare ***null*** reprezintă o **`proprietate inaplicabilă`** sau o **`informaţie necunoscută`**. Null este un simplu marcator care arată absenţa unei valori sau existenţa unei valori nedefinite. Această valoare care reprezintă un concept, nu poate lua locul nici unei valori reale din cadrul unui anumit domeniu de valori folosit pentru atributul respectv. De exemplu, pentru atributul ce reprezintă culoarea unei maşini, null înseamnă că în acel moment nu se cunoaşte culoarea maşinii respective.

Justificarea pentru integritatea entităţii este:

- Relaţiile din cadrul unei baze de date corespund entităţilor din lumea reală şi prin definiţie entităţile din lumea reală sunt distincte, având un identificator unic de un anumit fel.
- **Cheile primare** îndeplinesc funcţia de identificare unică în cadrul modelului relaţional
- În acest caz, valoarea de null a unei **chei primare** se află în contradicţie de termeni deoarece s-ar putea spune că există anumite entităţi care nu au identitate, ceea ce nu se poate întâmpla.

2.3.2 Integritatea referenţială

Constrângerea de ***integritate referenţială*** arată faptul că dacă o relaţie **`R2`** conţine o cheie externă **`FK`** corespunzătoare **cheii primare** **`PK`** a altei relaţii **`R1`**, atunci fiecare valoare a **`FK`** din **`R2`** trebuie fie să fie egală cu valoarea **`PK`** din anumite tupluri ale relaţiei **`R1`** fie să fie în întregime nulă (fiecare valoare a unui atribut participant în acea valoare **`FK`** trebuie să fie null). **`R1`** şi **`R2`** nu trebuie să fie neapărat distincte.

Justificarea constrângerii de integritate referenţială este:

- Dacă un anumit tuplu **`t2`** ce aparţine relaţiei **`R2`** se asociază cu un alt tuplu **`t1`** ce aparţine relaţiei **`R1`**, atunci tuplul **`t1`** trebuie să existe, altfel legătura nu are sens.
- Din acest motiv, valoarea unei chei externe trebuie să corespundă unei valori a **cheii primare** undeva în cadrul relaţiei referite, dacă valoarea cheii externe nu este nulă.
- Uneori, din motive practice, este necesar să se accepte pentru cheia externă şi valori nule

De exemplu, în relaţia **`OWNERS`** cheia externă este **`IdentificationNumber`**. Valoarea acestui atribut trebuie să corespundă valorilor aflate în relaţia **`CARS`** deoarece o persoană trebuie să posede o maşină existentă pentru a deveni proprietar. Dacă cheia externă are valoare nulă, aceasta înseamnă faptul că persoana nu posedă încă o maşină, dar ar putea cumpăra una existentă.

Pentru fiecare cheie externă din baza de date, proiectantul bazei de date trebuie să răspundă la trei întrebări importante:

- Poate cheia primară să accepte valori nule?

De exemplu, este corect să avem un proprietar care să posede o maşină necunoscută? Trebuie făcută observaţia că răspunsul la o astfel de întrebare depinde de contextul situaţiei din lumea reală reprezentat în baza de date.

- Ce ar trebui să se întâmple dacă se încearcă eliminarea valorii **cheii primare** aflată într-o asociere cu o cheie externă?

  De exemplu, ce se întâmplă dacă se încearcă să se elimine o maşină care se află în proprietatea unei persoane?

  De obicei există trei posibilităţi:

1. CASCADE – se foloseşte operaţia de ştergere **"în cascadă"** pentru a elimina tuplurile corespondente (tuplurile din relaţia în care se află cheia externă). În cazul de faţă, dacă se elimină o maşină, se elimină şi proprietarul acesteia.
2. RESTRICT - se foloseşte operaţia de ştergere **"cu restricţie"** pentru situaţia în care nu există tupluri corespondente (altfel, nu se acceptă). În cazul de faţă, o maşină poate fi eliminată din listă doar dacă nu aparţine unei persoane.
3. NULLIFIES – cheia externă va lua valoarea nulă în toate situaţiile de corespondenţă, iar tuplurile ce conţin **cheia primară** sunt apoi eliminate (desigur, o astfel situaţie nu se poate aplica în cazul în care cheia externă nu poate avea valori nule). În cazul de faţă, maşina poate fi eliminată din listă doar după ce valoarea atributului **IdentificationNumber** corespunzătoare fostului proprietar a fost setată pe null.

- Ce se întâmplă atunci când se doreşte actualizarea valorii unei **chei primare** aflată în relaţie cu o cheie externă?

  De obicei, există, de asemenea, trei posibilităţi

1. CASCADE – operaţia de actualizare **"în cascadă"** actualizează valoarea cheii externe a tuplurilor corespondente (inclusiv a tuplurilor relaţiei în care se află cheia externă). În cazul de faţă, dacă numărul de identificare a unei maşini se modifică, atunci trebuie să se modifice şi numărul de identificare a maşinii ce aparţine unei persoane.
2. RESTRICT – operaţia de actualizare **"cu restricţie"** în cazul în care nu există tupluri corespondente (altfel, nu se acceptă). În cazul de faţă, numărul de identificare a unei maşini poate fi actualizat numai dacă aceasta nu aparţine unei persoane.
3. NULLIFIES – cheia externă se setează pe null în toate cazurile de corespondenţă, iar tuplurile ce conţin **cheia primară** sunt actualizate (desigur, acest lucru nu este valabil dacă cheia externă nu poate accepta valori nule). În cazul de faţă, numărul de identificare a unei maşini poate fi actualizat numai după ce valoarea atributului **IdentificationNumber** a fostului proprietar a fost setată pe valoarea nulă.

2.3.3 Constrângeri de integritate semantică

O constrângere de ***integritate semantică*** se referă la înţelesul real al datelor. De exemplu, valoarea atributului referitor la numărul unei străzi din relaţia **OWNERS** trebuie să fie pozitiv, deoarece aşa se întâmplă în lumea reală.

O constrângere de integritate semantică poate fi văzută sub forma unui predicat folosit pentru păstrarea stării de corectitudine a tuturor instanţelor unei relaţii din cadrul unei baze de date.

Dacă utilizatorul încearcă să execute o operaţie care nu respectă constrângerea, sistemul trebuie fie să respingă operaţia sau posibil, în anumite situaţii complicate, să efectueze acţiuni de compensare pe unele obiecte ale bazei de date pentru a obţine asigurarea că rezultatele existente se află într-o stare corectă. În acest fel, limbajul folosit pentru speicifcarea unei constrângeri de integritate semantică trebuie să conţină nu doar posibilitatea de a defini diverse predicate specifice, ci şi specificarea acţiunilor compensatorii atunci când este nevoie.

Constrângerile de integritate semantică se introduc, de obicei de către administratorul bazei de date şi trebuie păstrate în catalogul sistemului sau în dicţionarul de date. Sistemul de gestiune al bazei de date monitorizează interacţiunile utilizatorilor pentru a se asigura de faptul că aceste constrângeri sunt respectate. Sistemele relaţionale de gestiune a bazelor de date permit introducerea unor tipuri de constrângeri de integritate semantică cum ar fi constrângerile de domeniu, constrângerile pentru valori nule, constrângeri de unicitate, constrângeri de verificare a unor condiţii impuse.

2.3.3.1 Constrângeri de domeniu

O ***constrângere de domeniu*** implică faptul că un atribut al unei relaţii ia valori într-un anumit domeniu de valori. O astfel de constrângere verifică dacă valorile luate de către atributul respectiv se încadrează în domeniul de valori prestabilit.

De exemplu, domeniul de valori al atributului **Street** din cadrul relaţiei **OWNERS** este de tip **CHAR(20)**, deoarece străzile au de obicei nume, iar domeniul atributului **Number** este de tip **NUMERIC**, deoarece numerele străzilor au valori numerice.

Există o serie de forme particulare ale constrângerilor de domeniu, numite constrângeri de format sau constrângeri de interval de valori. O ***constrângere de format*** poate fi ceva de tipul unui şablon de valori pentru date.

De exemplu, valorile atributului **IdentificationNumber** trebuie să fie de formatul **XX99XXX**, în care **X** reprezintă o literă din alfabet, iar **9** reprezintă o cifră. O ***constrângere de interval de valori*** specifică intervalul de valori în care poate lua valori un anumit atribut. De exemplu, valorile atributului **FabricationYear** se pot afla în intervalul 1950 şi 2010.

2.3.3.2 Constrângeri de tip Null

O ***constrângere de tip Null*** specifică faptul că valorile atributului respectiv nu pot fi nule. Pentru fiecare tuplu al unei instanţe a unei relaţii atributul trebuie să aibă o valoare care să existe în domeniul din care acesta îşi ia valorile. De exemplu, valorile atributelor **FirstName** şi **LastName** nu pot fi nule, ceea ce înseamnă că proprietarul unei maşini

trebuie să posede un nume.

O constrângere de tip Null este de obicei specificată folosind construcţia **`NOT NULL`** care urmează după numele atributului. În afară de NOT NULL, se mai poate folosi opţional şi cuvântul cheie adiţional “WITH DEFAULT” pentru ca sistemul să introducă o valoare implicită a atributului în cazul în care acesta are valoare nulă la introducere. WITH DEFAULT se poate aplica doar valorilor numerice (integer, decimal, float etc.) şi de tip dată calendaristică (date, time, timestamp). Pentru alte tipuri de date, valorile implicite trebuie introduse în mod explicit cu ajutorul limbajului de definire a datelor.

2.3.3.3 Constrângere de unicitate

O ***constrângere de unicitate*** arată faptul că valorile unui atribut trebuie să fie diferite, nefiind posibil ca două tupluri din cadrul unei relaţii să posede aceleaşi valori pentru aceleaşi atribute. De exemplu, în relaţia **`CARS`** valorile atributului **`SerialNumber`** trebuie să fie unice deoarece nu este posibil ca să avem două maşini şi doar un singur motor. O constrângere de unicitate este de obicei introdusă prin intermediul unui nume de atribut urmat de cuvântul **`UNIQUE`**. Se observă faptul că NULL este o valoare unică acceptată.

Obs:

NULL nu face parte din nici un domeniu de valori, motiv pentru care rezultatul unei operaţii de comparare SQL este întotdeauna “unknown” deşi nici una dintre valorile implicate în cadrul operaţiei nu este nulă. Din acest motiv, pentru a putea gestiona corect valorile nule, SQL introduce două predicate speciale, “IS NULL” şi “IS NOT NULL” pentru a verifica dacă există valori nule sau nu. Mai jos este prezentată o tabelă de adevăr care arată felul în care sunt gestionate de către SQL valorile nule ("Unknown"). Platformele sau producătorii de baze de date pot avea diverse modalităţi de gestionare a valorilor nule.

A	B	A OR B	A AND B	A = B
True	True	True	True	True
True	False	True	False	False
True	Unknown	True	Unknown	Unknown
False	True	True	False	False
False	False	False	False	False
False	Unknown	Unknown	False	False
Unknown	True	True	Unknown	Unknown
Unknown	False	Unknown	False	False
Unknown	Unknown	Unknown	Unknown	Unknown

A	NOT A
True	False
False	True
Unknown	Unknown

2.3.3.4 Constrângeri de verificare

O ***constrângere de verificare*** introduce o condiţie (un predicat) pe datele unei relaţii, care

verifică întotdeauna datele cu care se lucrează. Predicatul arată ce trebuie verificat şi, opţional, ce trebuie făcut dacă nu este respectată condiţia (răspuns la încălcarea condiţiei). Dacă se primeşte un răspuns de încălcare a condiţiei, operaţia este anulată şi se întoarce un cod de eroare corespunzător. La aplicarea constrângerii, sistemul verifică dacă starea curentă a bazei de date rămâne corectă şi după impunerea constrângerii. Dacă acest lucru nu se întâmplă, constrângerea este respinsă, altfel este acceptată şi impusă permanent din acel moment.

De exemplu, salariul unui angajat nu poate fi mai mare decât salariul directorului administrativ sau un director de departament nu poate avea mai mult de 20 de persoane subordonate. Pentru relaţiile folosite anterior, de exemplu, anul de fabricaţie al unei maşini nu poate fi mai mare decât anul în care aceasta a intrat în posesia unei persoane.

Acest tip de constrângere poate fi introdus uneori în cadrul unei baze de date prin intermediul restricţiei **`CHECK`** sau a unui declanşator. Constrângerea de verificare poate fi testată de către sistem înainte sau după operaţii cum ar fi cele de inserare, actualizare sau ştergere.

2.4 Algebra relaţională

Algebra relaţională este aplicată prin intermediul unui grup de operatori folosiţi la manipularea relaţiilor. Fiecare operator al algebrei relaţionale se aplică pe una sau două relaţii la intrare producând o relaţie nouă la ieşire.

Codd [2.3] a definit 8 astfel de operatori, câte 2 grupuri a câte 4 fiecare:

- Operaţiile tradiţionale pe mulţimi: reuniunea, intersecţia, diferenţa şi produsul cartezian
- Operaţii relaţionale speciale: selecţia, proiecţia, joncţiunea şi împărţirea.

2.4.1 Reuniunea

Reuniunea a două relaţii compatibile la reuniune **`R1`** şi **`R2`**, **`R1 UNION R2`**, reprezintă mulţimea tuturor tuplurilor **`t`** care aparţin atât lui **`R1`** cât şi lui **`R2`** sau ambelor.

Două relaţii sunt ***compatibile la reuniune*** dacă au acelaşi grad, iar atributul **`i`** al fiecăreia aparţine aceluiaşi domeniu de valori.

Notaţia formală pentru operaţia de reuniune este U.

Operaţia **`UNION`** este asociativă şi comutativă.

Figura 2.5 prezintă un exemplu de operaţie **`UNION`**. Operanzii sunt relaţiile **`R1`** şi **`R2`** iar rezultatul este o altă relaţie **`R3`** care are 5 tupluri.

R1

Name	Age	Sex
A	20	M
C	21	M
B	21	F

R2

Name	Age	Sex
D	20	F
A	20	M
E	21	F

R3= R1 U R2

Name	Age	Sex
A	20	M
C	21	M
B	21	F
D	20	F
E	21	F

Figura 2.5 – Exemplu de operaţie UNION dintre două relaţii: R1 şi R2

2.4.2 Intersecţia

Intersecţia dintre două relaţii compatibile la reuniune **`R1`** şi **`R2`**, **`R1 INTERSECT R2`**, reprezintă mulţimea tuturor tuplurilor **`t`** care aparţin atât lui **`R1`** cât şi lui **`R2`**.

Notaţia formală pentru operaţia de intersecţie este ∩.

Operaţia **`INTERSECT`** este asociativă şi comutativă.

Figura 2.6 prezintă un exemplu de operaţie **`INTERSECT`**. Operanzii sunt relaţiile **`R1`** şi **`R2`** iar rezultatul este o altă relaţie **`R3`** cu un singur tuplu.

R1

Name	Age	Sex
A	20	M
C	21	M
B	21	F

R2

Name	Age	Sex
D	20	F
A	20	M
E	21	F

R3= R1 ∩ R2

Name	Age	Sex
A	20	M

Figura 2.6 – Exemplu de operaţie INTERSECT dintre două relaţii: R1 şi R2

2.4.3 Diferenţa

Diferenţa dintre două relaţii compatibile la reuniune **`R1`** şi **`R2`**, **`R1 MINUS R2`**, reprezintă mulţimea tuturor tuplurilor **`t`** care aparţin relaţiei **`R1`** dar nu şi relaţiei **`R2`**.

Notaţia formală pentru operaţia de diferenţă este -

Operaţia de **DIFERENŢĂ** nu este asociativă şi comutativă.

Figura 2.7 prezintă un exemplu al operaţiei de **diferenţă.** Operanzii sunt relaţiile **R1** şi **R2** iar rezultatul este o altă relaţie **R3** care are 2 tupluri. Aşa cum se poate observa, rezultatul dintre **R1-R2** este diferit faţă de **R2-R1**.

R1

Name	Age	Sex
A	20	M
C	21	M
B	21	F

R2

Name	Age	Sex
D	20	F
A	20	M
E	21	F

R3= R1 - R2

Name	Age	Sex
C	21	M
B	21	F

R3= R2 – R1

Name	Age	Sex
D	20	F
E	21	F

Figura 2.7 – Exemplu de operaţie de diferenţă dintre 2 relaţii: R1 şi R2

2.4.4 Produsul cartezian

Produsul cartezian dintre două relaţii **R1** şi **R2**, **R1 TIMES R2**, este mulţimea tuturor tuplurilor **t** în care **t** reprezintă concatenarea unui tuplu **r** din relaţia **R1** şi a unui tuplu **s** din relaţia **R2**. Prin concatenarea unui tuplu $r = (r_1, r_2, \ldots, r_m)$ cu un tuplu $s = (s_{m+1}, s_{m+2}, \ldots, s_{m+n})$ se obţine tuplul $t = (r_1, r_2, \ldots, r_m, s_{m+1}, s_{m+2}, \ldots, s_{m+n})$.

R1 şi **R2** nu trebuie să fie compatibile la reuniune.

Notaţia formală pentru o operaţie de produs cartezian este ×.

Dacă **R1** are gradul **n** şi cardinalitatea **N1** iar relaţia **R2** are gradul **m** şi cardinalitatea **N2** atunci rezultă că relaţia **R3** are gradul **(n+m)** şi cardinalitatea **(N1*N2)**. Acest lucru este prezentat în *Figura 2.8*.

R1

Name	Age	Sex
A	20	M
C	21	M

R2

Name	Age	Sex
D	20	F
E	21	F

R3= R1 X R2

Name	Age	Sex	Name	Age	Sex
A	20	M	D	20	F
C	21	M	D	20	F
A	20	M	E	21	F
C	21	M	E	21	F

Figura 2.8 – Exemplu de produs cartezian între două relaţii

2.4.5 Selecţia

Operaţia de ***selecţie*** alege o submulţime de tupluri dintr-o relaţie. Este un operator unar care se aplică pe o singură relaţie. Submulţimea tuplurilor trebuie să satisfacă condiţia de selecţie sau predicatul.

Notaţia formală pentru operaţia de selecţie este:

σ **<condiţia de selecţie> (<relaţia>)**

în care **<condiţia de selecţie>** este

<atribut> <operator de comparare> <constanta>/<atribut>
[AND/OR/NOT <atribut> <operator de comparare>
<constanta>/<atribut>…]

Operatorul de comparare poate fi **<, >, <=, >=, =, <>** şi depinde de domeniul de valori al atributului sau de tipul de dată al constantei.

Gradul relaţiei rezultate este egal cu cel al relaţiei iniţiale pe care se aplică operatorul. Cardinalitatea relaţiei rezultate este mai mică sau egală cu cardinalitatea relaţiei iniţiale. Dacă se constată o cardinalitate scăzută se spune că selectivitatea condiţiei de selecţie este ridicată, iar dacă cardinalitatea este mare se spune că selectivitatea condiţiei de selecţie este redusă.

Operaţia de selecţie este comutativă

În *Figura 2.9*, sunt două exemple de operaţii de selecţie efectuate pe relaţia **R**. Prima condiţie de selecţie este **Age=20** iar rezultatul este relaţia **R1** iar a doua condiţie de selecţie este **(Sex=M) AND (Age>19)** iar rezultatul este relaţia **R2**.

R

Name	Age	Sex
A	20	M
M	21	F
B	20	F
F	19	M
A	20	F
R	21	F
C	21	M

R1= σ(Age=20)(R)

Name	Age	Sex
A	20	M
B	20	F
A	20	F

R2= σ(Sex=M AND Age>19)(R)

Name	Age	Sex
A	20	M
C	21	M

Figura 2.9 – Exemplu de operaţie de selecţie (cu două condiţii diferite de selecţie)

2.4.6 Proiecţia

Operaţia de ***proiecţie*** produce o altă relaţie prin selectarea unei submulţimi de atribute dintr-o relaţie existentă. Tuplurile duplicat din relaţia rezultantă sunt eliminate. Şi aici avem de a face cu un operator unar.

Notaţia formală pentru operaţia de proiecţie este:

π **<lista de atribute> (<relatie>)**

în care **<lista de atribute>** reprezintă submulţimea atributelor unei relaţii existente.

Gradul relaţiei rezultate este egal cu numărul de atribute din **<lista de atribute>** deoarece doar acele atribute apar în relaţia rezultantă. Cardinalitatea relaţiei rezultante este mai mică sau egală cu cardinalitatea relaţiei iniţiale. Dacă lista de atribute conţine o cheie candidat a relaţiei, atunci cardinalitatea este egală cu cardinalitatea relaţiei iniţiale. Dacă nu conţine o cheie candidat, atunci cardinalitatea poate fi mai mică datorită posibilităţii apariţiei tuplurilor duplicat, care sunt eliminate din relaţia rezultantă.

Proiecţia nu este comutativă.

În *Figura 2.10* sunt prezentate două exemple de operaţii de proiecţie efectuate asupra relaţiei **R**. În prima, proiecţia este efectuată pe atributele **Name** şi **Sex** iar rezultatul este relaţia **R1** iar în a doua, proiecţia este efectuată pe atributele **Age** şi **Sex** iar rezultatul este relaţia **R2**.

R

Name	Age	Sex
A	20	M
M	21	F
B	20	F
F	19	M
A	20	F

R1= π(Name, Sex)(R)

Name	Sex
A	M
M	F
B	F
F	M
A	F

R2= π(Age, Sex)(R)

Age	Sex
20	M
21	F
20	F
19	M

Figure 2.10 – Exemplu de operaţie de proiecţie (două liste diferite de atribute)

2.4.7 Joncţiunea

Operaţia de ***joncţiune*** produce cuplarea a două relaţii pe baza unei condiţii de joncţiune sau a unui predicat. Relaţiile trebuie să posede cel puţin un atribut comun pe care se aplică condiţia de joncţiune care are acelaşi domeniu de valori în ambele relaţii.

Notaţia formală pentru operaţia de joncţiune este:

R `<condiţia de joncţiune>` ►◄**S**

în care **`<condiţia de joncţiune>`** este

`<atribut din R> <operator de comparare> < <atribut din S>`

Operatorii de comparare pot fi **`<, >, <=, >=, =, <>`** şi depind de domeniile de valori ale atributelor.

Dacă relaţia **`R`** are atributele **`A1, A2, …, An`**, iar relaţia **`S`** are atributele **`B1, B2, …, Bm`** şi atributele **`Ai`** şi **`Bj`** au acelaşi domeniu de valori se poate defini o operaţie de joncţiune între relaţia **`R`** şi relaţia **`S`** punându-se o condiţie de joncţiune între atributele **`Ai`** şi **`Bj`**. Rezultatul este o altă relaţie **`T`** care conţine toate tuplurile **`t`** astfel încât **`t`** reprezintă concatenarea unui tuplu **`r`** care aparţine relaţiei **`R`** şi a unui tuplu **`s`** care aparţine relaţiei **`S`** dacă condiţia de joncţiune este adevărată. Acest tip de operaţie de joncţiune mai este cunoscut şi sub denumirea de ***theta-joncţiune***. Aceasta corespunde definiţiei conform căreia rezultatul unei joncţiuni trebuie să conţină două atribute identice din punct de vedere al valorilor pe care le au. Dacă unul dintre cele două atribute este eliminat, joncţiunea este

numită ***joncţiune naturală***.

Mai există şi alte forme de operaţii de joncţiune. Cea mai des folosită este operaţia numită ***echijoncţiune***, care înseamnă că operatorul de comparare este =.

Există situaţii în care nu toate tuplurile unei relaţii R au un tuplu corespondent în relaţia S. Acele tupluri nu vor apare în rezultatul operaţiei de joncţiune dintre relaţiile R şi S. În practică, uneori, este necesar să apară toate tuplurile în rezultatul final, astfel încât a fost creat un alt tip de joncţiune: ***joncţiunea externă***. Există 3 forme de joncţiune externă: ***joncţiunea externă stânga***, în care toate tuplurile relaţiei R apar în rezultatul final, ***joncţiunea externă dreapta***, în care în rezultatul final apar toate tuplurile relaţiei S şi ***joncţiune externă completă***, în care apar în rezultatul final toate tuplurile ce aparţin atât relaţiei R cât şi relaţiei S. Dacă nu există tuplu corespondent, sistemul va lua în considerare un tuplu ipotetic cu toate valorile atributelor setate pe valoarea nulă, iar acesta va fi folosit la concatenare.

În *Figura 2.11* se pot observa două relaţii **`R1`** şi **`R2`** cuplate printr-o condiţie de joncţiune în care **`LastName`** din relaţia **`R1`** este egal cu **`LastName`** din relaţia **`R2`**. Relaţia rezultantă este **`R3`**.

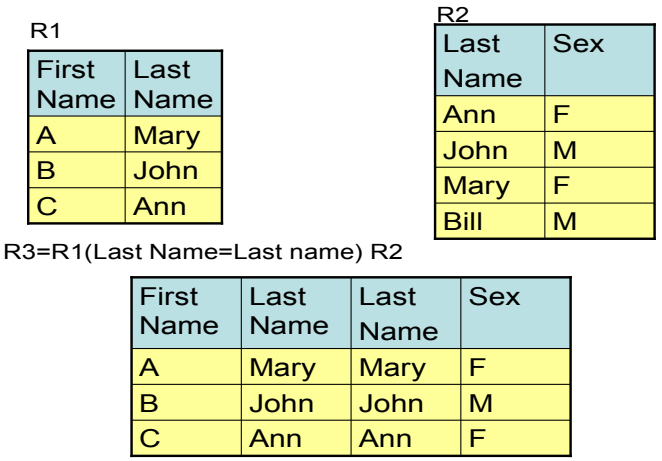


Figure 2.11 – Exemplu de operaţie de joncţiune

În *Figura 2.12* se pot vedea rezultatele unei joncţiuni naturale dintre **`R1`** şi **`R2`** şi rezultatul unei joncţiuni externe dreapta.

Joncțiune naturală

First Name	Last Name	Sex
A	Mary	F
B	John	M
C	Ann	F

Joncțiune externă dreapta

First Name	Last Name	Last Name	Sex
A	Mary	Mary	F
B	John	John	M
C	Ann	Ann	F
NULL	NULL	Bill	M

Figura 2.12 – Exemple de joncţiune naturală şi de joncţiune externă dreapta

2.4.8 Împărţirea

Operatorul de ***împărţire*** împarte relaţia **R1** de grad **(n+m)** la relaţia **R2** de grad **m** producând o relaţie de grad **n**. Atributul **(n+i)** al relaţiei **R1** şi atributul the **i** al relaţiei **R2** trebuie definite pe acelaşi domeniu. Rezultatul unei împărţiri dintre relaţia **R1** şi relaţia **R2** este o altă relaţie care conţine toate tuplurile care concatenate cu toate tuplurile relaţiei **R2** aparţin relaţiei **R1**.

Notaţia formală pentru operaţia de împărţire este ÷.

Figura 2.13 prezintă un exemplu de operaţie de împărţire dintre relaţia **R1** şi relaţia **R2.**

R1

Name	Sex
A	M
B	F
A	F
C	F
D	M
C	M

R2

Sex
M
F

R3= R1÷ R2

Name
A
C

Figura 2.13 – Exemplu de operaţie de împărţire

2.5. Calcul relaţional

Calculul relaţional este o alternativă la algebra relaţională fiind folosit pe partea de manipulare a modelului relaţional de date. Diferenţa dintre cele două este:

- Algebra relaţională introduce o colecţie de operaţii explicite, cum ar fi reuniunea, intersecţia, diferenţa, selecţia, proiecţia, joncţiunea etc. care se pot folosi cu scopul de **a crea** o anumită relaţie dorită pe baza unor relaţii existente în baza de date.
- Calculul relaţional oferă o notaţie cu ajutorul căreia se pot formula **definiţii** ale relaţiei dorite pe baza relaţiilor date.

De exemplu, să luăm interogarea *"Afişează numele complet şi oraşele proprietarilor de maşini care posedă o maşină roşie."*

Folosind algebra relaţională acest lucru se poate face:

- Se cuplează relaţia **OWNERS** cu relaţia **CARS** folosind atributele **IdentificationNumber**
- Alegerea din relaţia rezultantă doar a tuplurilor ale căror maşini au culoarea roşie: **Colour = "RED"**
- Se face o proiecţie pe rezultat astfel încât să se obţină doar atributele proprietarului maşinii **FirstName**, **LastName** şi **City**

Spre deosebire de algebra relaţională, calculul relaţional s-ar aplica astfel:

- Să se obţină atributele proprietarilor de maşini **FirstName**, **LastName** şi **City** astfel încât să existe o maşină cu acelaşi identificator **IdentificationNumber** şi care să posede culoarea roşie **RED**.

Aici, s-au declarat doar caracteristicile definitorii ale setului de tupluri dorit, sistemul fiind lăsat să decidă exact cum va fi făcut acest lucru. Se poate spune că formularea de calcul relaţional este *descriptivă* în timp ce algebra relaţională este *prescriptivă*. Calculul relaţional prezintă doar *care este problema* în timp ce algebra relaţională prezintă *cum trebuie soluţionată problema*.

De fapt, algebra relaţională şi calculul relaţional sunt echivalente, astfel încât pentru orice expresie din algebra relaţională există o expresie echivalentă în calculul relaţional şi invers. Între cele două există o corespondenţă de unu-la-unu, iar diferenţele nu reprezintă altceva decât diferenţe de stil. Calculul relaţional este mai apropiat de limbajul natural, în timp ce algebra relaţională este mai apropiată de un limbaj de programare.

Calculul relaţional se bazează pe o ramură a matematicii numită calcul **predicativ**. Kuhns [2.4] pare a fi părintele unei astfel de idei de folosire a calculului predicativ ca fundament al unui limbaj de lucru cu baze de date, dar Codd a fost primul care a propus conceptul de calcul relaţional ca tip de calcul predicativ destinat special bazelor de date relaţionale [2.3]. De asemenea, Codd a prezentat în [2.5] un limbaj explicit bazat pe calculul relaţional pe care l-a denumit **sublimbaj de date ALPHA** dar care nu a fost niciodată implementat în forma sa originală. Limbajul QUEL de la INGRES este astăzi foarte apropiat de sublimbajul de date ALPHA. Codd a prezentat şi un algoritm numit

algoritm de reducere, prin care o expresie oarecare din calculul relaţional poate fi convertită într-o expresie semantică echivalentă în algebra relaţională.

Există două tipuri de calcul relaţional:

- Calcul relaţional oprientat pe tupluri – se bazează pe conceptul de variabilă de tip tuplu
- Calcul relaţional orientat pe domeniu – se bazează pe conceptul de variabilă de tip domeniu

2.5.1 Calculul relaţional orientat pe tupluri

O ***variabilă de tip tuplu*** este acea variabilă care ia valori în cadrul unei relaţii. Valorile pe care le poate lua această variabilă reprezintă doar tupluri ale relaţiei respective. Cu alte cuvinte, dacă variabila **T** ia valori în relaţia **R**, atunci, în orice moment, **T** reprezintă un anumit tuplu **t** al relaţiei **R**.

O variabilă de tip tuplu se defineşte sub forma:

RANGE OF T IS X1; X2; …; Xn

în care **T** reprezintă variabila de tip tuplu, iar **X1, X2, …, Xn** sunt expresii de tip tuplu în calculul relaţional care reprezintă relaţiile **R1, R2, …, Rn**. Relaţiile **R1, R2, …, Rn** trebuie să fie toate compatibile la reuniune, iar atributele corespunzătoare trebuie denumite identic în fiecare dintre relaţii. Variabila de tip tuplu **T** ia valori în cadrul reuniunii relaţiilor respective. Dacă lista expresiilor de calcul relaţional de tip tuplu găseşte doar o singură relaţie **R** (situaţia obişnuită), atunci variabila de tip tuplu **T** ia valori doar din tuplurile existente în cadrul relaţiei respective.

Fiecare instanţă a unei variabile de tip tuplu poate fi **independentă** sau **cu legătură**. Dacă o variabilă de tip tuplu apare în contextul unei asocieri cu un atribut de forma **T.A**, în care **A** este un atribut al relaţiei în care **T** ia valori, aceasta este denumită variabilă de tip tuplu independentă. Dacă o variabilă de tip tuplu apare ca variabilă imediat după unul dintre cuantificatorii: **the existential quantifier** $\exists$ sau **the universal quantifier** $\forall$ aceasta este denumită variabilă de tip tuplu cu legătură.

O expresie de calcul relaţional de tip tuplu este definită astfel:

T.A, U.B, …, V.C WHERE f

în care **T, U, …, V** sunt variabile de tip tuplu, **A, B, …, C** sunt atribute ale relaţiilor asociate, iar **f** este o formulă de calcul care foloseşte variabilele independente **T, U, …, V**. Valoarea unei astfel de expresii se defineşte ca fiind o proiecţie a submulţimii produsului cartezian **T×U×…×V** (în care **T, U, …, V** iau toate valorile posibile) pentru care formula **f** este adevărată sau, dacă condiţia **"WHERE f"** nu este prezentă, o proiecţie a întregului porodus cartezian. Proiecţia se face pe atributele indicate prin notaţia cu punct **T.A, U.B, …, V.C**. Nici un element nu poate apare mai mult de o dată în cadrul listei respective.

De exemplu, interogarea *"Afişează numele complet şi oraşele proprietarilor de maşini care posedă o maşină roşie"* se poate reprezenta sub forma:

```
RANGE OF OWNERS IS
      OWNERS.FirstName, OWNERS.LastName, OWNERS.City WHERE
         ∃ CARS(CARS.IdentificationNumber=OWNERS.IdentificationNumber
                                        AND CARS.Color='RED')
```

Calculul relaţional orientat pe tupluri este în mod formal echivalent cu algebra relaţională.

Limbajul QUEL de la INGRES se bazează pe calculul relaţional orientat pe tupluri.

2.5.2 Calculul relaţional orientat pe domeniu

Lacroix şi Pirotte propun în [2.6] o alternativă la calculul relaţional numită calculul relaţional pe domeniu în care variabilele de tip tuplu sunt înlocuite cu variabile de domeniu. O variabilă de domeniu este acea variabilă care ia valori în cadrul unui anumit domeniu, nu în cadrul unei relaţii.

Fiecare instanţă a unei variabile de domeniu poate fi la rândul său **independentă** sau **cu legătură**. O variabilă de domeniu cu legătură apare imediat după variabilă împreună cu unul dintre cuantificatorii: **the existential quantifier** ∃ sau **the universal quantifier** ∀. În toate celelalte cazuri, variabila este o variabilă independentă.

Calculul relaţional orientat pe domeniu foloseşte **condiţii de apartenenţă**. O condiţie de apartenenţă are forma

R (termen, termen, …)

în care **R** este numele relaţiei, iar **termen** este o pereche de forma **A:v,** în care **A** este un atribut al **R** şi **v** este fie o variabilă de domeniu, fie o constantă. Condiţia este adevărată numai şi numai dacă există un tuplu al relaţiei **R** care să posede valorile specificate pentru atributele respective.

De exemplu, expresia

OWNERS(IdentificationNumber:'SB24MEA', City:'SIBIU') este o condiţie de apartenenţă care este adevărată dacă şi numai dacă există un tuplu al relaţiei **OWNERS** care să posede pentru atributul **IdentificationNumber** valoarea **SB24MEA** iar pentru atributul **City** valoarea **SIBIU**. Altfel, condiţia de apartenenţă

R (A:AX, B:BX, …)

este adevărată dacă şi numai dacă există un tuplu în **R** care să posede valoarea atributului **A** egală cu valoarea curentă a variabilei de domeniu **AX** (oricare ar fi), valoarea atributului **B** egală cu valoarea curentă a variabilei de domeniu **BX** (oricare ar fi) şi aşa mai departe.

De exemplu, interogarea *"Afişează numele complet şi oraşele proprietarilor de maşini care posedă o maşină roşie"* poate fi reprezentată sub forma:

```
FirstNameX, LastNameX, CityX WHERE ∃ IdentificationNumberX
(OWNERS (IdentificationNumber:IdentificationNumberX,
                    FirstName:FirstNameX, LastName:LastNameX, City:CityX)
```

```
    AND  CARS(IdentificationNumber:IdentificationNumberX, Color:'RED')
```

Calculul relaţional de domeniu este echivalent din punct de vedere formal cu algebra relaţională.

Lacroix şi Pirotte prezintă în [2.7] un limbaj numit ILL bazat pe calculul relaţional. Un alt limbaj relaţional bazat pe calculul relaţional de domeniu este Query-By-Example (QBE).

2.6 Rezumat

Acest capitol prezintă conceptele de bază ale modelului de date relaţional. Pentru o mai bună înţelegere, sunt explicate şi se prezintă exemple ale conceptelor : atribut, tuplu, relaţie, domeniu, schemă, cheie candidat, cheie primară, cheie alternativă şi cheie externă.

Capitolul mai prezintă şi constrângerile modelului relaţional de date. Au fost discutate o serie de tipuri de constrângeri, cum ar fi: integritatea entităţii, integritatea referenţială, integritatea semantică, prezentându-se avantajele şi rolul acestora în cadrul bazelor de date relaţionale.

S-au mai prezentat, de asemenea, în detaliu, operatorii folosiţi în algebra relaţională, cum ar fi reuniunea, intersecţia, diferenţa, produsul cartezian, selecţia, proiecţia, joncţiunea şi împărţirea. Este important de înţeles rolul jucat de către fiecare dintre aceşti operatori din algebra relaţională în mediul relaţional, deoarece toate interogările efectuate asupra unei baze de date relaţionale folosesc astfel de operatori.

Ca o alternativă la algebra relaţională a fost prezentat calculul relaţional pentru a putea lucra cu schema modelului relaţional de date. Diferenţele dintre acestea au fost prezentate în cursul capitolului de faţă, împreună cu calculul relaţional orientat pe tupluri care are în centru conceptul de variabilă de tip tuplu şi calculul relaţional orientat pe domeniu, care foloseşte conceptul de variabilă de domeniu.

2.7 Exerciţii

Pe parcursul acestui capitol s-au prezentat conceptele de bază folosite în cadrul modelului relaţional de date. Pentru a obţine o mai bună înţelegere a acestora să parcurgem următoarea situaţie preluată din lumea reală pe care să o modelăm apoi într-o bază de date relaţională:

- O companie are mai multe departamente. Fiecare departament are un număr corespunzător, un nume, un director administrativ, o adresă şi un buget. Două departamente diferite nu pot avea acelaşi identificator şi acelaşi director administrativ. Fiecare departament are un singur director administrativ şi angajaţi diferiţi. Pentru fiecare angajat trebie cunoscut numele, funcţia, salariul, data naşterii şi identificatorul personal care este unic.

Aceste informaţii se pot modela folosind o bază de date relaţională în care vor apare două relaţii:

- DEPARTMENTS
- EMPLOYEES

DeptNo, **DepName**, **Manager**, **Address**, **Budget** sunt atributele relaţiei **DEPARTMENTS**.

ID, **EmpName**, **Job**, **Salary**, **BirthDate**, **DepNo** sunt atributele relaţiei **EMPLOYEES**.

Fiecare atribut trebuie să posede un domeniu de valori asociat. De exemplu:

DEPARTMENTS	
DepNo	Numeric(2,0)
DepName	Character(20)
Manager	Numeric(3,0)
Address	Character(50)
Budget	Numeric(10,2)

EMPLOYEES	
ID	Numeric(3,0)
EmpName	Character(30)
Job	Character(10)
Salary	Numeric(7,2)
BirthDate	Date
DepNo	Numeric(2,0)

Fiecare relaţie trebuie să posede o **cheie primară**. Cheile candidat existente în cadrul relaţiei **DEPARMENTS** sunt: **DepNo** şi **Manager**. Una dintre acestea este **cheia primară** a relaţiei iar cealaltă va fi cheia alternativă. De exemplu, se poate alege **DepNo** drept **cheie primară** iar **Manager** drept cheie alternativă. **Cheia primară** a relaţiei **EMPLOYEES** este **ID**. Se vor folosi constrângerile de tip **cheie primară**. Unele dintre atribute nu pot avea valori nule, astfel încât trebuie folosite constrângeri **NOT NULL**.

În DB2 se pot crea aceste relaţii cu ajutorul comenzilor:

```
CREATE TABLE Departments (
      DepNo Numeric(2,0) NOT NULL PRIMARY KEY,
      DepName Char(20) NOT NULL,
      Manager Numeric(3,0) NOT NULL,
      Address Char(50),
      Budget Numeric(10,2)  );

CREATE TABLE Employees (
      ID Numeric(3,0) NOT NULL PRIMARY KEY,
      EmpName Char(30) NOT NULL,
      Job Char(10) NOT NULL,
      Salary Numeric(7,2),
      BirthDate Date NOT NULL,
      DepNo Numeric(2,0)  );
```

Între relaţiile **DEPARTMENTS** şi **EMPLOYEES** se va introduce o asociere. Fiecare angajat lucrează într-un sigur departament, ceea ce se reprezintă prin intermediul unei chei externe. Fie **DepNo** cheia externă a relaţiei **EMPLOYEES** care este asociată **cheii primare** **DepNo** din relaţia **DEPARTMENTS**.

În DB2 acest lucru se poate reprezenta folosind integritatea referenţială:

```
ALTER TABLE Employees ADD FOREIGN KEY (DepNo)
   REFERENCES Departments (DepNo)
   ON DELETE RESTRICT
```

```
ON UPDATE RESTRICT
ENFORCED ENABLE QUERY OPTIMIZATION;
```

Tuplurile relaţiilor **EMPLOYEES** şi **DEPARTMENTS** împreună cu valorile acestora, care respectă toate ipotezele de mai sus, pot fi văzute în *Figura 2.14*.

DEPARTMENTS Relation

DepNo	DepName	Manager	Address	Budget
2	Software	211	Bucharest	2000000,00
1	Accounting	422	Bucharest	500000,00
3	Hardware	111	Bucharest	4000000,00

EMPLOYEES Relation

ID	EmpName	Job	Salary	BirthDate	DepNo
211	John Smith	Engineer	2000,00	04/05/1966	2
123	Ann Adams	Accountant	1000,00	11/05/1975	1
311	Bill Jones	Programmer	1500,00	09/03/1980	2

Figura 2.14 – Relaţiile DEPARTMENTS şi EMPLOYEES

2.8 Întrebări recapitulative

1. Fie relaţia **FURNIZOR** cu patru atribute: **Id** – numărul de identificare al furnizorului (unic, nenul), **Name** – nume furnizor (nenul), **Address** – adresa furnizorului, **Discount** – reducerea de preţ oferită de furnizorul respectiv (nenulă, cu valori între 0 % şi 50 %). Arătaţi constrângerile necesare pentru definirea corectă a acestei relaţii.
2. Fie patru operaţii primitive algebrice: reuniunea, diferenţa, produsul cartezian, selecţia şi proiecţia. Celelalte se pot defini pe baza acestora. Daţi câte o definiţie pentru intersecţie, joncţiune şi împărţire folosind definiţiile operaţiilor de mai sus.
3. În cazul relaţiei definită anterior, la punctul 1, obţineţi numele furnizorului din New York care acordă o reducere de preţ mai mare de 5%, folosind operaţiile din algebra relaţională.

4. Se repetă întrebarea 3 folosind calculul relaţional orientat pe tupluri.
5. Se repetă întrebarea 3 folosind calculul relaţional orientat pe domeniu.
6. Care dintre următoarele afirmaţii definesc un atribut din punctul de vedere al modelului relaţional de date?
    A. O caracteristică din lumea reală modelată în cadrul bazei de date.
    B. O mulţime de valori atomice.
    C. O caracteristică a datelor.
    D. O mulţime ordonată de valori care descriu caracteristicile datelor.
    E. Nici una dintre cele enumerate
7. Care dintre următoarele sunt proprietăţi ale relaţiilor?
    A. Cheia primară nu poate avea valori nule.
    B. Într-o relaţie nu pot exista tupluri duplicat.
    C. Atributele au valori atomice.
    D. Într-o relaţie pot exista tupluri duplicat.
    E. Nici una dintre cele enumerate
8. O relaţie poate avea:
    A. Domeniu.
    B. Instanţă.
    C. Valoare.
    D. Grad.
    E. Nici una dintre cele enumerate
9. Care dintre următoarele afirmaţii este adevărată?
    A. Cheia primară este şi o cheie candidat.
    B. Fiecare relaţie are cel puţin o cheie externă.
    C. Cheile externe nu pot avea valori nule.
    D. Cheia primară poate fi şi cheie alternativă.
    E. Nici una dintre cele enumerate
10. La eliminarea unui tuplu dintr-o relaţie care are o cheie primară, care dintre următoarele opţiuni aplicate cheii externe va elimina toate tuplurile care au aceeaşi valoare în cadrul relaţiei ce conţine cheia externă?
    A. Restrict.
    B. Set to null.

C. Delete.

D. Cascade.

E. Nici una dintre cele enumerate

3

Capitolul 3 – Modelul conceptual de date

Acest capitol trece în revistă conceptele folosite la elaborarea unui model conceptual al bazei de date şi descrie în întregime modul de implementare al acestuia.

La încheierea acestui capitol ar trebui să puteţi:

- Oferi informaţiile necesare pentru a descrie cu acurateţe un anumit domeniu.
- Preveni greşelile şi neînţelegerile de comunicare.
- Ajuta la desfăşurarea discuţiilor.
- Stabili grupul de reguli folosite în domeniul care se modelează.
- Forma o bază solidă pentru proiectarea logică a bazei de date.
- Lua în considerare reguli şi legi care guvernează o anumită organizaţie.

3.1 Modelarea conceptuală, logică şi fizică: vedere de ansamblu

Termenii ***modelare conceptuală***, ***modelare logică*** şi ***modelare fizică*** sunt termeni des folosiţi atunci când se lucrează cu bazele de date.

Modelarea conceptuală prezintă informaţia aşa cum apare în cadrul unui anumit domeniu, identificând tipurile de entităţi şi relaţii din cadrul acestuia. Modelarea logică se bazează pe modelarea matematică, prezentând informaţiile şi entităţile într-o manieră complet normalizată în scopul eliminării redundanţelor. Modelarea fizică implementează modelul logic pentru un anumit produs şi versiune a unei baze de date. În cadrul acestui capitol ne vom concentra atenţia pe modelarea conceptuală, de la care încep toate, dar care trebuie văzută în relaţie cu modelarea logică şi fizică. La sfârşitul acestui capitol se va prezenta un studiu de caz de elaborare a unui model conceptual ce va fi transformat ulterior într-un model logic (Capitolul 4), şi apoi într-unul fizic (Capitolul 5).

Pentru a înţelege mai bine obiectivele clienţilor şi pentru a evita neînţelegerile ce ar putea apare se recomandă să se elaboreze modele care să pornească de la analiza cerinţelor. În acest scop IBM a creat o serie de instrumente ajutătoare cum ar fi Rational RequisitePro şi InfoSphere Data Architect. *Figura 3.1* prezintă modul în care cele trei categorii de modele, conceptual, logic şi fizic, se completează reciproc în cadrul ciclului de viaţă al modelării împreună cu instrumentele IBM care vin în sprijinul realizării acesteia.

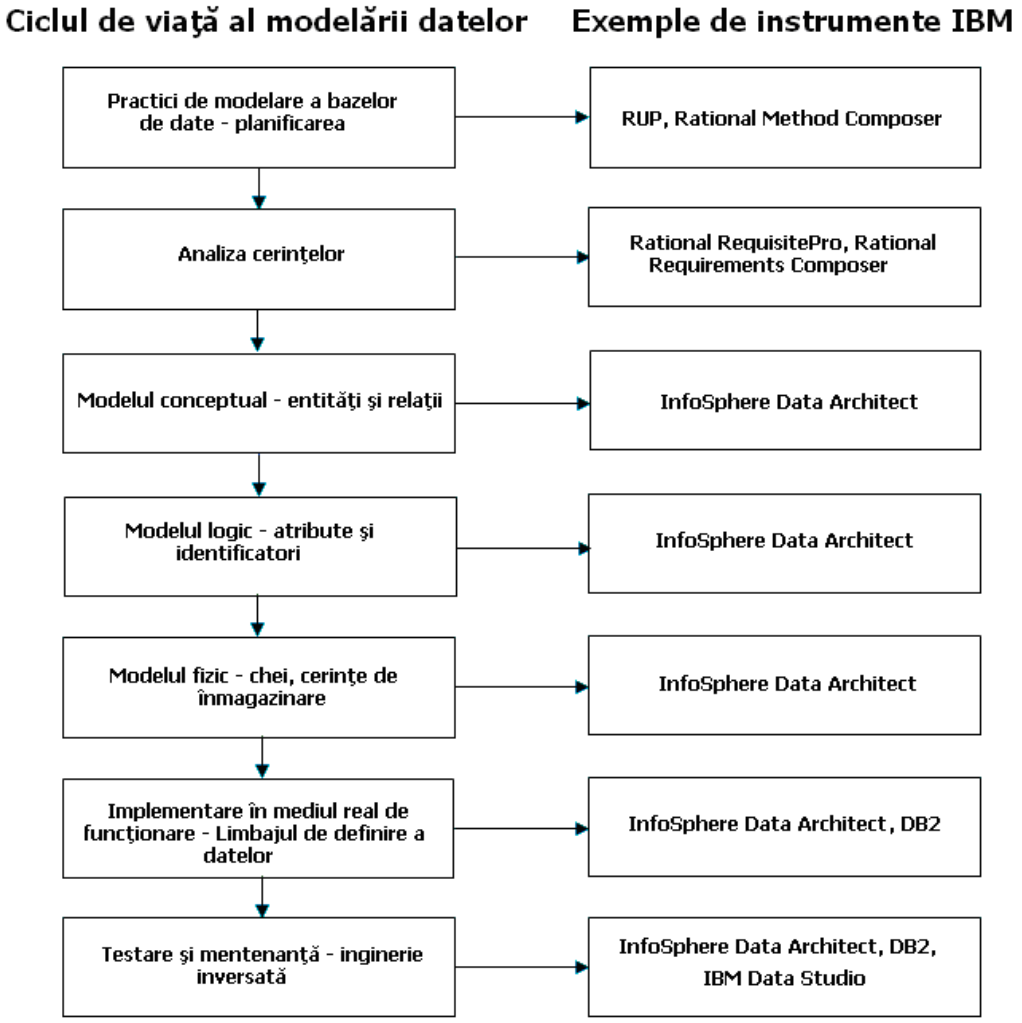


Figura 3.1 – Modelarea conceptuală, logică şi fizică în cadrul ciclului de viaţă al modelării datelor

Un model de date arată felul în care se reprezintă şi accesează datele, definind elementele de dată şi asocierile dintre acestea.

Datele reprezintă colecţii de litere, numere, fapte şi documente ce pot fi interpretate în diverse feluri. Datele sunt lipsite de semnificaţie dacă sunt luate ca atare, dar în momentul în care se prelucrează oferă informaţii de mare valoare. *Figura 3.2* prezintă o imagine sugestivă a relaţiei dintre date şi informaţii.

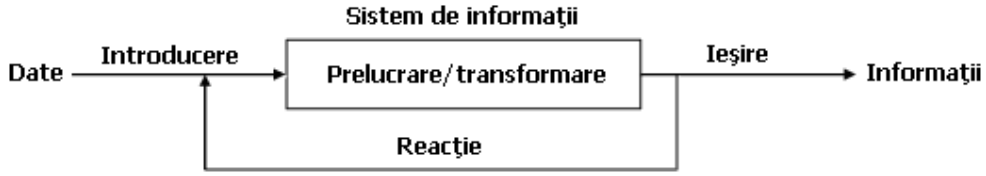


Figura 3.2 – Procesarea datelor

În această figură, datele reprezintă intrarea în sistemul de informaţii, unde se prelucrează şi se transformă, obţinându-se la ieşire informaţii. Un exemplu de date în contextul unei universităţi, poate fi numele unui student, iar un exemplu de informaţie în cadrul aceluiaşi context poate fi numărul acestora.

3.2 Ce este un model?

Un model reprezintă o abstractizare sau o reprezentare a lumii reale în care se introduc toate caracteristicile din cadrul domeniului de interes pentru utilizatorul de informaţii. Modelele se creează în scopul unei mai bune înţelegeri a unui proces, fenomen, sau activitate.

Modelele se utilizează, de obicei, pentru a oferi o reprezentare a obiectelor, evenimentelor şi tipurilor de relaţii dintre acestea precum şi pentru a introduce toate conceptele de bază dar şi regulile necesare unei bune comunicări dintre programatori şi utilizatori.

3.2.1 Model de date

Modelarea datelor stabileşte toate informaţiile necesare a fi obţinute din lumea reală. La modelarea datelor trebuie să se obţină asigurarea că acestea sunt reprezentate o singură dată în cadrul modelului. Aşa cum arată Connolly, “un model de date este un instrument folosit pentru abstractizarea datelor, a reprezentării în întregime a unei organizaţii precum şi pentru a ajuta utilizatorii să comunice clar şi precis referitor la datele existente în cadrul organizaţiei” [3].

Modelarea datelor reprezintă prima parte a procesului de proiectare şi elaborare a unei baze de date.

3.2.2 Modelul bazei de date

Modelul unei baze de date se foloseşte pentru a introduce date care descriu alte date şi care sunt cunoscute de obicei sub denumirea de metadate. Modelul unei baze de date este o colecţie integrată de concepte folosite la descrierea datelor, a relaţiilor dintre acestea, a semanticii datelor, precum şi a constrângerilor introduse. De obicei, modelul unei baze de date se folsoeşte la descrierea schemei bazei de date.

Tipurile de modele de baze de date sunt:

- Modelul extern de date. Acest model se foloseşte la obţinerea unei reprezentări a fiecărui utilizator, fiind cunoscut şi sub denumirea de univers al discursului. Principalul model folosit este modelul logic bazat pe înregistrări.

- Modelul conceptual de date. Acest model se foloseşte pentru a obţine o vedere generală asupra datelor şi este independent de orice implementare specifică a unui sistem de gestiune al bazelor de date. Modelul logic bazat pe obiecte este un astfel de model.
- Modelul intern de date. Acest model se foloseşte pentru a translata modelul conceptual de date într-un sistem specific de gestiune al bazelor de date. Modelul de date fizic este un astfel de model.

Cel mai răspândit model de date folosit astăzi este modelul relaţional al bazelor de date. Acest model este un model de tip entitate-relaţie şi foloseşte una dintre reprezentările modelului conceptual. Cele mai importante caracteristici ale modelului relaţional sunt cele referitoare la simplitate şi puternicul fundament teoretic oferit.

Componentele unui model de baze de date sunt

- Componenta structurală – care implică grupul de reguli comune folosite la elaborarea unei baze de date.
- Componenta de manipulare – stabileşte operaţiile ce pot fi utilizate în baza de date (pentru căutarea sau actualizarea datelor din baza de date, sau pentru modificarea structurii bazei de date).
- Componenta de integritate a datelor – un grup de reguli de integritate care asigură corectitudinea datelor.

3.2.3 Conceptele modelului conceptual de date

În cadrul procesului de elaborare a unui model de date trebuie creat mai întâi un model conceptual al datelor.

Modelul conceptual al datelor reprezintă o imagine mentală a unui obiect familiar fizic şi nu este specific nici unei baze de date reale. La nivel înalt, sunt descrise aspectele prin care o organizaţie doreşte să colecteze date precum şi asocierile dintre obiecte.

Obiectivul unui proiect conceptual al unei baze de date este acela de a construi un model conceptual de date. Pentru a face acest lucru se recomandă să se parcurgă următoarele etape:

1. Elaborarea diagramei entitate-relaţie. Mai întâi se creează tipurile de entităţi, stabilindu-se atributele şi tipurile de relaţii corespunzătoare.
2. Stabilirea constrângerilor de integritate. Identificarea şi documentarea constrângerilor de integritate, cum ar fi datele necesare, integritatea referenţială, constrângerile de domeniu ale atributelor, constrângerile organizaţei şi integritatea entităţii.
3. Revizuirea modelului final. În acest scop se elimină relaţiile de tip mulţi-la-mulţi, se elimină relaţiile recursive, se elimină supertipurile, se elimină relaţiile care au atribute şi se revăd relaţiile 1:1, care de obicei nu sunt necesare.

3.2.3.1 Modelul entitate-relaţie

Aşa cum s-a arătat în *Capitolul 1*, modelul entitate-relaţie a avut un mare succes din momentul în care a început să fie folosit sub forma unui instrument de proiectare a bazelor de date relaţionale. Pentru reprezentarea cerinţelor datelor în funcţie de tipul de bază de date cerut se foloseşte o diagramă entitate-relaţie. Diagrama entitate-relaţie este o formă de reprezentare a datelor structurate.

Conceptele folosite în modelul entitate-relaţie sunt:

- Tip de entitate
- Atribut
- Tip de relaţie
- Constrângere
- Domeniu
- Extensie
- Intensie

3.2.3.2 Entităţi şi tipuri de entităţi

Un tip de entitate reprezintă o mulţime de entităţi de acelaşi tip care au aceleaşi proprietăţi. Pentru reprezentarea unui tip de entitate se foloseşte un substantiv.

O entitate este o instanţă a unui tip de entitate. O entitate este un obiect, un concept sau fenomen de sine stătător şi bine definit din cadrul unui tip de entitate. O entitate poate fi:

- concretă (TEACHER sau STUDENT)
- abstractă (GRADE)
- un eveniment (EXAM)

Figura 3.3 arată exemple de entităţi şi tipuri de entităţi

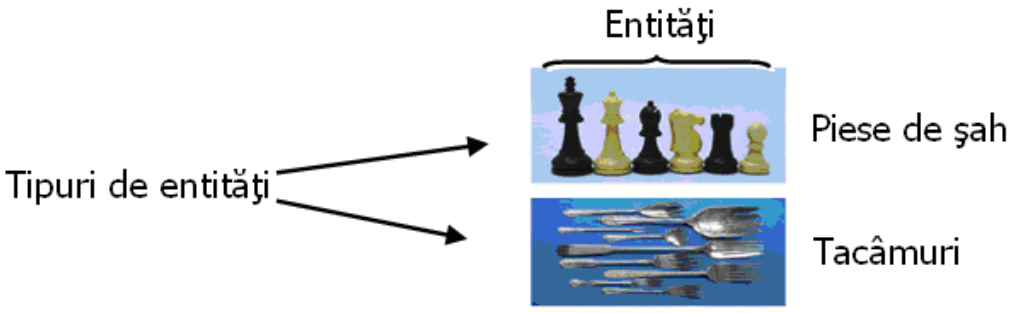


Figura 3.3 - Exemple de tipuri de entităţi şi entităţi

În funcţie de context, un anumit substantiv cum ar fi de exemplu TEACHER poate fi folosit atât ca entitate cât şi ca tip de entitate. De exemplu, dacă este nevoie să se reprezinte grupuri diferite de persoane, cum ar fi TEACHER sau STUDENT, se va crea un tip de

entitate numit PERSON care va avea entităţile TEACHER şi STUDENT.

Dacă este nevoie să se modeleze o universitate se va folosi TEACHER ca tip de entitate ce va avea entităţi cum ar fi PROFESSOR, ASSOCIATE PROFESSOR, ASSISTANT PROFESSOR.

Obs:

În modelul logic entităţile sunt numite tupluri, iar tipurile de entităţi sunt numite relaţii. De exemplu, se poate crea o relaţie numită PERSON care are două atribute: NAME şi ADDRESS. Din punct de vedere formal, acest lucru se va scrie cu ajutorul relaţiei PERSON=(NAME, ADDRESS)

3.2.3.3 Atribute

Un atribut este un tip de dată care descrie o proprietate a unui tip de entitate. Atributele determină, explică sau clasifică un tip de entitate.

Atributele au valori, care pot aparţine unor tipuri de date diverse, cum ar fi numere, şiruri de caractere, date calendaristice, imagini, sunete etc. Fiecare atribut poate avea o singură valoare pentru fiecare instanţă a unui tip de entitate.

În modelul fizic, atributele sunt numite coloane în cadrul unui tabel şi au un anumit domeniu de valori. Fiecare tabel are o listă de atribute (sau coloane).

Există următoarele tipuri de atribute:

- Simple (atomice) – Acest tip de atribut are o singură componentă. De exemplu, atributul **`Gender`** are o singură componentă şi poate lua doar două valori.
- Compuse – Un atribut compus este alcătuit din mai multe componente. De exemplu, atributul **`Name`** are componentele nume de familie şi prenume.
- Cu o singură valoare – Acest tip de atribut are o singură valoare pentru fiecare entitate. De exemplu, atributul **`Title`** are o singură valoare pentru fiecare cadru didactic.
- Cu valori multiple – Un atribut multivaloare are mai multe valori pentru fiecare entitate. De exemplu, o persoană poate avea mai multe numere de telefon. Conform definiţiei, fiecare atribut poate avea o singură valoare pentru fiecare instanţă a unui tip de entitate. În momentul în care se constată că avem de a face cu un atribut cu valori multiple, acesta va trebui transformat într-un nou tip de entitate.
- Derivat – Un atribut derivat are valorile obţinute pe baza calculelor făcute cu valorile altui atribut sau atribute. De exemplu, suma tuturor studenţilor dintr-o facultate. Atributul derivat nu trebuie să apară în nici unul dintre tabelele bazei de date, dar este folosit în modelul conceptual din motive de claritate sau este folosit în model pentru a putea oferi informaţii suplimentare programatorilor.
- Variabile – Acest tip de atribut are valori care se modifică în timp. De exemplu, anul

de studii al unui student.

- Constante – Acest tip de atribut îşi modifică valoarea foarte rar, dacă nu deloc.

> **Obs:**
>
> Se recomandă folosirea doar a atributelor constante; de exemplu, folosiţi data de început a studiilor unui student şi nu anul de studii.

- Obligatorii – Atributele obligatorii trebuie să posede neapărat o valoare. De exemplu, în situaţiile în care se urmăresc datele personale ale cuiva, este necesar să existe o valoare în câmpul **`Name`**.

> **Obs:**
>
> Folosind InfoSphere Data Architect, ca instrument de modelare a bazelor de date, veţi găsi opţiunea Required pe fila Attributes de pe pagina Properties în momentul selectării unui tip de entitate din diagrama entitate-relaţie. Acest lucru este prezentat în *Figura 3.4* de mai jos.

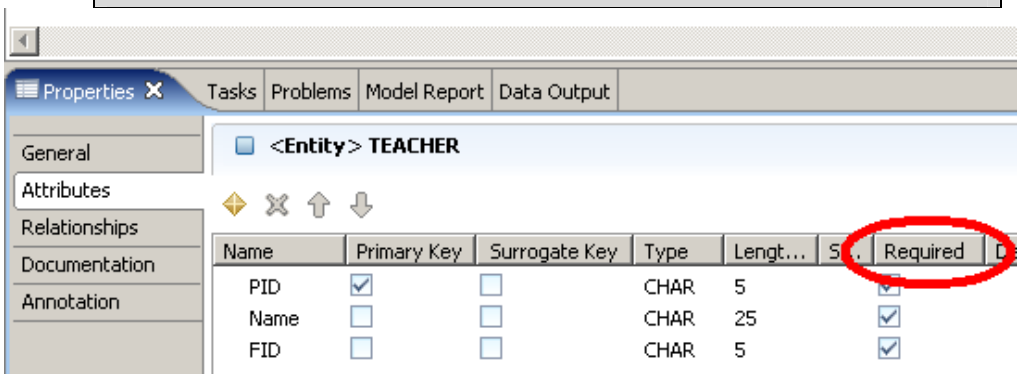


Figure 3.4 – Stabilirea unui atribut ca fiind obligatoriu

- Opţional – Atributele opţionale pot avea sau nu valori.
- Identificator unic – Acest tip de atribut are rolul de a deosebi o entitate de alta. De exemplu, într-o sală de curs se poate deosebi un student de altul prin utilizarea unui identificator unic.

În modelul logic se va folosi un alt concept pentru identificatorul unic, numit ***cheie***. O cheie este un atribut sau un grup de atribute dintr-o relaţie care are/au o valoare unică pentru fiecare tuplu al relaţiei. Cheia se foloseşte cu scopul de a obţine asigurarea că în cadrul unei relaţii nu se pot întâlni tupluri duplicat. Există mai multe tipuri de chei, fiecare dintre acestea având propriile caracteristici:

- **Cheie candidat** – O cheie candidat este un atribut sau un grup de atribute care identifică în mod unic un tuplu dintr-o relaţie.
- **Cheie primară** – O cheie primară este una dintre cheile candidat din cadrul unei

relaţii. Orice relaţie trebuie să posede o cheie primară care trebuie să îndeplinească cel puţin condiţiile de:

- Stabilitate. Valoarea unei chei primare nu trebuie să se modifice sau să devină nulă pe toată existenţa entităţii respective. De exemplu, în cazul unui student folosirea vârstei drept cheie primară nu este potrivită, deoarece aceasta se modifică în timp.
- Minimalitate. Cheia primară trebuie să fie alcătuită din cât mai puţine atribute care să asigure unicitatea.

- **Cheie alternativă** – O cheie alternativă este una dintre cheile candidat care nu a fost desemnată să îndeplinească rolul de cheie primară. Aceasta poate deveni în orice moment cheie primară dacă cea aleasă anterior nu mai este corespunzătoare.
- **Cheie surogat** – O cheie surogat acţionează la fel ca o cheie primară, dar aceasta nu există ca entitate în lumea reală. Deoarece cheia surogat nu spune nimic despre relaţia pentru care a fost desemnată, introducând date fără semnificaţie care conduc la creşterea artificială a bazei de date, se recomandă ca ori de câte ori este posibil să se renunţe la utilizarea acesteia. Cheile surogat pot fi definite atunci când se modelează o bază de date folosind instrumentul ajutător InfoSphere Data Architect, aşa cum se arată în *Figura 3.5*.

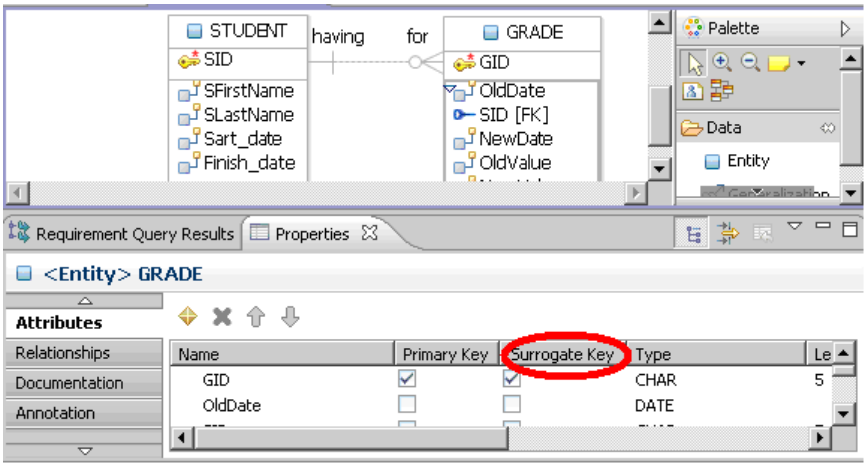


Figura 3.5 – Introducerea unei chei surogat

- **Chei simple** – Aceste chei sunt alcătuite dintr-un singur atribut.
- **Chei compuse** – Aceste chei sunt alcătuite din mai multe atribute.
- **Chei externe** – Aceste chei se folosesc, de obicei, atunci când există două sau mai multe relaţii în baza de date. Un atribut din cadrul unei relaţii trebuie să aibă un corespondent în cadrul celeilalte/celorlalte relaţii. Între cele două atribute se creează un tip de relaţie. O situaţie specială se întâlneşte la folosirea unui tip de relaţie unară. În acest caz, trebuie folosită o cheie externă în cadrul aceleiaşi relaţii.

3.2.3.4 Tipuri de relaţii

Un tip de relaţie reprezintă o mulţime de asocieri dintre două sau mai multe tipuri de entităţi şi se reprezintă de obicei prin folosirea unui verb. O relaţie reprezintă o instanţă a unui tip de relaţie stabilind legătura dintre tipurile de entităţi aflate în asociere. Aceste relaţii trebuie să reprezinte ceva important din cadrul modelului, deoarece între două tipuri de entităţi se pot stabili mai multe asocieri.

Un tip de relaţie se stabileşte între două tipuri de entităţi (sau între un tip de entitate şi el însuşi). Fiecare tip de relaţie trebuie citit în ambele sensuri, de la un tip de entitate către celălalt şi invers.

Un tip de relaţie se poate defini în mod formal, sub forma unei relaţii matematice între tipurile de entităţi pe care le asociază, aşa cum se prezintă în continuare. Pentru a realiza acest lucru se vor folosi următoarele variabile:

Tip de entitate: ***E***

Entitate: ***e***

Tip de relaţie: ***R***

Relaţie: ***r***

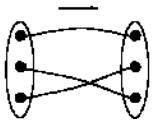
Având o mulţime de tipuri de entităţi $E_1, E_2 \ldots E_k$, o relaţie ***R*** stabileşte o regulă de corespondenţă între aceste tipuri de entităţi. O instanţă $r(E_1, E_2, \ldots, E_k)$ a relaţiei ***R*** arată că, la un anumit moment dat, în relaţia ***R*** se află tipurile de entităţi $E_1, E_2 \ldots E_k$

3.2.3.5 Constrângeri

În orice domeniu de activitate sunt permise doar anumite valori ale atributelor folosite. Pentru a face acest lucru, în modelul conceptual se folosesc constrângerile. O constrângere reprezintă o cerinţă pe care trebuie să o îndeplinească un tip de entitate în cadrul unei relaţii. Constrângerile pot face referire la un singur atribut al unui tip de entitate sau la un tip de relaţie dintre două tipuri de entităţi. De exemplu, "fiecare ***TEACHER*** trebuie să funcţioneze doar într-un singur ***DEPARTMENT***".

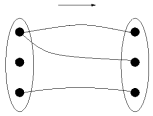
Tipurile de constrângeri sunt:

- Cardinalitatea – reprezintă numărul de tipuri de relaţii posibile pe care le poate avea fiecare tip de entitate. Tipurile de constrângeri de cardinalitate sunt prezentate mai jos. Fie o relaţie binară ***R*** între ***E*** (tipul de entitate din stânga din figura de mai jos) şi ***F*** (tipul de entitate din dreapta din figura de mai jos). Se spune că ***R*** poate fi:
  - Unu-la-unu (***1:1***) – dacă atât ***E*** cât şi ***F*** au câte o singură valoare participantă în relaţie, aşa cum se vede în figura de mai jos.

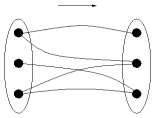


  - Unu-la-mulţi (***1:M***) - dacă ***E*** are o singură valoare iar ***F*** are mai multe valori

participante în relaţie, aşa cum se vede din figura de mai jos.



- Mulţi-la-mulţi (***M:M***) – dacă atât ***E*** cât şi ***F*** au mai multe valori participante în relaţie.



Tipurile de relaţii mulţi-la-mulţi nu sunt suportate în modelul relaţional de date şi trebuie rezolvate prin separarea tipului de relaţie original ***M:M*** în două tipuri de reaţie ***1:M***. De obicei, identificatorii unici ai celor două tipuri de entităţi participă la crearea identificatorului unic al celui de-al treilea tip de entitate.

- Participarea (opţionalitatea). Acest tip de constrângere arată dacă existenţa unui tip de entitate depinde de legătura acestuia cu un alt tip de entitate printr-un tip de relaţie. Participarea poate fi:
  - Totală sau obligatorie: Fiecare tip de entitate trebuie să participe într-o relaţie şi nu poate exista fără acea participare; participarea este obligatorie.
  - Partială sau opţională: Fiecare tip de entitate poate participa într-o relaţie; participarea nu este obligatorie.

Figura 3.6 prezintă atât constrângerile de cardinalitate cât şi pe cele de participare.

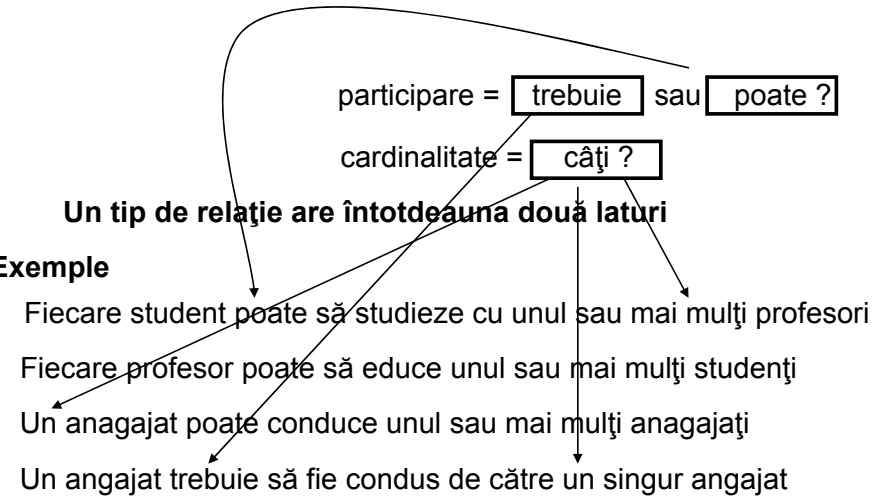


Figura 3.6 – Cardinalitatea şi participarea

- Subtipuri şi supertipuri

Atunci când un grup de instanţe are proprietăţi speciale cum ar fi atribute sau tipuri de relaţii care există doar pentru grupul respectiv, se recomandă să se împartă tipul de entitate corespunzător în subtipuri. Tipul de entitate de la nivelul superior este numit părinte şi reprezintă un supertip. Fiecare dintre grupuri reprezintă subtipuri şi se numesc copii.

Un subtip este alcătuit din toate atributele supertipurlui şi preia toate tipurile de relaţii ale acestuia. Pentru aputea exista, un supertip trebuie să aibă cela puţin două subtipuri asociate, care la rândul lor pot avea subtipuri asociate. Un subtip, de obicei, adaugă propriile atribute sau tipuri de relaţii faţă de cele moştenite de la supertip. Cu ajutorul supertipurilor şi subtipurilor se pot crea ierarhii.

Examplu

Un tip de entitate ***PERSON*** poate avea două subtipuri ***STUDENT*** şi ***TEACHER*** aşa cum se vede în *Figura 3.7* de mai jos.

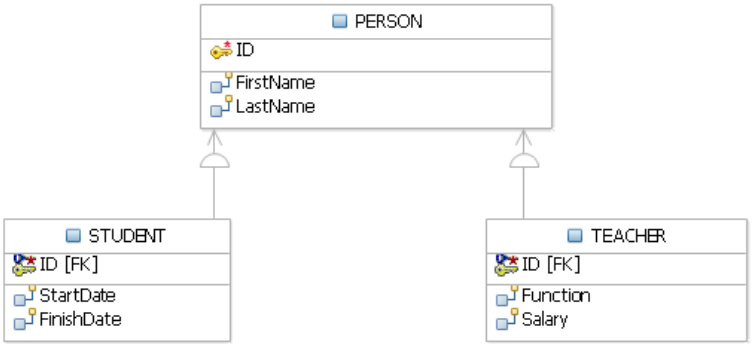


Figura 3.7 – Supertipul *PERSON* împreună cu subtipurile asociate *STUDENT* şi *TEACHER*

- Ierarhia

O ierarhie reprezintă o mulţime ordonată de obiecte. De exemplu, într-o şcoală, poate exista o ierhie de tipul celei prezentate în *Figura 3.8*.

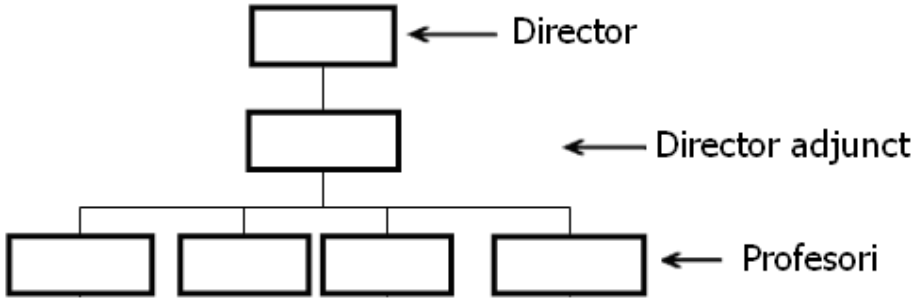


Figura 3.8 – Exemplu de ierarhie dintr-o şcoală

- Tipuri de relaţii unare – În situaţia acestui tip de constrângere, acelaşi tip de entitate

participă de mai multe ori în acelaşi tip de relaţie. Mai este cunoscut şi sub denumirea de tip de relaţie recursivă.

Exemplu

Directorul adjunct este şi el un profesor, astfel încât se poate realiza o asociere între profesorii subordonaţi folosind un tip de relaţie între tipul de entitate TEACHER şi el însuşi. Acelaşi tip de entitate participă de mai multe ori în acelaşi tip de relaţie. *Figura 3.9* arată un astfel de exemplu.

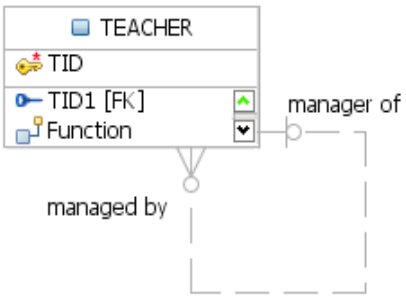


Figura 3.9 – Exemplu de tip de relaţie unară

- Date istorice

În anumite situaţii este nevoie să se păstreze datele folosite de-a lungul timpului în baza de date. De exemplu, la modificarea valorilor atributelor, sau atunci când se modifică tipurile de relaţii. Datele istorice se pot folosi sub forma unei constrângeri de timp. De exemplu, constrângerea poate specifica faptul că o dată calendaristică de final trebuie să urmeze întotdeauna după o dată calendaristică de început a unei activităţi oarecare. În modelul fizic se poate folosi o constrângere de tip `CHECK` în acest scop. Constrângerile de acest tip trebuie să aibă în vedere şi faptul că data de început nu poate fi modificată la o dată anterioară faţă de momentul începerii unei activităţi, sau data de început şi cea de sfârşit a unei activităţi nu pot fi identice.

Exemplu

Un student trebuie să termine studiile după data începerii acestora, nu înainte. În acest caz, data de final a studiilor unui student trebuie să fie după data de început a acestora.

> **Obs**:
>
> Pentru a introduce o constrângere de verificare, vedeţi secţiunea *Adding constraints* din cartea *Getting Started with InfoSphere Data Architect*, care face parte din seria actuală de cărţi.

Mai mult decât atât, se poate întâmpla ca atunci când se modifică datele existente, să se dorească să se păstreze o copie a valorilor anterioare. Acest lucru este posibil să se ceară atunci când datele au un caracter sensibil, cum ar fi, de exemplu, modificarea notelor unui student.

Exemplu

La modificarea notelor unui student este necesar să se păstreze înregistrarea a ceea ce se modifică, adică nota veche, nota nouă şi persoana care a făcut modificarea. *Figura 3.10* arată un exemplu de jurnalizare.

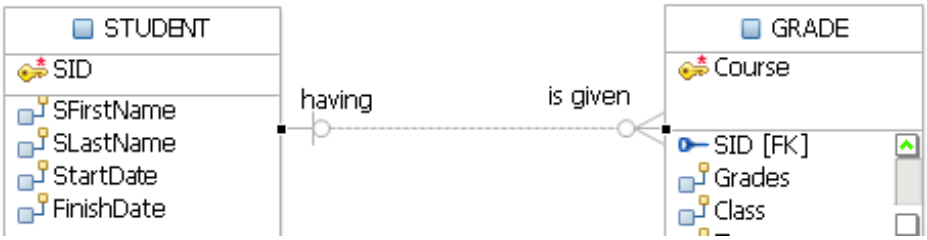


Figura 3.10 – Exemplu de jurnalizare

3.2.3.6 Domeniul de valori

Un domeniu de valori reprezintă mulţimea valorilor posibile ce poate fi asociată unuia sau mai multor atribute. Utilizatorul poate stabili domeniul de valori pe care îl doreşte. Modelele de domeniu permit utilizatorului stabilirea tipurilor de date necesare în domeniul respectiv.

Obs

Folosind InfoSphere Data Architect, este posibil să se folosească modele de domeniu pentru a restrânge valorile şi domeniile respective la nivelul tipurilor de date.

Clauza **`CHECK`** din standardul SQL permite introducerea de restricţii pe domenii. Clauza **`CHECK`** permite proeictantului schemei să introducă un predicat care trebuie validat de către orice valoare atribuită unei variabile al cărui tip se află în domeniul respectiv.

3.2.3.7 Extensia

Bazele de date se modifică în timp. Datele aflate într-o bază de date la un anumit moment dat formează extensia bazei de date. Extensia reprezintă mulţimea actuală de tupluri din cadrul unei relaţii şi este o instanţă a mulţimii de înregistrări şi de tipuri de relaţii dintre acestea.

3.2.3.8 Intensia

Intensia sau schema reprezintă modelul logic al unei baze de date şi se reprezintă prin intermediul tipurilor de entităţi. O instanţă prezintă şi restrânge structura tuplurilor permise. O instanţă este reprezentată prin intermediul tipurilor de entităţi şi este modelul unei baze de date. Operaţiile de manipulare ale datelor sunt permise numai dacă acestea respectă intensiile relaţiilor pe care le modifică.

3.3 Studiu de caz cu sistemul de management al unei biblioteci - Partea 1 din 3

Acest studiu de caz prezintă un sistem de management folosit în cadrul unei biblioteci. Se presupune că toate cerinţele sistemului au fost specificate de către client exact în aceşti

termeni:

"*Sistemul va gestiona informaţii referitoare la autori şi la cei care împrumută cărţi, păstrând un istoric al cărţilor împrumutate. Informaţiile referitoare la cei care împrumută cărţi au în vedere numele, adresa, e-mail-ul şi numărul de telefon. Informaţiile despre autori se referă la nume, adresă şi e-mal..*

Se pot introduce în sistem toate cărţile noi, cu autorii respectivi şi cu informaţiile referitoare la cei ce împrumută cărţi. Atunci când o persoană împrumută o carte, sistemul va înregistra data la care s-a împrumutat cartea respectivă şi va calcula numărul de zile până la care cartea trebuie returnată. Dacă cititorul aduce cartea mai târziu, trebuie să plătească o amendă în funcţie de numărul de zile de întârziere."

3.3.1 Elaborarea modelului conceptual

Se vor parcurge următorii paşi pentru elaborarea modelului conceptual:

Pasul 1 Identificarea tipurilor de entităţi

Pentru a identifica un tip de entitate, trebuie citite cu atenţie toate cerinţele subliniindu-se toate substantivele întâlnite în descriere. Se poate crea o listă cu toate aceste atribute, cu duplicate cu tot, din care se vor elimina ulterior toate cele ce nu prezintă interes pentru sistemul care se proiectează. În acest studiu de caz s-au întâlnit următoarele substantive:

Cărţi, autori, nume, adresă, e-mail, cititor, persoană, nume adresă, e-mail, telefon, data la care se face împrumutul, data la care trebuie returnată cartea, numărul de zile pe care se împrumută cartea, amenda.

Fiecare substantiv poate deveni un tip de entitate, dar o parte dintre acestea vor fi atribute. În această fază se va stabili care dintre aceste substantive sunt tipuri de entităţi. Se începe prin stabilirea dependenţelor. În exemplul de faţă, numele, adresa şi e-mail-ul depind de autor, iar numele, adresa, e-mail şi numărul de telefon depind de cititor.

Pornind de la această primă analiză, se vor stabili următoarele tipuri de entităţi: BIBLIOTECA, CARTE, AUTOR, CITITOR, CLIENT şi CEL CE ÎMPRUMUTA ca tipuri de entităţi din modelul nostru, deoarece acestea sunt obiecte de interes în cadrul sistemului. Rezultatul analizei este prezentat în tabelul de mai jos:

Nr. Crt.	Tip de entitate
1	BIBLIOTECI
2	CARTI
3	AUTORI
4	CITITORI
5	CLIENŢI

6	CEI CE ÎMPRUMUTĂ

Pasul 2 – Eliminarea tipurilor de entităţi duplicat

În momentul în care se scriu cerinţele se obişnuieşte să se folosească diverse cuvinte care înseamnă acelaşi lucru, motiv pentru care mai întâi trebuie să identificăm cuvintele redundante. În exemplul dat, există mai multe cuvinte sinonime: *cititor, client, cei ce împrumută cărţi*. În final trebuie să rămână doar unul dintre aceste sinonime. Vom alege *cititor*.

Un aspect deosebit de important este acela de a nu face greşeala de a introduce ca tip de entitate separată chiar numele sistemului, motiv pentru care cuvântul *biblioteca* trebuie eliminat.

În sfârşit, trebuie determinat tipul entităţilor, adică trebuie stabilit dacă un tip de entitate este puternic sau slab. Un tip de entitate este slab dacă existenţa sa depinde de existenţa unui alt tip de entitate. Un tip de entitate puternic este acela a cărui existenţă este independentă de orice alt tip de entitate. Acest lucru este necesar mai târziu, atunci când trebuie stabilit tipul de relaţie.

Rezultatele obţinute sunt prezentate în tabelul de mai jos:

Nr. crt.	Tip de entitate	Tip
1	CĂRŢI	Puternic
2	AUTORI	Puternic
3	CITITORI	Puternic

Pasul 3 – Lista atributelor fiecărui tip de entitate

În continuare se va obţine asigurarea că fiecare tip de entitate este neapărat necesar. Se verifică dacă nu cumva un anumit tip de entitate nu este de fapt un atribut al altui tip de entitate. De exemplu, numărul de telefon este un tip de entitate sau un atribut al tipului de entitate AUTORI?

Se va ţine legătura cu clientul pentru a stabili dacă nu cumva mai sunt necesare şi alte atribute pentru tipurile de entităţi identificate. În exemplul prezentat, am folosit doar câteva atribute pentru fiecare tip de entitate rămas, aşa cum se arată în tabelele următoare:

Tipul de entitate CITITORI

Nume atribut	Tip	Domeniu	Opţional
ID_CITITOR	Identificator unic	Text	Nu
NUME	Atribut compus	Text	Nu
EMAIL	Atribut cu o singură valoare	Text	Da

TEL	Atribut multivaloare	Text	Da
ADRESA	Atribut compus	Text	Da
ID_CARTE	Atribut cu o singură valoare	Text	Nu
DATA_IMPRUMUT	Atribut cu o singură valoare	Text	Nu
DURATA	Atribut derivat	Text	Nu
DATA_RETUR	Atribut derivat	Text	Nu

Tipul de entittate AUTORI

Nume atribut	Tip	Domeniu	Opţional
ID_AUTOR	Identificator unic	Text	Nu
NUME	Atribut compus	Text	Nu
EMAIL	Atribut cu o singură valoare	Text	Da
TEL	Atribut multivaloare	Text	Da
ADRESA	Atribut compus	Text	Da

Tipul de entitate CARTI

Nume atribut	Tip	Domeniu	Opţional
ID_CARTE	Identificator unic	Text	Nu
TITLU	Atribut cu o singură valoare	Text	Nu
EDITIE	Atribut cu o singură valoare	Numeric	Da
AN	Atribut cu o singură valoare	Numeric	Da
PRET	Atribut cu o singură valoare	Numeric	Da
ISBN	Atribut cu o singură valoare	Text	Da
PAGINI	Atribut cu o singură valoare	Numeric	Da
INTERVAL	Atribut cu o singură valoare	Text	Da

DESCRIERE	Atribut simplu	Text	Da

Aşa cum s-a arătat în secţiunea *3.2.3.3 Atribute*, sunt câteva lucruri care trebuie stabilite pentru atributele din tabelele de mai sus:

1. Trebuie să se ia o hotărâre referitoare la atributele compuse: se păstrează aşa cum sunt sau se separă în două sau mai multe. În cazul de faţă dacă clientul doreşte să facă o căutare după numele de familie sau după prenume este mai bine să se separe atributul compus. În acest fel vom avea în locul atributului NUME, atributele NUME şi PRENUME.
2. Atunci când se găseşte un atribut multivaloare, acesta trebuie transformat într-un alt tip de entitate. În cazul de faţă, nu este nevoie de mai multe numere de telefon, aşa că se va modifica tipul atributului din multivaloare în singură valoare.
3. DURATA este un atribut derivat care se poate calcula de către sistem pe baza atributului DATA_IMPRUMUT. În exemplul de faţă, fiecare carte poate fi împrumutată doar pe durata a 10 zile. Acest atribut trebuie eliminat.

După această analiză, rezultatele acestui pas sunt prezentate în tabelele de mai jos:

Tipul de entitate CITITOR

Nume atribut	Tip	Domeniu	Opţional
ID_CITITOR	Identificator unic	Text	Nu
NUME	Atribut cu o singură valoare	Text	Nu
PRENUME	Atribut cu o singură valoare	Text	Nu
EMAIL	Atribut cu o singură valoare	Text	Da
TEL	Atribut cu o singură valoare	Text	Da
ADRESA	Atribut compus	Text	Da
ID_CARTE	Atribut cu o singură valoare	Text	Nu
DATA_IMPRUMUT	Atribut cu o singură valoare	Text	Nu
DATA_RETUR	Atribut cu o singură valoare	Text	Nu

Tipul de entitate AUTORI

Nume atribut	Tip	Domeniu	Opţional
ID_AUTOR	Identificator unic	Text	Nu

NUME	Atribut cu o singură valoare	Text	Nu
PRENUME	Atribut cu o singură valoare	Text	Nu
EMAIL	Atribut cu o singură valoare	Text	Da
TEL	Atribut cu o singură valoare	Text	Da
ADRESA	Atribut compus	Text	Da

Tipul de entitate CARTI

Nume atribut	Tip	Domeniu	Opţional
ID_CARTE	Identificator unic	Text	Nu
TITLU	Atribut cu o singură valoare	Text	Nu
EDITIE	Atribut cu o singură valoare	Numeric	Da
AN	Atribut cu o singură valoare	Numeric	Da
PRET	Atribut cu o singură valoare	Numeric	Da
ISBN	Atribut cu o singură valoare	Text	Da
PAGINI	Atribut cu o singură valoare	Numeric	Da
INTERVAL	Atribut cu o singură valoare	Text	Da
DESCRIERE	Atribut cu o singură valoare	Text	Da

Pas 4 – Alegerea unui identificator unic

În cazul tipului de entitate CARTI, există posibilitatea de a alege identificatorul unic dintre atributele ID_CARTE şi ISBN. În cazul de faţă vom alege ID_CARTE, care reprezintă identificatorul unic al fiecărei cărţi din sistemul de management al bibliotecii, deoarece clientul nu dispune de un scaner manual, motiv pentru care, atât la împrumutul cât şi la returnarea cărţilor, bibliotecarul trebuie să introducă în câmpul ISBN toate cifrele acestuia ceea ce poate fi neplăcut şi ineficient.

Pas 5 – Stabilirea tipurilor de relaţii

În acest pas trebuie examinate asocierile multiple dintre diversele tipuri de entităţi, deoarece, de obicei, este posibil să se găsească mai multe tipuri de relaţii pentru o pereche de tipuri de entităţi. De exemplu, o carte poate fi scrisă de către o persoană sau poate fi împrumutată de către o persoană.

În continuare se vor identifica cardinalitatea şi participarea tipurilor de relaţii alese, eliminându-se tipurile de relaţii redundante. De asemenea, se va stabili dacă un tip de relaţie este puternic sau slab pe baza tipurilor de entităţi care intră în asociere. *Tipurile de relaţii slabe* se stabilesc între un tip de entitate puternic şi un tip de entitate slab. *Tipurile de relaţii puternice* se stabilesc între două tipuri de entităţi puternice. În instrumentul InfoSphere Data Architect acestea sunt cunoscute sub denumirea de *identifying*, respectiv *non-identifying*. O asociere de tip identifying se alege atunci când tipul respectiv de relaţie este unul în care unul dintre tipurile de entităţi subordonate este un tip de entitate dependent de existenţa altui tip de entitate. Tipurile de relaţii non-Identifying se aleg atunci când ambele tipuri de entităţi participante într-o relaţie sunt independente.

Pentru a stabili tipurile de relaţii trebuie să se identifice perechile de tipuri de entităţi, analizându-se toate scenariile posibile. O idee simplă ar fi folosirea tabelelor, aşa cum se vede mai jos:

	Tip de relaţie	**Identifying**	**Verbul din stânga**	**Verbul din dreapta**	**Cardinalitatea**	**Participarea**
1	CITITORI -> CARTI	Nu	Împrumută	Este împrumutată	Mulţi-la-mulţi	Poate
2	AUTORI - > CARTI	Nu	Scrie	Este scrisă	Mulţi-la-mulţi	Poate
3	AUTORI - > CARTI	Nu	Împrumută	Este împrumutată	Mulţi-la-mulţi	Poate

În tabelul de mai sus se observă 3 tipuri de relaţii mulţi-la-mulţi care trebuie rezolvate. Acest lucru se poate face prin descompunerea tipului de relaţie iniţială în două tipuri de relaţii unu-la-mulţi. În acest scop trebuie găsit un al treilea tip de entitate intermediar. În cazul de faţă vom avea:

1. CITITORI -> CARTI

Pentru a rezolva acest tip de relaţie se va folosi tipul de entitate COPIE care se creează în mod natural deoarece, de fapt, o persoană nu împrumută o carte, ci o copie a acesteia. Tipul de entitate COPIE va arăta astfel:

Tipul de entitate COPIE

Nume atribut	**Tip**	**Domeniu**	**Opţional**
ID_COPIE	Identificator unic	Text	Nu
STARE	Atribut cu o singură valoare	Text	Nu

2. AUTORI - > CARTI

Pentru a elimina tipul de relaţie mulţi-la-mulţi dintre tipul de entitate AUTORI şi tipul de entitate CARTI se creează tipul de entitate LISTA_AUTHORI. Noul tip de entitate va arăta astfel:

Tipul de entitate LISTA_AUTORI

Nume atribut	Tip	Domeniu	Opţional
ROL	Atribut cu o singură valoare	Text	Nu

În acest fel, noul tabel cu tipuri de relaţii va arăta astfel:

Tip de relaţie	Identifying	Verbul din stânga	Verbul din dreapta	Cardinalitatea	Participarea
CITITORI - > COPIE	Nu	Împrumută	Este împrumutată	Unu-la-mulţi	Poate
CARTI - > COPIE	Nu	Are	Este creată	Unu-la-mulţi	Poate
AUTORI - > LISTA_AUTORI	Nu	Apare	Are	Unu-la-mulţi	Poate
LISTA_AUTORI - > CARTI	Nu	Este creată	Are	Multi-la-multi	Poate

3. AUTORI - > CARTI

Cel de-al treilea tip de relaţie care trebuie de asemenea descompus se prezintă din nou din motive de uşiurinţă a citirii:

Tip de relaţie	Identifying	Verbul din stânga	Verbul din dreapta	Cardinalitatea	Participarea
AUTORI - > CARTI	Nu	Împrumută	Este împrumutată	Mulţi-la-mulţi	Poate

Parcurgând acest studiu de caz s-ar putea să vi se pară curios de ce avem un tip de entitate AUTORI şi alt tip de entitate CITITORI, la care cele mai multe dintre atribute sunt identice. De ce nu se creează un singur tip de entitate PERSOANA care să fie un supertip al subtipurilor AUTORI şi CITITORI? În definitiv, şi autorii pot împrumuta cărţi.

Acest model poate fi creat folosind un singur tip de entitate PERSOANA, dar există două motive pentru care nu vom proceda în acest fel:

- Simplitatea înţelegerii acestui model (în principal, de natură didactică, pentru a înţelege conceptele mai uşor)
- Simplitatea în momentul elaborării aplicaţiei corespunzătoare acestui model. Presupunând că sistemul se adresează bibliotecii unui oraş mic, şansele ca autorii cărţilor să locuiască în acel oraş sunt foarte reduse, ceea ce înseamnă faptul că şansele ca autorii cărţilor să împrumute cărţi de la biblioteca respectivă sunt şi acestea foarte reduse. În acelaşi timp de ce să avem în vedere un astfel de scenariu în aplicaţia pe care o elaborăm? În cazul puţin probabil ca autorul unei cărţi să locuiască în acel oraş, bibliotecarul poate să ceară acestuia să-şi creeze propriul cont de cititor. În acest fel programatorul nu mai trebuie să-şi facă griji cu privire la elaborarea unei singure interfeţe în care autorul este şi cititor. Desigur, o parte dintre aceste aspecte trebuie văzute ca reguli de domeniu şi trebuie interpretate ca atare.

Alte consideraţii care vor fi discutate ulterior vor avea în vedere constrângerile asupra datelor care se introduc în tabele. De exemplu, se poate introduce o restricţie prin care să se verifice dacă data de returnare a cărţilor împrumutate este ulterioară datei la care acestea au fost împrumutate. Se va discuta acest aspect atunci când se va disctua despre constrângerea **`CHECK`** în modelul fizic prezentat în Capitolul 5.

Pas 6 – Stabilirea regulilor de domeniu

Regulile de domeniu trebuie implementate prin intermediul unei aplicaţii. În cazul de faţă vom lua în considerare următoarele reguli de domeniu:

1. Doar administratorii sistemului pot modifica date
2. Se va permite ca sistemul să poată face rezervări de cărţi. Fiecare cititor poate să-şi vadă poziţia în lista de aşteptare pentru a obţine cartea de care este interesat.
3. Aplicaţia va trimite un mesaj prin e-mail prin care se va aminti cititorului de faptul că va trebui să returneze cartea în termen de 3 zile de la trimiterea mesajului.
4. Dacă cititorul nu returnează la timp cartea, acesta va fi amendat cu 0.1% din preţul cărţii pentru fiecare zi de întârziere.

Acest studiu de caz v-a prezentat o serie de recomandări de care este bine să ţineţi seama la proiectare. Totuşi, modelarea conceptuală este un proces iterativ, astfel încât este necesar să elaboraţi mai multe versiuni ale aceluiaşi model din care să o alegeţi apoi pe cea optimă. Nu există un răspuns ferm la întrebarea dacă un model este mai bun decât altul, dar cu siguranţă puteţi găsi soluţii mai bune decât altele.

Acest studiu de caz va continua în Capitolul 4, în care se va discuta despre modelarea logică a acestui sistem de management al unei biblioteci.

3.4 Rezumat

În cadrul acestui capitol s-a discutat despre modelarea conceptuală, explicându-se felul în care trebuie să se abordeze situaţiile atunci când se lucrează cu diagramele entitate-relaţie. Pe parcurs s-au prezentat o serie de concepte, cum ar fi: tipuri de entităţi, atribute,

tipuri de relaţie, constrângeri, domenii ş.a.m.d.

Folosirea unui instrument grafic, cum ar fi de exemplu InfoSphere Data Architect, se dovedeşte a fi de mare ajutor, în special atunci când se lucrează la proiecte foarte complexe, fiind nevoie de transmiterea acestor modele între diferitele persoane interesate de bunul mers al proiectului sau între cei care lucrează la modelele de date logice sau fizice. Conceptele de bază din modelarea conceptuală sunt prezentate şi în cartea *Getting Started with InfoSphere Data Architect*.

Acest capitol mai discută şi despre modul de organizare a modelului de date apelând la reguli pentru elaborarea modelului ER şi crearea de diagrame.

Deşi în această carte nu se urmăresc aspecte referitoare la programare, acest capitol vă oferă o bază solidă în înţelegerea modelării conceptuale prin folosirea de exemple realizate cu ajutorul instrumentului InfoSphere Data Architect.

3.5 Exerciţii

Fie următoarele tipuri de entităţi din cadrul unei universităţi:

- Facultate
- Profesor
- Functie
- Curs
- Student

Se cere să se realizeze o reprezentare grafică a tipurilor de relaţii şi entităţi. Reprezentaţi grafic toate aspectele care se modelează folosind o diagramă entitate-relaţie creată cu ajutorul instrumentului InfoSphere Data Architect. Pentru a obţine mai multe informaţii despre acest instrument urmăriţi cartea *Getting started with InfoSphere Data Architect* ce face parte din seria de cărţi din care face parte şi cea de faţă.

3.6 Întrebări recapitulative

1. Care dintre următoarele aparţin unui tip de entitate:
   A. O coloană dintr-un tabel
   B. Colecţie de obiecte sau concepte din lumea reală
   C. Date care descriu o caracteristică a unui obiect sau concept
   D. Mulţime de entităţi de acelaşi tip care au aceleaşi proprietăţi.
   E. Nici unul dintre cele enumerate.
2. Atributele variabile:
   A. Au valori care se modifică frecvent.
   B. Se modifică rar, dacă nu deloc
   C. Au valori ce provin de la alt atribut sau atribute

D. Au mai multe valori pentru fiecare entitate

E. Nici unul dintre cele enumerate.

3. Trebuie rezolvat un tip de relaţie M:M. Noul tip de relaţie are întotdeauna:

A. O participare opţională pe partea de mulţi

B. Participare obligatorie pe partea de unu

C. Participare obligatorie pe partea de mulţi

D. Redundanţă pe partea de mulţi

E. Recursivitate pe partea de unu

4. Care dintre următoarele afirmaţii este adevărată atunci când se vorbeşte despre modelarea conceptuală?

A. Un tip de entitate trebuie să apară o singură dată într-un model de date

B. Modelul conceptual trebuie să prezinte şi atributele derivate

C. Într-un model conceptual trebuie reprezentate toate datele posibile

D. Toate cele enumerate

E. Nici una dintre cele enumerate

5. Care dintre următoarele formulări NU reprezintă definiţia unui tip de entitate?

A. Un tip de entitate este o colecţie de obiecte sau concepte din lumea reală care au aceleaşi caracteristici.

B. Un tip de entitate este o instanţă a unor obiecte din lumea reală

C. Un tip de entitate este un obiect care are existenţă proprie şi care este distinct de toate celelalte.

D. Toate cele enumerate

E. Nici una dintre cele enumerate

6. Care dintre următoarele afirmaţii NU este adevărată referitor la un tip de relaţie?

A. Un tip de relaţie prezintă felul în care tipurile de entităţi se asociază unele cu altele.

B. Un tip de relaţie trebuie să existe întotdeauna între două tipuri de entităţi

C. Un tip de relaţie trebuie citit în ambele sensuri.

D. Un tip de relaţie este un substantiv

E. Toate cele enumerate.

7. Care dintre următoarele afirmaţii este adevărată atunci când se face referire la tipurile de relaţii obligatorii?

A. Prezintă felul în care de entităţie se asociază între ele.

B. Fiecare tip de entitate trebuie să participe într-o asociere.

C. Fiecare tip de entitate poate să participe într-o asociere

D. Există mai multe posibilităţi de asociere pentru fiecare tip de entitate

E. Nici una dintre cele enumerate.

8. Un grup de instanţe care au atribute sau tipuri de relaţii care există numai pentru acel grup este numit:

A. Constrângere

B. Cardinalitate

C. Supertip

D. Subtip

E. Toate cele enumerate

9. Denumiţi o clauză din standardul SQL, care permite restricţionarea domeniilor:

A. RELATIONSHIP

B. PRIMARY KEY

C. CHECK

D. CONSTRAINT

E. Nici una dintre cele enumerate

10. Datele existente într-o bază de date la un anumit moment dat sunt denumite:

A. Intensie

B. Extensie

C. Schemă

D. Instanţă

E. Nici una dintre cele enumerate

4

Capitolul 4 – Proiectarea bazelor de date relaţionale

Acest capitol prezintă conceptele de proiectare folosite în domeniul bazelor de date relaţionale care ajută la modelarea diverselor domenii din lumea reală sub forma bazelor de date relaţionale. Se vor oferi o serie de recomandări pentru crearea tabelelor, coloanelor şi stabilirea de asocieri dintre tabele astfel încât să se obţină minimum de redundanţă a datelor.

În cadrul acestui capitol se vor urmări:

- Modelarea obiectelor din lumea reală sub forma tabelelor relaţionale
- Identificarea problemelor şi minimizarea redundanţelor
- Identificarea dependenţelor şi introducerea acestora în cadrul proiectului bazei de date relaţionale
- Revizuirea tabelelor relaţionale pentru a obţine o proiectare optimă

4.1 Problema redundanţei

Redundanţa datelor înseamnă regăsirea aceloraşi date în mai multe locaţii din cadrul unei baze de date. Existenţa redundanţelor în cadrul unei scheme relaţionale semnifică o proiectare lipsită de optimizare a acesteia, datorită apariţiei anomaliilor:

- de inserare
- de ştergere
- de actualizare

Tabelul 4.1 prezintă un exemplu de redundanţă a datelor deoarece informaţiile despre colegiu şi informaţiile despre studenţi sunt păstrate la un loc. Folosind un astfel de model de proiectare, vor trebui introduse aceleaşi informaţii despre colegiu de mai multe ori, câte o dată pentru fiecare student înscris. De exemplu, informaţia despre nivelul colegiului se repetă de două ori, o dată pentru studentul *George Smith*, şi altă dată pentru studentul *Will Brown*.

ID_STUDENT	STUDENT	POZITIE	COLEGIU	NIVEL_COLEGIU
0001	Ria Sinha	6	Fergusson	4
0002	Vivek Kaul	15	PICT	5
0003	George Smith	9	IIT	1
0004	Will Brown	1	IIT	1

Tabel 4.1 – Exemplu de redundanţă a datelor (schema Student)

Obs:

Tabelul 4.1 se va folosi la toate exemplele din cadrul acestui capitol. Coloana **ID_STUDENT** care reprezintă numărul matricol al studentului este **cheie primară** în cadrul schemei Student.

4.1.1 Anomalii de inserare

O anomalie de inserare are loc atunci când, la introducerea datelor în cadrul unui rând al unui tabel, se constată că acest lucru nu se poate face până când nu se introduc alte date suplimentare în cadrul rândului (date care nu au nici o legătură cu datele ce trebuie introduse). De exemplu, introducerea informaţiilor despre student impune şi introducerea datelor despre colegiu (coloana NIVEL_COLEGIU din *Tabelul 4.2*).

ID_STUDENT	STUDENT	POZITIE	COLEGIU	NIVEL_COLEGIU
0005	Susan Fuller	10	Fergusson	

Tabelul 4.2 – Anomalii de inserare – Exemplu

În plus, se va constata o repetare a aceloraşi date în diverse locaţii din cadrul bazei de date deoarece datele referitoare la nivelul colegiului trebuie introduse pentru fiecare student nou introdus in tabel.

4.1.2 Anomalii la ştergere

O anomalie la ştergere apare atunci când la eliminarea unei înregistrări se pierd şi alte informaţii care nu au legătură cu datele eliminate, dar care au fost stocate ca parte a înregistrării care se elimină din cadrul tabelului.

De exemplu, prin eliminarea unui student se vor pierde şi informaţiile referitoare la colegiu şi la nivelul colegiului care reprezintă informaţii despre colegiul respectiv, neavând legătură cu studentul. Acest lucru se poate vedea în *Tabelul 4.3*, în care se pierd toate informaţiile referitoare la colegiul ‘IIT’ dacă se elimină ambii studenţi, *George Smith* şi *Will Brown,* care aparţin acestui colegiu.

ID_STUDENT	STUDENT	POZITIE	COLEGIU	NIVEL_COLEGIU
0003	George Smith	9	IIT	1
0004	Will Brown	1	IIT	1

Tabelul 4.3 – Anomalii la ştergere - Exemplu

4.1.3 Anomalii de actualizare

O anomalie de actualizare apare atunci când modificând datele unei entităţi într-o singură locaţie se va obţine o inconsistenţă a datelor faţă de alte date redundante existente în altă locaţie din cadrul bazei de date.

De exemplu, dacă se modifică nivelul colegiului la valoarea '1' în cazul studentului cu identificatorul **ID_STUDENT** 0003, atunci colegiul 'Fergusson' are un conflict de informaţie, deoarece va avea două nivele: 1 şi 4.

ID_STUDENT	STUDENT	POZITIE	COLEGIU	NIVEL_COLEGIU
0001	Ria Sinha	6	Fergusson	**4**
0003	George Smith	9	Fergusson	**1**

Tabelul 4.4 – Anomalii de actualizare – Exemplu

În secţiunile următoare, se vor sugera soluţii pentru rezolvarea problemelor cauzate de redundanţa datelor.

4.2. Descompuneri

Normalizarea folosită în proiectarea unei baze de date relaţionale presupune descompunerea schemei relaţionale în relaţii mai mici şi mai simple astfel încât să se evite apariţia redundanţelor. Ideea este aceea de a interoga relaţii mai mici pentru a obţine exact aceleaşi informaţii pe care le obţineam anterior din schema relaţională iniţială.

Schema Student din *Figura 4.1* prezintă un exemplu de normalizare pentru cazul de faţă.

În *Figura 4.1*, se prezintă descompunerea schemei Student (a) în relaţiile COLEGIU şi STUDENT din (b) şi (c). În acest fel, informaţia referitoare la nivelul colegiului nu este păstrată în mod redundant, oferind doar o singură valoare pentru fiecare colegiu existent în baza de date.

Colegiul de care aparţine un student se alege prin intermediul unui atribut al relaţiei STUDENT, astfel încât se poate restaura informaţia existentă în schema iniţială.

ID_STUDENT	STUDENT	POZITIE	COLEGIU	NIVEL_COLEGIU
0001	Ria Sinha	6	Fergusson	4
0002	Vivek Kaul	15	PICT	5
0003	George Smith	9	IIT	1
0004	Will Brown	1	IIT	1

(a) Student - Relatia COLEGIU

COLEGIU	NIVEL_COLEGIU
Fergusson	4
PICT	5
IIT	1

(b) Relatia COLEGIU

ID_STUDENT	STUDENT	POZITIE	COLEGIU
0001	Ria Sinha	6	Fergusson
0002	Vivek Kaul	15	PICT
0003	George Smith	9	IIT
0004	Will Brown	1	IIT

(c) Relatia STUDENT

Figura 4.1 – Exemplu de normalizare

Dacă am face o descompunere sub forma:

COLEGIU	NIVEL_COLEGIU

ID_STUDENT	STUDENT	POZITIE

se va constata că se va pierde informaţie, deoarece nu mai putem afla din ce colegiu face parte un student.

O altă posibilitate de descompunere ar fi:

COLEGIU	NIVEL_COLEGIU	ID_STUDENT

ID_STUDENT	STUDENT	POZITIE

În această situaţie descompunerea va conduce la introducerea de mai multe ori a nivelului colegiului (obţinem din nou redundanţă) pentru fiecare student introdus.

Deşi este necesar să realizăm descompuneri ale schemelor relaţionale pentru a evita apariţia redundanţelor, trebuie totuşi să fim atenţi să nu pierdem informaţia pe care o aveam în schema iniţială. Acest lucru conduce la concluzia că este posibil să refacem schema relaţională iniţială pe baza relaţiilor mai mici obţinute la descompunere. Pentru a obţine această reversibilitate a informaţiilor ne vom folosi de noţiunea de ***dependenţă funcţională***.

4.3. Dependenţe funcţionale

Dependenţa funcţională (DF) reprezintă un tip de constrângere de integritate care se bazează pe ideea de supercheie. Cu ajutorul dependenţelor funcţionale se identifică dependenţele dintre submulţimile atributelor unei relaţii. O definiţie formală a dependenţei funcţionale este prezentată mai jos:

Definiţie: Fie schema relaţională **R**, cu submulţimile de atribute **A** şi **B**. Se spune că avem o dependenţă funcţională **A** → **B** dacă valorile din **A** au o valoare corespunzătoare în **B** pentru fiecare instanţă **r** a relaţiei **R**.

Se spune că DF generalizează conceptul de supercheie deoarece valorile atributelor din mulţimea **A** determină în mod unic valorile atributelor din mulţimea **B** în cazul unei anumite relaţii date **R**. Cu alte cuvinte, se mai poate spune că avem o dependenţă funcţională **A** → **B** atunci când valorile lui **B** sunt dependente funcţional de **A**. Se spune 'funcţional' deoarece, asemănător conceptului de funcţie, o dependenţă funcţională ajută la realizarea unei corespondenţe dintre o mulţime de valori ale atributului **A** şi o altă mulţime de valori ale atributului **B**.

Pe relatia R există o DF dacă aceasta există pentru toate instanţele valabile **r** ale relaţiei **R**.

În acelaşi timp, pentru a verifica dacă o **instanţă r satisface o DF**, trebuie să verificăm dacă fiecare tuplu al instanţei r corespunde DF definite astfel încât **A** → **B** (**A** determină pe **B** sau **A** implică **B** pentru fiecare rând de date introduse în tabel la un anumit moment dat).

Dependenţa funcţională se citeşte sub forma "**A** îl determină pe **B**", "**B** este dependent de **A**" sau "**A** implică **B**" şi se notează sub forma "**A** → **B**".

Dacă $A_1 \rightarrow B_1$, $A_2 \rightarrow B_2$, $A_3 \rightarrow B_3$ … $A_n \rightarrow B_n$, atunci vom scrie:

$A_1\ A_2\ A_3 \ldots A_n \rightarrow B_1\ B_2\ B_3 \ldots B_n$

Mai jos se poate vedea o instanţă a relaţiei **R (A, B, C, D)** în care **A** → **D** :

A	B	C	D
a1	b1	c1	d1
a2	b2	c2	d1
a3	b3	c3	d1
a4	b3	c4	d2

Tabelul 4.5 – Dependenţa funcţională

Una dintre DF de mai sus este **A** ----→ **D**. Se poate observa cu uşurinţă de ce reciproca **D** ----→ **A** nu este adevărată. Dacă se dau aceleaşi valori în **D**, valorile coerspunzătoare din **A** se modifică aşa cum se vede în tuplurile **(a1 , b1 , c1, d1)** şi **(a2, b2, c2, d1)**.

Exemplu

Folosind schema Student din primul exemplu, se va prezenta în Tabelul 4.7, o DF, **ID_STUDENT** → COLEGIU **valabilă** pe schema Student, în timp ce DF STUDENT → COLEGIU nu este valabilă pe schema relaţiei deoarece pot exista studenţi care au acelaşi nume, dar să fie în colegii diferite.

ID_STUDENT	STUDENT	POZITIE	COLEGIU
0001	Ria Sinha	6	Fergusson
0002	Vivek Kaul	15	PICT
0003	George Smith	9	IIT
0004	Will Brown	1	IIT

Tabelul 4.7 – Exemplu de dependenţă funcţională

Următoarea mulţime de DF este adevărată:

{**ID_STUDENT** → COLEGIU , **ID_STUDENT** → STUDENT ,

ID_STUDENT → STUDENT → COLEGIU

ID_STUDENT → STUDENT → POZITIE}

Dependenţe funcţionale triviale – o dependenţă funcţională care este valabilă pentru toate valorile unui atribut al unei relaţii date este numită dependenţă funcţională trivială.

Exemplu

(prenume, nume) → prenume

În general, o dependenţă funcţională **A** → **B** este trivială dacă **B** este o submulţime a lui **A,** ceea ce înseamnă că **B** este inclusă în **A** (partea dreaptă este o parte a părţii stângi).

În proiectul unei baze de date suntem interesaţi, de fapt, mai mult de dependenţele funcţionale netriviale, deoarece acestea ajută la determinarea constrângerilor de integritate ale unei relaţii în timp ce dependenţele funcţionale triviale sunt evidente şi vor fi verificate în orice situaţie.

4.4 Proprietăţile dependenţelor funcţionale

Pentru a determina toate dependenţele funcţionale care există într-o mulţime dată de dependenţe funcţionale, se va folosi o proprietate importantă a DF numită Mulţimea de închidere a dependenţelor funcţionale.

Mulţimea de închidere a dependenţelor funcţionale – Toate dependenţele funcţionale care sunt deduse dintr-o mulţime de dependenţe funcţionale S, alcătuiesc mulţimea de închidere a dependenţelor funcţionale, notată cu, S^+. Mulţimea de închidere reprezintă, în esenţă, setul complet de valori ce pot fi determinate pe baza unui set cunoscut de valori

din cadrul unei relaţii cu ajutorul dependenţelor sale funcţionale.

Acest lucru conduce la faptul că orice instanţă r a relaţiei R care satisface mulţimea dată de dependenţe funcţionale, S, va satisface şi mulţimea de închidere a dependenţelor funcţionale, S^+.

În subsecţiunile următoare se va parcurge setul de reguli necesar pentru calculul mulţimii de închidere a dependenţelor funcţionale.

4.4.1 Axiomele lui Armstrong

Axiomele lui Armstrong mai sunt cunoscute şi sub denumirea de **'Reguli de inferenţă'** şi ne ajută să deducem toate dependenţele funcţionale ale unei mulţimi de dependenţe funcţionale date.

Există trei reguli de inferenţă:

1. ***Reflexivitatea*****:** Dacă **B** este o submulţime a atributului **A**, atunci **A** → **B**. (**dependenţă funcţională trivială**)

2. ***Augmentarea*****:** Dacă **A** → **B** iar **C** este un alt atribut, atunci **AC** → **BC**

Prin aplicarea reflexivităţii se obţine că **AC** → **B**.

3. ***Tranzitivitatea*****:** Dacă **A** → **B** şi **B** → **C**, atunci **A** → **C**.

Suplimentar, mai există şi alte reguli care pot fi dovedite cu ajutorul celor 3 axiome ce se folosesc pentru a uşura crearea mulţimii de închidere a dependedenţelor funcţionale şi care sunt cunoscute sub denumirea de **reguli adiţionale**.

1. ***Reuniunea*****:** Dacă **A** → **B** şi **A** → **C**, atunci **A** → **BC**.

2. ***Descompunerea*****:** Dacă **A** → **BC**, atunci **A** → **B** şi **A** → **C**.

Axiomele lui Armstrong sunt solide şi complete. Ele generează numai dependenţe funcţionale în mulţimea de închidere a dependenţelor funcţionale, S^+ şi produc mulţimea completă a dependenţelor funcţionale din S^+.

4.4.2 Determinarea mulţimii de închidere a atributelor

Există o metodă alternativă de a obţine mulţimea de închidere a dependenţelor funcţionale pornind de la o dependenţă funcţională dată. Aceasta poate fi folosită şi pentru a afla dacă o mulţime dată de atribute reprezintă o supercheie a unei relaţii.

Mulţimea de închidere a atributelor în cazul unui atribut, **A**, reprezintă mulţimea tuturor atributelor din cadrul unei relaţii **R** care pot fi determinate în mod unic de către **A**, pe baza unor dependenţe funcţionale date.

Obs: Dându-se inchidere(**A**), $\mathbf{A}^+$ fiind mulţimea care conţine toate atributele relaţiei R, se poate spune că atributul **A** este o supercheie a relaţiei R.

Calcul

Fie o relaţie **R** care are o mulţime de atribute. Se poate calcula mulţimea de închidere a

atributului **A**, ce va fi numită inchidere(**A**) astfel:

1. Mulţimea iniţială inchidere(**A**) = **A**

2. Pentru fiecare dependenţă funcţională dată, dacă **A** → **B**, atunci se adaugă **B** la inchidere(**A**), ceea ce înseamnă, inchidere(**A**) U **B**

3. Pentru orice submulţime a lui **A**, (fie **C** o submulţime a lui **A**), **A**→ **C** (dependenţă funcţională trivială) şi dacă **C** → **D**, dar **D** nu este o submulţime a lui **A**, atunci se adaugă **D** mulţimii inchidere(**A**)

4. Se repetă pasul 3 până nu mai rămân atribute ce pot fi adăugate mulţimii inchidere(**A**)

Exemplu

Fie o relaţie **R (A, B, C, D, E)** cu dependenţele funcţionale date **A** → **B, B** → **DE** şi **D** → **C**

Calcul

Pas 1. inchidere(**A**) = **A**

Pas 2. **A** → **B**, adică inchidere(**A**) = **A** U **B**, care se notează cu **AB**

Pas 3.

Prima iteraţie: **B** → **DE,** dar **B** a devenit o submulţime a inchidere(**A**), adică inchidere(**A**) = **ABDE**

A doua iteraţie: **AD** → **C,** dar **D** a devenit o submulţime a mulţimii inchidere(**A**), însă **C** nu este o submulţime a mulţimii inchidere(**A**), deci inchidere**(A), A^+ = ABDEC.**

În mod asemănător, inchidere(**B**), B^+ = **BDEC**

inchidere(**C**), C^+ = **C**

inchidere(**D**), D^+= **DC**

inchidere(**E**), E^+= **E**

4.4.3 Concluzie

Dependenţele funcţionale ajută la descompunerea corectă a relaţiilor astfel încât valorile aflate în legătură să poată fi păstrate într-un singur tabel.

În momentul în care datele sunt inserate în baza de date, acestea trebuie să se conformeze constrângerilor specificate. În afara constrângerilor de integritate, datele mai trebuie să se conformeze şi dependenţelor funcţionale.

Proprietăţile dependenţelor funcţionale ajută la alcătuirea mulţimii de închidere a dependenţelor funcţionale, astfel încât să obţinem o formulă structurată cu ajutorul căreia să deducem o mulţime exhaustivă de constrângeri pentru a modela cu eficienţă depdendenţele din lumea reală.

Axiomele lui Armstrong ne sunt de mare ajutor prin fapul că sunt solide şi complete. Închiderea atributelor pentru o mulţime dată de dependenţe funcţionale nu oferă doar o

alternativă la calculul mulţimii de închidere a dependenţelor funcţionale, dar ajută şi la identificarea supercheii unei relaţii verificând dacă o dependenţă funcţională dată, $\mathbf{X} \rightarrow \mathbf{Y}$ aparţine mulţimii de închidere a dependenţelor funcţionale.

4.5 Forme normale

Normalizarea reprezintă o procedură folosită la proiectarea bazelor de date relaţionale prin care se transformă schemele relaţionale într-o formă mult mai convenabilă. Scopul este acela de eliminare a redundanţelor din cadrul relaţiilor şi a problemelor generate de către acestea, numite anomalii de inserare, de ştergere şi de actualizare.

Formele normale trec dintr-una în alta obţinându-se în final un proiect optimizat. Normalizarea este un proces care se face pas cu pas, la fiecare pas producându-se o transformare a schemelor relaţionale într-o formă normală superioară. Fiecare formă normală conţine toate formele normale anterioare pe care se fac optimizări.

4.5.1 Prima formă normală (1FN)

O relaţie se consideră că se află în ***prima forma normală*** dacă toate atributele sale se află la nivel indivizibil sau atomic.

Nivelul atomic al valorilor asigură faptul că nu există ‘grupuri repetitive’. Acest lucru este necesar deoarece sistemul de management al bazelor de date relaţionale nu poate să păstreze o valoare decât la intersecţia unui rând cu o coloană. Grupurile repetitive se referă la faptul că s-ar putea păstra mai multe valori la intersecţia unui rând cu o coloană, iar un tabel în care s-ar întâmpla acest lucru nu ar mai putea fi considerat relaţional.

C. J. Date’s extinde această definiţie în [4.8], “Un tabel se află în prima formă normală dacă şi numai dacă satisface următoarele condiţii:

- Nu există o ordonare de sus în jos a rândurilor.
- Nu există o ordonare de la stânga la dreapta a coloanelor.
- Nu există rânduri duplicat.
- Fiecare intersecţie de rând cu coloană conţine exact o singură valoare din domeniul specificat (şi nimic altceva).
- Toate coloanele sunt de tip obişnuit [rândurile nu au componente ascunse cum ar fi identificatori de rând, identificatori de obiecte sau amprente de timp]. ”

O coloană care păstrează, de exemplu, rudele din cadrul unei familii nu se află la nivel atomic, deoarece se face referire la o mulţime de nume. În acelaşi timp, coloana ce conţine identificatorul unui angajat nu mai poate fi desfăcută în valori subordonate, astfel încât aceasta se află la nivel atomic.

Exemplu

Să urmărim tabelul de mai jos care prezintă relaţia *Filme*. În această relaţie, combinaţia de atribute {Titlu, An} alcătuieşte o cheie candidat.

Titlu	An	Tip	Regizor	Regizor_DN	Aparitie	Actori
Notting Hill	1999	Romantic	Roger M	05/06/1956	30	Hugh G Rhys I
Lagaan	2000	Drama	Ashutosh G	15/02/1968	50	Aamir K Gracy S

Tabelul 4.8 – Relaţia denormalizată *Filme*

Relaţia de mai sus nu se află în forma normală 1 nefiind nici măcar strict relaţională. Acest lucru se întâmplă deoarece apare atributul *Actori* ce conţine valori care pot fi descompuse în continuare. Pentru a o transforma într-o relaţie în forma normală 1, vom descompune tabelul în relaţiile **Film** şi **Distributie** aşa cum se vede din *Figura 4.2* de mai jos

TITLU	AN	TIP	REGIZOR	REGIZOR_DN	APARITIE
Notting Hill	1999	romantic	Roger M	05/06/1956	30
Lagaan	2000	drama	Ashutosh G	15/02/1968	50

Tabelul Filme

TITLU	AN	ACTORI
Notting Hill	1999	Hugh G
Notting Hill	1999	Rhys I
Lagaan	2000	Aamir K
Lagaan	2000	Gracy S

Tabelul Distributie

Figura 4.2 – Transformarea în prima formă normală. Exemplu de relaţie în forma normală 1

În *Figura 4.2*, la intersecţia dintre un rând şi o coloană a tabelului se află acum o valoare la nivel atomic care nu se mai poate descompune în continuare, astfel încât se poate spune că relaţia se află în forma normală 1, presupunând faptul că numele actorilor nu se descompun în nume şi prenume.

4.5.2 Forma normală 2 (2FN)

O relaţie se află în ***forma normală 2*** dacă se află în forma normală 1 şi fiecare atribut care nu face parte din cheie depinde de întreaga cheie, nu doar de o parte a acesteia.

Din definiţia de mai sus se poate trage concluzia că o relaţie care are cheia alcătuită dintr-un singur atribut se află cu siguranţă în forma normală 2.

Exemplu

Pentru a normaliza mai departe relaţia cu filme de mai sus, se vor elimina dependenţele referitoare doar la o parte a cheii. În acest exemplu, APARITIE depinde de AN. Acest lucru înseamnă că AN → APARITIE dar cheia candidat este {TITLU, AN}.

Astfel pentru a ajunge la forma normală 2, vom descompune în continuare tabelele de mai sus în ***relaţia FILME, relaţia APARITIE*** şi ***relaţia DISTRIBUTIE*** aşa cum se prezintă în *Figura 4.3*.

TITLU	AN	TIP	REGIZOR	REGIZOR_DN
Notting Hill	1999	romantic	Roger M	05/06/1956
Lagaan	2000	drama	Ashutosh G	15/02/1968

(a) Relatia FILME

AN	APARITIE
1999	30
2000	50

(b) Relatia APARITIE

TITLU	AN	ACTORI
Notting Hill	1999	Hugh G
Notting Hill	1999	Rhys I
Lagaan	2000	Aamir K
Lagaan	2000	Gracy S

(c) Relatia DISTRIBUTIE

Figura 4.3 – Trecerea în cea de-a doua formă normală

În figura de mai sus, fiecare atribut care nu face parte din cheie este acum depdendent de întreaga cheie şi nu doar de o parte a acesteia. Astfel, descompunerea de mai sus conduce la cea de-a doua formă normală.

4.5.3 Forma normală 3 (3NF)

O relaţie se află în ***forma normală 3*** dacă se află în forma normală 2 şi nu există atribut care nu este sau nu face parte din cheie care să depindă tranzitiv de un alt atribut care nu face parte din cheie. Acest lucru înseamnă faptul că fiecare atribut depinde direct de **cheia primară** şi nu printr-o relaţie tranzitivă în care un atribut **Z** poate depinde de un atribut **Y,** care nu este cheie iar **Y** depinde de **cheia primară** X.

Tranzitiv înseamnă, aşa cum am văzut anterior, dacă **X→Y** şi **Y→ Z**, atunci **X→Z**.

Acest lucru înseamnă că o relaţie aflată în forma normală 3 are atributele care nu fac parte din cheie mutual independente.

Exemplu

Pentru a normaliza în continuare relaţia aflată în forma normală 2, adică pentru a o aduce în forma normală 3, se vor elimina toate dependenţele tranzitive. În *Figura 4.3*, *REGIZOR_DN* depinde de *REGIZOR*, adică *REGIZOR* → *REGIZOR_DN*.

Dar cheia primară este {TITLU, AN}. În cazul de faţă {TITLU, AN} → REGIZOR şi REGIZOR → REGIZOR_DN adică există o dependenţă tranzitivă.

Din acest motiv, pentru a ajunge la forma normală 3 trebuie descompuse în continuare tabelele de mai sus în **relaţia FILME, relaţia REGIZOR**, **relaţia APARITIE** şi **relaţia DISTRIBUŢIE** aşa cum se vede din *Figura 4.4.*

TITLU	AN	TIP	REGIZOR	REGIZOR_DN
Notting Hill	1999	romantic	Roger M	05/06/1956
Lagaan	2000	drama	Ashutosh G	15/02/1968

(a) Relatia FILME

REGIZOR	REGIZOR_DN
Roger M	05/06/1956
Ashutosh G	15/02/1968

(b) Director Relation

AN	APARITIE
1999	30
2000	50

(c) Relatia APARITIE

TITLU	AN	ACTORI
Notting Hill	1999	Hugh G
Notting Hill	1999	Rhys I
Lagaan	2000	Aamir K
Lagaan	2000	Gracy S

(d) Relatia DISTRIBUTIE

Figura 4.4 – Transformarea în forma normală 3

În figura de mai sus, fiecare atribut care nu face parte din cheie este independent mutual şi dependent doar de cheie în totalitate. Astfel, descompunerea de mai sus ne-a adus în forma normală 3.

4.5.4 Forma normală Boyce-Codd (FNBC)

Forma normală Boyce-Codd este o versiune mai restrictivă a formei normale 3 care se aplică relaţiilor în care pot exista chei suprapuse.

Se spune că o relaţie se află în ***forma normală Boyce-Codd*** dacă se află în forma normală 3 şi fiecare dependenţă funcţională netrivială a acestei relaţii are o cheie candidat ca determinant, adică pentru fiecare $X \rightarrow Y$, X este o cheie.

Exemplu

Să considerăm relaţia ***Sali predare*** la un colegiu, aşa cum se prezintă în *Tabelul 4.9* de mai jos. Se presupune că fiecare profesor predă o singură disciplină.

Chei: {DISCIPLINA, ZI_PREDARE}, {ZI_PREDARE, PROFESOR}

DISCIPLINA	ZI_PREDARE	PROFESOR
Grafică	Luni	Dr. Arindham Singh
Baze de date	Luni	Dr. Emily Jose
Java	Miercuri	Dr. Prasad
Grafică	Marţi	Dr. Arindham Singh
Java	Joi	Dr. George White

Tabelul 4.9 – Relaţia SALI PREDARE

În relaţia de mai sus, nu există valori care să nu se afle la nivel atomic, motiv pentru care relaţia se află în forma normală 1.

Toate atributele fac parte din cheie, motiv pentru care relaţia este în formele normale 2 şi 3.

În relaţia de mai sus există o dependenţă funcţională Pofesor → Disciplina, dar atributul Profesor nu alcătuieşte singur cheia, motiv pentru care relaţia nu se află în forma normală FNBC. Pentru a o transforma în FNBC, se va descompune în relaţiile **EXPERTI_DOMENIU** şi **GRAFIC_PREDARE** aşa cum se vede din *Figura 4.5*.

DISCIPLINA	PROFESOR
Graphics	Dr. Arindham Singh
Databases	Dr. Emily Jose
Java	Dr. Prasad
Java	Dr. George White

(a) Relatia EXPERTI

DISCIPLINA	ZI_PREDARE	PROFESOR
Graphics	Monday	Dr. Arindham Singh
Databases	Monday	Dr. Emily Jose
Java	Wednesday	Dr. Prasad
Graphics	Tuesday	Dr. Arindham Singh
Java	Thursday	Dr. George White

(b) Relatia ORAR

Figura 4.5 – Transformarea în forma normală Boyce-Codd

Relaţia din *Tabelul 4.9* este descompusă acum în FNBC pe baza dependenţei funcţionale netriviale Profesor → Disciplina. Profesorul este acum cheie în *Figura 4.5 (a) Relatia EXPERTI*.

4.6 Proprietăţi ale descompunerii

În cadrul acestei secţiuni vom reveni la conceptul de descompunere pentru a trece în revistă câteva dintre proprietăţile de care avem nevoie.

Normalizarea reprezintă descompunerea unei scheme a unei relaţii în relaţii mai mici astfel încât fiecare dintre atribute să fie prezent în cel puţin una dintre noile relaţii, realizând astfel un proiect optimizat. Scopul este acela de a elimina redundanţele.

4.6.1 Pierderea de informaţie prin normalizare şi evitarea acesteia

Descompunerea unei relaţii $\mathbf{R}$ în relaţiile $\mathbf{X}$ şi $\mathbf{Y}$ se face **fără pierdere de informaţie** dacă nu se pierde, după descompunere, nici o informaţie din relaţia iniţială. Cu alte cuvinte, relaţia iniţială poate fi refăcută din relaţiile obţinute la descompunere fără a fi nevoie să se adauge rânduri suplimentare.

Prin urmare, dacă rezultatul descompunerii conduce la pierdere de informaţie faţă de relaţia iniţială $\mathbf{R}$ atunci descompunerea nu este corectă.

Fie $\mathbf{R}$ o relaţie care are o dependenţă funcţională $\mathbf{F}$ şi care se descompune în $\mathbf{R}_1$ şi $\mathbf{R}_2$. Descompunerea este fără pierdere de informaţie dacă una dintre următoarele dependenţe funcţionale este adevărată pentru relaţiile $\mathbf{R}_1$ şi $\mathbf{R}_2$:

$\mathbf{R}_1 \cap \mathbf{R}_2 \rightarrow \mathbf{R}_1$ sau $\mathbf{R}_1 \cap \mathbf{R}_2 \rightarrow \mathbf{R}_2$

Mai simplu, dacă atributele comune din relaţiile $\mathbf{R}_1$ şi $\mathbf{R}_2$ (adică, $\mathbf{R}_1 \cap \mathbf{R}_2$) alcătuiesc o supercheie fie pentru $\mathbf{R}_1$ fie pentru $\mathbf{R}_2$, atunci relaţia iniţială $\mathbf{R}$ nu a fost supusă unei pierderi de informaţie prin descompunere în relaţiile $\mathbf{R}_1$ şi $\mathbf{R}_2$.

O proprietate esenţială a descompunerii în proiectarea bazelor de date relaţionale este aceea că nu este permis ca aceasta să se facă cu pierdere de informaţie. O reunire în $\mathbf{R}$, fără pierdere de informaţie, a relaţiilor $\mathbf{X}$ şi $\mathbf{Y}$ obţinute prin descompunere conduce la lipsa pierderii de informaţie, nefiind nevoie să se adauge informaţie suplimentară ajutătoare.

Exemplu

Fie relaţia ***ANGAJATI***:

ID_ANG	NUME_ANG	EXP	ID_DEPT	NUME_DEPT

O descompunere ***fără pierdere de informaţie*** a relaţiei de mai sus este:

ID_DEPT	NUME_DEPT

Relaţia DEPARTAMENT

ID_ANG	NUME_ANG	EXP	ID_DEPT

Relaţia ANGAJAT

Prin reunirea relaţiilor de mai sus, (DEPARTAMENT) ∩ (ANGAJAT) = ID_DEPT iar ID_DEPT → {ID_DEPT, NUME_DEPT}, ceea ce înseamnă faptul că ID_DEPT este o supercheie a relaţiei ***DEPARTAMENT*** iar descompunerea se face fără pierdere de informaţie.

O ***descompunere cu pierdere de informaţie*** pentru situaţia de mai sus ar putea fi:

ID_DEPT	NUME_DEPT	NUME_ANG

Relaţia DEPARTAMENT

ID_ANG	NUME_ANG	EXP

Relaţia ANGAJAT

Prin reunirea relaţiilor de mai sus, (DEPARTAMENT) ∩ (ANGAJAT) = NUME_ANG care nu este o supercheie nici în tabelul cu departamente nici în cel cu angajaţi, motiv pentru care descompunerea se face cu pierdere de informaţie.

În exemplul anterior, NUME_ANG nu se ştie dacă este unic, motiv pentru care nu se poate spune cu siguranţă pentru ce departament lucrează un anumit angajat. Acest lucru conduce la pierderea de informaţie.

4.6.2 Păstrarea dependenţelor la descompunere

Prin descompunerea relaţiei **R** în relaţiile **X** şi **Y** se păstrează toate dependenţele funcţionale care există separat pe **X** şi **Y**, iar la reunire, se obţin toate dependenţele funcţionale care există în mulţimea de închidere a dependenţelor funcţionale F^+ din relaţia iniţială, **R**.

Pe o proiecţie a unei mulţimi de dependenţe funcţionale a relaţiei **R, F** reprezintă o mulţime a atributelor acesteia, iar **X** reprezintă mulţimea dependenţelor funcţionale de forma **C** → **D** în care **C** şi **D** aparţin lui **X**. Acest lucru se notează cu F_x.

În acest fel o descompunere a relaţiei **R** în relaţiile **X** şi **Y** îşi păstrează dependenţele dacă reuniunea mulţimilor de închidere a dependenţelor funcţionale din **X** şi **Y** este echivalentă cu mulţimea de închidere a dependenţelor funcţionale F^+ existente în **R.** Acest lucru înseamnă că prin F_x U F_y se obţine F^+ în care F_x reprezintă mulţimea dependenţelor funcţionale din F^+ care trebuie verificate numai în **X,** iar F_y reprezintă mulţimea dependenţelor funcţionale care se verifică doar în **Y**.

Acest lucru conduce la ideea că având o descompunere cu păstrare de dependenţe, dacă toate constrângerile se verifica pe tabelul obţinut prin descompunere, nu mai trebuie să facem verificarea şi pe tabelul iniţial.

Descompunerea cu păstrare de dependenţe nu presupune şi descompunerea fără pierdere de informaţie la cuplare şi invers. În timp ce joncţiunea fără pierdere de informaţie este obligatorie la descompunere, păstrarea dependenţelor nu este întotdeauna posibilă.

4.7 Acoperirea minimală

Acoperirea minimală, F_c reprezintă cea mai mică mulţime de dependenţe funcţionale astfel încât mulţimea de închidere a dependenţelor funcţionale atât a acoperirii minimale cât şi

mulţimii date de dependenţe funcţionale **F** ale relaţiei **R** sunt echivalente. Acest lucru conduce la $F^+ = F_c^+$

Scopul folosirii acoperirii minimale este acela de a face o verificare mai uşoară a constrângerilor la introducerea datelor într-un sistem de gestiune a bazelor de date relaţionale. Datele introduse în cadrul tabelelor trebuie să satisfacă constrângerile care există într-o anumită relaţie. Folosind acoperirea minimală, vom face un număr minim de verificări în raport cu mulţimea iniţială de dependenţe funcţionale care există într-o relaţie şi, prin urmare, verificarea constrângerilor este mult mai economică.

Acoperirea minimală, F_c provine din mulţimea **F** astfel încât:

1. Partea dreaptă a fiecărei dependenţe funcţionale din acoperirea minimală este un singur atribut.
2. Dacă se elimină un atribut din partea stângă a fiecărei dependenţe funcţionale, închiderea acoperirii minimale se schimbă.
3. Eliminarea unei dependenţe funcţionale din acoperirea minimală produce modificarea închiderii acoperirii minimale.

Acoperirea minimală a unei mulţimi date de dependenţe funcţionale nu este unică.

Atribute externe.

Pentru fiecare $\alpha \rightarrow \beta$ care există în **F**, se defineşte un atribut extern ca fiind acel atribut care dacă este eliminat din α şi β nu produce modificarea mulţimii de închidere a dependenţelor funcţionale.

Prin urmare, dacă se elimină **A** din α în F_c, se obţine $(F_c - \{\alpha \rightarrow \beta\}) \cup (\{\alpha - A\} \rightarrow \beta)$ iar prin eliminarea lui **B** din β în F_c se obţine $(F_c - \{\alpha \rightarrow \beta\}) \cup (\alpha \rightarrow \{\beta - B\})$.

Pentru a calcula acoperirea minimală F_c, se parcurg paşii:

1. Prin folosirea regulilor de descompunere puse la dispoziţie prin axiomele lui Armstrong se descompune fiecare dependenţă funcţională pentru a obţine un singur atribut în partea dreaptă.
2. Se reduce partea stângă a fiecărei dependenţe funcţionale la F_c eliminând toate atributele externe.
3. Se folosesc axiomele lui Armstrong pentru a reduce cât mai mult dependenţele funcţionale redundante rămase în acoperirea minimală astfel încât inchiderea sa să nu se modifice, ceea ce înseamnă că trebuie să se menţină egalitatea $F^+ = F_c^+$

Exemplu

Fie relaţia **R (A, B, C, D)** cu următoarea mulţime de dependenţe funcţionale **F**:

$A \rightarrow BC$,

$B \rightarrow C$,

$A \rightarrow B$,

AB → C,

AC → D

Pentru a determina acoperirea minimală F_c se urmează paşii prezentaţi antreior, modificările fiind marcate cu litere îngroşate:

Pas 1: Se reduce fiecare dependenţă funcţională din cadrul mulţimii, astfel încât partea dreaptă a expresiei să conţină un singur atribut

(folosind descompunerea: Dacă **X → YZ**, atunci **X → Y** şi **X → Z**).

A → B, **A → C,**

B → C,

A → B,

AB → C,

AC → D

Pas 2: Reducerea părţii stângi a expresiei pentru a elimina atributele externe, dacă există.

Avem **B → C** şi **AB → C** deci **A** este extern în **AB → C**.

Se înlocuieşte cu **B → C** care deja există în mulţimea dependenţelor funcţionale.

Asemănător, **A → C** şi **AC → D** astfel încât **C** este extern în **AC → D**.

Se înlocuiesşte cu **A → D**

Astfel mulţimea minimală devine:

A → B, **A → C**,

B → C,

A → B,

B → C,

A → D

Pas 3: Eliminarea dependenţelor funcţionale redundante

Avem duplicatele **A → B** şi **B → C**. Pe baza acestora se obţine relaţia tranzitivă:

A → C,

După care mulţimea minimală ne conduce la rezultatul:

A → B,

B → C,

A → D

Acoperirea minimală F_c **= { A → B , B → C, A → D }.**

4.8 Fuziunea schemelor în forma normală 3

Fuziunea schemelor în forma normală 3 este un concept care se aplică abordând tehnica de jos-în-sus şi este utilizat pentru a obţine o cuplare fără pierdere de informaţie şi cu păstrarea depdendenţelor la descompunerea unei relaţii în forma normală 3. Procesul este unul de tip de jos-în-sus deoarece se porneşte cu realizarea mulţimii minime de dependenţe funcţionale, creându-se schemele obţinute prin descompunere în mod direct prin adăugarea câte unei scheme pe rând în listă, obţinându-se astfel relaţiile în forma normală 3.

Schemele obţinute prin descompunere se bazează pe acoperirea minimală a unei mulţimi date de dependenţe funcţionale existente într-o relaţie $\mathbf{R}$. După aceea se verifică dacă schemele obţinute nu pierd, prin descompunere, informaţie la reunirea acestora. Dacă se pierde informaţie, se adaugă o altă schemă care conţine cheia candidat astfel încât să se rezolve problema descompunerii în forma normală 3.

Procedura folosită pentru descompunerea relaţiei $\mathbf{R}$ în $\mathbf{R}_1$, $\mathbf{R}_2$,$\mathbf{R}_3$ ..... , $\mathbf{R}_n$ este:

1. Pentru fiecare dependenţă funcţională, $\alpha \rightarrow \beta$ din acoperirea minimală $\mathbf{F}_c$, dacă nici una dintre schemele $\mathbf{R}_1$, $\mathbf{R}_2$,$\mathbf{R}_3$ ..... , $\mathbf{R}_n$ nu conţine α, β se adaugă $\mathbf{R}_i = (\alpha, \beta)$.
2. Pentru fiecare relaţie $\mathbf{R}_i$ din lista $\mathbf{R}_1$, $\mathbf{R}_2$,$\mathbf{R}_3$ ..... , $\mathbf{R}_n$ se verifică dacă cel puţin una dintre aceste relaţii conţine o cheie candidat din $\mathbf{R}$. Dacă nu conţine, se adaugă o altă relaţie mulţimii schemelor obţinute prin descompunere, $\mathbf{R}_{n+1}$, astfel încât $\mathbf{R}_{n+1}$ să fie cheie candidat în $\mathbf{R}$.

4.9 Descompunerea în forma normală 3

Descompunerea în forma normală 3 se poate realiza folosind metoda se sus-în-jos prin procesul de ***normalizare*** în care o relaţie este descompusă pe baza definiţei formei normale 3, după care se obţine asigurarea că nu s-a pierdut informaţie şi s-au păstrat dependenţele, verificându-se relaţiile obţinute. Secţiunea 4.5.3 prezintă această procedură.

În această secţiune vom prezenta un exemplu de descompunere în forma normală 3 a schemelor relaţionale urmând tehnica de abordare de jos-în-sus, folosind algoritmul explicat anterior.

Exemplu

Se porneşte de la schema relaţională **Carte(nume_carte, autor, autor_dn, vânzari, evaluare)** care are următoarea mulţime de dependenţe funcţionale:

(nume_carte, autor) → vanzari

autor → autor_dn

vanzari → evaluare

şi cheia candidat {nume_carte, autor}

Folosind algoritmul de fuziune descris în secţiunea anterioară, se descompune relaţia Carte în scheme aflate în forma normală 3 astfel:

Pas 1: Mulţimea de dependenţe funcţionale de mai sus reprezintă o acoperire minimală.

Pas 2: Pe baza depdendenţelor funcţionale date se obţin relaţiile:

(nume_carte, autor, vanzari)

(autor, autor_dn)

(vanzari, evaluare)

Pas 3: Deoarece relaţia (nume_carte, autor, vanzari) conţine cheia candidat, nu mai trebuie adăugate alte relaţii acestor scheme.

Descompunerea în forma normală 3 conduce la relaţiile:

(nume_carte, autor, vanzari)

(autor, autor_dn)

(vanzari, evaluare)

4.10 Forma normală 4 (4FN)

Forma normală 4 poate fi înţeleasă pe baza dependenţelor multivaloare. O relaţie se află în forma normală 4 atunci când în acea relaţie nu există decât o singură dependenţă multivaloare.

Pentru a înţelege acest lucru să vedem ce înseamnă o dependenţă multivaloare.

4.10.1 Dependenţe multivaloare

Se spune că un atribut **A** multi-determină un alt atribut **B** atunci când fiecărei valori a lui **A**, îi corespund mai multe valori în **B**.

Dependenţele multivaloare (DMV) se notează sub forma **A →→ B.** Aceasta înseamnă că A îl multi-determină pe **B, B** este multi-dependent de **A** sau **A** săgeată dublă către **B**.

O relaţie se află în forma normală 4 atunci când nu există două sau mai multe DMV în relaţia respectivă, iar atributele multivaloare sunt mutual independente. Din punct de vedere formal o relaţie **R(A,B,C)** se află în forma normală 4 dacă în respectiva relaţie există o DMV astfel încât **A →→ B** şi **A →→ C,** iar **B** şi **C** sunt independente, adică, de exemplu, C nu este o submulţime a lui B.

Exemplu

Fie relaţia ***inghetata*** prezentată în *Tabelul 4.10*.

Vanzator	Tip_I	Aroma_I
Amul	Scoop	Vanilie
Amul	Softy	Vanilie

Amul	Scoop	Ciocolata
Amul	Softy	Ciocolata
Baskin Robbins	Scoop	Ciocolata
Baskin Robbins	Sundae	Ciocolata
Baskin Robbins	Scoop	Capsuni
Baskin Robbins	Sundae	Capsuni
Baskin Robbins	Scoop	Unt şi zahăr brun
Baskin Robbins	Sundae	Unt şi zahăr brun

Tabelul 4.10 – Relaţia *inghetata* aflată în forma normală FNBC

Relaţia de mai sus se află în forma normală FNBC, astfel că toate atributele sunt parte a cheii candidat.

Există următoarele dependenţe multivaloare:

Vanzator →→ Tip_I

Vanzator→→ Aroma_I

În acest fel, în relaţia de mai sus, există o serie de redundanţe ce conduc către apariţia de anomalii. Din acest motiv, se aduce relaţia la forma normală 4, aşa cum se prezintă în *Figura 4.6*.

VANZATOR	Tip_I
Amul	Scoop
Amul	Softy
Baskin Robbins	Scoop
Baskin Robbins	Sundae

Relatia TIP_INGHETATA

VANZATOR	Aroma_I
Amul	Vanilla
Amul	Chocolate
Baskin Robbins	Chocolate
Baskin Robbins	Strawberry
Baskin Robbins	Butterscotch

Relatia AROMA_INGHETATA

Figura 4.6 – Relaţii în forma normală 4

Relaţiile din *Figura 4.6* sunt descompuse în forma normală 4 deoarece nu există decât o singură dependenţă multivalorică (DMV) mutual independentă în relaţia ce rezultă la descompunere.

4.11 Alte forme normale

Mai există alte trei forme normale suplimentare. Acestea sunt denumite ***forma normală 5*** [4.10], ***forma normală cu dependenţe de domeniu/cheie (FNDC)*** [4.11] şi ***forma normală 6*** [4.12, 4.13]. Aceste forme sunt folosite în cazuri speciale, motiv pentru care nu vor fi discutate aici. Pentru a afla mai multe informaţii despre aceste forme normale se pot urmări referinţele bibliografice specificate.

4.12 Studiu de caz cu sistemul de management al unei biblioteci - Partea 2 din 3

În această parte a studiului de caz ne vom ocupa de realizarea modelului logic al sistemului care porneşte de la modelul conceptual. Modelul logic are rolul de validare a modelului conceptual folosind tehnica normalizării, în care relaţiile sunt verificate pentru a vedea dacă îndeplinesc o serie de reguli numite forme normale. Modelul logic este necesar pentru a elimina redundanţele pe care modelul conceptual nu le poate detecta (în modelul conceptual nu există noţiunea de redundanţă).

Tabelul următor prezintă corespondenţa dintre modelul conceptual şi modelul logic din punct de vedere al folosirii termenilor:

Concept din modelarea conceptuală	Concept din modelarea logică
Nume de tip de entitate	Variabilă de tip relaţie, R
Tip de entitate	Relaţie
Entitate	Tuplu
Atribut	Atribut, A1, A2, etc.
Tip de relaţie	O pereche cheie primară-cheie externă
Identificator unic	Cheie primară

În exemplul nostru, transformarea modelului conceptual în model logic va însemna modificarea schemei după cum urmează (cheia primară este subliniată)

CITITOR = {ID_CITITOR, NUME, PRENUME, EMAIL, TEL, ADRESA, ID_CARTE, DATA_IMPRUMUT, DATA_RETUR}

AUTOR = {ID_AUTOR, NUME, PRENUME, EMAIL, TEL, ADRESA)

CARTE = {ID_CARTE, TITLU, EDITIE, AN, PRET, ISBN, PAGINI, INTERVAL, DESCRIERE}

COPIE = {ID_CARTE, STARE}

LISTA_AUTOR = {ROL}

Pentru a păstra tipurile de relaţii iniţiale în vederea respectării integrităţii datelor, şi pentru a evita pierderea de informaţie, trebuie introduse cheile externe corespunzătoare, motiv pentru care relaţiile devin după cum urmează (cheia externă este subliniată cu linie întreruptă):

CITITOR = {ID_CITITOR, ID_COPIE, NUME, PRENUME, EMAIL, TEL, ADRESA, DATA_IMPRUMUT, DATA_RETUR}

AUTOR = {ID_AUTOR, NUME, PRENUME, EMAIL, TEL, ADRESA}

CARTE = {ID_CARTE, TITLU, EDITIE, AN, PRET, ISBN, PAGINI, INTERVAL, DESCRIERE}

COPIE = {ID_COPIE, ID_CITITOR, ID_CARTE, STARE}

LISTA_AUTOR = {ID_AUTOR, ID_CARTE, ROL}

Acum să spunem că, aşa cum a fost stabilit cu ajutorul regulilor de domeniu, un cititor poate împrumuta doar o singură carte pe zi. Din acest motiv relaţia CITITOR trebuie să aibă cheia primară alcătuită din atributele: {ID_CITITOR, ID_COPIE, DATA_IMPRUMUT}.

Să verificăm acum fiecare relaţie pentru a vedea dacă respectă regulile formelor normale.

Pentru a valida dacă o relaţie se află în prima formă normală trebuie să ne asigurăm că nu există ‘gruprui repetitive’, deci atributele se află la nivel atomic. În acest exemplu nu toate atributele se află la nivel atomic, cum ar fi ADRESA din relaţia CITITOR şi din relaţia AUTOR. Acest atribut se poate descompune în {ADRESA, ORAS, TARA}. În acelaşi timp, se observă faptul că există grupuri repetitive în relaţia CITITOR, deoarece un cititor poate împrumuta mai multe cărţi în timp, astfel că relaţia CITITOR trebuie descompusă în CITITOR şi IMPRUMUT:

CITITOR = {ID_CITITOR, ID_COPIE, NUME, PRENUME, EMAIL, TEL, ADRESA}

IMPRUMUT = {ID_CITITOR, ID_COPIE, DATA_IMPRUMUT, DATA_RETUR}

După descompunere se constată faptul că toate relaţiile se află în forma normală 1.

Să vedem în continuare dacă ne aflăm şi în forma normală 2. Regula formei normale 2 spune că fiecare atribut al unei relaţii trebuie să depindă de întreaga cheie şi nu doar de o parte a acesteia. În exemplul nostru fiecare relaţie care are o cheie primară alcătuită dintr-un singur atribut, deci se află în mod automat în forma normală 2, astfel încât nu trebuie verificate decât relaţiile la care cheia primară este o cheie compusă. Verificând aceste relaţii se constată faptul că atât DATA_IMPRUMUT cât şi DATA_RETUR depind de ambele atribute ID_CITITOR şi ID_COPIE, iar ROL depinde de ID_AUTOR şi de

ID_CARTE, motiv pentru care se concluzionează că toate relaţiile se află în forma normală 2.

Pentru forma normală 3 trebuie verificat dacă există un atribut care nu face parte din cheie care să depindă de un alt atribut care nu face parte din cheie. În exemplul nostru, nu există o astfel de situaţie, motiv pentru care se constată că toate relaţiile se află în forma normală 3 şi că nu există motiv pentru continuarea descompunerii.

Acum ne putem întoarce la modelul conceptual şi putem face modificările necesare. Se va crea tipul de entitate IMPRUMUT şi se va crea tipul de relaţie corespunzător, stabilind cheia externă. Cu ajutorul unui instrument cum ar fi InfoSphere Data Architect (IDA) vom realiza foarte uşor astfel de modificări care sunt prezentate sintetic în figura de mai jos în care se vede modelul logic final:

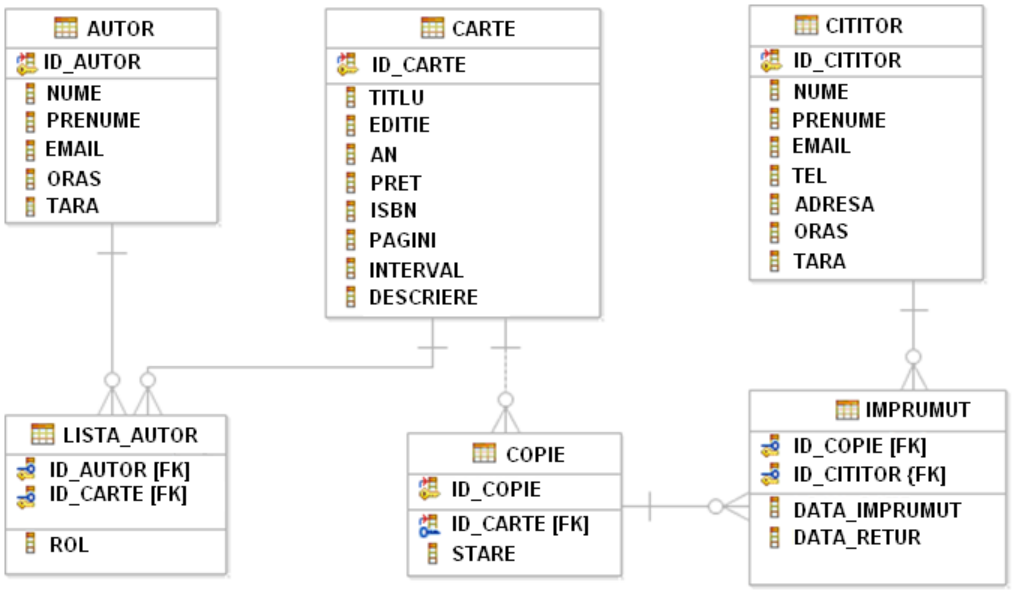


Figura 4.7 – Modelul logic final

O altă abordare ce poate fi la fel de corectă este şi aceea în care se porneşte cu modelul logic în care se introduc toate relaţiile şi atributele ce au fost identificate în partea I a studiului de caz (vezi capitolul precedent) după care se trece la rafinarea modelului prin introducerea cheilor primare şi normalizarea relaţiilor. La final trebuie să se ajungă la acelaşi model logic de mai sus.

Partea a III-a a acestui studiu de caz continuă în capitolul 5, unde vom arăta felul în care se transformă modelul logic în model fizic.

4.13 Rezumat

În cadrul acestui capitol a fost descris felul în care se modelează obiectele din lumea reală,

astfel încât să se obţină tabelele din cadrul bazei de date relaţionale, proprietăţile ce trebuie asociate acestor obiecte pentru a deveni coloane în aceste tabele, precum şi asocierile care se stabilesc între tabele pentru a fi cât mai aproape de ceea ce se întâmplă în realitate.

Existenţa datelor redundante în cadrul acestor tabele provoacă o serie de probleme, cum ar fi anomaliile de inserare, de actualizare sau de ştergere, motiv pentru care trebuie revizuite schemele relaţionale cu scopul declarat de a obţine o redundanţă cât mai mică posibil.

În acest scop s-au folosit formele normale cu ajutorul cărora se verifică tabelele bazei de date, astfel încât procesul de normalizare să ofere un proiect optim al bazei de date relaţionale. Fiecare formă normală superioară provine din una anterioară. Dependenţele funcţionale dintre atributele unui tabel ne ajută la realizarea descompunerii acestor tabele.

Proprietăţile dependenţelor funcţionale ale atributelor stabilite prin axiomele lui Armstrong şi mulţimea de închidere fac ca activitatea să devină mai eficientă pentru a putea efectua verificări cât mai economic posibil. Relaţiile trebuie descompuse în aşa fel încât să nu se piardă informaţie şi să se păstreze dependenţele existente iniţial.

Prin parcurgerea acestui capitol veţi putea face un proiect complet al unei baze de date relaţionale, astfel încât toate informaţiile existente în realitate să poată fi modelate în cadrul bazei de date, realizându-se totodată şi optimizările necesare pentru a gestiona eficient spaţiul de stocare, fără a avea redundanţe şi cu posibilitatea extragerii cu uşurinţă a datelor din baza de date.

4.14 Exerciţii

1. Să se determine atributele din mulţimea de închidere pentru fiecare atribut din relaţia dată **R(A,B,C,D,E)** pentru care mulţimea dependenţelor funcţionale este:

   {AB →C , A → DE , B → D, A → B , E→C }

   Să se identifice în acelaşi timp şi supercheia.

2. În ce formă normală se află relaţia următoare?

   Comanda (ID_PRODUS, ID_CLIENT , PRET, CANTITATE, TIP_COMANDA)

   În care (ID_PRODUS, ID_CLIENT) reprezintă cheia candidat, iar TIP_COMANDA are valorile: ‘lux’ dacă preţurile sunt peste 1000$ şi ‘obişnuit’ dacă preţurile sunt sub 1000$.

   Să se normalizeze în continuare relaţia pentru a o aduce la forma normală 3..

3. Să se determine acoperirea minimală a relaţiei **R (A, B, C, D)** pe baza mulţimii de dependenţe funcţionale:

   AB → C, B→C, A→CD

4. Să se aducă relaţia CARTE_BIBLIOTECA (ID_CARTE, NUME_CARTE, AUTOR, SUBIECT) la forma normală 3 fără a se pierde informaţie sau dependenţele

existente.

5. Să se determine mulţimea de închidere a dependenţelor funcţionale F^+ pentru relaţia

   R(A,B,C,D,E) care are următoarea mulţime de dependenţe funcţionale

   { **AB →C, A → DE, B → D, A → B, E→C** }

4.15 Întrebări recapitulative

1. Să se arate că următoarea descompunere a unei relaţii despre organizarea unei nunţi se face fără pierdere de informaţie sau dependenţe:

   COMANDA (ID_CLIENT, MIREASA, MIRE, BUGET)

   TIP_NUNTA (BUGET, TIP_NUNTA)

2. Fie relaţia ***Telefon mobil***:

MOBIL	BRAND	LOCATIE_BIROU
N93	Nokia	New Delhi
Diamond	HTC	Hyderabad
N97	Nokia	New Delhi
MotoSlim	Motorola	Mumbai
3315	Nokia	New Delhi
ZN50	Motorola	Mumbai

Relaţia TELEFON_MOBIL

   Care dintre următoarele comenzi de modificare în cazul acestei relaţii va produce o anomalie de actualizare?

   A. UPDATE **TELEFON_MOBIL** set MOBIL = ‘6600’ where MOBIL = ‘N97’

   B. UPDATE **TELEFON_MOBIL** set BRAND = ‘Samsung’ where MOBIL = ‘ZN50’

   C. UPDATE **TELEFON_MOBIL** set LOCATIE_BIROU = ‘Bangalore’ where MOBIL = ‘N97’

   D. UPDATE **TELEFON_MOBIL** set BRAND = ‘Samsung’ , MOBIL = ‘X210’ where MOBIL = ‘MotoSlim’

   E. Nici una dintre cele enumerate

3. Care este forma normală a ***relaţiei BIBLIOTECA_CARTE*** de mai jos:

   BIBLIOTECA_CARTE (ID_CARTE, NUME_CARTE, AUTOR, SUBIECT)

A. Prima formă normală

B. A doua formă normală

C. A treia formă normală

D. Forma normală Boyce - Codd

E. A patra formă normală

4. Care dintre următoarele nu poate fi determinată pe baza proprietăţilor dependenţelor funcţionale?

   A. Mulţimea de închidere a dependenţelor funcţionale, $\mathbf{F}^+$

   B. Descompunerea fără pierdere de informaţie

   C. $\mathbf{X} \rightarrow \mathbf{Y}$ aparţine lui $\mathbf{F}^+$

   D. Supercheia

   E. Nici una dintre cele enumerate

5. În cazul unei relaţii aflate în forma normală Boyce-Codd, dacă există mai multe dependenţe multivalorice, aceasta se află în forma normală 4 numai dacă:

   A. Dependenţele multivalorice sunt mutual independente.

   B. Dependenţele multivalorice se află în legătură unele cu altele.

   C. Cheia candidat determină dependenţele multivalorice.

   D. Cheile candidat sunt identice.

   E. Nici una dintre cele enumerate

6. Care dintre următoarele afirmaţii este incorectă?

   A. Descompunerea fără pierdere de informaţie trebuie să se facă întotdeauna.

   B. Dependenţele funcţionale sunt un fel de constrângeri de integritate.

   C. Păstrarea dependenţelor implică joncţiunea fără pierdere de informaţie şi viceversa.

   D. BCNF nu se poate obţine în orice situaţie.

   E. Nici una dintre cele enumerate

5

Capitolul 5 – Introducere în SQL

Limbajul structurat de interogare (SQL) este un limbaj de nivel înalt care permite utilizatorilor să opereze cu date relaţionale. Unul dintre aspectele de forţă ale limbajului SQL este acela că utilizatorii nu trebuie decât să specifice informaţia de care au nevoie fără a fi nevoiţi să cunoască felul în care aceasta este extrasă. Sistemul de management al bazei de date este responsabil cu stabilirea căii de acces pentru a ajunge la date. SQL lucrează cu mulţimi, adică va extrage rânduri din unul sau mai multe tabele.

SQL are trei sublimbaje, în funcţie de funcţionalitatea cerută:

- DDL – Data definition language (limbaj de definire a datelor) folosit pentru a defini, modifica sau elimina obiecte din baza de date
- DML – Data manipulation language (limbaj de manipulare a datelor) folosit pentru a citi şi modifica date
- DCL – Data control language (limbaj de control al datelor) folosit pentru a permite sau retrage autorizări

În cadrul acestui capitol se va face o prezentare a istoricului limbajului SQL şi se va arăta felul în care se lucrează cu ajutorul acestui limbaj. Ne vom concentra atenţia spre patru operaţii de bază din SQL folosite de cele mai multe orii în cadrul aplicaţiilor: Create, Read, Update şi Delete (CRUD).

5.1 Istoricul SQL

Don Chamberlin şi Ray Boyce de la compania IBM au elaborat limbajul SQL în anii 1970 în cadrul proiectului System R, care îşi propunea să ofere o implementare practică a modelului relaţional propus de către Codd.

La început, limbajul a fost denumit “Structured English Query Language” (limbaj structurat de interogare în limba engleză) sau SEQUEL, dar a fost schimbat ulterior în SQL deoarece SEQUEL aparţinea deja unei companii din Marea Britanie.

SQL este astăzi recunoscut ca limbaj standard pentru lucrul cu bazele de date relaţionale. SQL a fost adoptat ca limbaj standard în anul 1986 de către organizaţia ***American National Standards Institute*** (***ANSI***) şi de către ***International Standards Organization*** (***ISO***) în anul 1987. Din momentul în care a devenit standard, limbajul SQL a fost modificat

de şase ori. Ultima modificare a avut loc în anul 2008 iar versiunea este cunoscută sub denumirea de SQL:2008.

SQL este un limbaj care foloseşte limba engleză pentru a elabora cuvinte cheie şi construcţii specifice bazelor de date, fiind simplu de utilizat şi înţeles. Limbajul suportă diverse mecanisme de acces pentru a formula o serie de cerinţe şi proceduri. Pe parcursul secţiunilor care urmează se vor prezenta, unul câte unul, aceste mecanisme.

5.2 Definirea în SQL a schemei bazei de date relaţionale

Aşa cum s-a arătat în capitolele anterioare, schema unei baze de date relaţionale reprezintă o descriere formală a tuturor relaţiilor şi asocierilor din cadrul unei baze de date. Cu ajutorul limbajului SQL se poate realiza o implementare fizică a unei astfel de scheme (ceea ce mai este cunoscut şi sub denumirea de "model fizic de date"). Deşi cei mai mulţi dintre producătorii de baze de date respectă standardele ANSI şi ISO pentru limbajul SQL, există totuşi o serie de diferenţe de sintaxă la fiecare dintre aceştia. Din acest motiv, fiecare model fizic de date este specific unui anumit produs de baze de date. În această carte se va folosi produsul DB2 Express-C, versiunea gratuită a serverului de baze de date IBM DB2.

În cadrul unui model fizic de date există o serie de elemente şi proprietăţi care vor fi luate în discuţie în cadrul capitolului de faţă, prezentându-se totodată şi implementarea specifică în SQL.

5.2.1 Tipuri de date

La fel ca în cazul oricărui alt limbaj de programare şi bazele de date lucrează cu o serie de tipuri de date care se folosesc pentru stabilirea valorilor ce pot fi introduse în cadrul coloanelor unui tabel. Dintre tipurile de date folosite în limbajul SQL amintim: **`integer`**, **`float`**, **`decimal`**, **`char, date, time, blob`** etc.

De asemenea, există şi posibilitatea de a crea date definite de utilizator, dar acestea sunt limitate la combinaţii între tipurile de bază suportate. Tipuri de date definite de utilizatori pot fi, de exemplu, **`addresa`**, **`tara, telefon`**, **`CNP`**, **`cod postal`**, pentru a da doar câteva exemple.

5.2.1.1 Tipuri de date calendaristice

Toate produsele de baze de date suportă diverse funcţii şi tipuri de date calendaristice specifice. DB2 foloseşte următoarele tipuri de ate calendaristice:

- Date (YYYY-MM-DD)
- Time (HH:MM:SS)
- Timestamp (YYYY-MM-DD-HH:MM:SS:ssssss)

în care, ssssss, reprezintă o parte a timpului curent, în milisecunde.

Următoarele reprezintă o parte dintre funcţiile care folosesc tipurile de date calendaristice:

- Year

- Month
- Day
- Dayname
- Hour
- Minute
- Second
- Microsecond

5.2.2 Crearea unui tabel

Un tabel reprezintă o mulţime de date organizate şi păstrate în rânduri şi coloane. Un tabel poate conţine date, de exemplu, despre studenţi, profesori, discipline, cărţi etc.

Tipurile de entităţi din modelul conceptual se transformă în tabele în cadrul unei baze de date, iar atributele acestora se transformă în coloane ale tabelului.

Să începem prin crearea un tabel simplu cu o cantitate minimă de informaţie. De exemplu:

```
create table myTable (col1 integer)
```

Comanda de mai sus creează un tabel care are numele ***myTable***, cu o singură coloană numită ***col1*** care poate păstra date de tip ***integer***. Acest tabel poate accepta orice valoare întreagă sau nulă ce poate fi introdusă în coloana ***col1.*** Valorile nule sunt prezentate ulterior în cadrul acestei secţiuni.

5.2.2.1 Valori implicite

La introducerea datelor în cadrul unui tabel, câteodată poate fi util să se genereze automat valori implicite pentru unele dintre coloanele tabelului. De exemplu, atunci când utilizatorii unui site se înregistrează la site-ul respectiv, dacă lasă necompletat câmpul ***profession*** atunci coloana corespunzătoare din tabelul ***USERS*** ia valoarea ***Student***. Acest lucru este prezentat cu ajutorul următoarei comenzi:

```
CREATE TABLE USERS
    (NAME       CHAR(20),
     AGE        INTEGER,
     PROFESSION VARCHAR(30) with default 'Student')
```

Pentru a introduce o coloană care ia ca valoare un număr de departament introdus în mod incremental, pornind de la un număr introdus iniţial, se poate folosi comanda:

```
   CREATE TABLE DEPT
     (DEPTNO     SMALLINT      NOT NULL
                 GENERATED ALWAYS AS IDENTITY
                 (START WITH 500, INCREMENT BY 1),
      DEPTNAME   VARCHAR(36)   NOT NULL,
      MGRNO      CHAR(6),
      ADMRDEPT   SMALLINT      NOT NULL,
      LOCATION   CHAR(30))
```

Comanda SQL de mai sus crează tabelul DEPT, în care coloana DEPTNO are valori implicite începând cu 500 şi incrementate cu un pas. La introducerea rândurilor în acest tabel, nu trebuie introduse valori în câmpul DEPTNO, baza de date generând în mod automat valori pentru fiecare rând inserat.

5.2.2.2 Valori nule

O valoare nulă reprezintă o stare necunoscută. De exemplu, un tabel ce păstrează date despre notele studenţilor poate să permită valori nule. Acest lucru înseamnă pentru profesor faptul că studentul respectiv fie nu a fost încă evaluat, fie nu s-a prezentat la examen. Această valoare face distincţia faţă de valoarea zero care înseamnă faptul că studentul a dat examenul, dar a greşit la toate întrebările. Există situaţii în care nu se acceptă valori nule. De exemplu, dacă un câmp al unui tabel care conţine numele ţărilor trebuie să ofere o valoare aplicaţiei, atunci trebuie să se interzică apariţia valorilor nule, astfel:

```
create table myTable (name varchar(30), country varchar(20) NOT NULL)
```

Comanda de mai sus semnifică faptul că valorile nule nu sunt premise în coloana ***country,*** dar valorile duplicate sunt permise

5.2.2.3 Constrângeri

Constrângerile permit stabilirea unor reguli pentru datele introduse în cadrul unui tabel. Există următoarele tipuri de constrângeri:

- UNIQUE interzice introducerea valorilor duplicat în cadrul unui tabel. Acest lucru se specifică cu ajutorul comenzii CREATE TABLE prin folosirea cuvântului cheie UNIQUE. Se pot introduce valori nule. Constrângerii UNIQUE îi este întotdeauna asociat un index.
- PRIMARY KEY este similară constrângerii UNIQUE cu deosebirea faptului că sunt interzise valorile nule. Cheile primare au întotdeauna asociat un index.
- REFERENCES se foloseşte pentru a asigura integritatea referenţială care permite gestionarea relaţiilor dintre tabele. Acest lucru se va discuta în detaliu în secţiunea următoare.
- CHECK se foloseşte pentru a verifica dacă valorile introduse în cadrul unei coloane a unui tabel respectă regulile stabilite în cadrul constrângerii.

Exemplul următor prezintă definiţia unui tabel care are câteva constrângeri de tip CHECK şi o cheie primară:

```
CREATE TABLE EMPLOYEE
  (ID          INTEGER     NOT NULL  PRIMARY KEY,
   NAME        VARCHAR(9),
   DEPT        SMALLINT    CHECK (DEPT BETWEEN 10 AND 100),
   JOB         CHAR(5)     CHECK (JOB IN ('Sales','Mgr','Clerk')),
   HIREDATE    DATE,
   SALARY      DECIMAL(7,2),
   CONSTRAINT  YEARSAL     CHECK ( YEAR(HIREDATE) > 1986
                             OR SALARY > 40500 )
```

```
)
```

Înainte de a se introduce date în cadrul acestui tabel trebuie respectate patru constrângeri:

- Cheia primară pe coloana ID

  Aceasta înseamnă faptul că nu sunt premise valorile duplicat.

- Constrângerea CHECK pe coloana DEPT

  Se permite introducerea datelor în această coloană doar dacă valorile acestora sunt între 10 şi 100.

- Constrângerea CHECK pe coloana JOB

  Se permite introducerea datelor în această coloană doar dacă valorile acestora sunt 'Sales', 'Mgr' sau 'Clerk'.

- Constrângerea CHECK pe combinaţia de coloane HIREDATE şi SALARY

  Se permite introducerea datelor doar dacă data angajării este superioară anului 1986 iar salariul este mai mare de 40500.

5.2.2.4 Integritatea referenţială

Aşa cum s-a arătat în capitolul 2, integritatea referenţială permite stabilirea de asocieri între tabele. Prin folosirea unei perechi cheie primară-cheie externă se verifică validitatea datelor introduse. Integritatea referenţială reduce complexitatea codului aplicaţiei prin eliminarea nevoii de a face această verficare la nivel de aplicaţie.

Un tabel ale cărui valori depind de valorile altui tabel se numeşte ***dependent, sau tabel copil,*** iar un tabel la care se face referire se numeşte tabel ***de bază*** sau ***părinte***. Doar tabelele care au coloane cu restricţiile UNIQUE sau PRIMARY KEY pot fi referite de către alte tabele prin intermediul cheii externe.

Integritatea referenţială poate fi stabilită în cadrul definiţiei unui tabel sau după crearea unui tabel, aşa cum se prezintă în în exemplul de mai jos în care sunt prezentate trei sintaxe diferite:

Sintaxa 1:

```
CREATE TABLE DEPENDANT_TABLE
     (ID          INTEGER REFERENCES BASE_TABLE(UNIQUE_OR_PRIMARY_KEY),
      NAME        VARCHAR(9),
      :
      :
      :
     );
```

Sintaxa 2:

```
CREATE TABLE DEPENDANT_TABLE
     (ID          INTEGER,
      NAME        VARCHAR(9),
```

```
      :
      :
      :,
      CONSTRAINT constraint_name FOREIGN KEY (ID)
            REFERENCES BASE_TABLE(UNIQUE_OR_PRIMARY_KEY)
     );
```

Sintaxa 3:

```
CREATE TABLE DEPENDANT_TABLE
     (ID           INTEGER,
      NAME         VARCHAR(9),
      :
      :
      :
     );

ALTER TABLE DEPENDANT_TABLE
    ADD CONSTRAINT constraint_name FOREIGN KEY (ID)
    REFERENCES BASE_TABLE(UNIQUE_OR_PRIMARY_KEY);
```

În codul de mai sus, atunci când nu se specifică numele unei constrângeri, acesta este creat în mod automat de către sistemul DB2. Şirul generat are lungimea de 15 caractere, de exemplu ‘CC1288717696656’.

Ce se întâmplă atunci când aplicaţia trebuie să elimine un rând din tabelul de bază care este referit în tabelul copil? Aşa cum s-a arătat în *capitolul 2*, există trei reguli diferite care gestionează ştergerile şi actualizările din cadrul unui tabel, iar comportamentul de ştergere depinde de următoarele construcţii introduse la crearea tabelului:

- CASCADE

  Aşa cum arată şi numele, folosind această opţiune, operaţia se efectuează în cascadă şi în tabelele copil associate, iar valoarea se actualizează sau se elimină din tabelul părinte.

- SET NULL

  Folosind această opţiune, toate valorile din tabelul referit sunt trecute pe valoarea nulă

- NO ACTION

  Folosind această opţiune nu se produce nici o acţiune fiind necesară păstrarea integrităţii referenţiale atât înainte cât şi după executarea comenzii.

- RESTRICT

  Folosind această opţiune nu se permite valorilor care au referinţe către tabelul părinte să se modifice sau să fie eliminate.

Comanda de mai jos arată unde sunt specificate regulile de ştergere sau actualizare:

```
ALTER TABLE <TABEL_DEPENDENT>
```

```
   ADD CONSTRAINT <nume_constrangere>
    FOREIGN KEY <nume_coloana>
    ON DELETE <tip_de_actiune_la_stergere>
    ON UPDATE < tip_de_actiune_la_actualizare>
;
```

O acţiune de ştergere poate fi de tip CASCADE, SET NULL, NO ACTION sau RESTRICT. O acţiune de actualizare poate fi de tip NO ACTION sau RESTRICT.

5.2.3 Crearea unei scheme

La fel ca şi în cazul păstrării fişierelor în directoarele sistemului de fişiere ale unui sistem de calcul în care se păstrează fişierele ce aparţin aceluiaşi domeniu de interes, schema unei baze de date DB2 este un obiect al bazei de date ce permite păstrarea în comun a tuturor obiectelor ce aparţin bazei de date respective. În DB2, fiecare obiect are două părţi, numele schemei şi numele obiectului.

Pentru a crea o schemă se foloseşte comanda:

```
create schema mySchema
```

Pentru a crea un tabel în schema de mai sus se scrie:

```
create table mySchema.myTable (col1 integer)
```

Atunci când nu se specifică numele schemei, DB2 foloseşte o schemă implicită, care, de obicei, are numele identificatorului utilizatorului care se conectează la baza de date respectivă. Numele implicit al schemei se poate modifica folosind sintaxa SET CURRENT SCHEMA aşa cum se arată mai jos:

```
set current schema mySchema
```

5.2.4 Crearea unei vederi

O vedere este un tabel virtual ce provine din unul sau mai multe tabele sau din alte vederi. Este virtuală deoarece nu conţine date, ci doar o definiţie a unui tabel ce rezultă dintr-o operaţie de selecţie. De exemplu, pentru a crea o vedeere bazată pe tabelul EMPLOYEE trebuie scris:

```
CREATE VIEW MYVIEW AS

   SELECT LASTNAME, HIREDATE FROM EMPLOYEE
```

Odată ce o vedere este creată, se poate folosi la fel ca orice alt tabel. De exemplu, se poate folosi o comandă se selecţie pe aceasta:

```
SELECT * FROM MYVIEW
```

Vederile permit ascunderea datelor sau limitarea accesului la date, ceea ce înseamnă că acestea pot fi folosite si pe motive de securitate.

5.2.5 Crearea altor obiecte ale bazei de date

În mod asemănător comenzii CREATE TABLE folosită în SQL pentru crearea de tabele, mai există şi alte comenzi pentru fiecare dintre obiectele bazei de date, cum ar fi indecşi, funcţii, proceduri stocate, declanşatori, ş.a.m.d. Pentru informaţii suplimentare despre aceste comenzi vedeţi DB2 9.7 Information Center.

5.2.6 Modificarea obiectelor bazei de date

Un obiect odată creat poate fi modificat pentru a răspunde cerinţelor impuse. Eliminarea şi recrearea obiectului este o variantă de a face aceste modificări, dar acest lucru poate avea uneori efecte inacceptabile.

O modalitate mai bună de modificare a obiectelor bazei de date este aceea de utilizare a comenzii SQL ALTER. De exemplu, dacă se doreşte să se modifice definiţia unui tabel astfel încât să nu se permită introducerea de valori nule într-o anumită coloană a acestuia, se poate folosi conmanda:

```
alter table myTable alter column col1 set not null
```

În mod asemănător, şi alte modificări, cum ar fi adăugarea sau eliminarea unei coloane, stabilirea sau eliminarea **cheii primare**, ş.a.m.d., se pot realiza folosind sintaxa corespunzătoare `alter table`. Comanda ALTER se mai poate folosi şi pentru alte obiecte ale bazei de date.

5.2.7 Redenumirea obiectelor unei baze de date

Odată create obiectele bazei de date, acestea pot fi redenumite folosind comanda SQL, RENAME. Pentru a redenumi un obiect al unei baze de date se poate folosi sintaxa:

```
RENAME <tip obiect> <nume obiect> to <nume nou>
```

În care tipul obiectului poate fi, de exemplu, un tabel, un spaţiu rezervat tabelului, sau un index. Nu toate obiectele bazei de date pot fi redenumite după creare.

Pentru a redenumi o coloană, se va folosi comanda SQL ALTER TABLE împreună cu RENAME. De exemplu:

```
ALTER TABLE <nume tabel> RENAME COLUMN <nume coloană> TO <nume nou>
```

5.3 Manipularea datelor folosind limbajul SQL

Această secţiune arată felul în care se folosesc operaţiile de citire, actualizare sau ştergere a datelor în SQL.

5.3.1 Extragerea datelor

Extragerea datelor în SQL este o operaţie ce permite citirea (extragerea) rândurilor şi coloanelor dintr-un tabel relaţional. Acest lucru se poate face cu ajutorul comenzii SELECT.

Dacă tabelul are numele **myTable**, cea mai simplă comandă de extragere a datelor din cadrul acestui tabel este:

```
select * from myTable
```

Caracterul special **'*'**, reprezintă toate coloanele din cadrul tabelului. Folosirea **'*'** în cadrul unei interogări nu este recomandată în practică, deoarece se obţine mai multă informaţie decât este necesară. De obicei, nu sunt necesare toate coloanele unui tabel, caz în care trebuie specificată o listă de coloane. De exemplu,

```
select col1, col2 from myTable
```

extrage **col1** şi **col2** cu toate rândurile tabelului **myTable** în care **col1** şi **col2** sunt numele coloanelor din care se extrag datele.

5.3.1.1 Ordonarea rezultatelor

Comanda SELECT întoarce rezultatele fără a respecta nici o ordine. Dacă se foloseşte aceeaşi comandă SELECT de mai multe ori se va întoarce acelaşi grup de rânduri, dar într-o ordine diferită. Pentru a fi siguri că rezultatele vor fi afişate în aceeaşi ordine în permanenţă, fie crescător, fie descrescător, se va folosi clauza ORDER BY.

De exemplu, comanda de mai jos returnează rezultatul ordonând datele din coloana col1 crescător:

```
SELECT col1 FROM myTable ORDER BY col1 ASC
```

ASC provine de la ordonarea crescătoare, care este implicită. Ordonarea descrescătoare a datelor se face folosind **DESC** aşa cum se vede mai jos:

```
SELECT col1 FROM myTable ORDER BY col1 DESC
```

5.3.1.2 Cursoare

Un cursor este un mecanism al bazei de date prin care se păstrează rezultatele obţinute din execuţia unei clauze SELECT. Sintaxa prin care se declară, se deschide, se parcurge şi se închide un cursor se prezintă mai jos:

```
DECLARE <nume cursor> CURSOR [WITH RETURN <tinta retur>]
      <comanda SELECT >;
OPEN  <nume cursor>;
FETCH <nume cursor> INTO <variabile>;
CLOSE <nume cursor>;
```

În loc să returneze unei aplicaţii toate datele unei comenzi SQL dintr-o dată, cursorul permite aplicaţiei să proceseze rândurile unul câte unul. Folosind comenzile FETCH împreună cu o buclă din cadrul aplicaţiei, progrmatorii pot ajunge de la un rând la altul cu ajutorul cursorului aplicând anumite reguli unuia dintre acestea sau în funcţie de conţinutul acestuia. De exemplu, următorul fragment de cod adună toate salariile angajaţilor cu ajutorul unui cursor.

```
...
  DECLARE p_sum INTEGER;
  DECLARE p_sal INTEGER;
  DECLARE c CURSOR FOR
            SELECT SALARY FROM EMPLOYEE;
  DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
  SET p_sum = 0;
  OPEN c;
  FETCH FROM c INTO p_sal;
  WHILE(SQLSTATE = '00000') DO
     SET p_sum = p_sum + p_sal;
     FETCH FROM c INTO p_sal;
  END WHILE;
  CLOSE c;
...
```

Cursoarele sunt folosite pe scară largă pentru a parcurge mai multe rânduri din cadrul unui tabel şi pentru a le prelucra cu ajutorul aplicaţiilor.

5.3.2 Introducerea datelor

Pentru a introduce date în cadrul unui tabel se foloseşte comanda INSERT. Există mai multe modalităţi de a introduce date. De exemplu, se poate introduce câte un rând pentru fiecare comandă INSERT, mai multe rânduri pentru fiecare comandă INSERT sau toate rândurile folosind o interogare, aşa cum se arată în exemplele următoare.

În acest prim exemplu, comanda de inserare introduce un singur rând în tabelul **myTable**.

```
insert into myTable values (1);
insert into myTable values (1, 'myName', '2010-01-01');
```

În cel de-al doilea exemplu, comanda introduce mai multe (trei) rânduri în tabelul **myTable**.

```
insert into myTable values (1),(2),(3);
insert into myTable values (1, 'myName1','2010-01-01'),
                           (2, 'myName2','2010-02-01'),
                           (3, 'myName3','2010-03-01');
```

În sfârşit, în cel de-al treilea exemplu, comanda de inserare introduce toate rândurile unei subinterogări, "**select * from myTable2**" în tabelul **myTable**.

```
insert into myTable (select * from myTable2)
```

5.3.3 Ştergerea datelor

Pentru a şterge rânduri dintr-un tabel se foloseşte comanda DELETE. Prin specificarea unei condiţii în clauza WHERE se pot elimina dintr-un tabel unul sau mai multe rânduri folosind o singură comandă. De exemplu, comanda următoare şterge din tabelul **myTable** toate rândurile în care **col1 > 1000.**

```
DELETE FROM myTable WHERE col1 > 1000
```

Trebuie avută mare atenţie la folosirea comenzii de ştergere. Dacă în comanda de ştergere nu apare şi clauza **WHERE** comanda DELETE va şterge toate rândurile din tabel.

5.3.4 Actualizarea datelor

Pentru a actualiza datele existente în cadrul unui tabel se foloseşte comanda UPDATE. Folosind o singură comandă de actualizare împreună cu condiţia WHERE se pot actualiza mai multe rânduri din cadrul unui tabel. Pentru fiecare rând desemnat a fi actualizat, comanda poate modifica una sau mai multe coloane.

De exemplu:

```
UPDATE myTable SET col1 = -1 WHERE col2 < 0
UPDATE myTable SET col1 = -1,
                   col2 = 'a',
                   col3 = '2010-01-01'
               WHERE col4 = '0'
```

Trebuie avută mare atenţie la folosirea comenzii de actualizare. Dacă în comanda de actualizare nu apare şi clauza **WHERE** aceasta va actualiza toate rândurile din tabel.

5.4 Joncţiuni ale tabelelor

O comandă SQL simplă de selecţie este acea comandă care extrage valorile din una sau mai multe coloane ale unui singur tabel. Următoul nivel de complexitate apare atunci când trebuie extrase date din două sau mai multe tabele, ceea ce conduce la multiple posibilităţi referitoare la felul în care pot fi obţinute rezultatele.

În cadrul unei comenzi SQL există două posibilităţi de cuplare a tabelelor:

1. Inner join (joncţiunea internă)
2. Outer join (joncţiunea externă)

Acestea sunt detaliate în secţiunile următoare.

5.4.1 Joncţiunea internă

O ***joncţiune internă*** este cea mai obişnuită formă de joncţiune folosită într-o comandă SQL. Aceasta poate fi de tipurile:

- Equi-join (echi-joncţiunea)
- Natural join (joncţiunea naturală)
- Cross join (joncţiunea încrucişată)

5.4.1.1 Equi-join

Acest tip de joncţiune apare atunci când două tabele sunt cuplate pe baza egalităţii coloanelor specificate; de exemplu:

```
SELECT *
  FROM student, enrollment
  WHERE student.enrollment_no=enrollment.enrollment_no

          SAU
SELECT *
  FROM student
       INNER JOIN enrollment
       ON student.enrollment_no=enrollment.enrollment_no
```

5.4.1.2 Joncţiunea naturală

O ***joncţiune naturală*** este o versiune îmbunătăţită de echi-joncţiune în care coloanele care se cuplează nu au nevoie de nici o specificaţie. Sistemul selectează automat coloana cu acelaşi nume din tabele şi aplică operatorul de egalitate asupra acesteia. Joncţiunea naturală elimină toate atributele duplicat, ca în exemplul:

```
SELECT *
  FROM STUDENT
  NATURAL JOIN ENROLLMENT
```

Joncţiunile naturale provoacă mai multe ambiguităţi decât uşurinţa de folosire. De exemplu, pot apare probleme atunci când tabelele care se cuplează au mai multe coloane cu acelaşi nume, sau atunci când tabelele nu au acelaşi nume pentru coloanele care se cuplează. Cele mai multe dintre sistemele de baze de date nu suportă joncţiunea naturală.

5.4.1.3 Joncţiunea încrucişată

O ***joncţiune încrucişată*** este un produs cartezian simplu al tabelelor care se cuplează. De exemplu:

```
SELECT *
FROM STUDENT, ENROLLMENT
```

5.4.2 Joncţiuni externe

O ***joncţiune externă*** este o formă specializată de joncţiune folosită în comenzile SQL. Într-o joncţiune externă, primul tabel specificat în comanda SQL din clauza FROM se apreciază ca fiind tabelul din stânga, iar celălalt tabelul din dreapta. O joncţiune externă poate fi de trei tipuri:

- Left outer join (joncţiune externă stânga)
- Right outer join (joncţiune externă dreapta)
- Full outer join (joncţiune externă completă)

Figura 5.1 prezintă cele trei tipuri de joncţiuni externe.

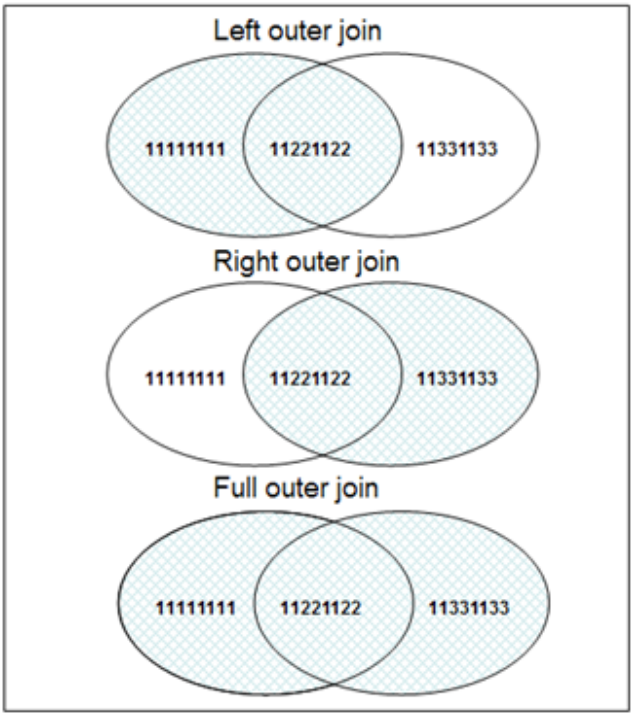


Figura 5.1 – Tipurile de joncţiuni externe

În secţiunile următoare, se va descrie mai în detaliu fiecare dintre aceste joncţiuni. Pentru o mai bună înţelegere a fiecărei situaţii, exemplele sunt oferite folosind tabelele din *Figura 5.2*.

Tabelul STUDENT

Name	Year	Enrollment no
John Smith	1	11221122
Raul Chong	2	11331133

Tabelul INREGISTRARE

Name	Subject ID	Enrollment no
Physics	100	11221122
Mathematics	200	11111111

Figura 5.2 – Tabelele folosite la exemplificarea joncţiunilor externe

5.4.2.1 Joncţiunea externă stânga

În acest tip de joncţiune, rezultatul va fi o reuniune a rezultatelor unei echi-joncţiuni,

incluzând şi rândurile ce nu au corespondent ale tabelului din STÂNGA. De exemplu, următoarea comandă va returna rândurile prezentate *Figura 5.3*.

```
SELECT *
  FROM STUDENT
  LEFT OUTER JOIN ENROLLMENT
  ON STUDENT.ENROLLMENT_NO = ENROLLMENT_NO
```

Joncțiunea externă stânga

Name	Year	Enrollment no	Name	Subject ID
John Smith	1	11221122	Physics	100
Raul Chong	2	11331133	-	-

Figura 5.3 – Rezultatul unei joncţiuni externe stânga

5.4.2. Joncţiunea externă dreapta

În cazul acestui tip de joncţiune, rezultatul reprezintă reuniunea unei echi-joncţiuni, care include şi rândurile care nu au corespondent din tabelul din drepata. De exemplu, următoarea comandă întoarce rândurile prezentate în *Figura 5.4.*

```
SELECT *
  FROM STUDENT
  RIGHT OUTER JOIN ENROLLMENT
  ON STUDENT.ENROLLMENT_NO = ENROLLMENT_NO
```

Joncțiunea externă dreapta

Name	Year	Enrollment no	Name	Subject ID
John Smith	1	11221122	Physics	100
-	-	11111111	Mathematics	200

Figura 5.4 – Rezultatul unei joncţiuni externe dreapta

5.4.2.3 Joncţiune externă completă

În acest caz, rezultatul este o reuniune a rezultatelor unei echi-joncţiuni, care include şi rândurile ce nu au corespondent nici în tabelul din STÂNGA nici în tabelul din DREAPTA. De exemplu, comanda următoare va returna rândurile prezentate în *Figura 5.5.*

```
SELECT *
  FROM STUDENT
  FULL OUTER JOIN ENROLLMENT
  ON STUDENT.ENROLLMENT_NO = ENROLLMENT_NO
```

Joncțiunea extenă completă

Name	Year	Enrollment no	Name	Subject ID
John Smith	1	11221122	Physics	100
Raul Chong	2	11331133	-	-
-	-	11111111	Mathematics	200

Figura 5.5 – Rezultatul unei joncţiuni externe complete

Fiecare tip de joncţiune returnează seturi de date diferite, motiv pentru care acestea trebuie folosite în mod explicit pentru fiecare dintre cerinţele aplicabile domeniului. De exemplu, dacă se doreşte o listă a studenţilor înregistraţi la oricare dintre cursuri, dar şi a studenţilor care nu au fost înregistraţi nicăieri, probabil că va fi nevoie de o joncţiune externă stânga.

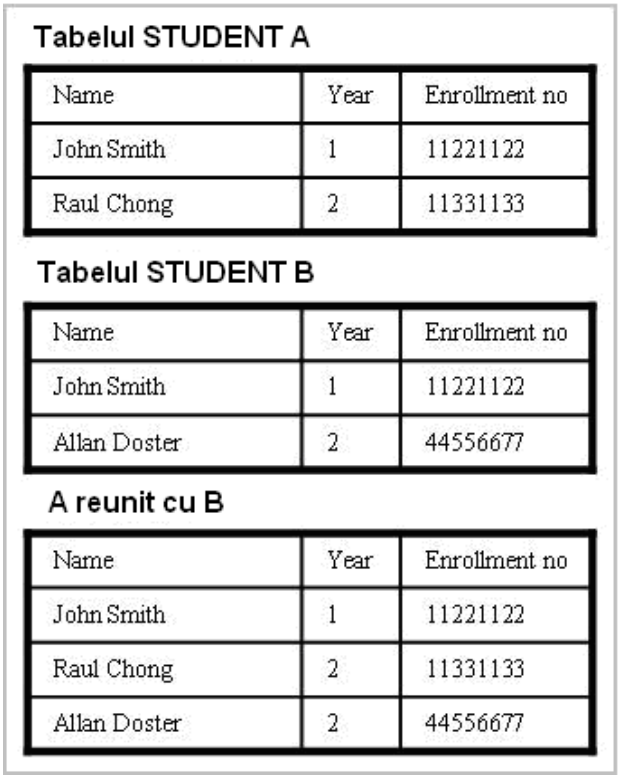
5.5 Operaţiile de reuniune, intersecţie şi diferenţă

Operatorii de reuniune, intersecţie şi diferenţă folosiţi în teoria mulţimilor sunt folosiţi şi îm comenzile SQL atât timp cât cerinţele necesare sunt respectate.

5.5.1 Reuniunea

Operatorul **`Union`** poate fi folosit pentru a reuni două seturi de date care au aceleaşi definiţii ale coloanelor şi care apar în aceeaşi ordine. Operatorul de reuniune elimină duplicatele din rezultat. De exemplu, următoarea comandă întoarce rândurile prezentate în *Figura 5.6.*

```
SELECT * FROM student_table_a
UNION
SELECT * FROM student_table_b
```



Tabelul STUDENT A

Name	Year	Enrollment no
John Smith	1	11221122
Raul Chong	2	11331133

Tabelul STUDENT B

Name	Year	Enrollment no
John Smith	1	11221122
Allan Doster	2	44556677

A reunit cu B

Name	Year	Enrollment no
John Smith	1	11221122
Raul Chong	2	11331133
Allan Doster	2	44556677

Figure 5.6 – Exemplu cu operatorul de reuniune

În *Figura 5.6*, se observă faptul că operatorul de reuniune elimină rândurile duplicat. Pot exista situaţii în care eliminarea rândurilor duplicat nu este necesară. În astfel de cazuri se foloseşte operatorul **`UNION ALL`** ca în exemplul următor:

```
SELECT * from student_table_a
UNION ALL
SELECT * from student_table_b
```

Figura 5.7 prezintă un exemplu de rezultat în care se foloseşte operatorul UNION ALL.

A reunit cu tot ce este în B

Name	Year	Enrollment no
John Smith	1	11221122
Raul Chong	2	11331133
John Smith	1	11221122
Allan Doster	2	44556677

Figura 5.7 – Exemplu de folosire a operatorului UNION ALL

5.5.2 Intersecţia

Operatorul de intersecţie **INTERSECT** returnează un rezultat în care se prezintă datele care sunt comune ambelor tabele, folosindu-se comanda:

```
select * from student_table_a
INTERSECT
select * from student_table_b
```

Figura 5.8 prezintă un exemplu de rezultat al comenzii anterioare.

A intersectat cu B

Name	Year	Enrollment no
John Smith	1	11221122

Figura 5.8 – Exemplu de rezultat obţinut după folosirea operatorului INTERSECT

Operatorul de intersecţie va returna toate datele comune care există în ambele tabele A şi B, iar acestea vor fi afişate o singură dată, chiar dacă în cele două tabele există mai multe rânduri duplicat. Pentru a returna toate datele duplicat în rezultatul final, se va folosi operatorul INTERSECT ALL, ca în exemplul:

```
select * from student_table_a
INTERSECT ALL
select * from student_table_b
```

5.5.3 Diferenţa (EXCEPT)

Operatorul de diferenţă (EXCEPT) întoarce ca rezultat doar datele care există în tabelul din STÂNGA.

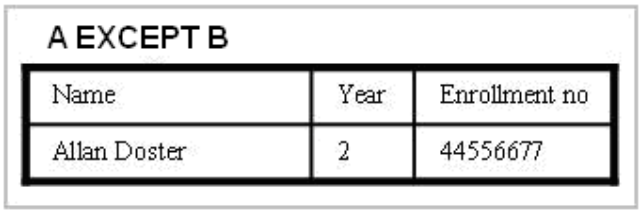
Din punct de vedere logic,

A EXCEPT B = A MINUS [A INTERSECT B]

De exemplu:

```
select * from student_table_a
EXCEPT
select * from student_table_b
```

va întoarce rezultatul din *Figura 5.9.*



A EXCEPT B

Name	Year	Enrollment no
Allan Doster	2	44556677

Figura 5.9 – Exemplu de rezultat obţinut după aplicarea operatorului EXCEPT

Operatorul **EXCEPT** va returna datele care există în tabelul A, dar nu şi în tabelul B, iar datele comune sunt afişate o singură dată, chiar dacă există mai multe rânduri duplicat în tabelul A. Pentru a returna toate datele, inclusiv duplicatele, se va folosi operatorul EXCEPT ALL. De exemplu:

```
select * from student_table_a
EXCEPT ALL
select * from student_table_b
```

5.6 Operatori relaţionali

Operatorii relaţionali reprezintă teste şi operaţii de bază care se pot aplica datelor. Aceşti operatori conţin:

- Operaţii matematice de bază, cum ar fi '+', '-', '*' şi '/'
- Operatori logici, cum ar fi 'AND', 'OR' şi 'NOT'
- Operatori de manipulare a şirurilor de caractere, cum ar fi 'CONCATENATE', 'LENGTH', 'SUBSTRING'
- Operatori de comparare, cum ar fi '=', '<', '>', '>=', '<=' and '!='
- Operatori de grupare şi agregare
- Alte operaţii, cum ar fi DISTINCT

Vom elimina din discuţie operatorii matematici de bază, operatorii logici şi cei de comparaţie, şi ne vom opri asupra celorlalţi operatori.

5.6.1 Operatori de grupare

Operatorii de grupare efectuează operaţii cu două sau mai multe seturi de date obţinându-se un rezultat totalizator. De exemplu, să spunem că avem un tabel care conţine lista tuturor studenţilor, precum şi cursurile la care aceştia sunt înregistraţi. Fiecare student se poate înregistra la mai multe cursuri, fiecare curs având mai mulţi studenţi înregistraţi. Pentru a obţine numărul studenţilor înregistraţi la cursuri se foloseşte comanda SQL:

```
select count(*) from students_enrollment
```

Dacă se doreşte să se obţină numărul studenţilor înscrişi la fiecare curs, trebuie ordonate datele tabelului pe baza cursurilor existente, după care se numără studenţii înscrişi la cursul respectiv. Acest lucru se face folosind clauza GROUP BY astfel:

```
select course_enrolled, count(*)
            from students_enrollment
            group by course_enrolled

---------------Resultset----------------
COURSE_ENROLLED            STUDENT_COUNT
------------------------- -------------
English                               10
Mathematics                           30
Physics                               60
```

Gruparea se poate face şi pe mai multe coloane, caz în care ordonarea se face începând cu coloana cea mai din stânga.

5.6.2 Operatori de agregare

Operatorii care acţionează pe două sau mai multe tupluri sau rânduri, întorcând un rezultat scalar se numesc operatori de agregare. Aceştia ar putea fi: COUNT, SUM, AVERAGE, MINIMUM, MAXIMUM ş.a.m.d. Aceşti operatori se folosesc împreună cu clauza GROUP BY aşa cum se prezintă în comenzile SQL de mai sus.

5.6.3 Clauza HAVING

HAVING este un operator special ce poate fi folosit numai împreună cu clauza GROUP BY pentru a selecta anumite rânduri din rezultatul obţinut prin grupare. Priviţi exemplul din secţiunea operatorilor de grupare; se vor prezenta doar acele cursuri care au mai puţin de 5 studenţi înregistraţi; se va folosi clauza HAVING aşa cum se vede mai jos:

```
SELECT course_enrolled, count(*)
            FROM students_enrollment
            GROUP BY course_enrolled
            HAVING count(*) < 5
```

Pentru a reduce în continuare numărul de rânduri din rezultatul scalar se va folosi clauza WHERE; dar această clauză nu poate fi folosită la gruparea rândurilor.

5.7 Subinterogări

Atunci când o interogare se foloseşte în cadrul altei interogări, interogarea exterioară se numeşte interogarea principală sau interogarea părinte, iar interogarea interioară se numeşte subinterogare. Această subinterogare poate întoarce o singură valoare scalară, unul sau mai multe tupluri, sau valori nule. Subinterogările se execută primele, după care se execută interogările părinte folosind datele furnizate de către subinterogare.

5.7.1 Subinterogări ce returnează o valoare scalară

Valorile scalare reprezintă o singură valoare a oricărui atribut sau entitate, cum ar fi de exemplu NUME, VARSTA, CURS, AN etc. Interogarea următoare foloseşte o subinterogare care întoarce o valoare scalară, care apoi se foloseşte în interogarea părinte pentru a obţine rezultatul dorit:

```
SELECT name FROM students_enrollment
      WHERE age = ( SELECT min(age) FROM students )
```

Interogarea de mai sus întoarce o listă a celor mai tineri studenţi. Subinterogarea “SELECT min(age) FROM students” întoarce o valoare scalară care arată studentul cu vârsta cea mai mică. Interogarea părinte întoarce o listă a tuturor studenţilor a căror vârstă este egală cu valoarea returnată de subinterogare.

5.7.2 Subinterogări care returnează mai multe valori

Atunci când subinterogarea returnează un grup de rezultate ce conţin mai multe valori din cadrul unei coloane (cum ar fi lista numelor) sau o mulţime de valori ce provin din mai multe coloane (cum ar fi numele, vârsta şi data naşterii studenţilor) atunci se spune că subinterogarea returnează valori multiple. De exemplu, pentru a obţine o listă a studenţilor care sunt înscrişi la un anumit curs oferit de către departamentul de calculatoare, se va folosi subinterogarea:

```
SELECT name FROM students
      WHERE course_enrolled IN
      (
            SELECT distinct course_name
            FROM courses
            WHERE department_name = 'Computer Science'
      )
```

În acest exemplu, subinterogarea întoarce o listă a tuturor cursurilor oferite de departamentul de calculatoare, iar interogarea exterioară afişează toţi studenţii înscrişi la cursurile obţinute cu ajutorul subinterogării.

Se observă faptul că pot exista mai multe variante de a obţine acest rezultat. Exemplele prezentate în cadrul acestui capitol arată metodele de lucru şi nu comanda SQL optimă.

5.7.3 Subinterogări corelate

Atunci când o subinterogare se execută pentru fiecare rând al tabelului părinte, în loc de a fie executată dintr-o singură trecere, (aşa cum se întâmplă în exemplul de mai sus), aceasta este denumită subinterogare corelată. De exemplu:

```
SELECT dept, name, marks
FROM final_result a WHERE marks =
        (
          SELECT max(marks) FROM final_result WHERE dept = a.dept
        )
```

Comanda de mai sus caută o listă a studenţilor pe departamente care au primit cele mai mari note în departamentele respective. Pentru fiecare rând din tabelul din STÂNGA, subinterogarea găseşte max(marks) pentru departamentul rândului curent şi dacă valoarea notei din rândul curent este egală cu rezultatul obţinut din subinterogare, atunci se adaugă rezultatului ce va fi furnizat de către interogarea exterioară.

5.7.4 Subinterogări în clauzele FROM

O subinterogare poate fi folosită şi într-o caluză FROM aşa cum se vede din exemplul următor:

```
SELECT dept, max_marks, min_marks, avg_marks
FROM
     (
      SELECT dept,
             max(marks) as max_marks,
             min(marks) as min_marks,
             avg(marks) as avg_marks
      FROM final_result GROUP BY dept
     )
WHERE (max_marks - min_marks) > 50 and avg_marks < 50
```

Interogarea de mai sus foloseşte subinterogări într-o caluză FROM. Subinterogarea returnează nota minimă, maximă şi media notelor pe fiecare deparatament. Interogarea exterioară foloseşte datele pe care le filtrează în continuare prin adăugarea unor condiţii de filtrare în clauza WHERE a interogării exterioare.

5.8 Corespondenţa dintre conceptele folosite la programarea orientată pe obiecte şi cele folosite la bazele de date relaţionale

Mulţi programatori folosesc astăzi limbaje de programare orientată pe obiecte, dorind în acelaşi timp să acceseze baze de date relaţionale. Tabelul următor arată corespondenţa dintre unele concepte folosite în programarea orientată pe obiecte şi conceptele folosite în bazele de date relaţionale. Lista din tabelul de mai jos nu este exhaustivă.

Conceptele din programarea orientată pe obiecte – elemente de clasă	Concepte ale bazelor de date relaţionale
Nume	Nume tabel
Atribut	Nume coloană
Metodă	Procedură stocată
Constructor/Destructor	Declanşator
Identificator obiect	Cheie primară

Table 5.1 Corespondenţa dintre conceptele folosite la programarea orientată pe obiecte şi cele folosite la bazele de date relaţionale

Bibliotecile folosite la realizarea corespondenţei obiectual-relaţional, cum ar fi Object-Hibernate sunt foarte populare la oferirea unui cadru pentru realizarea unei astfel de transformări. pureQuery, o nouă tehnologie introdusă de către IBM, oferă suport suplimentar şi îmbunătăţiri ale performanţelor în acest domeniu. Pentru mai multe informaţii despre această tehnologie, vedeţi cartea gratuită *Getting started with pureQuery,* care face parte din această serie de cărţi.

5.10 Studiu de caz cu sistemul de management al unei biblioteci - Partea 3 din 3

Partea finală a studiului de caz arată modul de transfornare a modelului logic în model fizic, în care se vor crea obiectele specifice bazelor de date DB2. Tabelul de mai jos prezintă corespondenţa dintre conceptele modelelor conceptual, logic şi fizic:

Conceptele modelării conceptuale	Conceptele modelării logice	Conceptele modelării fizice
Numele tipului de entitate	Variabila relaţie, R	Nume tabel
Tip de entitate	Relaţie	Tabel
Entitate	Tuplu	Rând
Atribut	Atribut, A1, A2, etc.	Coloană
Tip de relaţie	O pereche cheie primară-cheie externă	Constrângere
Identificator unic	Cheie primară	Cheie primară

Transformarea din modelul logic în modelul fizic este simplă. Din modearea logică s-au

obţinut toate relaţiile şi asocierile dintre acestea pentru a crea sistemul de management al unei biblioteci. Tot ceea ce mai rămâne de făcut este specificarea domeniilor de valori (tipurilor de date) pentru fiecare atribut din cadrul fiecărui tabel, împreună cu constrângerile corespunzătoare. Sugerăm folosirea următoarelor prefixe la acordarea numelor constrângerilor:

- PRIMARY KEY: pk_
- UNIQUE: uq_
- DEFAULT: df_
- CHECK: ck_
- FOREIGN KEY: fk_

Să privim din nou fiecare relaţie, după adăugarea tipurilor de date şi constrângerilor:

Relaţia CITITOR

Nume atribut	Domeniu	Sub-domeniu	Opţional	Constrângere
ID_CITITOR	Text	CHAR(5)	Nu	Pk_
NUME	Text	VARCHAR(30)	Nu	
PRENUME	Text	VARCHAR(30)	Nu	
EMAIL	Text	VARCHAR(40)	Da	
TEL	Text	VARCHAR(15)	Da	
ADRESA	Text	VARCHAR(75)	Da	
ORAS	Text	CHAR(3)	Nu	
TARA	Text	DATE	Nu	

Relaţia AUTOR

Nume atribut	Domeniu	Sub-domeniu	Opţional	Constrângere
ID_AUTOR	Text	CHAR(5)	Nu	Pk_
NUME	Text	VARCHAR(30)	Nu	
PRENUME	Text	VARCHAR(30)	Nu	
EMAIL	Text	VARCHAR(40)	Da	

TEL	Text	VARCHAR(15)	Da	
ADRESA	Text	VARCHAR(75)	Da	
ORAS	Text	VARCHAR(40)	Da	
TARA	Text	VARCHAR(40)	Da	

Relaţia CARTE

Nume atribut	Domeniu	Sub-domeniu	Opţional	Constrângere
ID_CARTE	Text	CHAR(5)	Nu	Pk_
TITLU	Text	VARCHAR(40)	Nu	
EDITIE	Numeric	INTEGER	Da	
An	Numeric	INTEGER	Da	
PRET	Numeric	DECIMAL(7,2)	Da	
ISBN	Text	VARCHAR(20)	Da	
PAGINI	Numeric	INTEGER	Da	
INTERVAL	Text	VARCHAR(10)	Da	
DESCRIERE	Text	VARCHAR(100)	Da	

Relaţia IMPRUMUT

Nume atribut	Domeniu	Sub-domeniu	Opţional	Constrângere
ID_CITITOR	Text	CHAR(5)	Nu	Pk_, fk_
ID_COPIE	Text	VARCHAR(30)	Nu	Pk_, fk_
DATA_IMPRUMUT	Text	DATE	Nu	< RETURN_DATE
DATA_RETUR	Text	DATE	Nu	

Relaţia COPIE

Nume atribut	Domeniu	Sub-domeniu	Opţional	Constrângere
ID_COPIE	Text	CHAR(5)	Nu	Pk_
ID_CARTE	Text	VARCHAR(30)	Nu	Fk_
STARE	Text	VARCHAR(30)	Nu	

Relaţia AUTOR_LISTA

Nume atribut	Domeniu	Sub-domeniu	Opţional	Constrângere
ID_AUTOR	Text	CHAR(5)	Nu	Pk_, fk_
ID_CARTE	Text	VARCHAR(30)	Nu	Pk_, fk_
ROL	Text	VARCHAR(30)	Nu	

În acest moment se pot crea tabelele folosind sintaxa:

```
CREATE TABLE AUTHOR
       (
       AUTHOR_ID CHAR(5) CONSTRAINT AUTHOR_PK PRIMARY KEY(AUTHOR_ID) NOT
       NULL,
       LASTNAME VARCHAR(15) NOT NULL,
       FIRSTNAME VARCHAR(15) NOT NULL,
       EMAIL VARCHAR(40),
       CITY VARCHAR(15),
       COUNTRY CHAR(2)
       )

CREATE TABLE AUTHOR_LIST
       (
       AUTHOR_ID CHAR(5) NOT NULL CONSTRAINT AUTHOR_LIST_AUTHOR_FK FOREIGN
       KEY(AUTHOR_ID) REFERENCES AUTHOR (AUTHOR_ID),
       BOOK_ID CHAR(5) NOT NULL,
       ROLE VARCHAR(15) CONSTRAINT AUTHOR_LIST_PK PRIMARY KEY
       (AUTHOR_ID,BOOK_ID) NOT NULL
       )

CREATE TABLE BOOK
       (
       BOOK_ID CHAR(3) CONSTRAINT BOOK_PK PRIMARY KEY(BOOK_ID)
```

```
      CONSTRAINT AUTHOR_LIST_BOOK_FK FOREIGN KEY(BOOK_ID)  REFERENCES BOOK
      (BOOK_ID) NOT NULL,
      TITLE VARCHAR(40) NOT NULL,
      EDITION INTEGER,
      YEAR INTEGER,
      PRICE DECIMAL(7 , 2),
      ISBN VARCHAR(20),
      PAGES INTEGER,
      AISLE VARCHAR(10),
      DESCRIPTION VARCHAR(100)
      )

CREATE TABLE COPY
      (
      COPY_ID CHAR(5) CONSTRAINT COPY_PK PRIMARY KEY(COPY_ID) NOT NULL,
      BOOK_ID CHAR(5) CONSTRAINT COPY_BOOK_FK FOREIGN KEY(BOOK_ID)
      REFERENCES BOOK(BOOK_ID) NOT NULL,
      STATUS VARCHAR(10)
      )

CREATE TABLE LOAN
      (
      COPY_ID CHAR(5) CONSTRAINT LOAN_COPY_FK FOREIGN KEY(COPY_ID)
      REFERENCES COPY(COPY_ID) NOT NULL,
      BORROWER_ID CHAR(5) CONSTRAINT LOAN_BORROWER_FK FOREIGN KEY
      (BORROWER_ID)REFERENCES BORROWER (BORROWER_ID) NOT NULL,
      LOAN_DATE DATE NOT NULL,
      LOAN_DAYS INTEGER NOT NULL,
      RETURN_DATE DATE CONSTRAINT LOAN_PK PRIMARY KEY(COPY_ID,
      BORROWER_ID)
      )

CREATE TABLE BORROWER
      (
      BORROWER_ID CHAR(5) NOT NULL CONSTRAINT BORROWER_PK PRIMARY KEY
      (BORROWER_ID),
      LASTNAME VARCHAR(15) NOT NULL,
      FIRSTNAME VARCHAR(15) NOT NULL,
      EMAIL VARCHAR(40),
      PHONE VARCHAR(15),
      ADDRESS VARCHAR(60),
      CITY VARCHAR(15),
      COUNTRY CHAR(2)
      )
```

Folosind InfoSphere Data Architect se poate transforma în mod automat modelul logic în model fizic, generându-se în acelaşi timp şi codul DDL corespunzător.

5.9 Rezumat

În cadrul acestui capitol se face o prezentare de ansamblu a limbajului SQL împreună cu unele dintre caracteristicile sale. Pornind de la standardele ISO/ANSI SQL producătorii introduc o serie de alte caracteristici şi funcţionalităţi proprii. Aceste caracteristici influenţează proiectarea şi arhitectura internă a produsului, oferind performanţe superioare funcţiilor din standardul SQL. Una dintre aceste caracteristici este mecanismul de indexare folosit în cadrul bazelor de date. Comportamentul de bază al unui index este acelaşi în toate bazele de date, dar producătorii pot introduce caracteristici suplimentare pentru a îmbunătăţi scrierile/citirile în/din baza de date prin intermediul algoritmilor proprii. Pentru detalii şi pentru obţinerea unei liste complete a comenzilor SQL, a cuvintelor cheie şi a sintaxei, apelaţi la SQL Reference Guide [5.3].

5.10 Exerciţii

1. Creaţi un tabel cu coloane de tip întreg, dată calendaristică şi text cu valori implicite.
2. Introduceţi 10 rânduri în cadrul acestui tabel folosind comanda SQL INSERT.
3. Scrieţi o comandă SQL cu o subinterogare ce are o joncţiune de tip INNER JOIN.
4. Scrieţi o comandă SQL cu o subinterogare ce are o clauză GROUP BY.
5. Scrieţi o comandă SQL ce are o funcţie agregat şi clauzele WHERE, HAVING.
6. Scrieţi o comandă SQL ce foloseşte ORDER BY pe mai multe coloane.

5.11 Întrebări recapitulative

1. Care sunt categoriile de bază ale limbajului SQL referitor la funcţionalitate?
   A. Definirea datelor
   B. Modificarea datelor
   C. Controlul datelor
   D. Toate cele enumerate
   E. Nici una dintre cele enumerate
2. Cine a inventat limbajul SQL?
   A. Raymond F. Boyce
   B. E F Codd
   C. Donald D. Chamberlin
   D. A şi C

E. Nici unul dintre cei enumeraţi

3. Limbajul SQL a fost adoptat ca limbaj standard de către
   A. American National Standards Institute
   B. Bureau of International Standards
   C. International Standards Organizations
   D. Toate cele enumerate
   E. Nici una dintre cele enumerate
4. Care dintre următoarele reprezintă corespondenţa corectă dintre domeniul orientat pe obiecte şi domeniul relaţional?
   A. Nume – Nume tabel
   B. Atribut – Nume coloana
   C. Metodă – Procedură stocată
   D. Toate cele enumerate
   E. Nici una dintre cele enumerate
5. Care dintre următoarele funcţii sunt specializate pentru lucrul cu date calendaristice?
   A. Year
   B. Dayname
   C. Second
   D. Toate cele enumerate
   E. Nici una dintre cele enumerate
6. Care este modalitatea implicită de ordonare a datelor în SQL?
   A. Crescător
   B. Descrescător
   C. Ordine aleatoare
   D. Toate cele enumerate
   E. Nici una dintre cele enumerate
7. Nu se pot introduce mai multe rânduri cu o singură comandă INSERT
   A. Adevărat
   B. Fals
8. Care dintre următoarele sunt tipuri corecte de joncţiune internă?
   A. Echi-joncţiunea

B. Joncţiunea naturală

C. Joncţiunea încrucişată

D. Toate cele enumerate

E. Nici una dintre cele enumerate

9. Care dintre următoarele sunt tipuri corecte de joncţiuni externe?

A. Joncţiunea externă stânga

B. Joncţiunea externă dreapta

C. Joncţiunea externă completă

D. Toate cele enumerate

E. Nici una dintre cele enumerate

10. Operatorul Union păstrează duplicatele în rezultat.

A. Adevărat

B. Fals

6

Capitolul 6 – Proceduri stocate şi funcţii

Procedurile stocate şi funcţiile sunt obiecte ale bazei de date care pot încorpora comenzi SQL şi elemente de logică a aplicaţiei. Prin păstrarea unei părţi a logicii aplicaţiei în baza de date se obţine o creştere a performanţelor prin reducerea considerabilă a traficului de pe reţea între baza de date şi aplicaţie. Suplimentar, aceste obiecte oferă o locaţie centralizată de păstrare a codului, astfel încât acesta se poate refolosi.

În cadrul acestui capitol se va discuta despre:

- Utilizarea produsului IBM Data Studio pentru elaborarea de funcţii şi proceduri stocate
- Lucrul cu funcţii SQL
- Lucrul cu proceduri stocate

6.1 IBM Data Studio

IBM Data Studio se foloseşte la elaborarea de proceduri stocate şi de funcţii definite de către utilizator. IBM Data Studio este un mediu de dezvoltare şi de administrare bazat pe Eclipse fiind gratuit. El nu este încorporat în DB2, fiind oferit sub forma unei imagini separate, cu două apariţii:

- IDE: permite partajarea aceluiaşi Eclipse cu alte produse cum ar fi InfoSphere Data Architect şi produsele Rational. De asemenea se oferă suport şi pentru servicii de date Web.
- Stand-alone: Această versiune oferă aproape aceeaşi funcţionalitate ca şi versiunea IDE dar fără a oferi suport pentru serviciile de date Web, fiind mult mai mică ca dimensiune.

Obs:

Pentru o mai bună cunoaştere a produsului IBM Data Studio, vedeţi cartea gratuită, în format electronic, *Getting started with IBM Data Studio for DB2* care este parte a seriei de cărţi gratuite elaborate în cadrul programului DB2 on Campus.

În cadrul acestui capitol vom folosi versiunea stand-alone ce poate fi descărcată de la adresa *ibm.com/db2/express*. Figura 6.1 prezintă produsul IBM Data Studio 2.2.

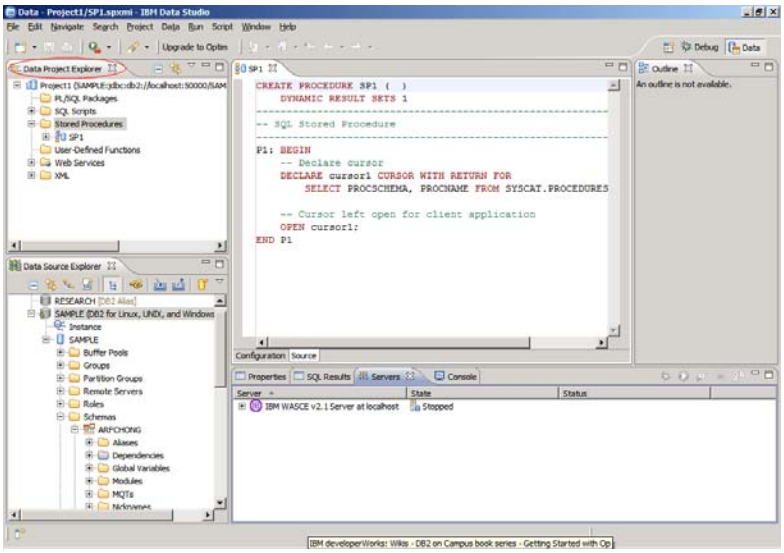


Figura 6.1 – IBM Data Studio 2.2

În acest capitol ne vom concentra atenţia pe vederea *Data Project Explorer* marcată în colţul din stânga sus al figurii. Această vedere este destinată dezvoltării pe parte de server.

6.1.1 Crearea unui proiect

Înainte de a elabora proceduri stocate, sau funcţii definite de către utilizator, folosind mediul Data Studio, trebuie creat un proiect. Din meniul Data Studio se alege *File -> New -> Project* după care se alege *Data Development Project*, aşa cum se arată în *Figura 6.2*.

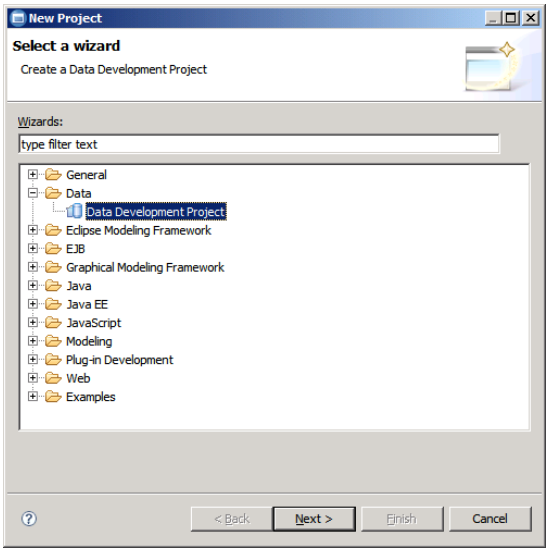


Figura 6.2 – Crearea unui proiect de date

Se urmează paşii din programul ajutător pentru a introduce numele proiectului şi pentru a specifica baza de date care se asociază acestuia. Dacă nu se poate realiza conectarea la baza de date, se apasă butonul *New* din secţiunea *Select Connection* ceea ce va conduce la apariţia unei ferestre, aşa cum se vede în *Figura 6.3*.

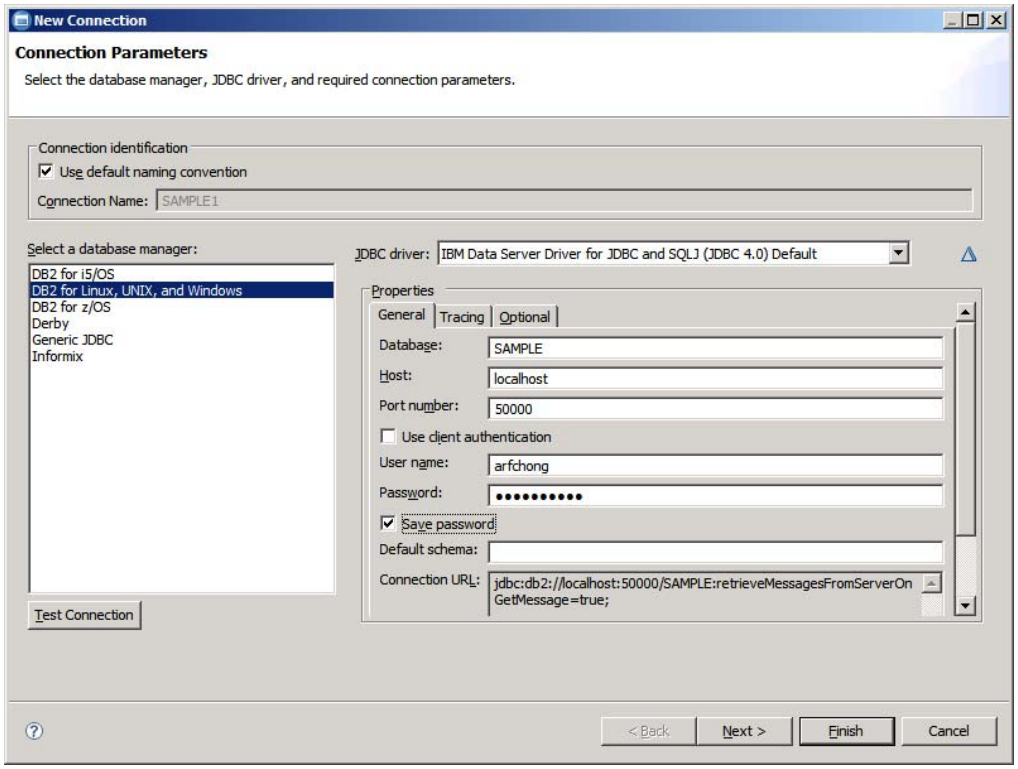


Figura 6.3 – Parametrii unei conexiuni noi

În *Figura 6.3*, trebuie selectată opţiunea *DB2 for Linux, UNIX and Windows* în câmpul *Select a database manager field* din partea stângă a figurii. Din meniul derulant *JDBC driver*, implicit după alegerea *DB2 for Linux, UNIX and Windows* este JDBC type 4 driver afişat sub denumirea *IBM Data Server Driver for JDBC and SQLJ (JDBC 4.0) Default*. Se va alege acest driver implicit. Pentru câmpul *host*, se poate introduce o adresă IP sau o adresă locală. În exemplul prezentat IBM Data Studio şi baza de date DB2 se află pe acelaşi calculator, astfel încât se introduce ***localhost***. Se testează conexiunea cu baza de date cu ajutorul butonului *Test Connection* care se vede în colţul din stânga al figurii. Dacă testul este reuşit se apasă *Finish*, iar numele bazei de date se adaugă listei de conexiuni ce pot fi asociate proiectului. Se alege baza de date, după care se apasă butonul *Finish* iar proiectul apare în vederea *Data Project Explorer*. În cadrul acestei vederi dacă se apasă pe semnul "+" se poate expanda proiectul astfel încât se pot vedea diverse foldere, cum ar fi pachete PL/SQL, scripturi SQL, proceduri stocate etc.

6.2 Lucrul cu proceduri stocate

O procedură stocată este un obiect al bazei de date care încorporează comenzi SQL şi elemente de logică a aplicaţiei. Aceasta ajută la îmbunătăţirea performanţelor prin reducerea traficului de date de pe reţea.

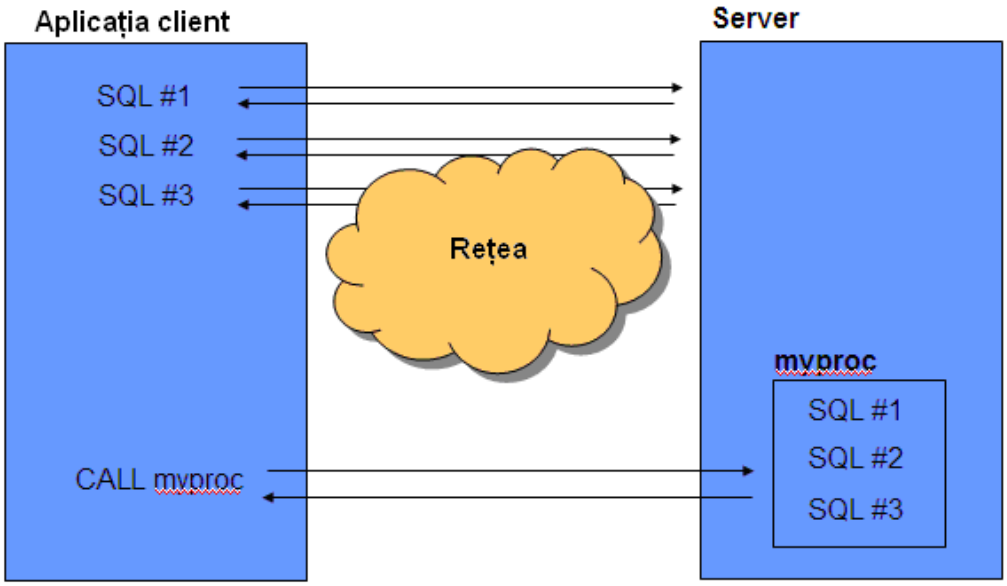


Figura 6.2 – Reducerea traficului pe reţea prin folosirea procedurilor stocate

În colţul din stânga sus al figurii, se pot vedea câteva comenzi SQL executate una după cealaltă. Fiecare comandă SQL se trimite de la client la serverul de date, iar serverul de date întoarce rezultatul înapoi la client. Prin executarea mai multor astfel de comenzi SQL, creşte traficul pe reţea. În partea de jos a figurii se vede o metodă alternativă care propune un trafic mai redus. Această a doua metodă apelează o procedură stocată, numită ***myproc*** păstrată pe server, care conţine acelaşi cod SQL; după care, pe partea de client (în stânga), se foloseşte comanda CALL pentru a apela procedura. Cea de-a doua metodă este mult mai eficientă, deoarece există un singur apel al comenzii care se trimite pe reţea, fiind returnat un singur rezultat înapoi la client.

Procedurile stocate pot fi utile în cadrul bazei de date şi din motive de securitate. De exemplu, se poate permite utilizatorilor să acceseze tabele sau vederi numai prin intermediul procedurilor stocate, ceea ce asigură serverul şi ţine la distanţă utilizatorii care altfel ar putea accesa informaţii cheie pe care nu ar trebui să le obţină. Acest lucru este benefic deoarece utilizatorii nu au nevoie de anumite privilegii explicite pe tabele sau vederi, ci doar de acele privilegii care să le permită accesarea procedurilor stocate.

6.2.1 Tipuri de proceduri

În principal, există două tipuri de proceduri stocate: procedurile SQL şi procedurile externe. Procedurile SQL sunt scrise în limbaj SQL, iar procedurile externe sunt scrise într-un limbaj

gazdă, dar mai există şi alte diferenţe importante între acestea atât în comportament cât şi în pregătire.

Procedurile SQL şi procedurile externe sunt alcătuite dintr-o parte de definire şi codul propriu-zis al acestora. Atât procedurile SQL cât şi procedurile externe au nevoie de următoarele informaţii:

- Numele procedurii.
- Parametrii de intrare şi de ieşire.
- Limbajul în care se scrie procedura. Pentru procedura SQL acesta este SQL.
- Informaţiile ce trebuie folosite la apelul procedurii, cum ar fi opţiunile adoptate la execuţie, timpul de execuţie al acesteia precum şi dacă procedura întoarce sau nu un rezultat.

Exemplul următor prezintă definiţia unei proceduri SQL.

```
CREATE PROCEDURE UPDATESALARY          (1)
 (IN EMPNUMBR CHAR(10),                (2)
  IN RATE DECIMAL(6,2))
 LANGUAGE SQL                          (3)
  UPDATE EMP                           (4)
   SET SALARY = SALARY * RATE
   WHERE EMPNO = EMPNUMBR
```

În acest exemplu:

1. Numele procedurii este UPDATESALARY.
2. Există doi parametri, EMPNUMBR ce are tipul de date CHAR(10), şi RATE ce are tipul de date DECIMAL(6,2). Ambii sunt parametri de intrare.
3. LANGUAGE SQL arată faptul că este vorba de o procedură SQL, astfel încât corpul procedurii furnizează ceilalţi parametri.
4. Corpul procedurii conţine o comandă SQL UPDATE care actualizează valorile din tabelul cu angajaţi.

Exemplul următor prezintă o definiţie a unei proceduri externe echivalente scrisă în limbajul COBOL. Programul procedurii stocate care actualizează salariile anagajaţilor se numeşte UPDSAL.

```
CREATE PROCEDURE UPDATESALARY          (1)
 (IN EMPNUMBR CHAR(10),                (2)
 IN RATE DECIMAL(6,2))
 LANGUAGE COBOL                        (3)
 EXTERNAL NAME UPDSAL;                 (4)
```

În acest exemplu:

1. Numele procedurii stocate este UPDATESALARY.

2. Procedura are doi parametri, EMPNUMBR ce are tipul de date CHAR(10) şi RATE care are tipul de date DECIMAL(6,2). Ambii sunt parametri de intrare.
3. LANGUAGE COBOL arată că avem o procedură externă, astfel încât codul procedurii stocate se află într-un program COBOL separat.
4. Numele modulului care se încarcă şi care conţine programul executabil al procedurii stocate este UPDSAL.

6.2.2 Crearea unei proceduri stocate

Pentru a crea o procedură stocată Java, PL/SQL sau SQL PL în Data Studio, se urmează paşii de mai jos. Se observă faptul că procedurile stocate scrise în alte limbaje de programare nu pot fi create cu Data Studio. În continuare se va prezenta, ca exemplu, o procedură SQL (reprezentată prin SQL PL ca limbaj al procedurii stocate), dar paşi asemănători se aplică şi în situaţia folosirii limbajelor Java sau PL/SQL.

Pas 1: Scrierea sau generarea codului procedurii stocate

Atunci când se doreşte crearea unei proceduri stocate, se apasă butonul din dreapta al mouse-ului atunci când acesta se află deasupra folderului Stored Procedures şi se alege *New -> Stored Procedure*. Se introduce informaţia cerută în programul ajutător *New Stored Procedure* adică, proiectul căruia i se asociază procedura, numele şi limbajul procedurii, precum şi comenzile SQL folosite în cadrul procedurii. Implicit, Data Studio oferă un exemplu de comandă SQL. Se păstrează valorile implicite şi se apasă butonul *Finish* fiind creată procedura cu ajutorul unui şablon de cod, aşa cum se vede din *Figura 6.3*.

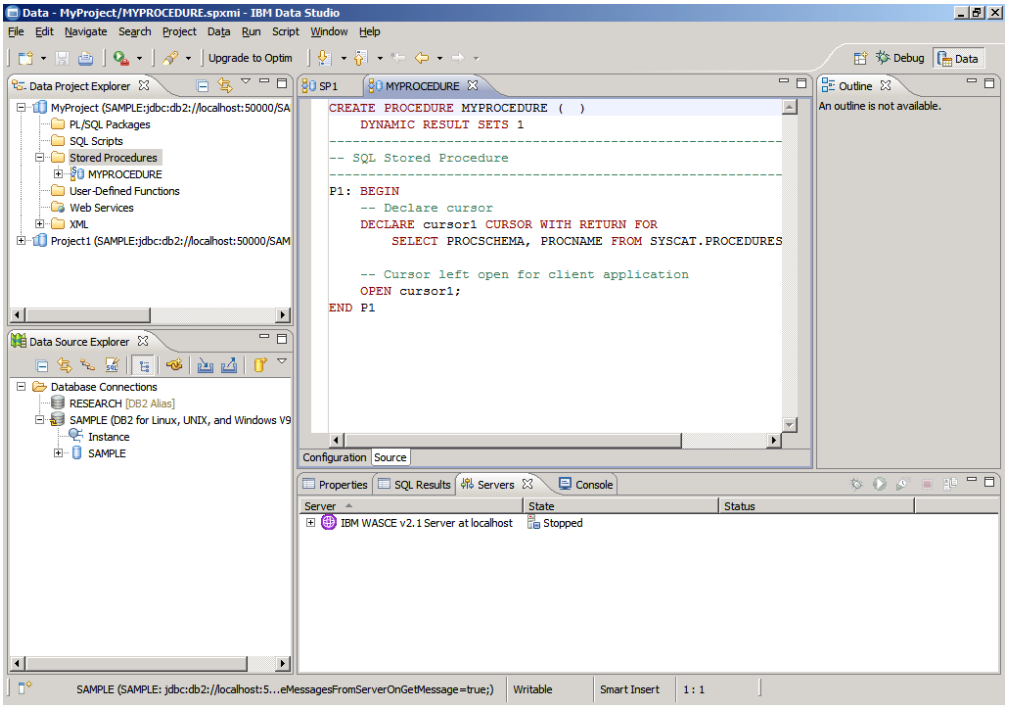


Figura 6.3 – Exemplu de procedură stocată

În *Figura 6.3*, se prezintă codul procedurii ***MYPROCEDURE*** aşa cum a fost generat. Tot acest cod se poate înlocui cu un alt cod propriu. Pentru simplitate, se va folosi în cadrul acestui capitol exemplul de procedură de mai sus, ca şi cum l-am fi scris noi.

Pas 2: Punerea în mediul de funcţionare a procedurii stocate

După ce a fost creată, procedura trebuie depusă în mediul real de funcţionare, motiv pentru care se merge în vederea *Data Project Explorer*, se apasă butonul din dreapta al mouse-ului şi se alege *Deploy*. Punerea în mediul real de funcţionare este esenţială pentru comanda **`CREATE PROCEDURE`**, deoarece compilează procedura şi o stochează în baza de date, aşa cum se vede din *Figura 6.4*.

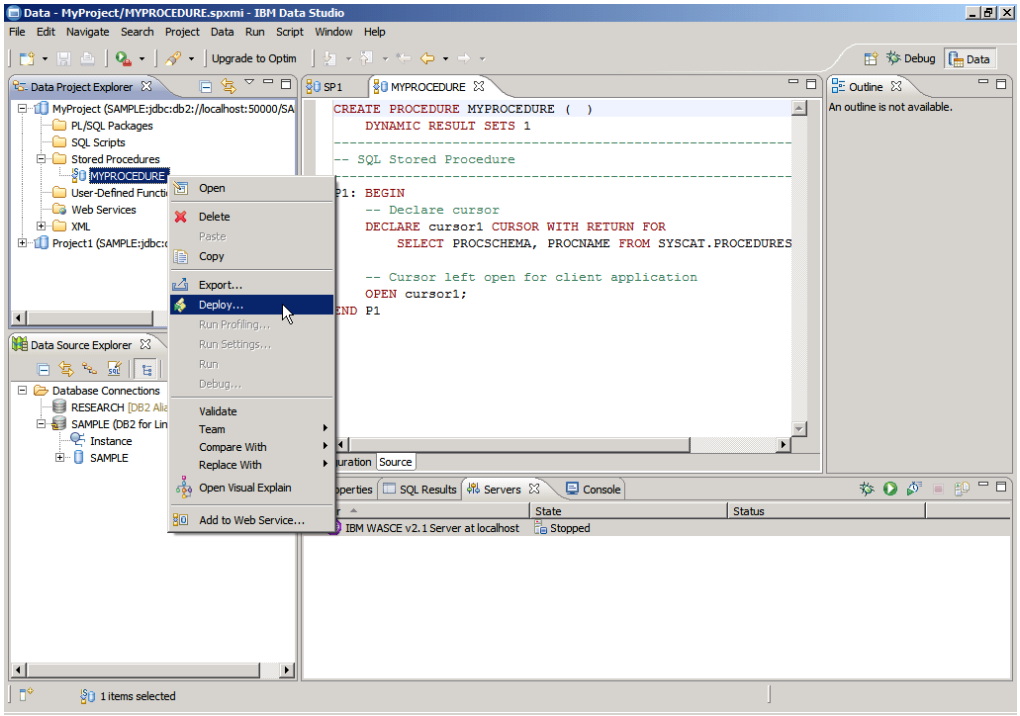


Figura 6.4 – Punerea în mediul de funcţionare a procedurii stocate

După selectarea opţiunii *Deploy*, în secţiunea *Deploy options*, se lasă valorile implicite şi se apasă butonul *Finish*.

Pas 4: Executarea procedurii

După depunerea procedurii în mediul de funcţionare, aceasta poate fi executată prin selectare, apăsarea butonului din dreapta al mouse-ului şi alegerea comenzii *Run*. Rezultatele vor apare pe fila *Results* în colţul din dreapta jos al interfeţei grafice Data Studio aşa cum se vede în *Figura 6.5*.

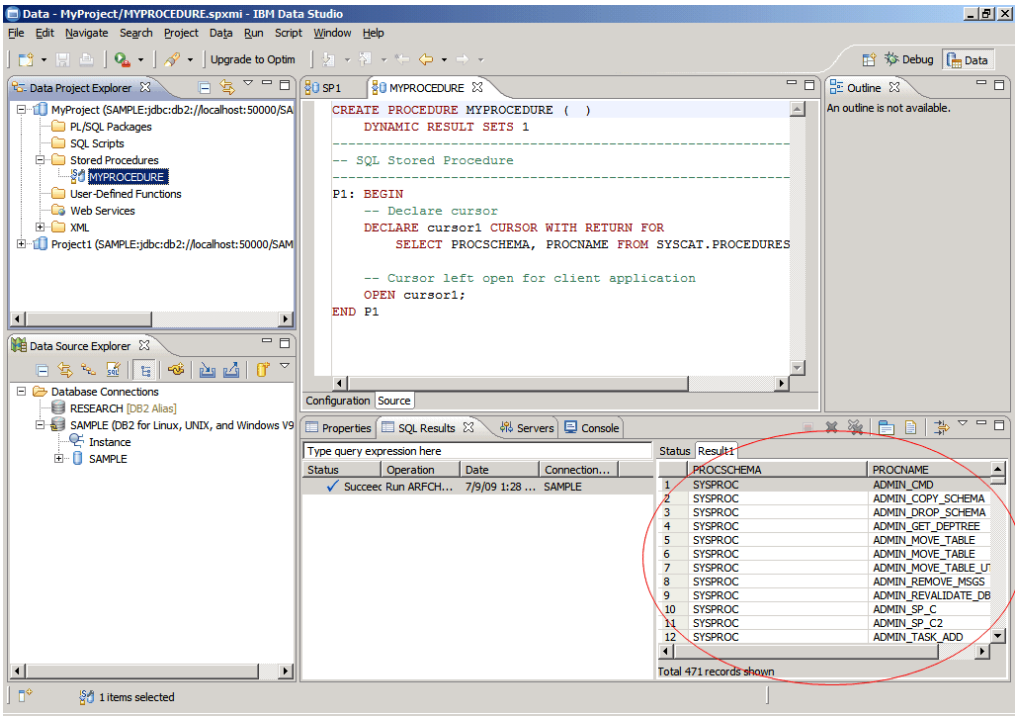


Figura 6.5 – Rezultatul obţinut după executarea procedurii stocate

Pentru a executa o procedură stocată din DB2 Command Window sau din Command Editor, se poate folosi comanda **`CALL <nume procedura>`**. De reţinut este faptul că mai întâi trebuie asigurată conexiunea la baza de date, deaorece acesta este locul în care se află procedura stocată. *Figura 6.6* prezintă acest lucru.

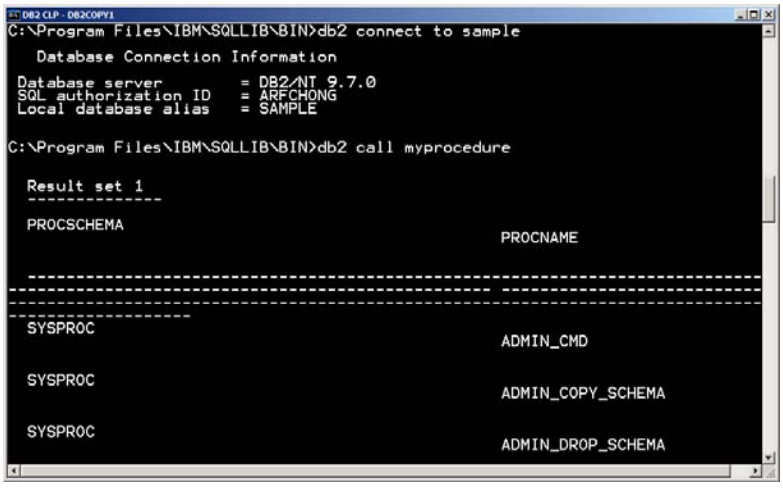


Figura 6.6 – Apelul unei proceduri stocate din DB2 Command Window

Aşa cum se procedează cu o procedură stocată în DB2 Command Window, la fel se procedează şi într-o aplicaţie Java, C, Visual Basic ş.a.m.d. Nu trebuie decât să se folosească sintaxa corectă pentru fiecare tip de limbaj de programare.

6.2.3 Modificarea şi eliminarea unei proceduri stocate

Există două modalităţi de modificare a unei proceduri stocate existente:

1. Se elimină procedura existentă şi se recreează din nou cu noua definiţie.
2. Se foloseşte sintaxa ‘CREATE OR REPLACE PROCEDURE' în locul sintaxei ‘CREATE PROCEDURE'.

Se mai poate folosi şi comanda ALTER PROCEDURE, dar aceasta poate fi folosită doar pentru modificarea anumitor proprietăţi ale procedurii şi nu codul propriu-zis.

Pentru a elimina o procedură, se foloseşte numele complet al acesteia în comanda ‘DROP PROCEDURE’ aşa cum se prezintă în exemplul de mai jos.

```
drop procedure myschema.EMPLOYEE_COUNT
```

Pentru a modifica o procedură este de preferat să se folosească sintaxa **`CREATE OR REPLACE PROCEDURE`** în locul eliminării şi recreării procedurii, deoarece eliminarea unei proceduri poate avea alte consecinţe, cum ar fi invalidarea altor obiecte ce depind de procedura respectivă. Folosind sintaxa **`CREATE OR REPLACE PROCEDURE`** o astfel de situaţie nu va mai avea loc.

6.3 Operarea cu funcţii

O funcţie SQL este o metodă sau o comandă care poate avea zero sau mai mulţi parametri de intrare, dar care returnează o singură valoare. Toate bazele de date au o serie de funcţii predefinite destinate efectuării de operaţii matematice, manipulării cu şiruri de caractere sau introducerii altor metode specifice bazelor de date. Utilizatorul poate, la rândul său, să-şi elaboreze propriile funcţii, cunoscute sub denumirea de funcţii definite de către utilizator, pentru a-şi rezolva anumite cerinţe care nu pot fi realizate cu funcţiile puse la dispoziţie de către sistem.

6.3.1 Tipuri de funcţii

Funcţiile pot fi clasificate în următoarele tipuri în funcţie de comportamentul acestora şi de setul de date de intrare cu care operează:

- Scalare
  - Agregat
  - Şir de caractere
- Tabelare

În plus, mai pot exista funcţii create în cadrul sistemului care sunt funcţii oferite de către sistem sau funcţii definite de către utilizator. O funcţie definită de către utilizator este o

extensie a limbajului SQL. Este un program de dimensiuni reduse ce poate fi scris în mod asemănător unui subprogram scris într-un limbaj gazdă. O funcţie definită de către utilizator, reprezintă de multe ori cea mai bună soluţie într-o aplicaţie SQL, deoarece aceasta poate fi apelată în cadrul unei comenzi SQL. În DB2, se pot crea funcţii scalare sau tabelare folosind SQL PL, PL/SQL, C/C++, Java, CLR (Common Language Runtime), şi OLE (Object Linking and Embedding).

6.3.1.1 Funcţii scalare

O funcţie scalară este o funcţie care, pentru fiecare set de unul sau mai mulţi parametri scalari întoarce o singură valoare. Funcţiile scalare se folosesc de obicei la manipularea şirurilor de caractere sau la efectuarea de operaţii matematice simple în cadrul comenzilor SQL. Funcţiile scalare nu pot conţine comenzi SQL care să modifice starea bazei de date, adică comenzile INSERT, UPDATE şi DELETE nu sunt permise.

De exemplu, funcţia predefinită LENGTH întoarce lungimea unui şir de caractere, aşa cum se prezintă mai jos:

```
SELECT length('Mary')
    FROM sysibm.sysdummy1
```

Comanda SQL de mai sus executată atunci când se realizează conexiunea la baza de date DB2 întoarce valoarea 4 care reprezintă lungimea şirului de caractere 'Mary'.

Funcţiile scalare pot fi folosite oriunde în cadrul unei comenzi SQL acolo unde este posibil, adică, de exemplu, într-o listă de selecţie, sau într-o clauză FROM. De exemplu:

```
SELECT EMPNO, LASTNAME, YEAR(CURRENT DATE - BRTHDATE)
     FROM EMPLOYEE
     WHERE WORKDEPT = 'D01'
```

Exemplul de mai sus introduce funcţia YEAR care este folosită pentru a extrage anul după efectuarea operaţiei "CURRENT DATE - BRTHDATE".

Funcţiile scalare predefinite se execută eficient deoarece se execută pe serverul bazei de date ca parte a unei comenzi SQL de care se asociază. Atunci când se foloseşte în cadrul unor predicate, o funcţie scalară poate îmbunătăţi performanţa generală a interogării. Atunci când o funcţie scalară se aplică pe mai multe rânduri, aceasta poate acţiona sub forma unui filtru, limitând numărul de rânduri ce trebuie returnate către client.

6.3.1.1.1 Funcţii agregat

O funcţie agregat este o funcţie care, pentru fiecare set de unul sau mai mulţi parametri scalari, returnează o singură valoare scalară. De exemplu, AVG(COL_NAME) întoarce valoarea medie a coloanei ’COL_NAME’.

6.3.1.1.2 Funcţii folosite pentru operarea cu şiruri de caractere

O funcţie folosită pentru operarea cu şiruri de caractere este o funcţie care acceptă cel puţin unul sau mai mulţi parametri scalari şi zero sau mai mulţi parametri de tip întreg pentru o singură valoare scalară (de tip şir de caractere sau întreg). De exemplu, SUBSTR('abcdefghi',3,4) are trei parametri (şirul sursă, subşirul locului de începere şi

numărul de caractere începând cu locul de pornire) şi returnează subşirul corespunzător. În exemplul de faţă, rezultatul este 'cdef'.

6.3.1.2 Funcţii tabelare

Funcţiile tabelare returnează ca rezultat un tabel. Acestea pot fi folosite în clauza FROM a unei interogări. Funcţiile tabelare, spre deosebire de funcţiile scalare, pot modifica starea bazei de date, motiv pentru care se pot folosi comenzile INSERT, UPDATE şi DELETE. Exemple de funcţii tabelare predefinite în SQL sunt SNAPSHOT_DYN_SQL() şi MQREADALL(). Funcţiile tabelare sunt asemănătoare vederilor, dar deoarece permit modificări ale datelor (INSERT, UPDATE şi DELETE) sunt mult mai puternice.

Mai jos se prezintă un exemplu de funcţie tabelară care afişează angajaţii din departamente:

```
CREATE FUNCTION getEnumEmployee(p_dept VARCHAR(3))
RETURNS TABLE
  (empno CHAR(6),
   lastname VARCHAR(15),
   firstnme VARCHAR(12))
SPECIFIC getEnumEmployee
RETURN
  SELECT e.empno, e.lastname, e.firstnme
    FROM employee e
    WHERE e.workdept=p_dept
```

6.3.2 Crearea unei funcţii

La fel ca în cazul procedurilor stocate, se poate folosi instrumentul IBM Data Studio pentru a crea o funcţie definită de către utilizator, singura diferenţă fiind aceea că trebuie apăsat butonul din dreapta al mouse-ului atunci când acesta se află deasupra folderului *user-defined functions*, urmând apoi toţi paşii descrişi anterior în cazul creării unei proceduri.

Comanda CREATE FUNCTION se foloseşte pentru a înregistra sau defini o funcţie definită de către utilizator sau un model de funcţie pe serverul de baze de date curent. Mai jos se prezintă o sintaxă simplificată de utilizare a acestei comenzi:

```
>>-CREATE--+------------+--FUNCTION--nume-functie-------------->
           '-OR REPLACE-'

      .-IN------.
>--(--+---------+--nume-parametru--| tip-data1 |--+-------------+--|-)-->
      |         |                                '-| clauza-implicita |-'
      +-OUT-----+
      '-INOUT---'

>-- RETURNS--+-| tip-data2 |-------------+--| lista-optiuni |----->
             '-+-ROW---+--| lista-coloane |-'
               '-TABLE-'
>--| corp-functie-SQL |-------------------------------------><
```

De exemplu, următoarea funcţie SQL PL întoarce un şir de caracatere în ordine inversă:

```
CREATE FUNCTION REVERSE(INSTR VARCHAR(40))
     RETURNS VARCHAR(40)
     DETERMINISTIC NO EXTERNAL ACTION CONTAINS SQL
     BEGIN ATOMIC
      DECLARE REVSTR, RESTSTR VARCHAR(40) DEFAULT '';
      DECLARE LEN INT;
      IF INSTR IS NULL THEN
         RETURN NULL;
      END IF;
      SET (RESTSTR, LEN) = (INSTR, LENGTH(INSTR));
      WHILE LEN > 0 DO
       SET (REVSTR, RESTSTR, LEN)
         = (SUBSTR(RESTSTR, 1, 1) CONCAT REVSTR,
            SUBSTR(RESTSTR, 2, LEN - 1),
            LEN - 1);
      END WHILE;
      RETURN REVSTR;
END
@
```

Pentru informaţii complete despre sintaxa CREATE FUNCTION precum şi a opţiunilor posibile, vedeţi DB2 v9.7 Information Center.

6.3.3 Apelarea unei funcţii

Funcţiile pot fi apelate în cadrul unei comenzi SQL sau în cadrul oricărei alte operaţii de manipulare a datelor. Funcţiile nu pot fi apelate în mod explicit folosind comanda ‘CALL’. Funcţiile obişnuite se apelează dintr-o comandă SELECT sau VALUES. De exemplu, funcţia REVERSE definită anterior poate fi apelată astfel:

```
SELECT reverse(col_name) from myschema.mytable

              SAU

VALUES reverse('abcd')
```

În cazul unei funcţii tabelare, aceasta se apelează într-o clauză FROM a unei comenzi SQL deoarece întoarce ca rezultat un tabel. Atunci când se foloseşte funcţia specială TABLE() trebuie utilizat un alias după apelare. De exemplu, pentru a apela funcţia tabelară `getEnumEmployee` creată anterior în cadrul acestei secţiuni, se va folosi o comandă SQL de genul celei prezentate în *Figura 6.1* de mai jos:

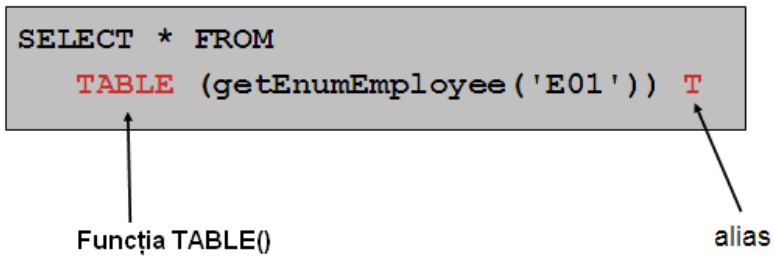


Figura 6.1 – Apelarea unei funcţii tabelare.

6.3.4 Modificarea şi eliminarea unei funcţii

Există două modalităţi de modificare a unei funcţii existente definită de către utilizator:

1. Eliminarea funcţiei existente şi recrearea acesteia cu o nouă definiţie.
2. Folosirea sintaxei 'CREATE OR REPLACE FUNCTION' în locul comenzii 'CREATE FUNCTION'.

Mai există şi posibilitatea folosirii comenzii ALTER FUNCTION, dar aceasta se foloseşte mai degrabă pentru modificarea unor proprietăţi şi nu pentru modificarea codului propriu-zis.

Pentru a elimina o funcţie se foloseşte numele complet al acesteia în cadrul comenzii 'DROP FUNCTION' aşa cum se vede din exemplul de mai jos:

```
DROP FUNCTION myschema.reverse
```

Pentru modificarea unei funcţii se preferă folosirea sintaxei **`CREATE OR REPLACE FUNCTION`** în locul eliminării şi recreării funcţiei respective, deoarece eliminarea unei funcţii poate avea alte consecinţe, cum ar fi invalidarea altor obiecte din baza de date care depind de acea funcţie. Folosind sintaxa **`CREATE OR REPLACE FUNCTION`** astfel de situaţii nu mai pot să apară.

6.4 Rezumat

Procedurile stocate şi funcţiile sunt instrumente foarte importante şi utile pentru implementarea metodelor specifice domeniului care nu sunt disponibile în cadrul sistemului în mod implicit. Procedurile stocate şi funcţiile au nevoie de permisiunea 'EXECUTE' pentru ca utilizatorii să le poată apela. Acestea oferă o modalitate de îmbunătăţire a performanţelor prin reducerea traficului de pe reţea, prin centralizarea codului în cadrul bazei de date şi prin creşterea securităţii acesteia.

6.5 Exerciţii

1. Creaţi o procedură stocată SQL PL folosind IBM Data Studio. Folosind baza de date SAMPLE din DB2, procedura trebuie să preia identificatorul unui angajat ca parametru de intrare şi să extragă toate coloanele corespunzătoare acestui angajat din tabelul EMPLOYEE.

2. Să se creeze o funcţie definită de către utilizator cu aceleaşi cerinţe din exerciţiul (1).
3. Să se creeze o funcţie SQL care nu acceptă parametri de intrare şi returnează numlele unei zile a săptămânii sub forma unui şir de caractere (luni, marţi etc.) prin preluarea datei curente a sistemului ca dată de intrare.
4. Să se creeze o funcţie SQL care are o dată calendaristică ca parametru de intrare şi returnează numele unei zile a săptămânii sub forma unui şir de caractere (luni, marţi etc.).
5. Să se creeze o procedură stocată fără parametri de intrare care să returneze numărul de tabele din baza de date.
6. Să se creeze o procedură care primeşte numele unui tabel ca parametru de intrare şi întoarce numărul de rânduri din acel tabel.

6.6 Întrebări recapitulative

1. Care dintre următoarele este o funcţie corectă?
   A. Select
   B. Update
   C. Count
   D. Delete
   E. Nici una dintre cele enumerate
2. Care dintre următoarele nu este o funcţie corectă?
   A. avg
   B. count
   C. insert
   D. substr
   E. Nici una dintre cele enumerate
3. Care dintre următoarele este o funcţie scalară?
   A. avg
   B. count
   C. max
   D. substr
   E. Toate cele enumerate
4. Care dintre următoarele nu este o funcţie scalară?
   A. trim

B. upper

C. min

D. substr

E. Nici una dintre cele enumerate

5. Care dintre următoarele este o funcţie agregat?

A. lcase

B. year

C. max

D. substr

E. Nici una dintre cele enumerate

6. Care dintre următoarele nu este o funcţie agregat?

A. sum

B. min

C. max

D. len

E. Nici una dintre cele enumerate

7. Care dintre următoarele este o funcţie pentru operarea cu şiruri de caractere?

A. avg

B. year

C. max

D. substr

E. Nici una dintre cele enumerate

8. Care dintre limbajele de programare sunt suportate de către Data Studio pentru a crea o procedură stocată?

A. SQL PL

B. PL/SQL

C. Java

D. Toate cele enumerate

E. Nici unul dintre cele enumerate

9. O funcţie poate fi apelată folosind comanda VALUES

A. Adevărat

B. Fals

10. O procedură poate fi apelată folosind comanda ‘CALL’

A. Adevărat

B. Fals

7

Capitolul 7 – Folosirea SQL în aplicaţii

În capitolele precedente s-a discutat despre limbajul structurat de interogare (SQL), care reprezintă limbajul standardizat de lucru cu bazele de date, de manipulare a obiectelor din cadrul acestora şi de extragere a datelor existente. Acest capitol va prezenta conceptele cu ajutorul cărora poate fi apelat codul SQL din cadrul aplicaţiilor care, în general, sunt scrise în limbajele obişnuite cum ar fi C, C++, Java, .NET şi altele.

În cadrul acestui capitol vă veţi familiariza cu:

- Conceptul de tranzacţie
- Operarea cu cod SQL încorporat
- Diferenţele dintre codul SQL static şi dinamic
- Interfeţele de programare a aplicaţiilor pentru baze de date (API), cum ar fi ODBC, CLI şi JDBC
- Tehnologia pureQuery

7.1 Folosirea SQL în aplicaţii: vedere de ansamblu

SQL este limbajul standard ce permite utilizatorilor să comunice cu serverul bazei de date, dar pentru a scrie aplicaţii complexe de mari dimensiuni care folosesc baze de date, utilizarea doar a limbajului SQL nu este suficientă. Limbajele folosite la elaborarea aplicaţiilor, cum ar fi C, C++ sau Java oferă utilizatorilor un control mai riguros şi o logică funcţională mai puternică. Aceste limbaje, cunoscute sub denumirea de ***limbaje gazdă*** se pot integra bine cu limbajul SQL pentru a putea comunica cu baza de date în cadrul aplicaţiilor. În acest caz, se spune că limbajul SQL este încorporat în aplicaţia gazdă.

Alte tehnici permit utilizarea directă a apelurilor către interfaţa de programare a aplicaţiilor (API) pentru a obţine accesul la baza de date. De exemplu, ODBC, CLI şi JDBC sunt astfel de interfeţe de programare aplicaţiilor folosite în cazul bazelor de date.

Toate tehnicile de folosire a limbajului SQL precum şi a modului în care acestea interacţionează cu baza de date sunt prezentate în *Figura 7.1* de mai jos şi vor fi discutate în detaliu în secţiunile următoare.

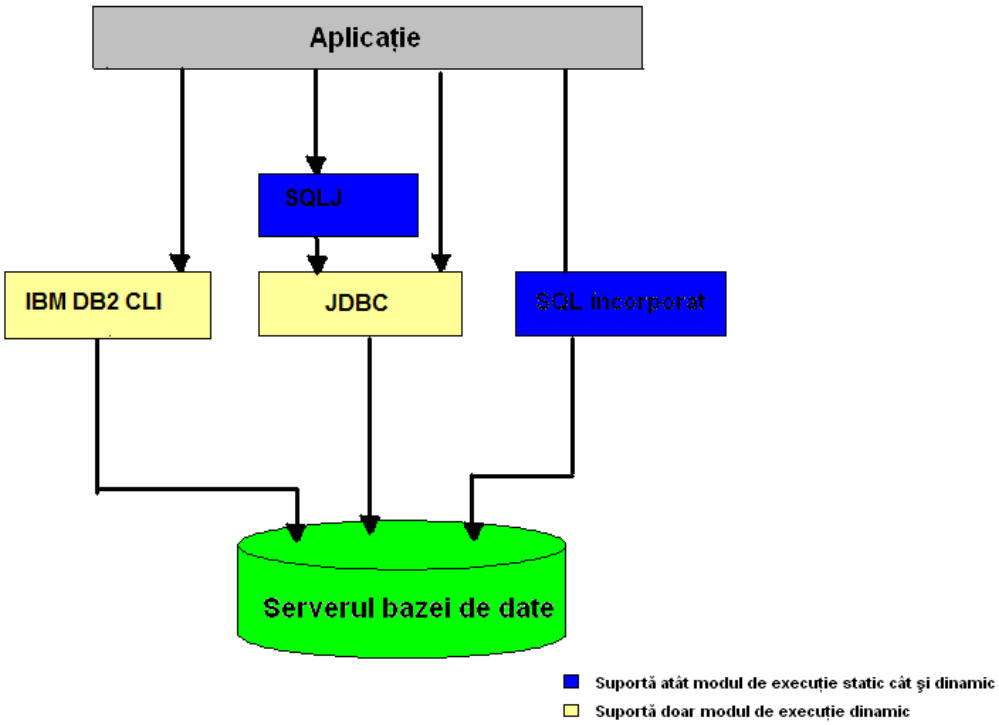


Figura 7.1 – Rezumatul tuturor tehnicilor folosite în aplicaţiile SQL

7.2 Ce este o tranzacţie?

Înainte de a intra în detaliile diferitelor tehnici utilizate în aplicaţiile ce folosesc SQL, vă veţi familiariza cu conceptul de tranzacţie. O ***tranzacţie*** sau ***unitate de operare*** este un grup de operaţii efectuate în baza de date care trebuie executate cu succes în întregime pentru ca tranzacţia să reuşească. Dacă una dintre operaţii nu reuşeşte, tranzacţia este anulată.

De exemplu, dacă o bancă trebuie să transfere 1000 de dolari din contul A în contul B, atunci trebuie să reuşească fiecare dintre următoarele secvenţe înainte să se efectueze transferul:

- Se reduce balanţa contului A cu valoarea de 1000
- Se măreşte balanţa contului B cu valoarea de 1000

În SQL, cerinţele de mai sus se rezolvă cu ajutorul a două comenzi SQL. Dacă una sau mai multe dintre aceste comenzi eşuează, transferul de bani nu reuşeşte. Din acest motiv se spune că cele două comenzi din exemplul de mai sus alcătuiesc o tranzacţie. Dacă o tranzacţie nu reuşeşte, se realizează o rulare înapoi în baza de date astfel încât datele să-şi păstreze consistenţa, adică să se afle în starea de dinainte de începerea tranzacţiei. Cele mai întâlnite exemple de aplicaţi ce folosesc tranzacţii sunt, de exemplu, sistemele de rezevări online, sistemele de cumpărături online, ş.a.m.d.

7.3 SQL încorporat

Aşa cum îi arată şi numele, SQL încorporat presupune încorporarea codului SQL în cadrul aplicaţiilor gazdă scrise în limbaje de programare de nivel înalt, cum ar fi C, C++, sau COBOL. Aplicaţia gazdă conţine logica de bază necesară, apelând comenzile SQL la nevoie pentru a obţine informaţii din baza de date.

Atunci când se întâmplă acest lucru, apare o întrebare: cum se pot compila aplicaţiile scrise în limabjul de programare de nivel înalt care conţin cod SQL încorporat, deoarece nici unul dintre limbajele de programare de acest tip nu poate compila şi valida sintaxa SQL?

Răspunsul la această întrebare este oferit prin intermediul operaţiei de pre-compilare. În cazul sistemului DB2, acesta are un pre-compilator care efectuează conversiile necesare sintaxei SQL direct în apeluri de servicii API la execuţia DB2. Codul pre-compilat este apoi compilat şi ataşat folosind instrumentele de dezvoltare ale limbajului respectiv.

Comenzile SQL încorporate în limbajul gazdă trebuie identificate cu ajutorul unei propoziţii sau a unui bloc. În cazul C, C++ şi COBOL, comanda de iniţializare este EXEC SQL. Această comandă este urmată de comanda SQL care se va executa şi se termină cu simbolul (;), care reprezintă terminatorul comenzii. De exemplu:

```
EXEC SQL
UPDATE employee.details
       SET emp_desig = 'Mgr' WHERE emp_desig = 'Asst Mgr';
```

Comenzile SQL pot fi încorporate şi în cadrul aplicaţiilor Java, iar aplicaţiile ce conţin cod SQL încorporat sunt denumite aplicaţii **SQLJ**. În mod asemănător comenzii de iniţializare din C, EXEC SQL, în aplicaţiile SQLJ se foloseşte comanda de iniţializare **#sql**.

De exemplu, comanda SQL UPDATE din exemplul anterior arată astfel în aplicaţia SQLJ:

```
#sql {
 UPDATE employee.details
        SET emp_desig = 'Mgr' WHERE emp_desig = 'Asst Mgr'};
```

Ambele exemple prezentate anterior conţin comenzi ***SQL statice*** deoarece numele tabelului, numele de coloane şi alte obiecte ale bazei de date sunt cunoscute şi rămân constante de fiecare dată când rulează aplicaţia.

Dacă aceste obiecte ar fi introduse în program în timpul execuţiei acestuia, vom vorbi despre comenzi ***SQL dinamice*** deoarece comanda SQL este determinată la momentul execuţiei. Vom discuta despre comenzile statice şi dinamice mai târziu, în secţiunile următoare.

7.3.1 Comenzi SQL statice

Comenzile SQL statice reprezintă modalitatea tradiţională de elaborare a aplicaţiilor cu SQL încorporat. Aplicaţiile ce conţin cod SQL static sunt destinate situaţiilor în care aplicaţiile vor folosi de fiecare dată când interacţionează cu baza de date aceleaşi comenzi

SQL. De exemplu, o aplicaţie care actualizează stocul de inventar, va folosi întotdeauna aceeaşi comandă SQL pentru a modifica stocul. În mod asemănător, sistemul de rezervare online va actualiza întotdeauna acelaşi tabel şi va marca aceeaşi coloană pentru rezervare. Pentru a vedea situaţia rezervărilor la un moment dat, se va folosi aceeaşi comandă SELECT pe acelaşi tabel.

O aplicaţie SQL încorporată în care sintaxa SQL este cunoscută complet dinainte şi în care comenzile SQL sunt încorporate în codul sursă al aplicaţiei este cunoscută sub denumirea de ***aplicaţie SQL încorporată static***. Singura intrare (intrări) care trebuie introduse în comenzile SQL din cadrul aplicaţiei este (sunt) valorile de date curente ce trebuie inserate în cadrul tabelelor sau valorile predicatelor din comenzile SQL. Aceste valori de intrare sunt introduse în comenzile SQL cu ajutorul **variabilelor gazdă**.

7.3.1.1 Variabile gazdă

Variabilele gazdă sunt variabile ale limbajului de programare ce pot fi folosite doar pentru procesarea codului SQL static. Aceste variabile gazdă trebuie declarate în aplicaţie înainte de a fi folosite. Se recomandă ca aceste variabile să fie iniţializate folosind valori implicite, în momentul declarării. O altă recomandare este aceea de a se adăuga la numele variabilei gazdă sufixul '**_hv**' to pentru a face diferenţa de celelalte variabile sau de nume de coloane.

Să vedem fragmentul de cod SQL încorporat într-o aplicaţie scrisă în limbajul C prezentat în *Lista 7.1*

```
EXEC SQL
  SELECT emp_name, emp_dept, emp_salary
    INTO :name_hv, :dept_hv, :salary_hv
    FROM employee.details
   WHERE emp_id = :id_hv ;

EXEC SQL
  UPDATE employee.details
         SET emp_salary = :new_salary_hv
         WHERE emp_id = :id_hv ;
```

Lista 7.1 - Fragmentul de cod SQL încorporat într-o aplicaţie scrisă în limbajul C

În ambele comenzi, numele tabelului (**`employee.details`**) şi numele coloanelor (**`emp_name, emp_dept`** etc.) sunt toate hard-codate. Introducerea informaţiei în comenzile SQL la momentul execuţiei se face cu ajutorul variabilelor gazdă (**`id_hv, dept_hv`** etc.). Simbolul (**:**) din faţa fiecărei variabile gazdă este parte a sintaxei codului SQL încorporat.

7.3.1.2 Structura unei aplicaţii cu cod SQL încorporat

Spre deosebire de limbajul gazdă, toate aplicaţiile care au cod SQL încorporat sunt alcătuite din următoarele trei elemente principale necesare pregătirii şi executării comenzilor SQL.

- O secţiune declarativă, DECLARE SECTION, pentru declararea variabilelor gazdă.
- Corpul principal al aplicaţiei, care este alcătuit din pregătirea şi execuţia comenzilor SQL.
- Plasarea de elemente care fie încheie cu succes, fie rulează înapoi modificările făcute de comenzile SQL (dacă este necesar).

Lista 7.2 prezintă un fragment de cod SQL încorporat într-o aplicaţie scrisă în limbaj C care foloseşte cele trei elemente.

```
int getDetails( int employee_id, double new_salary)
{
    int ret_code = 1;
    EXEC SQL INCLUDE SQLCA;

    EXEC SQL BEGIN DECLARE SECTION;
    sqlint32 id_hv = 0;   // identificatorul angajatului
    char name_hv[129] = {0};     // numele angajatului
    char dept_hv[129] = {0};     // departamentul angajatului
    double salary_hv = 0; // salariul angajatului

EXEC SQL END DECLARE SECTION;
// Se copiaza identificatorul si salariul din această functie
// în variabilele gazda
id_hv = employee_id;
salary_hv = new_salary;

// Se foloseste comanda UPDATE pentru a introduce noul salariu al unui
// angajat
EXEC SQL
 UPDATE employee.details
        SET emp_salary = :salary_hv WHERE emp_id = :id_hv;

if (SQLCODE < 0)
   {
    printf("\n UPDATE SQL Error:%ld\n",SQLCODE);
    EXEC SQL ROLLBACK; // Tranzactia se ruleaza inapoi
    ret_code = 0; // eroare
   }
else
   {
    EXEC SQL COMMIT; // Tranzactia s-a realizat cu succes

// Se foloseste o comanda SELECT pentru a parcurge salariile
// actualizate
   EXEC SQL
```

```
    SELECT emp_name, emp_dept, emp_salary
      INTO :name_hv, :dept_hv, :salary_hv
      FROM employee.details
     WHERE emp_id  = :id_hv;

   if (SQLCODE < 0)
   {
     printf("\n SELECT SQL Error:%ld\n",SQLCODE);
     Ret_code = 0;
   }
   else
   {
    // Se afiseaza salariile actualizate
       printf("\n Employee name: %s",name_hv);
       printf("\n Employee Id: %d",id_hv);
       printf("\n Employee Department: %s",dept_hv);
       printf("\n Employee New Salary: Rs. %ld p.a",salary_hv);
   }
  }
  return ret_code;
}
```

Lista 7.2 – Structura unei aplicaţii scrisă în limbajul C ce conţine cod SQL încorporat

Exemplul de mai sus arată felul în care comenzile SQL pot fi încorporate în cadrul unei aplicaţii scrise în limbajul de programare C. Alte limbaje de programare cum ar fi C++, COBOL, Java, trebuie să folosească aceleaşi trei elemente principale ca în exemplul de mai sus.

7.3.1.3 Zona de comunicaţii SQL, SQLCODE şi SQLSTATE

Zona de comunicaţii SQL (SQLCA) este o structură de date care se foloseşte ca mediu de comunicare dintre serverul bazei de date şi clienţii săi. Structura de date SQLCA conţine un număr de variabile care sunt actualizate la sfârşitul fiecărei execuţii. SQLCODE este una dintre variabilele din structura de date care este setată pe valoarea 0 (zero) după fiecare execuţie SQL. Dacă comanda SQL se încheie cu un mesaj de avertizare, variabila trece pe o valoare pozitivă, diferită de zero, iar dacă se încheie cu eroare trece pe o valoare negativă.

Valorile SQLCODE pot reprezenta fie probleme de hardware, fie de sistem de operare; de exemplu, atunci când sistemul de fişiere este plin, sau atunci când apare o eroare la accesarea unui fişier. Se recomandă să se urmărească valorile returnate de către SQLCODE după fiecare execuţie SQL aşa cum se arată în aplicaţia de mai sus.

SQLSTATE este o altă variabilă oferită de către SQLCA ce păstrează coduri de retur sub formă de şiruri de caractere care prezintă rezultatul celei mai recente comenzi SQL executate. SQLSTATE prezintă un mesaj generic, standardizat, folosit de către diverşi producători de baze de date.

7.3.1.4 Paşii necesari pentru compilarea unei aplicaţii SQL statice

Aşa cum s-a arătat anterior aplicaţiile SQL încorporate trebuie să fie pre-compilate înainte de a fi folosite de pre-compilatorul DB2. Pre-compilatorul verifică comenzile SQL din cadrul codului sursă, le înlocuieşte cu API-urile echivalente folosite la execuţie de DB2 şi suportate de către limbajul gazdă şi rescrie rezultatul (împreună cu comenzile SQL comentate) într-un fişier nou care poate fi compilat şi asociat cu ajutorul instrumentelor de dezvoltare folosite de limbajul gazdă.

Pre-compilatorul DB2 se apelează cu ajutorul comenzii DB2 **`PRECOMPILE`** (sau **`PREP`**). Pre-compilatorul DB2 efectuează următoarele activităţi:

1. Validează sintaxa SQL pentru fiecare comandă SQL codată şi introduce tipurile de date corespunzătoare pentru variabilele gazdă prin compararea acestora cu tipurile de date ale coloanelor. De asemenea se determină metoda de conversie a datelor ce trebuie folosită la parcurgerea sau inserarea datelor în baza de date.

2. Reevaluează asocierile cu obiectele bazei de date, creează **planurile de acces** şi le introduce în cadrul unui **pachet** din baza de date. Planul de acces al unei comenzi SQL reprezintă calea optimă către obiectele bazei de date la care face referire codul SQL. Optimizatorul DB2 estimează planurile de acces la momentul pre-compilării unei comenzi SQL încorporate static. Această estimare se bazează pe informaţia disponibilă în cadrul optimizatorului la momentul pre-compilării, care este urmărită prin intermediul cataloagelor sistemului.

   Suplimentar, fiecare aplicaţie este **asociată** pachetului corespunzător din baza de date. Avantajul păstrării acestor planuri de acces în baza de date este acela că ori de câte ori se execută aplicaţia, se parcurge şi se execută planul de acces corespunzător din cadrul pachetului, ceea ce asigură o creştere a vitezei de execuţie a codului SQL, deoarece este cunoscută dinainte calea optimă de accesare a obiectelor din baza de date.

   În acest moment se pot stabili comenzile SQL statice al căror plan de acces poate fi determinat, cunoscut dinainte şi păstrat în baza de date pentru a creşte viteza de execuţie. Din acest motiv, în cazul unei comenzi SQL care urmează să fie încorporată static în cadrul unei aplicaţii, sintaxa acesteia trebuie cunoscută la momentul pre-compilării.

Odată ce aplicaţia SQL încorporată este pre-compilată şi introdusă în baza de date, aceasta poate fi compilată şi asociată cu ajutorul instrumentelor de dezvoltare ale limbajului gazdă. De asemenea, este posibil să se amâne asocierea aplicaţiei până la momentul execuţiei. Acest pas poate fi făcut cu ajutorul clauzei **BINDFILE** <*fisier_de_asociere*> din comanda PRECOMPILE, care creează un fişier de asociere ce conţine datele necesare creării pachetului din baza de date. Acest <*fisier_de_asociere*> poate fi apoi utilizat de către utilitarul DB2 **`BIND`** pentru a crea un pachet în baza de date căruia să-i ataşezeze aplicaţia. Această tehnică este cunoscută sub denumirea de *asociere întârziată*

Figura 7.2 de mai jos descrie întregul proces de compilare a codului SQL. Această diagramă reprezintă setul general de acţiuni ce trebuie efectuate pe perioada compilării şi

poate fi diferit în funcţie de compilatorul folosit.

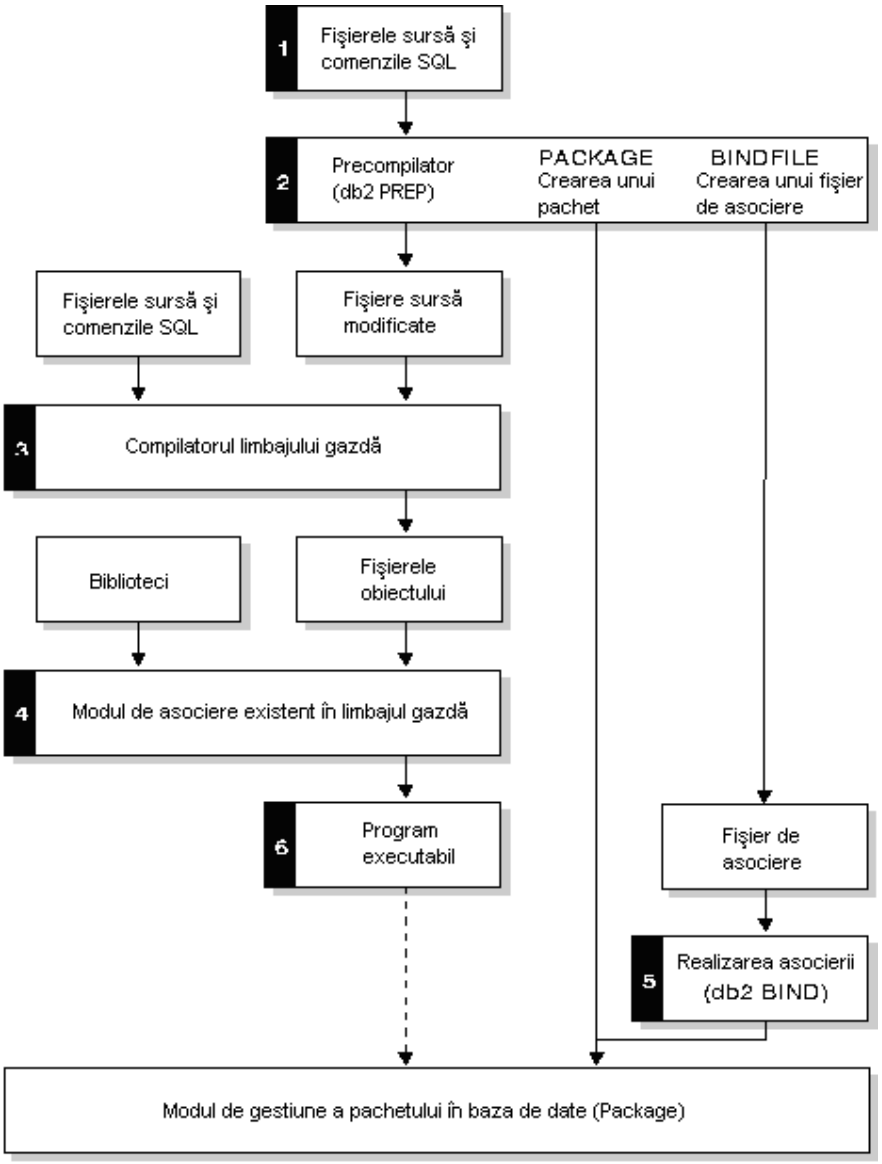


Figura 7.2 – Compilarea şi execuţia codului static SQL

În cazul aplicaţiilor **SQLJ**, asemănător pre-compilatorului pentru aplicaţii SQL încorporate, există un modul, **SQLJ translator** care detectează clauzele SQL din cadrul aplicaţiilor SQLJ pe care le converteşte în comenzi JDBC. JDBC se va prezenta mai în detaliu în secţiunile următoare.

7.3.2 Comenzi SQL dinamice

Comenzile SQL dinamice conţin parametri ale căror valori nu sunt cunoscute până la momentul execuţiei, fiind introduse cu ajutorul aplicaţiei. Comenzile SQL dinamice se întâlnesc foarte des în cadrul aplicaţiilor interactive în care utilizatorul introduce cea mai mare parte a informaţiilor necesare pentru a crea comanda SQL dorită. Să luăm de exemplu aplicaţia prezentată în *Figura 7.3*, "Formular de căutare a unui angajat", care face parte dintr-o aplicaţie de resurse umane din cadrul unei companii.

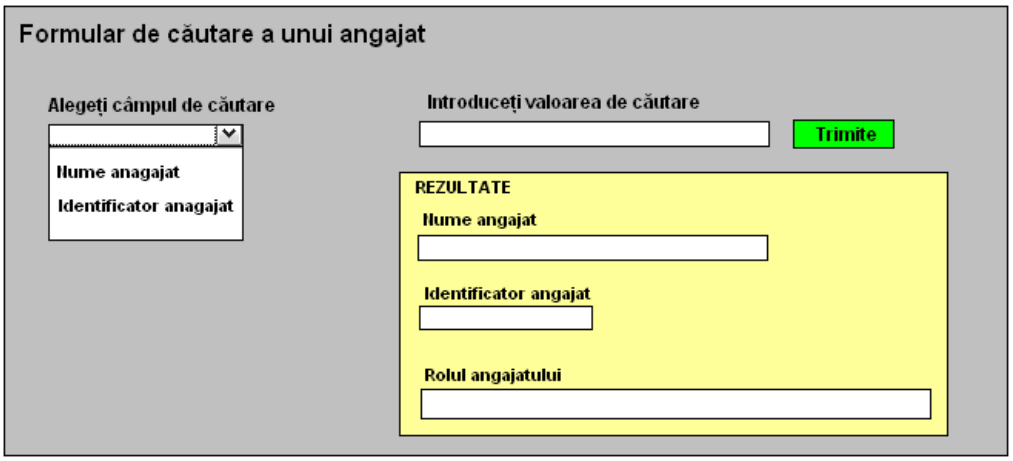


Figura 7.3 - Formular de căutare a unui angajat

Formularul de mai sus permite utilizatorului să ofere detaliile necesare stabilirii rolului unui angajat. Valorile de intrare sunt fie *Nume angajat* fie *Identificator angajat*. În cazul acestui tip de aplicaţii, codul SQL nu poate fi cunoscut decât în momentul execuţiei, fiind un exemplu de necesitate a folosirii codului SQL dinamic.

Spre deosebire de tehnica folosirii statice a codului SQL, **planurile de acces** pentru interogările SQL dinamice pot fi generate numai în momentul execuţiei.

7.3.2.1 Structura aplicaţiei SQL încorporate

Lista 7.3 prezintă codul corespunzător Formularului de căutare aunui anagajat din *Figura 7.3*. Acesta arată o demonstraţie a felului în care se execută o comandă SQL în mod dinamic din cadrul unei aplicaţii încorporate.

```
int findEmployee(char * field, char * value, char * emp_name, char *
emp_id, char * emp_desig)
{
 int ret_code = 1;
 char sqlstmt[250] = {0};
 EXEC SQL INCLUDE SQLCA;

 EXEC SQL BEGIN DECLARE SECTION;
  char name_hv[129] = {0};  // Nume angajat
  char desig_hv[129] = {0}; // Rolul angajatului
  char id_hv[10] = {0};     // Identificator angajat
  char value_hv[250] = {0}; // valoare camp
 EXEC SQL END DECLARE SECTION;
// Copiaza valoarea din campul de cautare obtinuta cu aceasta functie in
// sirul de caractere sqlstmt
sprintf(sqlstmt,"SELECT emp_name, emp_id, emp_desig
                  FROM employee.details
                 WHERE %s = ?",field);
// Copiaza valoarea obtinuta cu aceasta functie in
// variabila gazda
     strcpy(value_hv,value) ;

// Pregateste mai intai comanada SQL dianmica
// Comanda va crea planul de acces la executie
   EXEC SQL PREPARE dynsqlstmt FROM :sqlstmt;
   if (SQLCODE <0)
   {
     printf("\n Error during Prepare statement:%ld",SQLCODE);
     ret_code = 0; //error
   }
   else
   {
     EXEC SQL DECLARE cur1 CURSOR FOR :dynsqlstmt;
     if (SQLCODE <0)
     {
      printf("\n Error during Declare Cursor:%ld",SQLCODE);
      ret_code = 0; //error
      }
      else
      {
         EXEC SQL OPEN cur1 USING :value_hv;
         if (SQLCODE <0)
         {
           printf("\n Error during Open Cursor:%ld",SQLCODE);
           ret_code = 0; //error
```

```
          }
          else
          {
            EXEC SQL FETCH cur1 INTO :name_hv, :id_hv, :design_hv;
            if (SQLCODE <0)
            {
              printf("\n Error during Fetch cursor:%ld",SQLCODE);
              ret_code = 0; //eroare
            }
            else
            {
             EXEC SQL CLOSE cur1;
              if (SQLCODE <0)
              {
                printf("\n Error during Close cursor:%ld",SQLCODE);
                Ret_code = 0; //eroare
              }
             else
             {
               // Copiaza rezultatele obtinute in variabilele
              // corespunzatoare
               strcpy(emp_name,:name_hv);
               strcpy(emp_id,:id_hv);
               strcpy(emp_desig,:desig_hv);
              }
            }
         }
       }
     }
  return ret_code;
}
```

Lista 7.3 – Cod SQL încorporat în C cu elemente statice şi dinamice

În lista de mai sus, informaţiile referitoare la câmpul de căutare al formularului angajaţilor (pe care utilizatorul îl completează la execuţie) este mai întâi reţinută în varaibila numită **field**. Această informaţie este copiată în comanda SQL fiind păstrată apoi într-o variabilă de tip şir de caractere (vezi **sqlstmt** de mai sus).

Să spunem că utilizatorul alege să caute angajatul după câmpul **`Identificator angajat`**. În acest caz, comanda SQL (fără valorile predicatului) va arăta astfel:

```
SELECT emp_name, emp_id, emp_desig from employee.details WHERE emp_id =?
```

Semnul întrebării folosit în cadrul comenzii se numeşte **marcator de parametru**. Aceşti marcatori se folosesc în locul valorilor predicatelor şi arată faptul că valorile curente se vor introduce mai târziu, la execuţie. Deoarece valorile exacte ale predicatelor nu sunt necesare la generarea planului de acces, informaţia SQL existentă este suficientă în acest

moment pentru a crea acest plan. Planul de acces este elaborat folosind comanda SQL dinamică '**de pregătire** ' (vezi EXEC SQL PREPARE din lista de mai sus).

Dacă utilizatorul introduce date în câmpul **`Nume angajat`** codul SQL devine:

```
SELECT emp_name, emp_id, emp_desig from employee.details WHERE emp_name =?
```

În acest fel comanda SQL completă este generată doar la execuţie, iar pregătirea acestei comenzi va conduce la un plan de acces diferit de cel anterior.

Odată ce comanda a fost pregătită cu succes, este posibil să se execute folosind valorile predicatelor introduse de către utilizator. În cazul unei comenzi SELECT acest lucru se realizează prin declararea mai întâi a unui cursor pe comanda SQL, după care acesta se deschide şi se introduc valorile predicatelor, iar la final se folosesc rezultatele obţinute în urma interogării în cadrul variabilelor gazdă. La terminarea operaţiei cursorul trebuie închis.

1. Codul de mai sus conţine unele comenzi SQL statice necesare pregătirii comenzii, declarării cursorului ş.a.m.d. Aceasta înseamnă că astfel de aplicaţii au nevoie de pre-compilare sau traducere a codului SQL (pentru aplicaţiile SQLJ), deoarece mai sunt încă dependente de comenzile SQL statice.
2. Dacă există o modalitate de înlocuire a tuturor comenzilor SQL statice cu echivalentele API, pre-compilarea/ traducerea codului SQL al aplicaţiei nu ar mai fi necesare deloc. Întrebarea care apare într-o astfel de situaţie este cum ar putea fi scrise astfel de aplicaţii SQL dinamice care să nu necesite deloc comenzi SQL statice (nici pentru comenzile PREPARE, EXECUTE, CURSOR ş.a.m.d.). răspunsul la această întrebare este dat în secţiunile următoare.

7.3.3 Comenzi SQL statice sau dinamice?

În ceea ce urmează se vor prezenta câteva dintre diferenţele existente între modurile de execuţie statică şi dinamică ale aplicaţiilor ce au cod SQL încorporat:

1. Spre deosebire de comenzile SQL statice, planurile de acces ale comenzilor dinamice sunt generate la execuţie, motiv pentru care comenzile dinamice trebuie să fie pregătite în cadrul aplicaţiei.
2. Timpul necesar pentru generarea planului de acces la execuţie face ca aplicaţiile ce conţin cod SQL dinamic să fie un pic mai lentre decât cele care conţinj cod SQL static. Totuşi, aplicaţiile ce conţin cod SQL dinamic sunt mult mai flexibile decât cele ce conţin cod SQL static, ceea ce face ca primele să fie mult mai robuste.
3. Uneori, o comandă SQL dinamică poate să aibă performanţe mai bune decât una statică echivalentă deoarece poate folosi ultimele statistici disponibile în baza de date în momentul execuţiei. Planul de acces generat în cazul unei comenzi SQL statice este memorat dinainte şi poate deveni depăşit de noile statistici din baza de date, ceea ce nu este cazul pentru comenzile dinamice.
4. Unul dintre avantajele comenzilor SQL dinamice asupra comenzilor statice se poate vedea chiar dacă aplicaţia nu este modificată sau actualizată. Dacă se modifică.**partea statică a comenzii SQL** este necesară refacerea planului de

acces. Aceasta presupune pre-compilarea aplicaţiei şi refacerea asocierilor cu pachetele. În cazul comenzilor SQL dinamice, deoarece planurile de acces sunt generate la execuţie, nu sunt necesare pre-compilarea sau reasocierea.

7.4 Interfeţele de programare a aplicaţiilor pentru bazele de date

Aşa cum am văzut anterior, deşi codul SQL încorporat are posibilitatea de executare dinamică a comenzilor SQL, acesta tot mai conţine comenzi SQL statice care necesită pre-compilare sau traducere.

Mai mult decât atât, pentru a elabora planurile de acces, este necesară o conexiune la baza de date deoarece pre-compilarea aplicaţiilor cu cod SQL încorporat necesită informaţii statistice preluate din catalogul bazei de date.

Aceasta ne aduce într-o altă zonă a elaborării de aplicaţii SQL care utilizează interfeţele de programare a aplicaţiilor pentru bazele de date (APIs). Aceste interfeţe de programare a aplicaţiilor sunt, în totalitate, dinamice prin natură şi pot fi elaborate în mod independent de produsul de baze de date utilizat, fără a fi necesară pre-compilarea.

Interfeţele de programare a aplicaţiilor pentru bazele de date, aşa cum le sugerează şi numele, reprezintă un grup de interfeţe de programare a aplicaţiilor puse la dispoziţie de către producătorii de baze de date pentru comunicarea cu diverse limbaje de programare cum ar fi C, C++, Java ş.a.m.d. Prin intermediul acestora se oferă programatorilor un mecanism de interacţiune cu bazele de date din cadrul unei aplicaţii doar prin simpla apelare a acestor *interfeţe de apelare a codului SQL.*

Stratul intermediar dintre aplicaţie şi serverul bazei de date, care face posibilă interacţiunea, este **driver-ul de conectare la baza de date**. Producătorii de baze de date oferă ei înşişi astfel de drivere şi, oadată ce bibliotecile driverelor sunt asociate cu bibliotecile codului sursă, codul sursă al aplicaţiei poate fi compilat şi executat cu uşurinţă.

Aplicaţiile pot fi elaborate actualmente, în mod independent de bazele de date cu care lucrează, fără a fi nevoie să existe conexiune cu baza de date în momentul compilării. Aceasta permite programatorilor să elaboreze aplicaţii fără a fi nevoie să cunoască sintaxa codului SQL încorporat. Secţiunile următoare prezintă în detaliu aceste drivere necesare pentru realizarea conectivităţii.

7.4.1 ODBC şi driverul IBM Data Server CLI

Pentru a se păstra o anumită standardizare la nivelul bazelor de date între toţi producătorii care oferă drivere de conectivitate, X/Open Company şi SQL Access Group au elaborat împreună o specificaţie de apelare a interfeţei SQL numită X/Open Call Level Interface. Scopul acestei interfeţe este acela de a creşte portabilitatea aplicaţiilor prin oferirea independenţei de producător. Cele mai multe dintre specificaţiile X/Open Call Level Interface au fost acceptate ca parte a standardului ISO Call Level Interface International Standard (ISO/IEC 9075-3:1995 SQL/CLI).

Microsoft a elaborat o interfaţă de apelare a codului SQL numită Open Database Connectivity (ODBC) care se foloseşte pe sistemele de operare Microsoft şi care se

bazează pe un draft preliminar al X/Open CLI. Totuşi, ODBC nu se mai limitează la sistemele de operare Microsoft, iar actualmente, există multe implementări şi pe alte platforme.

Driverul IBM Data Server CLI este o interfaţă de apelare creată pentru DB2 care se bazează pe specificaţiile ODBC de la Microsoft® şi pe specificaţiile International Standard for SQL/CLI. Aceste specificaţii au fost alese ca bază pentru DB2 Call Level Interface pentru a răspunde standardelor din domeniu şi pentru a oferi o soluţie mai simplă programatorilor obişnuiţi cu alte asemenea tipuri de interfeţe. Pe lângă acestea au fost adăugate anumite extensii specifice DB2 pentru ca programatorii să se poată folosi de caracteristicile specifice acestui produs. DB2 este o interfaţă de programare a aplicaţiilor destinată limbajelor de programare C şi C++ pentru lucrul cu baze de date relaţionale care folosesc apeluri de funcţii pentru a executa comenzi SQL dinamice introduse sub formă de argumente ale acestor funcţii.

Chiar şi pentru comenzile PREPARE şi EXECUTE există comenzi corespunzătoare cum ar fi **SQLPrepare()** şi, respectiv, **SQLExecute()**, care acceptă comenzi SQL ca argument. Aceasta înseamnă că nu mai este nevoie de comenzi statice în aplicaţii care să se introducă cu comanda EXEC. De exemplu, să luăm o comandă SQL dinamică care trebuie pregătită cu ajuorul API-ului SQLPrepare(). Trebuie să ne amintim că, prin folosirea codului SQL încorporat, obţinem acelaşi lucru ca atunci când folosim comanda EXEC SQL PREPARE.

```
SQLCHAR *stmt = (SQLCHAR *)"UPDATE employee.details SET emp_id = ? WHERE
emp_name = ? ";

/* pregatirea comenzii */
int rc = SQLPrepare(hstmt, stmt, SQL_NTS);
```

IBM CLI mai oferă şi alte interfeţe, cum ar fi SQLConnect(), SQLFetch(), SQLExecute ş.a.m.d.

Specificaţiile ODBC mai presupun şi existenţa unui mediu de operare, în care driverele ODBC sunt încărcate în mod dinamic la execuţie de către un manager de drivere în funcţie de sursa de date (numele bazei de date) introdusă în cererea de conectare. Driverul IBM DB2 Call Level Interface corespunde standardelor ODBC 3.51, ceea ce înseamnă faptul că acţionează ca un diver ODBC atunci când este încărcat de un manager de drivere.

Figura 7.4 prezintă locul în care este nevoie de folosirea driverului IBM DB2 CLI în cadrul mediului de dezvoltare a aplicaţiilor ce încorporează cod SQL dinamic. De asemenea se observă faptul că driverul IBM DB2 CLI se poate comporta la fel ca orice alt driver ODBC atunci când este încărcat prin intermediul managerului de drivere ODBC.

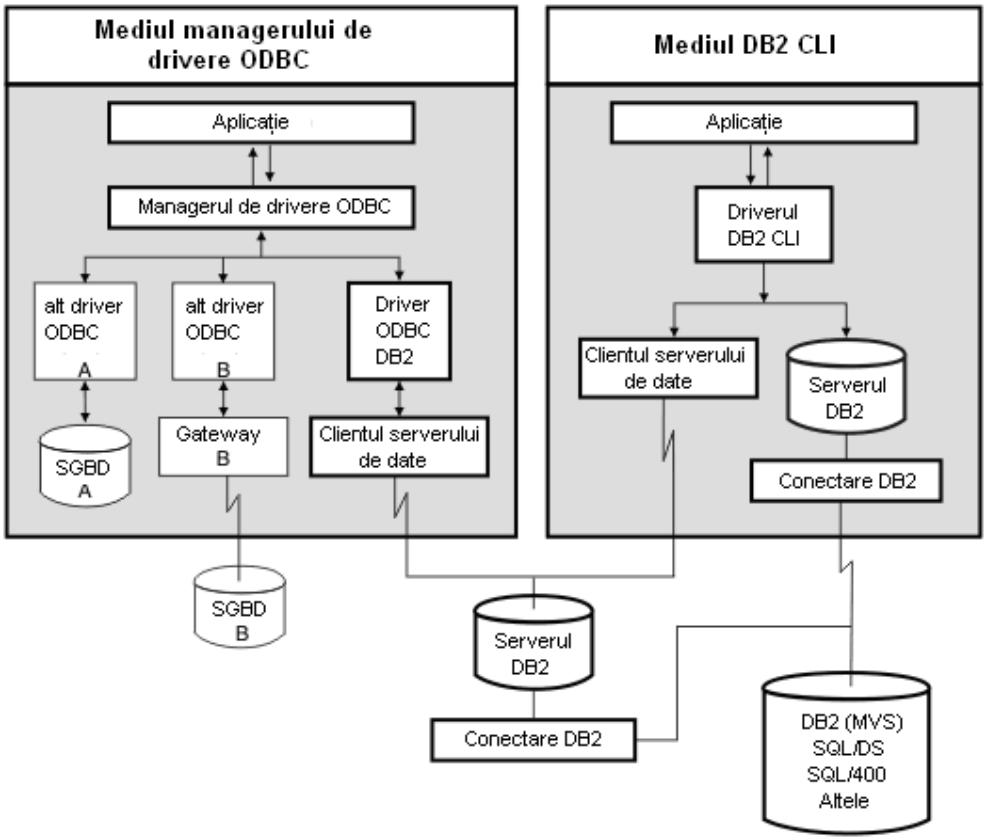


Figura 7.4 - IBM DB2 CLI şi ODBC

7.4.2 JDBC

JDBC înseamnă Java Database Connectivity. Aşa cum îi arată şi numele, este o interfaţă de programare a aplicaţiilor SQL similară cu ODBC şi CLI, dar folosită pentru aplicaţiile Java.

La eleborarea unei aplicaţii CLI, bibliotecile dependente CLI trebuie asociate aplicaţiei. În mod asemănător, în JDBC, pachetele relevante Java care conţin suportul pentru API-urile Java trebuie importate.

Un exemplu de comandă SQL ce poate fi apelată din cadrul unei aplicaţii JDBC este prezentat în *Lista 7.4*.

```
// SQL pentru SELECT. Informatiile despre nume, tara, strada si provincie
// ce sunt oferite de utilizator sunt păstrate in variabilele
// corespunzatoare.
String sqlSel = "select "+name+", " +country+", "+street+", "+province+",
"+zip+" from CUSTOMER where Customer = ?";

//pregatirea comenzii SELECT
PreparedStatement pstmt=con.prepareStatement(sqlSel);
pstmt.setString (1, "custCountry");         //alege parametrul the Input
pstmt.execute();                            //executa comanda SELECT
ResultSet result = pstmt.getResultSet ();  //obtine rezultatul si seteaza
                                            //valorile

List<Customer> custList = new ArrayList<Customer>();

while (result.next ()) {
        Customer cust = new Customer();
        cust.name = result.getString (1);
        cust.country = result.getString (2);
        cust.street = result.getString (3);
        cust.province = result.getString (4);
        cust.zip = result.getString (5);
        custList.add (cust);
   }
}catch (SQLException e) {e.pringStackTrace ();}
```

Lista 7.4 – Fragment de cod cu folosirea JDBC

În fragmentul de cod de mai sus:

- Comanda completă SQL, odată alcătuită, este mai întâi introdusă în cadrul unei variabile de tip şir de caractere (sqlSel de mai sus).
- Comanda dinamică SQL este apoi pregătită folosind **marcatori pentru parametru** în locul valorilor predicatului.
- Înainte de execuţie, valorile curente sunt **asociate** marcatorilor corespunzători parametrilor. Comanda JDBC "`pstmt.setString (1, "custCountry")`" va înlocui **primul** marcator de parametru în comanda SQL dinamică cu valoarea 'custCountry'.
- După execuţie, programatorii trebuie să mapeze rezultatele returnate de JDBC pe obiectele Java corespunzătoare.

7.5 pureQuery

Comenzile SQL dinamice oferă flexibilitate programatorilor, dar au nevoie de activităţi suplimentare. Pentru fiecare comandă SQL care se execută, comanda trebuie mai întâi pregătită în cadrul aplicaţiei, după care valorile curente ale predicatelor trebuie asociate

marcatorilor de parametri asociaţi, iar rezultatele primite de la driverul JDBC se mapează cu obiectele Java corespunzătoare.

pureQuery este o platformă oferită programatorilor Java pentru a exploata avantajele comenzilor SQL dinamice fără a fi preocupaţi de activităţile suplimentare de pregătire şi mapare.

Să luăm fragmentul de aplicaţie JDBC prezentat în *Lista 7.4*. Aceeaşi comandă SQL SELECT, atunci când este scrisp folosind pureQuery, are mai puţine linii de cod, aşa cum se vede din *Lista 7.5*.

```
//Numele, tara, strada şi provincia sunt variabile ce se vor
//popula la executie folosind datele introduse de utilizator.
String sqlSel = "select "+name+", " +country+", "+street+", "+province+",
"+zip+" from CUSTOMER where Customer = ?";

Data data = DataFactory.getData (con);

//se executa comanda Select si se obtine lista clientilor
List<Customer> customerList = data.queryList (sqlSel, Customer.
class, "custCountry");
```

Lista 7.5 – Fragment de cod cu folosirea pureQuery

API-ul pureQuery `queryList` va executa comanda `sqlSel` ce are valoarea predicatului "`custCountry`" returnând rezultatul interogării în `cutomerList`. La fel ca şi în JDBC, comanda SQL se generează tot la execuţie, deşi codul scris are mai puţine linii de cod în comparaţie cu aplicaţia JDBC. În acest fel, nu doar se maximizează viteza de elaborare a aplicaţiei, dar se obţine şi o reducere a complexităţii.

Metoda de mai sus cu folosirea pureQuery se numeşte **metoda inline,** care suportă executarea dinamică a codului SQL. pureQuery suportă şi execuţia statică a aplicaţiilor SQL folosind **metoda adnotării**.

Folosind metoda inline, comenzile SQL sunt elaborate sub forma unor obiecte Java de tip şir de caractere şi transmise API-ului pureQuery API. Pe de altă parte, folosind metoda adnotării, şirul SQL se defineşte ca fiind o **adnotare** pureQuery. Metoda adnotărilor folosită de pureQuery este următoarea:

- @Select (care marchează interogările SQL)
- @Update (care marchează comenzile DML SQL)
- @Call (care marchează comenzile CALL SQL).

Să luăm următoarea comandă SELECT:

```
SELECT Name, Country, Street, Province, Zip FROM customer where
Customer =?
```

În cazul pureQuery, tot ceea ce trebuie făcit este să:

- Se introducă codul SQL SELECT în marcajul corespunzător.

- Se declare o funcţie definită de către utilizator care este folosită pentru a executa acelaşi cod SQL.

De exemplu, în *Lista 7.6*, comanda SQL SELECT este plasată în marcajul @Select.

```
public interface CustomerData
{
//Selecteaza PDQ_SC.CUSTOMER pe baza parametrilor şi populează componenta
//Customer cu rezultatele
@Select(sql="select Name, Country, Street, Province,Zip from CUSTOMER
where Customer =?")
Customer getCustomer(int cid);
}
```

Lista 7.6 – Comentariul @Select

Pentru a executa acum comanda SQL, aplicaţia nu mai are nevoie să implementeze interfaţa de mai sus **`CustomerData`**. pureQuery implementează aceste interfeţe definite de către utilizator prin utilizarea unui utilitar intern numit generator pureQuery, care creează stratul de acces la date necesar aplicaţiei.

Aplicaţia poate creea în mod direct o varaibilă pentru **`CustomerData`** folosind API-ul pureQuery API apelând în mod direct metoda **`getCustomer()`** pentru a executa comanda SQL de mai sus, aşa cum se vede în *Lista 7.7*.

```
// se foloseste DataFactory pentru a instatia interfata definita de
// utilizator
CustomerData cd = DataFactory.getData(CustomerData.class, con);
// se executa codul SQL pentru getCustomer() si se obtin rezultatele
// pentru componentele Customer
Iterator<Customer> cust = cd.getCustomer();
```

Lista 7.7 – Execuţia comenzii SQL folosind pureQuery şi metoda adnotării

Rezultatul obţinut cu ajutorul utilitarului de generare este un fişier de implementare Java (**`CustomerDataImpl.java`** în exemplul de mai sus) al interfeţei definite de către utilizator (**`CustomerData)`**. Acest fişier de implementare Java conţine codul SQL curent precum şi definiţia metodelor declarate (`getCustomer`).

Folosind un astfel de stil de programare, programatorul aplicaţiei introduce toate comenzile SQL împreună cu metodele corespunzătoare în cadrul interfeţelor. Aceste metode sunt apoi folosite pentru a executa comenzile SQL în cadrul aplicaţiilor. În acest fel, comenzile SQL sunt **separate** de logica aplicaţiei în codul aplicaţiei.

Stilul de programare ce foloseşte metoda adnotării pureQuery permite modul de execuţie static al aplicaţiilor, în timp ce metoda inline foloseşte execuţia dinamică. În funcţie de cerinţele utilizatorului se poate folosi oricare dintre aceste stiluri de programare pureQuery pentru a elabora o aplicaţie Java cu baze de date.

Mai multe detalii referitoare la tehnicile de lucru folosind pureQuery găsiţi la adresele:

http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.datatools.javat

ool.runtime.overview.doc/topics/helpindex_pq_sdf.html

http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.datatools.javatool.runtime.overview.doc/topics/helpindex_pq_sdf.html

7.5.1 IBM pureQuery Client Optimizer

O foarte interesantă caracteristică suportată de către pureQuery, este pureQuery Client Optimizer. În secţiunile anterioare am discutat despre reducerea de performanţă evidenţiată în aplicaţiile SQL dinamice. pureQuery Client Optimizer oferă o tehnică de reducere a pierderilor de performanţă. Aplicaţia JDBC dinamică existentă poate fi optimizată pentru a rula anumite comenzi SQL static, fără a face nici o modificare codului aplicaţiei. Pentru folosirea pureQuery Client Optimizer, tebuie efectuaţi următorii paşi:

- Atunci când se execută mai întâi aplicaţia dinamică, pureQuery Client Optimizer preia comenzile SQL din aplicaţie într-un fişier pureQueryXml.
- Comenzile SQL din fişierul XML sunt apoi separate în pachete, cu ajutorul instrumentului în linie de comandă **Configure**.
- Folosind staticBinder, aceste pachete sunt apoi create în cadrul serverului bazei de date, făcându-se o asociere între acestea şi aplicaţie.
- În sfârşit, la executarea aplicaţiei, trebuie setat modul ‘static’, care permite ca anumite comenzi din SQL să se execute în mod static.
- În principiu, fiecare comandă SQL preluată din aplicaţie corespunde unei comenzi SQL din fişierul XML. În momentul în care se găseşte o astfel de corespondenţă, se parcurg liniile pachetului respectiv cu ajutorul fişierului XML şi, deoarece SQL este deja asociat unui anumit pachet existent în serverul bazei de date, comanda poate fi executată static. Fiecare comandă SQL care nu are o corespondenţă este executată în modul dinamic.

7.6 Rezumat

În acest capitol am discutat despre diverse tehnici de utilizare a limbajului SQL în cadrul aplicaţiilor. S-a început prin descrierea conceptului de "tranzacţie" după care a urmat descrierea modului în care comenzile SQL pot fi încorporate în cadrul aplicaţiilor.

Capitolul explică diferenţele dintre modurile static şi dinamic de execuţie, arătându-se faptul că modul static oferă o perfomanţă superioară faţă de modul dinamic de execuţie, în timp ce modul dinamic oferă o flexibilitate mult mai bună la programare precum şi abilitatea de a elabora şi executa aplicaţii fără a fi nevoie de pre-compilare şi conectare la baza de date. Alegerea între modul de lucru static şi cel dinamic se face în funcţie de cerinţele aplicaţiei şi de proiectarea acesteia, neexeistând reguli prin care unul să-l fie preferat celuilalt.

Aplicaţiile încorporate SQL şi SQLJ pot suporta ambele, atât cod SQL static cât şi dinamic. Totuşi, chiar dacă aplicaţiile SQL încorporate folosesc preponderent comenzi SQL dinamice, tot mai este nevoie de unele comenzi SQL statice, ceea ce înseamnă faptul că

aplicaţiile tot mai trebuie pre-compilate şi conectate la baza de date.

Un concept diferit de utilizare a API-ului bazei de date, cum ar fi ODBC, CLI şi JDBC introduce modul dinamic complet de elaborare a aplicaţiilor. Acest concept surmontează multe dintre problemele datorate încorporării codului SQL. Principalul avantaj este acela că programatorii pot scrie cod standard ce poate fi folosit cu orice sistem de gestiune al bazelor de date făcând foarte mici modificări.

La final, în capitol se discută despre folosirea tehnicii pureQuery, o tehnologie IBM ce permite fructificarea avantajelor oferite de către codul SQL dinamic fără a fi preocupaţi de activităţile suplimentare de pregătire sau de mapare a obiectelor.

7.7 Exerciţii

Proiectaţi în întregime o aplicaţie pentru sistemul de management al unei biblioteci care trebuie să efectueze următoarele activităţi:

A. Întroducerea cărţilor noi în baza de date.

B. Regăsirea cărţilor după numele autorului sau după titlu.

C. Rezervaţi în baza de date un spaţiu de păstrare a datei de retur a cărţilor împrumutate.

D. La o anumită dată se va afişa lista cărţilor împrumutate cu data la care trebuie returnate.

Încercaţi să stabiliţi care dintre interogări sunt mai potrivite pentru executarea dinamică şi care sunt mai potrivite pentru executarea statică.

7.8 Întrebări recapitulative

1. Execuţia statică a unei comenzi SQL presupune că planul de acces:
   A. Se va genera doar la momentul execuţiei aplicaţiei.
   B. Este generat la momentul pre-compilării sale.
   C. Ambele
   D. Nici una
2. Executarea dinamică a comenzii SQL:
   A. Nu necesită existenţa informaţiei complete despre sintaxa SQL de dinainte.
   B. Necesită informaţia completă despre sintaxa SQL în faza de pre-compilare.
   C. Ambele
   D. Nici una
3. Codul SQL încorporat în aplicaţiile C/C++:
   A. Nu necesită pre-compilarea DB2.

B. Necesită pre-compilarea DB2.

4. SQLJ este:
   A. O tehnică de încorporare a codului SQL în aplicaţiile Java.
   B. Un apel programabil SQL la nivel de interfaţă.
5. ODBC vine de la:
   A. Open Database Community
   B. Open Database Connectivity
   C. Open Source Database Community
   D. Open Database Connection.
   E. Nici una dintre cele enumerate
6. Care dintre următoarele suportă rularea atât în mod static cât şi dinamic?
   A. JDBC
   B. SQLJ
   C. DB2 CLI
   D. Toate cele enumerate
   E. Nici una dintre cele enumerate
7. Care dintre următoarele aplicaţii necesită la pre-compilare/compilare o conexiune la baza de date?
   A. JDBC
   B. SQLJ
   C. DB2 CLI
   D. Toate cele enumerate
   E. Nici una dintre cele enumerate
8. Marcatorii de parametri (‘?’) se folosesc ca înlocuitori pentru:
   A. Valorile predicatelor introduse la momentul execuţiei
   B. Nume de coloane, nume de tabele etc.
   C. Comanda SQL însăşi.
   D. Toate cele enumerate
   E. Nici una dintre cele enumerate
9. Tehnica pureQuery numită ‘annotated style’ suportă:
   A. Comenzi SQL dinamice

B. Comenzi SQL statice

C. Toate cele enumerate

D. Nici una dintre cele enumerate

10. pureQuery Client Optimizer:

A. Necesită modificarea codului aplicaţiilor JDBC existente astfel încât acestea să poată fi executate în modul static.

B. Nu necesită modificarea codului aplicaţiilor JDBC existente astfel încât acestea să poată fi executate în modul static.

8

Capitolul 8 – Limbaje de interogare pentru XML

În ziua de astăzi datele din lumea reală pot fi reprezentate cu ajutorul mai multor modele de date. Modelul tradiţional relaţional de reprezentare a datelor folosit pe scară largă de peste zece ani are nevoie de modificări pentru a putea lucra cu date obţinute din surse diverse. Uneori, datele nu pot fi reprezentate în mod structurat, aşa cum impune modelul relaţional. De fapt, majoritatea datelor cu care se lucrează astăzi se află fie în format semi-structurat fie în format nestructurat. Datele nestructurate pot fi poze sau imagini, motiv pentru care datele structurate au nevoie de metadate asociate acetora. Datele semi-structurate nu impun un format rigid, aşa cum sunt înregistrările din cadrul unui tabel, motiv pentru care sunt mult mai flexibile putând reprezenta diverse tipuri de obiecte din cadrul unui domeniu.

XML este o metodă foarte bună de reprezentare a datelor semi-structurate fiind folosită cu succes de ceva timp. XML a devenit de facto standardul folosit la schimbul de informaţii pe Internet. Datorită faptului că în ultimul timp din ce în ce mai multe tranzacţii se produc online pe Internet, a apărut necesitatea urmăririi tuturor acestor tranzacţii, dar şi a păstrării acestora pentru o utilizare ulterioară. Unele organizaţii păstrează aceste informaţii pentru audit şi cerinţe de compatibilitate, iar altele pe motive de creştere a performanţelor prin efectuarea de analize a datelor ce se păstrează. Acest lucru necesită o bază de date foarte puternică care oferă suport real pentru păstrarea eficientă şi interogarea unui volum mare de date în format XML. DB2 este un astfel de server de date care oferă suport nativ atât pentru datele relaţionale cât şi pentru cele în format XML, motiv pentru care este cunoscut sub denumrea de server hibrid de date.

8.1 Vedere de ansamblu asupra tehnologiei XML

Denumirea XML vine de la eXtensible Markup Language (limbaj extensibil de marcare). XML este un model de date ierarhic alcătuit din noduri de tipuri diferite legate între ele printr-o relaţie de tip părinte/copil. Un model de date XML poate fi reprezentat în format de tip text sau binar.

8.1.1 Elementele XML şi obiectele bazei de date

Un element XML este conceptul de bază al unui document XML. Fiecare document XML trebuie să aibă cel puţin un element XML, cunoscut şi sub numele de rădăcină sau elementul documentului. Acest element poate avea în continuare, oricâte atribute şi

elemente copil.

Aveţi mai jos un exemplu de element XML simplu

```
<name>I am an XML element</name>
```

Fiecare element XML are câte o etichetă de început şi una de sfârşit. În exemplul de mai sus <name> reprezintă eticheta de început, iar </name> reprezintă eticheta de sfârşit. Elementul mai are şi o valoare de tip text: “I am an XML element”.

Mai jos se prezintă un document XML care are mai multe elemente XML şi un atribut.

```
<employees>
      <employee id=”121”>
            <firstname>Jay</firstname>
            <lastname>Kumar</lastname>
            <job>Asst. manager</job>
            <doj>2002-12-12</doj>
      </employee>
</employees>
```

În exemplul de mai sus, <employees> reprezintă rădăcina documentului XML care are un element subordonat numit <employee>. Elementul <employee> are, la rândul său, elemente subordonate şi un atribut numit ***id*** cu valoarea ***121***. În DB2, întregul document XML poate fi păstrat sub forma unei singure valori într-o coloană a unui tabel. De exemplu, structura tabelului de mai jos:

```
department(id integer, deptdoc xml)
```

Coloana ***id*** păstrează valoarea de tip întreg ***id*** a fiecărui departament, în timp ce coloana ***deptdoc*** care este de tip XML, va păstra câte un document XML pe departament. Din acest motiv, întregul document XML este tratat ca şi cum ar fi o singură valoare/obiect în cadrul tabelului. În *Figura 8.1* puteţi urmări o reprezentare grafică a felului în care sunt păstrate datele XML în sistemele DB2.

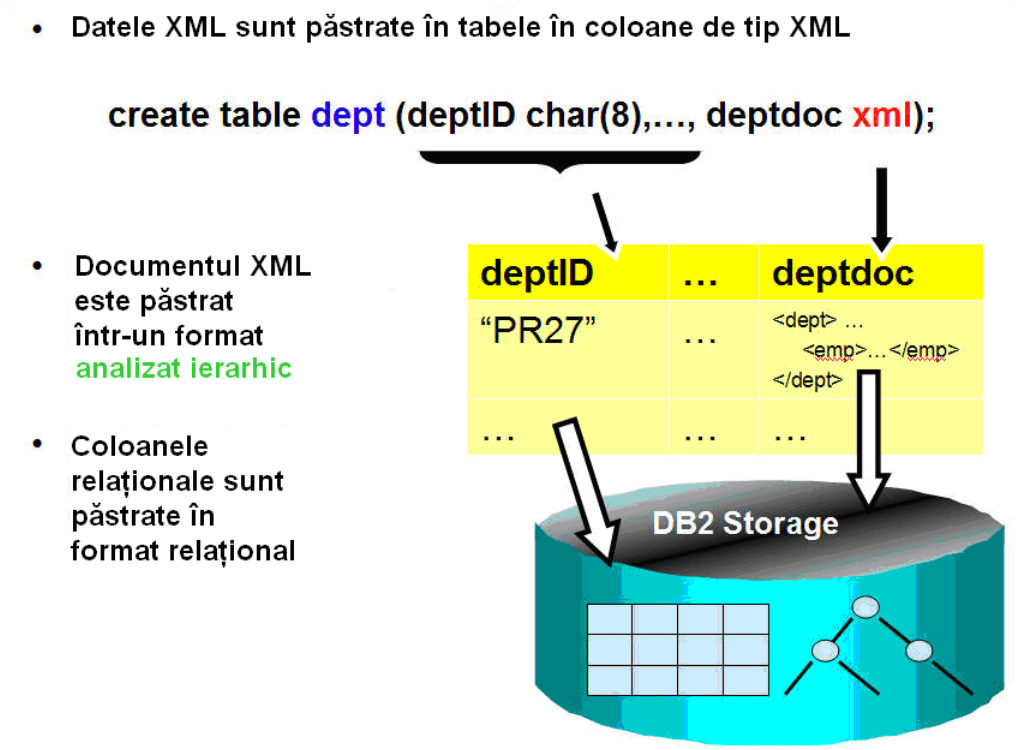


Figura 8.1 – Stocarea nativă în format XML

În DB2 documentul XML este analizat în momentul inserării, iar formatul ierarhic analizat este păstrat în cadrul bazei de date. Datorită acestui fapt nu este necesară analizarea documentului la momentul interogării, ceea ce conduce la o creştere a performanţelor de interogare.

8.1.2 Atributele XML

Atributele sunt întotdeauna părţi constitutive ale elementelor care oferă informaţii suplimentare despre acestea. Mai jos aveţi un exemplu de elemennt cu două atribute, ***id*** şi ***uid*** ce au valorile ***100-233-03*** şi respectiv ***45***:

```
<product id="100-233-03" uid="45"/>
```

De remarcat faptul că în cadrul unui element numele atributelor trebuie să fie unice, ceea ce înseamnă că acelaşi element nu poate avea două atribute cu acelaşi nume.

De exemplu, următoarea definiţie de element va genera o eroare în momentul analizei.

```
<product id="100-233-03" id="10023303"/>
```

De remarcat este şi faptul că valorile atributelor se află întotdeauna între ghilimele.

În schema unui document XML, atributele sunt definite după ce au fost definite toate celelalte elemente din cadrul unui element complex. Mai jos aveţi un exemplu de element

complex care are un singur atribut şi două elemente subordonate.

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
   <xs:attribute name=”empid” type=”xs:integer”/>
  </xs:complexType>
</xs:element>
```

8.1.3 Nume de domeniu

Numele de domeniu XML oferă un mecanism de marcare a unui atribut şi a unui nume de element pentru a evita conflictul de nume în cadrul unui document XML. De exemplu, dacă o companie de asigurări de sănătate primeşte informaţii despre asigurat de la o altă companie sub forma unui document XML, este foarte probabil ca două sau mai multe companii să aibă acelaşi nume de element, dar care să reprezinte lucruri diferite în formate diferite. Prin marcarea acestor elemente cu ajutorul unui nume de domeniu se rezolvă conflictul de nume. În XML, un nume poate fi marcat cu ajutorul unui nume de domeniu. Un nume marcat are două părţi:

- Un nume de domeniu de tip Uniform Resource Identifier (URI)
- Un nume local

De exemplu, ***http://www.acme.com/names*** este un nume de domeniu de tip URI, iar *customer* este numele local. De multe ori, numele marcate folosesc un prefix pentru numele de domeniu în locul unui nume de tip URI. De exemplu, elementul customer care aparţine domeniului URI ***http://www.acme.com/names*** mai poate fi scris şi sub forma *acme:customer,* în care acme este prefixul numelui de domeniu corespunzător ***http://www.acme.com/names*.** Prefixele numelor de domeniu pot fi declarate în antetul XQuery, aşa cum se vede mai jos.

Declare namespace *acme* "*http://www.acme.com/names*"

De asemenea, prefixul numelui de domeniu poate fi declarat în cadrul constructorului elementului, aşa cum se vede mai jos.

```
<book xmlns:acme="http://www.acme.com/names">
```

De remarcat este faptul că numele de domeniu aparţine aceluiaşi domeniu cu cel al elementului declarat, ceea ce înseamnă că toate elementele subordonate pot face referire la acest nume de domeniu.

Următoarele prefixe de nume de domeniu sunt predefinite şi nu trebuie folosite ca prefixe de domeniu definite de utilizator:

```
Xml, xs, xsi, fn, xdt.
```

Se mai pot declara, de asemenea, nume de domeniu implicite, adică un nume de domeniu fără prefix astfel:

```
declare default element namespace 'http://www.acme.org/names'
```

(în antetul XQuery)

```
<book xmlns="http://www.acme.com/names">
```

(în elementul constructor)

8.1.4 Document Type Definition

Document Type Definition (DTD) este o schemă de document XML cu ajutorul căreia se pot stabili structurile ce sunt permise a fi create în cadrul unui document XML. Cu ajutorul acestor scheme se stabileşte lista elementelor şi a atributelor ce pot apare în cadrul documentului pe care îl validează. Schema DTD poate fi declarată în interiorul unui document XML sau în exteriorul acestuia făcându-se o referire pentru verificare.

Mai jos puteţi vedea un exemplu de schemă DTD.

```
<!DOCTYPE TVSCHEDULE [

<!ELEMENT PRODUCTS (PRODUCT+)>
<!ELEMENT PRODUCT (NAME,PRICE,DESCRIPTION)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT PRICE (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>

<!ATTLIST PRODUCTS ID CDATA #REQUIRED>
]>
```

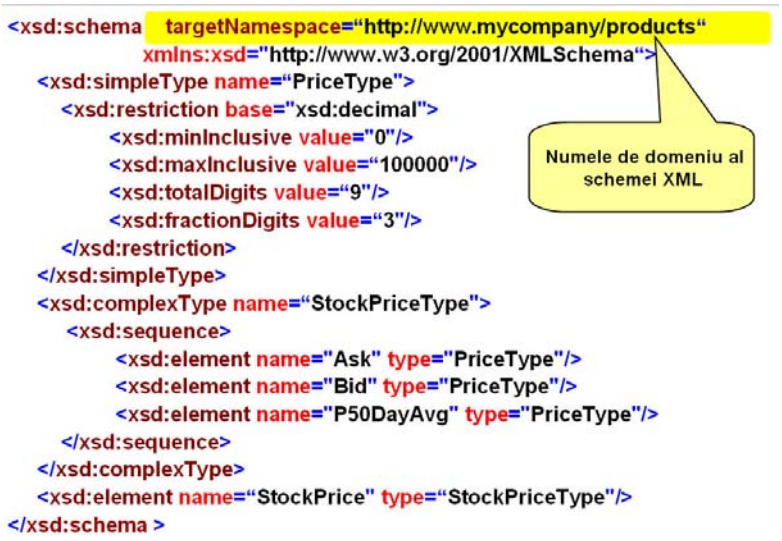
Ambele documente de mai jos sunt validate cu schema DTD de mai sus:

```
<PRODUCTS>
  <PRODUCT ID=”100-200-43”>
    <NAME>Laptop</NAME>
    <PRICE>699.99</PRICE>
    <DESCRIPTION>This is a Laptop with 15 inch wide screen, 4 GB RAM, 120
GB HDD </DESCRIPTION>
  </PRODUCT>
</PRODUCTS>
```

```
<PRODUCTS>
   <PRODUCT ID=”100-200-56”>
     <NAME>Printer</NAME>
     <PRICE>69.99</PRICE>
     <DESCRIPTION>This is a line printer </DESCRIPTION>
   </PRODUCT>
   <PRODUCT ID=”100-200-89”>
     <NAME>Laptop</NAME>
     <PRICE>699.99</PRICE>
     <DESCRIPTION>This is a Laptop with 13 inch wide screen, 4 GB RAM, 360
GB HDD </DESCRIPTION>
   </PRODUCT>
</PRODUCTS>
```

8.1.5 Schema XML

O schemă XML stabileşte structura, conţinutul şi tipurile de date acceptate în cadrul unui document XML. Aceasta poate fi alcătuită din unul sau mai multe documente de tip schemă. Un document de tip schemă poate avea un nume de domeniu.




```
<xsd:schema targetNamespace="http://www.mycompany/products"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:simpleType name="PriceType">
    <xsd:restriction base="xsd:decimal">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="100000"/>
      <xsd:totalDigits value="9"/>
      <xsd:fractionDigits value="3"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="StockPriceType">
    <xsd:sequence>
      <xsd:element name="Ask" type="PriceType"/>
      <xsd:element name="Bid" type="PriceType"/>
      <xsd:element name="P50DayAvg" type="PriceType"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="StockPrice" type="StockPriceType"/>
</xsd:schema >
```

Figura 8.2 - Schema XML: exemplu

O schemă XML este o variantă de schemă DTD. Schema XML descrie structura unui document XML. Limbajul folosit mai este denumit şi XML Schema Definition (XSD). Schemele XML sunt mult mai puternice decât cele DTD, deoarece acestea oferă un control mai bun asupra documentelor XML. Prin folosirea schemelor XML, în afara tipurilor de date obişnuite, cum ar fi integer, date, decimal şi datetime, se pot crea şi tipuri de date definite de utilizator, tipuri de elemente complexe etc. Se pot specifica lungimea, valorile minime sau maxime, tipurile de valori de şiruri de caractere permise sau enumerările. De asemenea, se mai pot specifica modurile de apariţie a elementelor în cadrul unui document

XML. Un alt avantaj al folosirii schemei XML faţă de utilizarea schemei de tip DTD este şi acela al posibilităţii acordării de nume de domeniu. Suplimentar, schema XML oferă suport pentru moştenirea tipului. Schema XML devine o recomandare a organizaţiei W3C începând cu luna mai a anului 2001.

În continuare se prezintă un exemplu de schemă XML alcătuită din trei documente de tip schemă şi două nume de domeniu.

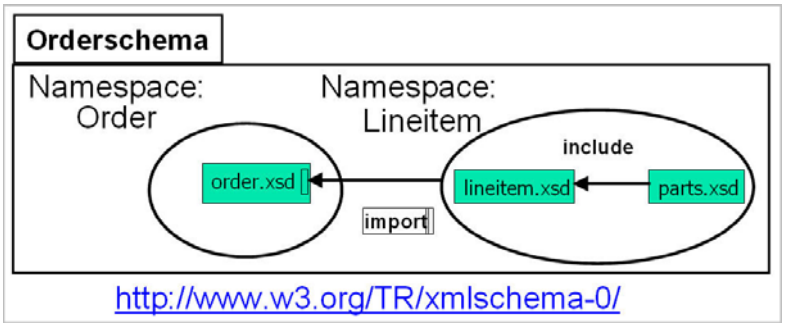


Figura 8.3 – Mai multe nume de domeniu în cadrul unei scheme XML

În DB2, utilizarea schemelor XML este opţională şi se aplică pe fiecare document XML. Aceasta înseamnă că se pot folosi acelaşi coloane pentru păstrarea ambelor tipuri de documente XML, atât pentru documente ce au scheme asociate, cât co pentru documente fără scheme asociate. Din acest motiv, nu este nevoie de o anumită schemă pe o coloană ce conţine documente XML. Validarea documentelor XML se face pe document (adică pe fiecare rând). Se pot folosi zero sau mai multe scheme pe fiecare coloană XML. De asemenea, se poate ca o singură coloană a unui tabel să conţină atât documente XML validate cât şi nevalidate. DB2 mai permite utilizatorilor să interzică inserarea de documente care nu sunt validate în cadrul coloanelor XML cu ajutorul clauzei 'IS VALIDATED' în cadrul unei comenzi SELECT.

8.2 XML Schema

XML reprezintă o aplicaţie a limbajului standard generalizat de marcare Standard Generalized Markup Language (SGML), fiind foarte puternic şi uşor de folosit. XML nu are limitări în ceea ce priveşte complexitatea structurală sau numele de domeniu. Fiind un metalimbaj, poate fi folosit la crearea altor limbaje specifice pieţei sau industriei. XML suportă o mare varietate de tipuri de date şi constrângeri de integritate, ce vor fi discutate în cele ce urmează.

8.2.1 Tipuri simple

Specificaţiile W3C XML Schema introduc o serie de tipuri simple de date, aşa cum se prezintă în *Figura 8.4*.

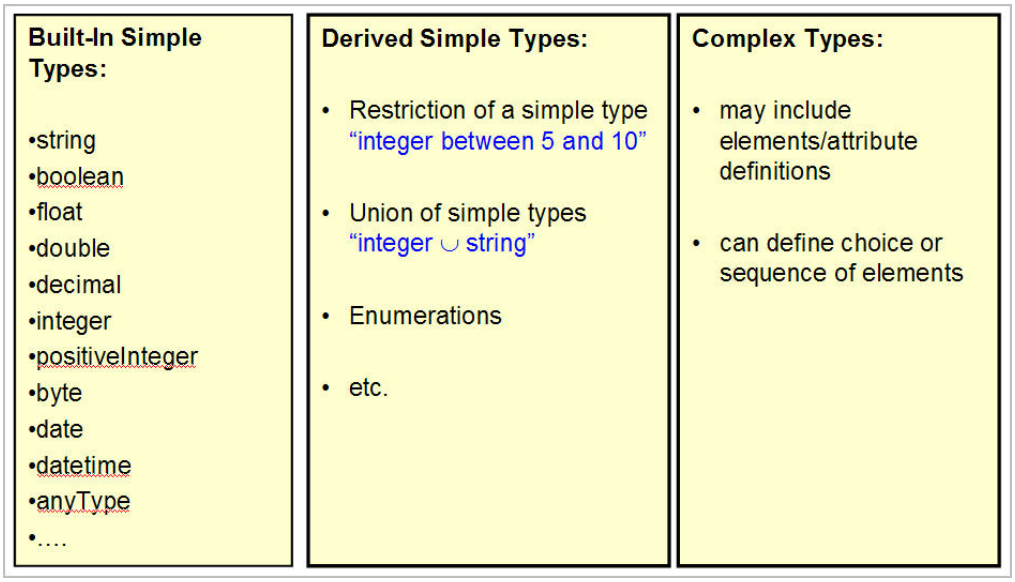


Figure 8.4 – Tipuri de date în XML Schema

În *Figura 8.4* sunt vizibile o serie de tipuri de date simple, dar există multe altele ce nu apar în figură şi care pot fi folosite pentru a defini elementele existente în documentele XML. În afara tipurilor de date simple se mai pot folosi şi tipuri de date definite de utilizator, create pe baza celor simple.

De exemplu, mai jos este prezentat un tip de dată definit de utilizator, 'myInteger', creat pe baza tipului simplu de date xs:integer şi care permite folosirea doar a valorilor din intervalul -2 la 5. Obţinerea unui tip de date definit de utilizator dintr-un tip de date simplu este numită derivaţie.

```
<xs:simpleType name= "myInteger" >
<xs:restriction base= "xs:integer" >
<xs:minInclusive value = "-2" />
<xs:maxExclusive value = "5" />
</xs:restriction>
</xs:simpleType>
```

Acest tip de derivaţie este denumit derivaţie prin restricţie.

Mai jos este prezentat un alt exemplu de derivaţie care este folosit pentru a face o enumeraţie:

```
<xs:simpleType name= "passGrades" >
<xs:restriction base= "xs:string" >
<xs:enumeration  value = "A" />
<xs:enumeration  value = "B" />
<xs:enumeration  value = "C" />
</xs:restriction>
</xs:simpleType>
```

Orice element de tip passGrades poate avea doar una dintre cele trei posibile valori (A, B sau C). Orice altă valoare luată de acest element va genera un mesaj de eroare din partea schemei XML.

De asemenea, se mai pot introduce şabloane de valori ce pot fi păstrate în cadrul unui element, aşa cum se prezintă în exemplul de mai jos:

```
<xs:simpleType name= "CapitalNames" >
<xs:restriction base= "xs:string" >
<xs:pattern  value = "([A-Z]( [a-z]*)?)+" />
</xs:restriction>
</xs:simpleType>
```

Alte două tipuri de derivaţie sunt derivaţiile de tip listă şi derivaţiile de tip reuniune. Mai jos este prezentat un exemplu de derivaţie de tip listă:

```
<xs:simpleType name= "myintegerList" >
<xs:list itemType= "xs:integer" />
</xs:simpleType>
```

Acest tip de dată poate fi folosit pentru a defini atribute sau elemente care acceptă spaţii libere între grupurile de numere întregi, cum ar fi de exemplu: "1 234 333 -32321".

În continuare este prezentat un exemplu de derivaţie prin reuniune.

```
<xs:simpleType name= "intordate" >
<xs:union memberTypes= "xs:integer xs:date" />
</xs:simpleType>
```

Acest tip de dată poate fi folosit pentru a defini atribute sau elemente care acceptă spaţiul liber ca marcator de separare între liste de numere întregi, ca în exemplul: “1 223 2001-10-26". De remarcat este faptul că în acest caz sunt alăturate tipuri de date diferite (tipul de dată întreg şi tipul de dată calendaristic) care alcătuiesc membrii listei.

8.2.2 Tipuri complexe

Tipurile complexe reprezintă o descriere a structurii de marcare care se bazează pe tipurile de date simple pentru elaborarea unei construcţii formate din elemente sau atribute individuale care alcătuiesc tipul de dată complex. Cu alte cuvinte, elementele unui tip de dată complex conţin alte elemente/atribute. Un element complex poat fi gol, sau poate conţine alte elemente, text, sau ambele, dar şi atribute. În continuare este prezentat un exemplu de tip complex de dată.

```
  <xs:complexType name=”employeeType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
```

Pentru a defini un element de tipul complex prezentat anterior, se foloseşte sintaxa:

```
<xs:element name="employee" type=“employeeType”>
```

O altă modalitate de creare a unui tip complex de dată este prezentată mai jos:

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Diferenţa dintre cele două, constă în faptul că definiţia primului tip complex de dată este independentă de element şi poate fi reutilizată la definirea altor elemente.

8.2.3 Constrângeri de integritate

Schemele XML permit diverse modalităţi de identificare şi asociere a diferitelor părţi de informaţie. Se pot, de exemplu, emula în mod direct tipurile de atribute ID şi IDREF pe baza schemei DTD, folosind tipurile de dată xs:ID sau xs:IDREF din schema XML. În schemele XML aceste două tipuri de date mai pot fi folosite şi pentru elemente, nu doar pentru atribute, aşa cum este cazul în schemele DTD. De asemenea se mai poate impune respectarea unicităţii elementelor prin folosirea tipului de element xs:unique, aşa cum se poate vedea din exemplul următor.

```
<xs:element name =”book” >
<xs:complexType>
….
</xs:complexType>
<xs:unique name=”book”>
<xs:selector xpath=”book”/>
<xs:field xpath=”isbn”/>
</xs:unique>
</xs:element>
```

Acest lucru permite obţinerea asigurării că există un singur ISBN pentru fiecare carte în cadrul documentului XML respectiv.

Se mai pot folosi şi tipurile de elemente xs:key şi xs:keyref pentru a asigura respectarea

constrângerilor de integritate. O cheie este o constrângere de unicitate cu restricţia suplimentară că toate nodurile corespondente trebuie să existe. Definiţia este asemănătoare cu definiţia unui element de tip unic:

```
<xs:element name =”book” >
<xs:complexType>
….
</xs:complexType>
<xs:key name=”book”>
<xs:selector xpath=”book”/>
<xs:field xpath=”isbn”/>
</xs:key>
</xs:element>
```

Pentru a face referire la această cheie se foloseşte xs:keyref. De remarcat este faptul că atributul de referinţă al elementului de tip xs:keyref trebuie să refere un element de tip xs:key sau xs:unique definit în cadrul aceluiaşi element sau al unuia dintre elementele părinte.

8.2.4 Evoluţia schemelor XML

Unul dintre motivele folosirii pe scară largă a tehnologiei XML este flexibilitatea acesteia la modelarea datelor, care se caracterizează prin efectuarea cu uşurinţă a modificărilor pentru a corespunde schemei. În zilele noastre, fiecare segment al industriei are o specificaţie proprie sub forma unui document de tip schemă XML. Aceste standarde sunt în permanentă modificare, fiind necesară respectarea compatibilităţii, motiv pentru care este necesară adaptarea rapidă la aceste modificări fără a afecta aplicaţiile. DB2 oferă o serie de posibilităţi de gestionare a modificărilor apărute în cadrul schemelor. De exemplu, în DB2 se pot păstra documente XML fără a fi nevoie de a păstra şi asocierea cu schemele corespunzătoare în cadrul aceleiaşi coloane. O altă situaţie poate fi aceea de a păstra documente XML compatibile cu diferite scheme XML în cadrul aceleiaşi coloane de tip XML. DB2 mai oferă şi o funcţie de verificare a compatibilităţii dintre două versiuni de scheme XML. Dacă le găseşte compatibile oferă opţiunea de a înlocui versiunea mai veche cu cea curentă.

Dacă documentele existente în baza de date trebuie validate cu ajutorul unei scheme de validare care urmează a fi modificată, există două modalităţi de face acest lucru în DB2 pureXML:

- Dacă cele două scheme sunt asemănătoare (*compatibile*), se poate înregistra schema nouă în cadrul XML Schema Repository (XSR), înlocuind schema veche şi continuând validarea. Ambele nume de scheme (numele SQL şi numele URI) rămân aceleaşi în cadrul ambelor scheme.
- În cazul în care cele două scheme XML nu sunt asemănătoare (*nu sunt compatibile)*, noua schemă se va înregistra cu un nume SQL nou şi un nume nou pentru URI.

Odată cu trecerea la noua schemă compatibilă, prin folosirea XMLVALIDATE, se poate continua referirea la noua schemă folosind numele SQL existent, sau numele URI cu condiţia ca numele URI să rămână neschimbat atât pentru documentele existente cât şi pentru cele noi. De obicei, trecerea la noua schemă compatibilă se face atunci când modificările petrecute în cadrul schemei sunt nesemnificative.

De exemplu, să urmărim situaţia în care apar o serie de modificări minore în cadrul unei scheme. Paşii ce trebuie urmaţi presupun înlocuirea schemei existente cu schema nouă, modificată, pentru a realiza cu succes operaţia în XSR:

1. Se apelează procedura stocată XSR_REGISTER sau se execută comanda REGISTER XMLSCHEMA pentru a înregistra schema cea nouă în XSR. De remarcat este faptul că nu trebuie validate documentele cu ajutorul noii scheme XML, dacă nu se urmăreşte decât înlocuirea vechii scheme cu una nouă, aşa cum se arată în pasul următor.
2. Se apelează procedura stocată XSR_UPDATE sau se execută comanda UPDATE XMLSCHEMA pentru a actualiza noua schemă XML în XSR prin înlocuirea schemei vechi.

Odată realizată înlocuirea, doar schema cea nouă mai poate fi folosită.

Dacă se foloseşte opţiunea ***dropnewschema*** în procedura stocată XSR_UPDATE sau în comanda XMLSCHEMA, atunci schema cea nouă poate fi folosită doar cu numele celei vechi şi nu cu numele folosit la înregistrare.

8.3 XPath

XPath 2.0 este un limbaj folosit la prelucrarea valorilor care corespund modelului de date XQuery/XPath Data Model (XDM). XPath foloseşte expresii de cale ce verifică calea ce trebuie parcursă pentru a naviga prin documentele XML. Acesta reprezintă un aspect foarte important atât în XSLT cât şi în XQuery şi este o recomandare W3C.

8.3.1 Modelul de date XPath

XDM are trei reprezentări ale documentelor XML. Valorile din XDM sunt secvenţe ce conţin zero sau mai multe obiecte ce pot fi:

- Valori atomice, cum ar fi cele de tip integer, string, sau Boolean
- Noduri XML cum ar fi documente, elemente, atribute sau text

Expresiile XQuery sau XPath lucrează cu instanţele XDM şi produc instanţe XDM. O secvenţă este o instanţă XDM ce conţine o colecţie ordonată de zero sau mai multe obiecte. Un obiect poate fi sau o valoare atomică, aşa cum s-a arătat mai sus, sau un nod. Valorile atomice sau nodurile pot fi introduse în orice ordine în cadrul unei secvenţe. Secvenţele nu pot fi imbricate. Dacă se combină două sau mai multe secvenţe, rezultatul obţinut este o secvenţă care conţine toate obiectele existente în secvenţele sursă.

De exemplu, introducerea unei secvenţe (<x/>, <y/>, 45) între două obiecte ale secvenţei

("beta", 2) conduce la obţinerea secvenţei ("beta", <x/>, <y/>, 45, 2). Notaţia utilizată în acest exemplu este consistentă faţă de sintaxa folosită pentru crearea secvenţei în XQuery. Întreaga secvenţă este introdusă între paranteze, iar obiectele se separă prin virgule.

8.3.2 Noduri de tip document

Un nod de tip document este un nod în cadrul unui document XML care marchează începutul documentului XML. Fiecare document XML trebuie să posede un nod de tip document, iar într-un document XML nu pot exista mai multe noduri de tip document. În DB2, în cazul unei secvenţe returnată prin intermediul unei comenzi XQuery sau XPath, aceasta trebuie tratată sub forma unui document XML, folosindu-se nodul constructor. *Figura 8.5* prezintă diverse tipuri de noduri ce pot alcătui un model de date XML.

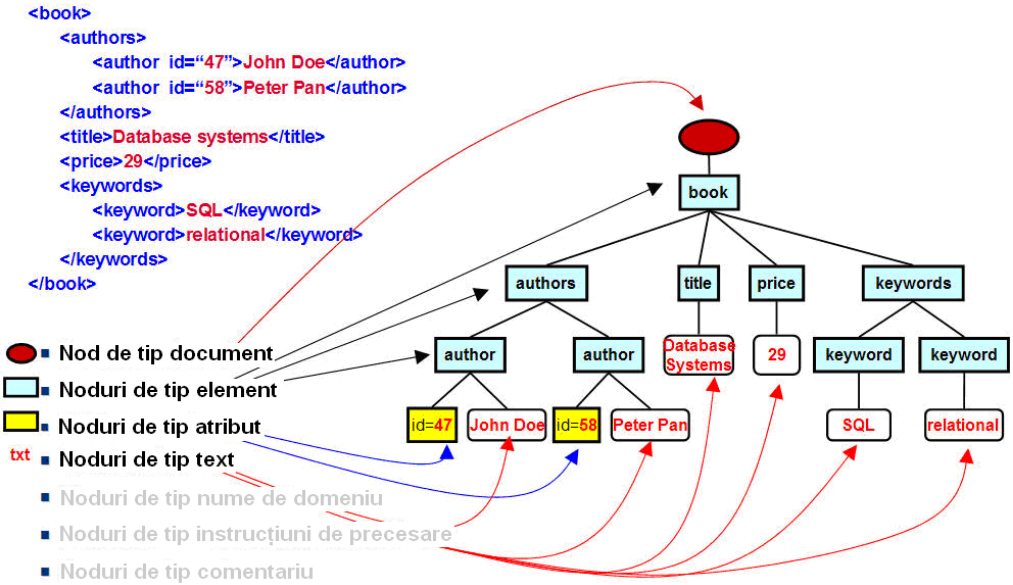


Figura 8.5 – Model de date XML Data Model: tipuri de noduri

De exemplu, următoarea comandă conţine o expresie de conţinut care returnează un document XML care conţine un element rădăcină numit customer-list:

```
document
{
<customer-list>
   {db2-fn:xmlcolumn('MYSCHEMA.CUSTOMER.INFO')/ns1:customerinfo/name}
</customer-list>
}
```

Într-un document XML alcătuit corect, primul nod este nodul rădăcină.

De exemplu, în documentul XML de mai jos, elementul <products> este un nod de tip

document. Nodul de tip document este, uneori, numit şi nodul rădăcină.

```
<products>
<product pid="100-201-01"><description>
<name>Ice Scraper, Windshield 4 inch</name>
<price>3.99</price></description>
</product>
</products>
```

8.3.3 Expresii de cale

Expresiile de cale se folosesc de obicei în XPath. Acestea sunt alcătuite din unul sau mai mulţi paşi separaţi prin intermediul unui slash (/) sau dublu slash (//). Fiecare pas produce o secvenţă de noduri care se folosesc sub forma unor noduri de tip conţinut pentru pasul următor. Valoarea unei expresii de cale este secvenţa obiectelor obţinute la ultimul pas al expresiei de cale.

Primul pas stabileşte punctul de pornire sau locaţia. De obicei, se foloseşte o funcţie sau o variabilă care întoarce secvenţa de noduri. Primul "/" înseamnă că acea cale începe de la nodul rădăcină. "//" înseamnă că acea cale începe cu o secvenţă alcătuită din nodul rădăcină şi toţi descendenţii săi.

8.3.4 Parcurgere avansată în XPath

Fiecare pas poate reprezenta o etapă de tip axă sau o expresie de filtrare. O etapă de tip axă este alcătuită din trei părţi:

- O axă opţională care specifică direcţia deplasării prin documentul XML sau prin fragmentul de document;
- Un nod de testare care stabileşte criteriile folosite la selecţia nodurilor; şi
- Zero sau mai multe predicate care filtrează secvenţa produsă în etapa respectivă.

Rezultatul etapei de tip axă este o secvenţă de noduri, fiecare nod având ataşată o valoare ce corespunde poziţiei din cadrul secvenţei. Fiecare nod poate fi accesat prin intermediul poziţiei sale.

8.3.5 Semantica XPath

Tabelul de mai jos prezintă diversele axe suportate în DB2.

Axa	Descrierea	Direcţia
Proprie	Întoarce nodul de tip conţinut.	Spre înainte
Copil	Întoarce copilul nodului de tip conţinut	Spre înainte
Descendent	Întoarce descendenţii nodului de tip conţinut	Spre înainte

Descendent sau propriu	Întoarce nodul de tip conţinut împreună cu descendenţii săi	Spre înainte
Părinte	Întoarce părintele nodului de tip conţinut	Spre înapoi
Atribut	Întoarce atributele nodului de tip conţinut	Spre înainte

Un nod de testare reprezintă o condiţie ce trebuie să fie adevărată pentru fiecare nod care este selectat prin intermediul unei axe. Nodul de testare poate fi sau un test de nume sau un test de tip.

Un test de nume filtrează nodurile pe baza numelor lor. Este alcătuit din QName sau un caracter înlocuitor şi, atunci când este folosit, selectează nodurile (elemente sau atribute) care corespund QNames. QNames corespunde dacă QName expandat al nodului este egal cu QName expandat din testul de nume. Două QNames expandate sunt egale dacă aparţin aceluiaşi domeniu de nume, iar numele lor locale sunt identice.

Tabelul de mai jos arată toate testele de nume ce pot fi folosite.

Test	**Descriere**
QName	Corespunde tuturor nodurilor al căror QName este acelaşi cu QName specificat
NCName.*	Corespunde tuturor nodurilor al căror nume de domeniu URI este acelaşi cu numele de domeniu la care este asociat prefixul specificat.
*.NCName	Corespunde tuturor nodurilor al căror nume local este acelaşi cu NCName specificat
*	Corespunde tuturor nodurilor

Un test de tip al nodurilor de filtrare depinde de tipul acestora. Tabelul de mai jos arată toate testele de tip suportate în DB2.

Test	**Descriere**
node()	Corespunde oricărui tip de nod
text()	Corespunde oricărui nod de tip text
comment()	Corespunde oricărui nod de tip comentariu
processing-instruction()	Corespunde oricărui nod de tip instrucţiune de procesare
element()	Corespunde oricărui nod de tip element

attribute()	Corespunde oricărui nod de tip atribut
Document-node()	Corespunde oricărui nod de tip document

Există două sintaxe pentru paşii de tip axă: abreviată şi neabreviată. Sintaxa neabreviată este alcătuită dintr-un nume de axă şi un test al nodului care se separă prin două puncte (::). În sintaxa abreviată, axa este omisă prin folosirea de notaţii prescurtate.

Tabelul de mai jos arată sintaxa abreviată suportată în DB2.

Sintaxa abreviată	**Descriere**
Nu este specificată axa	child:: cu excepţia cazurilor în care testul de nod este de tip attribute(). În acest caz, se omite prescurtarea axei pentru attribute::
@	attribute::
//	/descedent-or-self::node() cu excepţia cazurilor în care apare la începutul expresiei de cale. În acest caz, paşii de tip axă seletează rădăcina arborelui plus toate nodurile descendente.
.	self::node()
..	Parent::node()

De exemplu, următorul tabel preizntă expresiile de cale echivalente

Expresia de cale	**Expresia de cale cu sintaxa abreviată**
/dept/emp/firstname/child::node()	/dept/emp/firstname
/dept/emp//firstname/parent::node()/@id	/dept/emp/firstname/../@id

8.3.6 Interogări XPath

În DB2 se pot scrie expresii XPath prin încorporarea acestora în XQuery.

De exemplu, următoarea interogare XQuery returnează numele (numele elementelor) tuturor produselor existente în coloana DESCRIPTION din tabelul PRODUCT.

```
XQuery db2-fn:xmlcolumn('PRODUCT.DESCRIPTION')/product/description/name
Execution result
1
--------------------------------------------------
<name>Snow Shovel, Basic 22 inch</name>
<name>Snow Shovel, Deluxe 24 inch</name>
<name>Snow Shovel, Super Deluxe 26 inch</name>
<name>Ice Scraper, Windshield 4 inch</name>
  4 record(s) selected.
```

Aşa cum se vede, ***xmlcolumn*** este o funcţie XQuery care preia un argument de tip şir de caractere de forma NUMESCHEMA.NUMETABEL.NUMECOLOANAXML. Dacă tabelul pe care se execută interogarea se află în schema implicită, atunci nu trebuie specificat decât NUMETABEL.NUMECOLOANAXML. Db2-fn reprezintă domeniul de nume de care aparţine funcţia xmlcolumn. Funcţia xmlcolumn returnează toate documentele XML existente în coloana specificată.

DB2 are şi o altă funcţie numită ***sqlquery*** care ajunge la acelaşi rezultat cu funcţia ***xmlcolumn***. Diferenţa dintre cele două funcţii constă în abilitatea funcţiei ***sqlquery*** de a folosi documente XML specifice spre deosebire de documentele XML returnate prin intermediul funcţiei ***xmlcolumn***. De exemplu, interogarea XQuery de mai jos întoarce doar numele acelor produse al căror pid este 100-201-01, în care pid este o coloană relaţională.

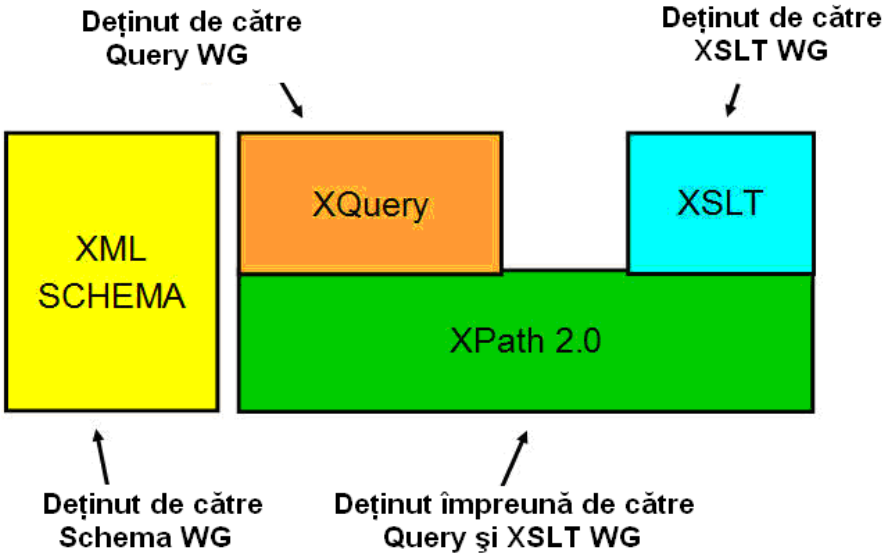
```
XQuery db2-fn:sqlquery("select DESCRIPTION from PRODUCT where pid= '100-
201-01'")/product/description/name

Execution result :
1
--------------------------------------------------
<name>Ice Scraper, Windshield 4 inch</name>

  1 record(s) selected.
```

8.4 XQuery

XQuery este un limbaj de interogare folosit în cadrul surselor de date XML. XQuery a fost elaborat de grupul de lucru al organizaţiei W3C şi se află astăzi în faza de recomandare. XQuery este folosit ca standard industrial pentru interogarea documentelor XML, oferind acces declarativ la datele din documentele XML la fel cum face limbajul SQL în cazul datelor relaţionale. *Figura 8.6* prezintă limbajul XQuery şi alte standarde XML asociate. Figura arată faptul că atât XPath cât şi XQuery folosesc ca bază XPath apelând la expresiile de cale XPath.



"Grupul de lucru W3C XML Query împreună cu grupul de lucru W3C XML Schema şi grupul de lucru W3C XSL au lucrat la un loc pentru a obține o serie de specificații care să lucreze în comun." "XQuery se foloseşte pentru a prelua date din mai multe baze de date, din fişiere XML, din documente Web, aflate la distanță, chiar din scripturi CGI şi produce rezultate în format XML ce pot fi procesate cu ajutorul XSLT."

www.w3.org/XML/Query/

Figura 8.6 - XQuery şi standardele asociate

8.4.1 Fundamentele limbajului XQuery

Fiecare interogare XQuery are două părţi:

- Prologul, care este opţional, este alcătuit din declaraţii ce stabilesc mediul de execuţie al interogării.
- Corpul interogării care este alcătuit dintr-o expresie ce pezintă rezultatul interogării.

Datele de intrare şi de ieşire ale unei interogări XQuery sunt întotdeauna instanţe XDM.

Mai jos este prezentat un exemplu de interogare XQuery, în care comanda “declare default element namespace” reprezintă prologul, iar corpul este alcătuit din expresii FLWOR. Expresiile FLWOR vor fi prezentate într-o secţiune ulterioară.

```
XQUERY
declare default element namespace "http://posample.org";
for $cust in db2-fn:xmlcolumn('CUSTOMER.DESCRIPTION')
return $cust/Name/LastName;
```

Corpul unei interogări XQuery poate fi alcătuit din oricare sau din toate expresiile ce urmează:

- Literale sau variabile

- Expresii Path
- Predicate
- Construcţii If ..then..else
- Constructori
- Operatori de comparare
- Expresii FLWOR

8.4.2 Expresii FLWOR

Expresiile FLWOR reprezintă partea esenţială a limbajului XQuery. FLWOR este o abreviere provenită din expresiile FOR LET WHERE ORDER BY şi RETURN ale XQuery.

De multe ori, expresiile FLWOR sunt comparate cu comenzile is SELECT FROM WHERE ORDER BY din limbajul SQL.

Mai jos este prezentată o expresie FLWOR simplă în XQuery.

```
XQuery
for $x in db2-fn:xmlcolumn('PRODUCT.DESCRIPTION')
let $p := $x/product/description/name
where $x/product/description/price < 3
return <cheap_products> {$p} </cheap_products>
```

Clauza “for” parcurge secvenţa şi asociază variabila obiectelor din cadrul secvenţei, câte unul odată. În cazul de faţă, $x este asociată secvenţei de documente XML existente în coloana DESCRIPTION. Pentru fiecare iteraţie a clauzei “for”, $x reţine câte un obiect (instanţa XDM) din grupul de documente XML corespunzător variabilei $p. De exemplu, dacă documentul XML este asociat variabilei $x se obţin patru nume de produse, după care toate numele produselor vor deveni parte a secvenţei şi vor fi asociate variabilei $p.

Clauza “where” este asemănătoare clauzei where din SQL şi filtrează grupul de rezultate pe baza unui anumit predicat. În exemplul de faţă, clauza where selectează doar acele produse al căror preţ este mai mic de “3”.

Clauza “return” arată instanţa XDM care se returnează. Instanţa XDM poate fi returnată imediat ce este extrasă cu ajutorul interogării sau printr-o combinaţie de elemente suplimentare existente în clauza return, aşa cum se poate vedea în exemplul de mai sus.

Mai jos sunt prezentate alte câteva exemple de expresii FLWOR:

```
1. XQuery for $i in (1 to 3) return $i
             SAU
2. XQuery for $x in db2-fn:xmlcolumn('PRODUCT.DESCRIPTION')//text()

Execution Result
1
-----
1
2
3
```

Interogarea de mai sus va returna toate nodurile de tip text din toate documentele existente în coloana DESCRIPTION a tabelului PRODUCT.

8.4.3 Joncţiuni în XQuery

Într-o interogare XQuery pot exista joncţiuni efectuate pe mai multe surse XML. Acest lucru este foarte util atunci când este nevoie să se interogheze datele XML din mai multe coloane XML. Mai jos este prezentat un exemplu de utilizare a joncţiunilor în XQuery.

```
for $book in db2-fn:xmlcolumn('BOOKS.DOC')/book
for $entry in db2-fn:xmlcolumn('REVIEWS.DOC')/entry
where $book/title = $entry/title
return <review>
{$entry/review/text()}
</review>;
```

În exempllul de faţă avem o joncţiune efectuată pe două coloane XML numite DOC din două tabele diferite, numite BOOKS şi REVIEWS. Coloana DOC din tabelul BOOKS păstrează toate informaţiile principale despre cărţi, cum ar fi titlul, autorul şi preţul cărţilor în format XML. Coloana DOC din tabelul REVIEWS păstrează informaţiile referitoare la recenzii, cum ar fi titlul cărţii şi recenzorii cărţii, împreună cu recenzia propriu-zisă. Interogarea XQuery de mai sus returnează recenziile (doar nodurile de tip text) acelor cărţi care există atât în tabelul BOOKS cât şi în tabelul REVIEWS.

8.4.4 Funcţii definite de utilizator

Potrivit specificaţiilor XQuery, se pot crea funcţii definite de utilizator în cadrul interogărilor XQuery ce pot fi invocate asemănător funcţiilor obişnuite pe perioada execuţiei interogării respective. DB2 are o serie de funcţii predefinite ca pot fi folosite în cadrul unei interogări XQuery. Acestea sunt funcţii de manipulare a şirurilor de caractere, a numerelor şi a datelor calendaristice. De asemenea, se mai pot folosi funcţii pentru a efectua operaţii pe secvenţe, cum ar fi afişarea inversă a secvenţei etc. De exemplu, următoarea interogare XQuery foloseşte funcţia "`contains`" din DB2:

```
XQuery
declare default element namespace "http://posample.org";
for $x in db2-fn:xmlcolumn('PRODUCT.DESCRIPTION')
let $p := $x/product/description/name
where $x/product/description/name/fn:contains(text(),'Scraper')
return $p

Execution result
1
--------------------------------------------------------------------
<name>Ice Scraper, Windshield 4 inch</name>
1 record(s) selected.
```

8.4.5 XQuery şi XML Schema

XQuery şi XML Schema sunt ambele standarde elaborate de către grupurile de lucru ale organizaţiei W3C. Specificaţiile XML Schema se referă la asocierile dintre tipul de date şi instanţele XDM.

DB2 are o serie de funcţii predefinite care transformă în mod explicit o serie de valori ale nodurilor în tipuri de scheme XML; de exemplu, interogarea XQuery de mai jos foloseşte funcţia predefinită xs:double.

```
XQuery
declare default element namespace "http://posample.org";
for $x in db2-fn:xmlcolumn('PRODUCT.DESCRIPTION')
let $p := $x/product/description/name
where $x/product/description/xs:double(price) = 3.99
return $p
```

8.4.6 Gruparea şi agregarea

XQuery suportă atât gruparea cât şi agregarea datelor XML.

Mai jos este un exemplu de grupare a datelor XML preluate din două coloane XML din două tabele diferite. Interogarea grupează numele clienţilor din tabelul CUSTOMER.

Clauza “`for`” efectuează iteraţia în documentele ce conţin informaţii despre clienţi şi asociază fiecare element numit city variabilei *$city*. Pentru fiecare element city, clauza “let” face o asociere între variabila *$cust-names* şi o listă neordonată a tuturor clienţilor din oraşul respectiv. Interogarea returnează elementele city care conţin numele oraşelor precum şi numele elementelor imbricate ale tuturor clienţilor din oraşul respectiv.

```
XQuery
for $city in fn:distinct-values(db2-fn:xmlcolumn('CUSTOMER.INFO')
    /customerinfo/addr/city)
let $cust-names := db2-fn:xmlcolumn('CUSTOMER.INFO')
    /customerinfo/name[../addr/city = $city]
order by $city
return <city>{$city, $cust-names} </city>
```

Interogarea întoarce rezultatul:

```
Execution result :
1
<city>Aurora
    <name>Robert Shoemaker</name>
</city>
<city>Markham
    <name>Kathy Smith</name>
    <name>Jim Noodle</name>
</city>
<city>Toronto
    <name>Kathy Smith</name>
    <name>Matt Foreman</name>
    <name>Larry Menard</name>
</city>
```

Mai jos este prezentat un exemplu de agregare în XQuery. În acest exemplu, interogarea face o iteraţie pe elementul PurchaseOrder din anul 2005 şi asociază elementul cu variabila $po în clauza “`for`”. Expresia de cale $po/item/ mută apoi poziţia conţinutului către fiecare element din cadrul elementului PurchaseOrder. Expresia (price * quantity) determină valoarea totală a acelui obiect. Funcţia fn:sum adaugă secvenţa rezultată valorii totale a fiecărui obiect. Clauza “let” asociază rezultatul funcţiei fn:sum variabilei $revenue. Clauza “order by” sortează apoi rezultatele pe baza veniturilor totale ale fiecărei comenzi de achiziţie.

```
for $po in db2-fn:xmlcolumn('PURCHASEORDER.PORDER')/
      PurchaseOrder[fn:starts-with(@OrderDate, "2005")]
    let $revenue := sum($po/item/(price * quantity))
    order by $revenue descending
    return
      <tr>
          <td>{string($po/@PoNum)}</td>
          <td>{string($po/@Status)}</td>
          <td>{$revenue}</td>
      </tr>
```

8.4.7 Cuantificarea

Expresiile de cuantificare întorc adevărat sau fals în funcţie de satisfacerea unei anumite condiţii de către unele sau toate obiectele din una sau mai multe secvenţe. De exemplu:

```
some $i in (1 to 10) satisfies $i mod 7 eq 0
every $i in (1 to 5) , $j in (6, 10) satisfies $i < $j
```

Expresiile de cuantificare încep cu un cuantificator: some or every. Cuantificatorul este urmat de una sau mai multe clauze care asociază variabile cu secvenţele returnate de către expresii. În primul exemplu, $i reprezintă variabila, iar (1 la 10) reprezintă secvenţa. În cel de-al doilea exemplu, avem două variabile $i şi $j care se asociază la (1 la 5) şi, respectiv, (6 la 10). În continuare se foloseşte o expresie de testare în care apar variabilele asociate. Expresia de testare se foloseşte pentru a determina dacă unele sau toate variabilele asociate satisfac o anumită condiţie. În primul exemplu, condiţia este $i mod 7 este egal cu 0. Calificatorul acestei expresii este “some”, existând o valoare pentru care expresia de testare este adevărată, astfel că rezultatul este adevărat. În cel de-al doilea exemplu, condiţia este $i este mai mic decât $j. Calificatorul este “every”, astfel că trebuie verificat dacă fiecare valoare a lui $i este mai mică decât fiecare valoare a lui $j. În acest caz rezultatul este adevărat.

8.5 XSLT

XSLT este acronimul pentru eXtensible Stylesheet Language Transformations, care este parte a standardului XSL şi care descrie felul în care se transformă (modifică) structura unui document XML într-o altă structură în cadrul altui document XML. Aceasta ajută la reprezentarea aceloraşi date în diverse formate. De exemplu, detaliile comenzilor de achiziţie păstrate sub forma unor documente XML pot fi transformate, folosind o serie stiluri, în diverse formate. Acelaşi document poate fi prezentat pe un site Web în format tabelar prin folosirea etichetelor HTML. *Figura 8.7* arată felul în care se foloseşte XSLT.

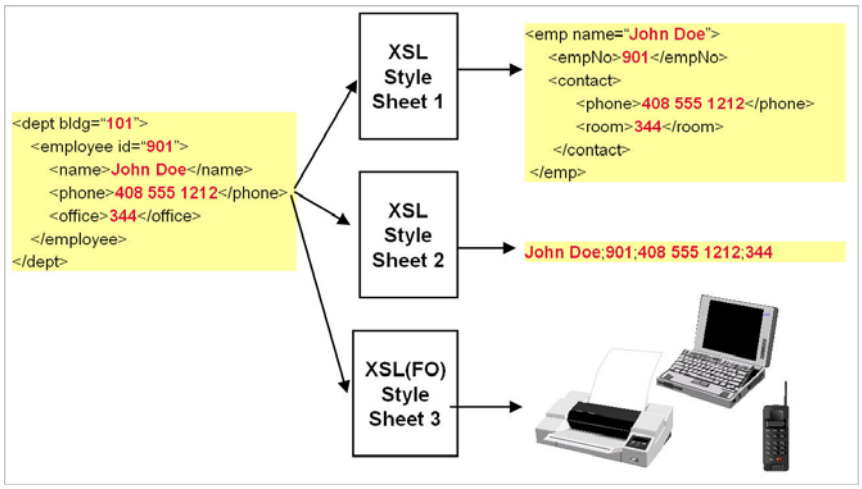


Figura 8.7 - XSL (XML): o singură sursă, mai multe formate

În DB2 9.7, pentru a efectua transformări XSLT sunt necesare două lucruri.

1. Documentul XML sursă
2. XSLT Stylesheet

XSLT Stylesheet reprezintă tot un document XML corect alcătuit, motiv pentru care poate fi păstrat într-o coloană XML a unui tabel din DB2. Mai jos se prezintă un exemplu de document XSLT Stylesheet.

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
    <h2>Product Details</h2>
    <table border="1">
      <tr >
        <th>Name</th>
        <th>Price</th>
      </tr>
      <xsl:for-each select="product/description">
        <tr>
          <td><xsl:value-of select="name"/></td>
          <td><xsl:value-of select="price"/></td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Exemplu de document sursă ce trebuie transformat:

```
<products>
     <product pid="100-201-01">
       <description>
         <name>Ice Scraper, Windshield 4 inch</name>
         <details>Basic Ice Scraper 4 inches wide, foam handle</details>
         <price>3.99</price>
       </description>
     </product>
    </products>
```

DB2 9.7 pune la dispoziţie funcţia ***XSLTransform*** care preia documentul XML sursă şi fişierul de stil XSLT ca date de intrare şi returnează rezultatul transformării (tot un

document XML).

De exemplu, următoarea comandă SELECT care foloseşte funcţia ***XSLTransform*** întoarce documentul HTML care poate fi salvat sub forma unui fişier .html

```
SELECT XSLTRANSFORM (description USING stylesheet AS CLOB (10M))  FROM
product where pid like '100-201-01'

Execution Result :
1
-------------------------------------------------------
<html>
<body>
<h2>Product Details</h2>
<table border="1">
<tr>
<th>Name</th><th>Price</th>
</tr>
</table>
</body>
</html>

1 record(s) selected.
```

8.6 SQL/XML

SQL/XML stabileşte un mecanism standard de utilizare a datelor XML împreună cu comenzi SQL. Este un standard ANSI/ISO care defineşte o serie de extensii bazate pe SQL (funcţii) care permit efectuarea de operaţii atât pe datele relaţionale cât şi pe documentele XML. De asemenea se mai introduce XML ca tip de dată. Aceste funcţii sunt tratate ca extensii ale comenzilor SQL. Se mai stabilesc şi funcţiile care analizează documentele XML şi care validează documentele XML în conformitate cu schema corespunzătoare. Există şi alte funcţii cum ar fi XMLCONCAT sau XMLAGG ce pot fi folosite pentru a aduna mai multe fragmente XML într-un singur document XML de dimensiuni mai mari.

8.6.1 Transformarea relaţiilor în documente XML

SQL/XML are funcţii de transformare a datelor relaţionale în format XML şi invers. Există funcţii de validare, analizare şi serializare a documentelor XML. Funcţiile de publicare sunt necesare atunci când este nevoie să se trimită date în format XML către o aplicaţie. De exemplu, dacă se invocă un serviciu Web care returnează cotaţia stocului ce are un anumit cod (sub forma unui document XML) atunci stocul, care este păstrat în tabelele relaţionale poate fi adus în format XML, după care serviciul Web poate trimite documentul XML către aplicaţia care are nevoie de acesta. În multe situaţii, s-ar putea să fie necesară folosirea echivalentului relaţional al unor fragmente XML către instrumente ce folosesc acest format pentru a-l folosi la prelucrarea ulterioară.

8.6.2 Păstrarea şi publicarea documentelor XML

Funcţiile de publicare sunt funcţii SQL/XML care se folosesc pentru a transforma datele relaţionale în format XML şi invers.

8.6.3 Funcţii SQL/XML

Să vedem câteva astfel de funcţii şi sintaxa acestora.

8.6.3.1 XMLELEMENT şi XMLATTRIBUTES

XMLELEMENT şi XMLATTRIBUTES sunt două dintre cele mai folosite funcţii de publicare. Aşa cum îi sugerează şi numele, funcţia XMLELEMENT creează un element XML pe baza unor date relaţionale. De exemplu, următoarea comandă SELECT produce rezultatele de mai jos.

```
Select XMLELEMENT(NAME "firstname", firstnme) from employee
Execution Result:
1
-----------------------------------------
<FIRSTNAME>CHRISTINE</FIRSTNAME>
<FIRSTNAME>MICHAEL</FIRSTNAME>
<FIRSTNAME>SALLY</FIRSTNAME>
<FIRSTNAME>JOHN</FIRSTNAME>
<FIRSTNAME>IRVING</FIRSTNAME>
<FIRSTNAME>EVA</FIRSTNAME>

6 record(s) selected.
```

În acest exemplu, NAME este un cuvânt cheie care arată că şirul ce urmează după acesta reprezintă numele unui element ce trebuie creat. Şirul arată că numele elementului este urmat de un nume de coloană, care este firstname, în acest exemplu. Funcţia XMLATTRIBUTES se utilizează pentru a crea atribute pe baza datelor relaţionale. De exemplu, comanda SELECT va produce rezultatul de mai jos.

```
Select XMLELEMENT(NAME "emp" , XMLATTRIBUTES(EMPNO AS "employee_num" ))
from employee
Execution Result :
1
-----------------------------------------------------
<EMP EMPLOYEE_NUM="000010"/>
<EMP EMPLOYEE_NUM="000020"/>
<EMP EMPLOYEE_NUM="000030"/>
<EMP EMPLOYEE_NUM="000050"/>
<EMP EMPLOYEE_NUM="000060"/>
6 record(s) selected.
```

Funcţia XMLATTRIBUTES preia atât numele atributului cât şi valoarea acestuia sub forma unui singur parametru în format A AS "B", în care A reprezintă valoarea atributului, care poate fi o coloană relaţională (empno) urmată de cuvântul cheie AS şi de un şir de caractere care arată numele atributului (employee_num). De remarcat este faptul că elementul EMP nu are nici o valoare de tip text.

Mai jos este prezentat un alt exemplu de utilizare a funcţiei XMLATTRIBUTES în care elementul EMP are două atribute (NUM şi SALARY). De asemenea mai are şi două elemente subordonate (FIRST şi LAST), care indică numele şi prenumele unui angajat.

```
Select
XMLELEMENT(NAME "emp" ,
 XMLATTRIBUTES(EMPNO AS "NUM",SALARY as "salary" ),
XMLELEMENT(NAME "first" , firstnme),
XMLELEMENT(NAME "last", lastname) )
 from employee
Execution Result :
1
-------------------------------------------------------------
<EMP NUM="000010"
SALARY="152750.00"><FIRST>CHRISTINE</FIRST><LAST>HAAS</LAST></EMP>
<EMP NUM="000020"
SALARY="94250.00"><FIRST>MICHAEL</FIRST><LAST>THOMPSON</LAST></EMP>
<EMP NUM="000030"
SALARY="98250.00"><FIRST>SALLY</FIRST><LAST>KWAN</LAST></EMP>
<EMP NUM="000050"
SALARY="80175.00"><FIRST>JOHN</FIRST><LAST>GEYER</LAST></EMP>
<EMP NUM="000060"
SALARY="72250.00"><FIRST>IRVING</FIRST><LAST>STERN</LAST></EMP>
```

8.6.3.2 XMLQUERY

Unul dintre avantajele folosirii SQL/XML doar cu interogări XQuery este acela că se pot extrage atât date relaţionale cât şi date în format XML din acelaşi tabel în acelaşi timp. XMLQUERY este o funcţie SQL/XML ce permite efectuarea acestei operaţii. Funcţia XMLQUERY preia o constantă de tip expresie XQuery, o expresie ce întoarce valori în format XML. De exemplu, următoarea comandă SELECT întoarce identificatorii şi numele tuturor produselor din tabelul PRODUCT.

```
SELECT pid , XMLQUERY('$DESCRIPTION/product/description/name'  ) AS
"PRODUCTNAME"    FROM product
Execution Result :
PID        PRODUCTNAME
------------------------------------------------------------
100-100-01 <name>Snow Shovel, Basic 22 inch</name>
100-101-01 <name>Snow Shovel, Deluxe 24 inch</name>
100-103-01 <name>Snow Shovel, Super Deluxe 26 inch</name>
100-201-01 <name>Ice Scraper, Windshield 4 inch</name>
  4 record(s) selected.
```

Numele produselor se găsesc în coloana DESCRIPTION care conţine documente XML ce descriu produsul. De remarcat este faptul că dacă sunt mai multe coloane DESCRIPTION (ca rezultat al joncţiunii dintre două sau mai multe tabele) atunci se poate folosi clauza PASSING aşa cum se vede mai jos.

```
SELECT pid , XMLQUERY('$D/product/description/name' PASSING
prod.description as "D"  ) AS "PRODUCTNAME"    FROM product prod,
purchaseorder po
```

Dacă documentele XML conţin nume de domenii, atunci interogarea trebuie rescrisă, aşa cum se vede în exemplul următor:

```
SELECT pid , XMLQUERY('$DESCRIPTION/*:product/*:description/*:name'  ) AS
"PRODUCTNAME"    FROM product
Execution Result :
PID        PRODUCTNAME
------------------------------------------------------------------------
100-100-01 <name xmlns="http://posample.org">Snow Shovel, Basic 22
inch</name>
100-101-01 <name xmlns="http://posample.org">Snow Shovel, Deluxe 24
inch</name>
100-103-01 <name xmlns="http://posample.org">Snow Shovel, Super Deluxe 26
inch</name>
100-201-01 <name xmlns="http://posample.org">Ice Scraper, Windshield 4
inch</name>
  4 record(s) selected.
```

În acest exemplu, funcţia XMLQUERY returnează o secvenţă XML din toate documentele XML din coloana respectivă. Dacă se doreşte să se extragă doar un anumit produs pe baza unui predicat, atunci trebuie folosită funcţia XMLEXISTS împreună cu XMLQUERY.

De exemplu, următoarea comandă SELECT va extrage numele produselor care au pid 100-103-01 şi preţul de 49.99

```
SELECT pid , XMLQUERY('$DESCRIPTION/*:product/*:description/*:name'  ) AS
"PRODUCTNAME"    FROM product
 where
 XMLEXISTS('$DESCRIPTION/*:product[@pid="100-103-
01"]/*:description[*:price="49.99"]')
Execution Result :
PID        PRODUCTNAME
--------------------------------------------------------------------
100-103-01 <name xmlns="http://posample.org">Snow Shovel, Super Deluxe 26
inch</name>
  1 record(s) selected.
```

Un aspect important de utilizare a funcţiilor XMLEXISTS şi XMLQUERY, pentru a corespunde condiţiilor cerute de predicat, este acela că se elimină secvenţele goale. Acestea sunt returnate de XMLQUERY ca rânduri ce nu corespund. XMLEXISTS va returna adevărat sau fals în funcţie de îndeplinirea condiţiei din predicat. Funcţia XMLQUERY va returna apoi o secvenţă pentru fiecare valoare corespondentă adevărată returnată de funcţia XMLEXISTS eliminând astfel secvenţele goale.

De exemplu, dacă comanda SELECT cu predicate este scrisă fără a folosi funcţia XMLEXISTS aşa cum se vede mai jos, rezultatul va crea confuzie. În comanda SELECT de mai jos, predicatul este parte a constantei de tip expresie XQuery introdus ca parametru al funcţiei XMLQUERY

```
SELECT pid , XMLQUERY('
$DESCRIPTION/*:product[@pid="100-103-
01"]/*:description[*:price="49.99"]/*:name')
AS "PRODUCTNAME"
 FROM product
Execution Result:
PID        PRODUCTNAME
--------------------------------------------------------------------
100-100-01
100-101-01
100-103-01 <name xmlns="http://posample.org">Snow Shovel, Super Deluxe 26
inch</name>
100-201-01
4 record(s) selected.
```

Motivul pentru care există 3 secvenţe goale este acela că XMLQUERY întoarce întotdeauna o secvenţă. Se va întoarce o secvenţă corespondentă dacă predicatul este îndeplinit şi o secvenţă goală dacă predicatul nu este îndeplinit. De aceea se recomandă folosirea funcţiei XMLEXISTS împreună cu funcţia XMLQUERY pentru a realiza corespondenţa.

8.6.3.3 XMLAGG

Funcţia XMLAGG întoarce o secvenţă ce conţine un obiect pentru fiecare valoare nenulă dintr-un set de valori XML. De exemplu, următoarea comandă SELECT va pune angajaţii

ce aparţin unui singur departament într-un singur grup:

```
SELECT
    XMLELEMENT (NAME "Department",
           XMLATTRIBUTES (e.workdept AS "name" ),
           XMLAGG ( XMLELEMENT (NAME "emp", e.firstnme)  )
              ) AS "dept_list"
FROM employee e
GROUP BY  e.workdept;
```

Rezultatul execuţiei va fi::

```
dept_list
--------------------------------------------------------------------
<Department name="A00">
<emp>CHRISTINE</emp>
<emp>VINCENZO</emp>
<emp>SEAN</emp>
<emp>GREG</emp>
</Department>
<Department name="B01">
<emp>MICHAEL</emp>
</Department>
<Department name="C01">
<emp>SALLY</emp>
<emp>DELORES</emp>
<emp>HEATHER</emp>
<emp>KIM</emp>
</Department>
<Department name="D21">
<emp>EVA</emp>
<emp>MARIA</emp>
</Department>
<Department name="E01">
<emp>JOHN</emp>
</Department>
```

8.7 Interogarea documentelor XML păstrate în tabele

În afara creării de documente XML pe baza datelor relaţionale folosind funcţii SQL/XML se mai pot folosi şi interogări XQuery pentru a interoga documentele păstrate în coloanele de tip XML. Dacă este nevoie să se folosească comanda SQL SELECT atunci se poate folosi funcţia XMLQUERY din SQL/XML căreia i se trece, sub forma unui parametru al acestei funcţii, interogarea XQuery.

Alegerea limbajului de interogare depinde de tipul de date ce trebuie extrase. Dacă sunt necesare doar date în format XML, atunci există posibilitatea utilizării XQuery sau a funcţiei

XMLQUERY, într-o comandă SELECT. Dacă trebuie extrase atât date relaţioanle cât şi date în format XML, atunci alegerea ideală este folosirea comenzilor SELECT împreună cu XMLQUERY.

8.8 Modificarea datelor

Păstrarea documentelor XML în interiorul unei baze de date este un aspect important al managementului datelor. Aceasta reprezintă o cerinţă obligatorie a zilelor noastre în scopul compatibilizării cu standardele existente şi de a răspunde rapid la cererile de modificare solicitate de către clienţi. DB2 nu numai că oferă o stocare eficientă a documentelor XML, păstrând formatul nativ al acestora, dar oferă şi modalităţi de manipulare a datelor XML stocate. Se oferă o serie de funcţii cum ar fi XMLPARSE şi XMLSERIALIZE, pentru a converti prezentarea sub formă de text a documentelor XML în format nativ XML şi invers.

8.8.1 XMLPARSE

Aşa cum s-a arătat mai sus, XMLPARSE este o funcţie care analizează un text XML transformându-l în format nativ XML (formatul ierahic de tip arbore). În mod implicit, la inserare, DB2 face o analiză a textului XML, dar dacă utilizatorul doreşte să analizeze în mod explicit textul XML, atunci trebuie folosită funcţia XMLPARSE.

De exemplu, următoarea comandă INSERT mai întâi analizează textul XML şi dacă textul este corect alcătuit, continuă cu introducerea documentului XML în coloana specificată din tabel.

```
INSERT INTO PRODUCT(PID,DESCRIPTION) VALUES('100-345-01', XMLPARSE(
DOCUMENT  '<product xmlns="http://posample.org" pid="100-100-01">
<description><name>Snow Shovel, Basic 22 inch</name>
<details>Basic Snow Shovel, 22 inches wide, straight handle with D-
Grip</details>
<price>9.99</price>
<weight>1 kg</weight>
</description>
</product> '))
```

Un alt motiv de utilizare a funcţiei XMLPARSE este acela că se permite specificarea modului de tratare a spaţiilor libere. Implicit, DB2 elimină spaţiile libere dintre elemente, dar dacă utilizatorul doreşte să le păstreze, atunci poate folosi clauza PRESERVE WHITESPACE aşa cum se arată mai jos.

```
INSERT INTO PRODUCT(PID,DESCRIPTION) VALUES('100-345-01', XMLPARSE(
DOCUMENT  '<product xmlns="http://posample.org" pid="100-100-01">
<description><name>Snow Shovel, Basic 22 inch</name>
<details>Basic Snow Shovel, 22 inches wide, straight handle with D-
Grip</details>
<price>9.99</price>
<weight>1 kg</weight>
</description>
</product> ' PRESERVE WHITESPACE))
```

8.8.2 XMLSERIALIZE

Aşa cum îi arată şi numele, funcţia XMLSERIALIZE se foloseşte pentru a serializa structura arborescentă XML în format şir de caractere/binar. Se permite ca arborele să fie serializat în formate de tip CHAR/CLOB/BLOB. Acest tip de serializare este necesară atunci când trebuie manipulat documentul XML în format text, deoarece orice dată returnată de către o interogare XQuery este în format XML.

De exemplu, următoarea comandă SELECT va parcurge documentele XML din coloana DESCRIPTION a tabelului PRODUCT şi le va returna sub forma unor obiecte serializate de tip CLOB ce aparţin produsului ce are pid ‘10010001’.

```
SELECT XMLSERIALIZE(DESCRIPTION AS CLOB(5K)) FROM PRODUCT WHERE PID LIKE
‘10010001’
```

Execution Result:

```
1
--------------------------------------------------------------------------
<product xmlns="http://posample.org" pid="100-100-01">
<description><name>Snow Shovel, Basic 22 inch</name>
<details>Basic Snow Shovel, 22 inches wide, straight handle with D-
Grip</details><price>9.99</price><weight>1 kg</weight>
</description>
</product>
1 record(s) selected.
```

8.8.3 Expresia TRANSFORM

Expresia TRANSFORM a fost introdusă în versiunea DB2 9.5 şi permite utilizatorului să modifice documentele XML existente în coloanele de tip XML. Expresia de transformare aparţine XQuery Facilitatea de actualizare permite următoarele operaţii ce pot fi efectuate într-o instanţă XDM.

- Inserarea unui nod
- Eliminarea unui nod
- Modificarea unui nod prin modificarea unor proprietăţi, dar cu păstrarea identităţii
- Crearea unei copii modificate a unui nod cu o nouă identitate.

Mai jos este prezentat un exemplu de utilizare a expresiei de transformare care actualizează un document XML al unui client cu identificatorul cid=1000. Expresia de transformare actualizează atât valoarea elementului phone number cât şi valoarea atributului ‘type’ la valoarea ‘home’.

Documentul XML înainte de comanda de actualizare:

```
<customerinfo>
        <name>John Smith</name>
        <addr country="Canada">
                <street>Fourth</street>
                <city>Calgary</city>
                <state>Alberta</state>
                <zipcode>M1T 2A9</zipcode>
        </addr>
        <phone type="work">963-289-4136</phone>
</customerinfo>
```

Comanda de actualizare cu expresia de transformare:

```
update customer
set info = xmlquery( 'copy $new := $INFO
                modify (
       do replace value of $new/customerinfo/phone with "416-123-4567",
       do replace value of $new/customerinfo/phone/@type with "home"    )
                return  $new')
where cid = 1000;
```

Documentul XML după actualizare:

```
<customerinfo>
        <name>John Smith</name>
        <addr country="Canada">
                <street>Fourth</street>
                <city>Calgary</city>
                <state>Alberta</state>
                <zipcode>M1T 2A9</zipcode>
        </addr>
        <phone type="home">416-123-4567</phone>
</customerinfo>
```

8.9 Rezumat

XML este un model de date foarte flexibil fiind foarte potrivit pentru un anumit gen de aplicaţii. Este o alegere ideală pentru aplicaţiile care necesită modificări, actualizări de scheme şi pentru obiectele care sunt prin natura lor de tip ierarhic. Posibilitatea de a reprezenta date semi-strucutrate face din XML o alegere bună pentru a face schimb de date precum şi pentru integrarea datelor atunci când acestea provin din surse diverse. DB2 oferă suport pentru păstrarea documentelor XML în forma lor nativă, gestionând eficient şi permiţând interogarea cu uşurinţă a documentelor XML. Se oferă utilizatorului posibilitatea alegerii limbajului pe care îl doreşte. Astfel se poate utiliza XQuery sau SQL/XML, în funcţie de datele accesate, dar şi de preferinţe. DB2 mai oferă avantajul flexibilităţii în ceea ce priveşte înmagazinarea schemelor şi a validării instanţelor de documente XML relativ la aceste scheme.

Caracteristica de transformare existentă în DB2 oferă o modalitate uşoară de a face modificări în cadrul documentelor XML fără a fi necesare modificări la nivelul aplicaţiei.

8.10 Exerciţii

1. Să se creeze un tabel cu o singură coloană relaţională (cid) şi o coloană de tip XML. Apoi să se insereze documente XML cu următoarea structură:

```
<customer id=”C62”>
  <firstname>Pete</firstname>
  <lastname>Bush</lastname>
  <address>
       <door>No 34</door>
       <building>Galaxy Apartment</building>
       <road>Meera road</road>
       <city>Mumbai</city>
       <zip>411202</zip>
  </address>
</customer>
```

2. Să se insereze datele provenite de la 5 clienţi sub forma unor documente XML folosind aceeaşi structură ca cea de mai sus, folosind comanda INSERT. Identificatorii cid ai celor 5 clienţi sunt respectiv, 10, 13, 15, 20, 23.
3. Să se efectueze o comandă SELECT pe un tabel pentru a extrage toate datele relaţionale şi XML.
4. Să se scrie interogarea XQuery necesară pentru a obţine toate documentele XML păstrate în tabel.
5. Să se scrie interogarea XQuery necesară pentru a extrage documentele XML selectate din cadrul coloanei XML al cărei identificator cid este cuprins între 10 şi 20

8.11 Întrebări recapitulative

1. Care dintre următoarele funcţii SQL/XML nu este o funcţie de publicare?
   A. XMLELEMENT
   B. XMLATTRIBUTE
   C. XMLCONCAT
   D. XMLPARSE
   E. Nici una dintre cele enumerate
2. Fie următoarea definiţie de tabel:

```
create table clients(
id           int primary key not null,
```

```
name         varchar(50),
status       varchar(10),
contactinfo  xml )
```

şi datele XML aflate în coloana contactinfo

```
<customerinfo>
       <name>Kathy Smith>
       <addr country=”Canada>
              <city>Toronto</city>
              <prov-state>Ontario</prov-state>
              <zip>M5H-4C9</zip>
       </addr>
</customerinfo>
<customerinfo>
       <name>Amit Singh>
       <addr country=”Canada>
              <city>Markham</city>
              <prov-state>Ontario</prov-state>
              <zip>N9C-3T6</zip>
       </addr>
</customerinfo>
```

Care este rezultatul următoarei interogări?

```
select xmlquery('$c/customerinfo/addr/city[1]'
passing info as "c") \
from xmlcustomer \
where xmlexists('$c/customerinfo/addr[prov-state="Ontario"]' \
passing xmlcustomer.info as "c")
```

A. Toronto

   Markham

B. <City>Toronto</City>

   <City>Markham</City>

C. <City>Toronto

   Markham</City>

D. Nici unul dintre cele enumerate

3. Analizarea unui document XML reprezintă un proces de conversie a:

   A. Datelor relaţionale din formatul relaţional în format ierarhic.

   B. Datelor XML din formatul şir de caractere serializat în forma ierarhic

C. Datelor XML din formatul ierarhic în formatul şir de caractere serializat

D. Datelor XML din formatul ierarhic în formatul relaţional

E. Nici una dintre cele enumerate

4. FLWOR vine de la:

A. For, Lower, Where, Or, Run

B. From, Let, Where, Or, Reset

C. For, Let, Where, Order by, Reset

D. For, Let, Where, Order by, Return

5. Când se foloseşte o interogare XQuery împreună cu o interogare SQL:

A. Trebuie lucrat cu atenţie deoarece XQuery este case sensitive, iar SQL nu.

B. Nu trebuie să vă îngrijoraţi deoarece ambele sunt case insensitive.

C. Trebuie lucrat cu atenţie deoarece SQL este case sensitive, iar XQuery nu.

D. Nu trebuie să faceţi griji.

E. Nici una dintre cele enumerate

6. Ce face următoarea interogare XQuery:

```
" xquery db2-fn:xmlcolumn('NNXML1.XMLCOL')/a/b "
```

A. Extrage elementul b, care este subordonat elementului a, din coloana XML numită XMLCOL a tabelului NNXML1

B. Extrage elementul a, care este subordonat rădăcinii b, din coloana XML numită XMLCOL a tabelului NNXML1

C. Extrage elementul b, care este subordonat rădăcinii a, din coloana XML numită NNXML1 a tabelului XMLCOL

D. Extrage elementul a in coloana XML numită XMLCOL a tabelului NNXML1

7. Dacă următorul tabel are un singur document XML în coloana DOC ca mai jos.

Descriere tabel: CONTEST (DOC XML)

```
<dept bldg="111">
    <employee id="901">
       <name>Ajit Patil</name>
       <phone>567 789 1342</phone>
       <office>124</office>
   </employee>
   <employee id="922">
       <name>Peter Jose</name>
       <phone>121 768 3456</phone>
```

```
            <office>213</office>
          </employee>
        </dept>
```

Care dintre următoarele interogări va returna numele elementului <name>Peter Jose</name>?

A. db2-fn:xmlcolumn('CONTEST.DOC')/dept/employee[@id="922"]/name

B. select xmlquery('$d/dept/employee[@id="922"]/name' passing DOC as "d") from contest

C. Ambele

D. Nici una dintre cele enumerate

8. Care dintre următoarele este echivalentă cu interogarea XQuery dată?

```
xquery db2-fn:xmlcolumn('CONTEST.DOC')/dept/employee[@id="922"]/name
```

A. xquery db2-fn:xmlcolumn(‘CONTEST.DOC’) /dept/employee/name[../@id=“922”]

B. xquery db2-fn:xmlcolumn(‘CONTEST.DOC’) /dept/employee[../@id=“922”]/name

C. xquery db2-fn:xmlcolumn(‘CONTEST.DOC’) /dept/employee/name[@id=“922”]

D. Nici una dintre cele enumerate

9. În DB2 9.7, care dintre următoarele este adevărată atunci când se foloseşte o comandă de actualizare, dacă expresia XPath ‘$new/customer/phone ‘ întoarce mai multe elemente phone?

```
update customer
set info =
 xmlquery('copy $new := $information
   modify do replace value of $new/customer/phone with "091-454-8654"
   return  $new')
where cid = 67;
```

A. Comanda UPDATE nu reuşeşte şi apare un mesaj de eroare

B. Comanda UPDATE va înlocui toate elementele phone cu noul element phone ce are valoarea de tip text "091-454-8654"

C. Comanda UPDATE înlocuieşte doar prima apariţie a elementului phone cu noul element phone ce are valoarea de tip text "091-454-8654"

D. Nici una dintre cele enumerate

9

Capitolul 9 – Securitatea bazelor de date

Odată cu dezvoltarea organizaţiilor ce au ca obiect de activitate tehnologia informaţiei, s-a acumulat un volum foarte mare de date din diferite domenii de activitate. Toate aceste date pot sta la baza unor decizii foarte importante, ceea ce înseamnă faptul că datele au devenit extrem valoroase pentru organizaţii, astfel încât este necesar să se acorde mare atenţie securităţii acestora. Din aceste motive, fiecare persoană din cadrul organizaţiei trebuie atenţionată şi responsabilizată referitor la breşele de securitate ce pot apare luându-se măsuri pentru a proteja datele din domeniul în care lucrează.

În acest capitol vom discuta despre necesitatea asigurării securităţii în bazele de date, despre conceptele de control al accesului şi de siguranţă în cadrul sistemelor de management al bazelor de date precum şi despre o serie de alte aspecte care privesc politicile şi procedurile de securitate.

9.1 Securitatea bazelor de date: vedere de ansamblu

De cele mai multe ori, problemele de securitate sunt complexe şi pot implica aspecte legale, sociale, sau etice, dar şi aspecte legate de politicile implementate sau referitoare la controlul echipamentelor fizice. Securitatea unei baze de date se referă la protecţia bazei de date împotriva pericolelor intenţionate sau nu, folosind elemente de control ce pot fi sau nu bazate pe echipamente de calcul.

Analiza securităţii unei baze de date nu include doar serviciile oferite de sistemul de gestiune al bazei de date, dar şi o serie de aspecte asociate bazei de date şi securităţii mediului.

Mai mult, aspectele legate de securitate nu se referă doar la datele existente în baza de date, deoarece breşele de securitate pot afecta şi alte părţi ale sistemului, care, ca rezultat, pot afecta baza de date.

În consecinţă, prin concentrarea atenţiei doar asupra securităţii bazei de date nu se va obţine o bază de date sigură. Toate componentele sistemului trebuie să fie sigure: baza de date, reţeaua, sistemul de operare, clădirea în care se află baza de date, dar şi persoanele care accesează sistemul.

Figura 9.1 prezintă vederea de ansamblu asupra securităţii unei baze de date.

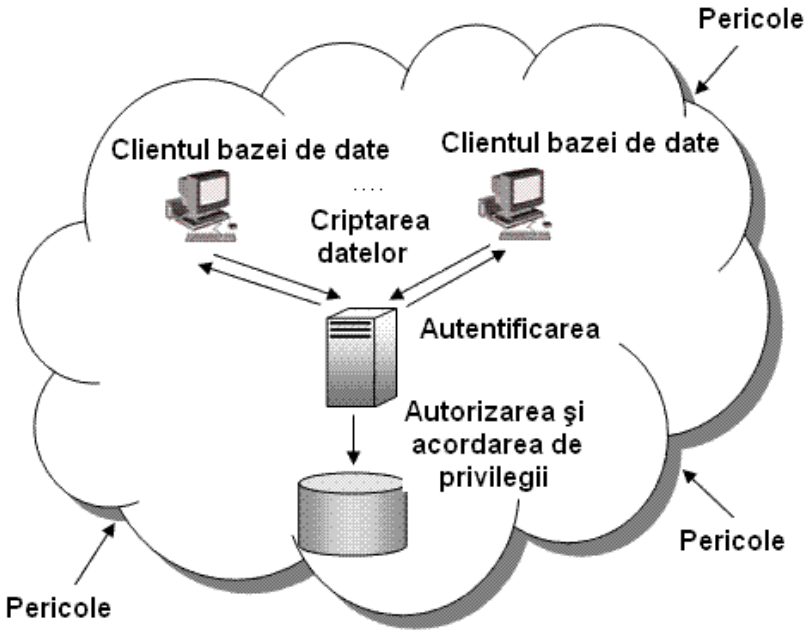


Figura 9.1 – Securitatea bazei de date

Proiectarea şi implementarea unei baze de date sigure implică atingerea următoarelor obiective:

- Caracterul privat, care înseamnă că datele nu pot fi cunoscute de persoane neautorizate;
- Integritatea, care înseamnă faptul că doar utilizatorii autorizaţi pot modifica datele;
- Disponibilitatea, care înseamnă faptul că utilizatorilor autorizaţi nu trebuie să li se interzică accesul;

Pentru a atinge aceste obiective, trebuie eleborată o politică de securitate în care să se facă referire la măsurile ce trebuie impuse. În particular, trebuie determinaţi utilizatorii care pot accesa baza de date precum şi datele la care aceştia au acces. Suplimentar, trebuie stabilite şi operaţiile permise pe setul de date.

Pentru a atinge aceste obiective se poate apela la mecanismul de securitate oferit de către sistemul de gestiune al bazei de date sau la cel oferit de către sistemul de operare. Orice alt mecanism exterior, cum ar fi securizarea accesului în clădirea în care se află baza de date, nu face obiectul discuţiilor din cadrul acestui capitol. Persoanele responsabile cu securitatea bazelor de date sunt numite administratori de baze de date (ABD) şi trebuie să aibă în vedere diversele pericole care pândesc sistemul.

ABD stabilesc regulile de autorizare prin care se stabilesc celelalte persoane care vor avea acces la baza de date, care parte a acesteia poate fi accesată de către ce utilizator, precum şi operaţiile permise acestora.

9.1.1 Necesitatea asigurării securităţii bazei de date

Motivul pentru care securitatea unei baze de date a devenit un aspect atât de important este acela al creşterii cantităţii şi importanţei cruciale a datelor care sunt colectate şi păstrate pe sistemele de calcul. Orice pierdere a disponibilităţii datelor sau a pierderii efective a acestora poate avea dimensiuni catastrofice. O bază de date reprezintă o resursă colectivă esenţială care trebuie asigurată în mod adecvat, folosind elementele de control cele mai potrivite.

Pericolele ce pot afecta securitatea datelor pot reprezenta orice situaţie sau eveniment intenţionate sau nu care ar putea afecta în mod negativ un sistem şi, la final, întreaga organizaţie. Distrugerile pot fi tangibile (pierderea datelor) sau intangibile (pierderea credibilităţii sau încrederii clientului). Orice pericol trebuie văzut ca o breşă potenţială în securitatea sistemului, care dacă reuşeşte, va afecta întreaga organizaţie.

Câteva exemple de pericole ce pot fi luate în considerare:

- Datele folosite de către un utilizator ce are acces la disc
- Colectarea sau copierea neautorizată a datelor
- Modificarea programelor
- Acces ilegal al unor persoane rău intenţionate
- Furtul datelor, programelor sau echipamentelor
- Instruirea inadecvată a personalului
- Prezentările neautoirizate de informaţii
- Calamităţi (foc, inundaţii, explozii)
- Ruperea sau deconectarea cablurilor
- Viruşi

Cât de mult suferă o organizaţie afectată de materializarea unor astfel de pericole depinde de o serie de factori, cum ar fi existenţa măsurilor de securitate şi a planurilor de măsuri pentru situaţii speciale.

De exemplu, dacă apare o defecţiune hardware care modifică capacitatea de stocare, toate activităţile de procesare trebuie întrerupte până la remedierea problemei. Perioada de inactivitate şi viteza de restaurare a bazei de date depind de posibilitatea utilizării de elemente alternative hardware şi software, de perioada de timp trecută de la realizarea ultimei copii de siguranţă, de timpul necesar pentru restaurarea sistemului, precum şi de faptul că datele pierdute nu pot fi restaurate şi recuperate.

Organizaţiile trebuie să identifice tipurile de pericole la care se expun şi să adopte planuri şi măsuri corespunzătoare care să aibă în vedere şi costul implementării acestora.

Pierderea unei cantităţi mari de timp, efort şi bani pentru rezolvarea unor pericole care ar putea aduce inconveniente minore poate conduce la ineficienţă, din punct de vedere economic. De fapt, fiecare afacere influenţează tipurile de pericole ce trebuie avute în

vedere, deoarece unele dintre acestea pot apare foarte rar. Totuşi, şi aceste pericole trebuie luate în considerare dacă au un impact semnificativ asupra organizaţiei.

Nu contează cât de sigur pare a fi un sistem de calcul, securitatea nu poate fi obţinută decât dacă şi mediul este sigur.

Următoarele pericole trebuie avute în vedere în orice plan complet de securitate a datelor:

- Furtul şi frauda. Aceste acţiuni pot fi întreprinse de către persoane şi pot sau nu să modifice datele. În acest caz, atenţia trebuie îndreptată către fiecare dintre posibilele locaţii în care datele sau aplicaţiile se află la un moment dat în mod fizic. Securitatea fizică concretă poate fi realizată astfel încât persoanele neautorizate să nu aibă posibilitatea să obţină accesul la încăperile în care se află calculatoarele, serverele sau fişierele acestora. Prin folosirea unui firewall se realizează protecţia la accesul neautorizat la părţi ale bazei de date prin stabilirea de conexiuni de comunicare.
- Pierderea caracterului privat sau al confidenţialităţii. Confidenţialitatea se referă la necesitatea de a păstra secretul datelor. Acest lucru prezintă un aspect major pemntru orice organizaţie, în timp ce caracterul privat are în vedere protecţia datelor cu caracter personal. Pierderea caracterului privat poate conduce la pierderea competitivităţii, iar pierderea controlului asupra caracterului privat poate conduce la şantaje, probleme sociale sau furt de parole. Unele dintre aceste aspecte pot conduce la probleme cu aspect juridic ale organizaţiei din care face parte persoana respectivă.
- Pierderea integrităţii datelor. Dacă integritatea datelor este compromisă, datele vor deveni invalide sau corupte. În acest caz, organizaţia va avea de suferit pierderi importante sau va lua decizii greşite pe baza detelor eronate.
- Pierderea disponibilităţii. Acest lucru înseamnă faptul că datele, sistemul, sau ambele, nu pot fi accesate. Uneori acest fenomen este acompaniat de modificarea datelor şi poate conduce la dificultăţi operaţionale severe. O astfel de situaţie poate apare ca rezultat al sabotajului, al pierderii conexiunii de reţea, al aplicaţiilor defectuoase, sau ca urmare a acţiunii viuşilor.
- Pierderea accidentală a datelor. Aceasta poate fi rezultatul unei erori umane, sau a breşelor din cadrul software-ului sau hardware-ului. Pentru a evita pierderea accidentală a datelor, organizaţia trebuie să stabilească proceduri clare de autorizare a utilizatorilor, a instalării software-urilor sau a mentenanţei hardware. La fel ca în toate cazurile în care este implicat personal uman, anumite pierderi sunt inevitabile, dar este de dorit ca prin folosirea de politici şi proceduri adecvate pierderile sa fie cât mai mici.

Securitatea bazei de date îşi propune să minimizeze pierderile cauzate de evenimentele amintite anterior într-o modalitate eficientă din punct de vedere al costurilor, fără a impune utilizatorilor constrângeri insuportabile. Deoarece criminalitatea pe calculator este în plină expansiune, iar acest tip de infracţiuni poate ameninţa toate componentele unui sistem, introducerea de măsuri de securitate adecvate devine vitală.

Cele mai folosite măsuri ce se pot lua pentru a asigura protecţia şi integritatea datelor sunt: controlul accesului, folosirea vederilor, controlul integrităţii şi criptarea. Este de asemenea necesar să se stabilească cele mai adecvate politici şi proceduri de securitate care se referă la personal şi la controlul fizic al accesului.

9.1.2 Controlul accesului

În practică, există două concepte importante referitoare la securitatea datelor. Cele două sunt cunoscute sub denumirea de ***control discreţionar*** şi ***control obligatoriu***. În ambele situaţii, unitatea de dată sau obiectul de dată care se protejează poate diferi de la întreaga bază de date la câteva rânduri. Folosind controlul discreţionar, utilizatorul va avea diferite drepturi de acces, cunoscute sub denumirea de privilegii acordate pe anumite obiecte. Evident, există o serie de limitări din punct de vedere al drepturilor pe care un utilizator le are asupra unui anumit obiect.

De exemplu, în cazul în care se foloseşte controlul discreţionar, un utilizator poate avea acces la obiectul X din baza de date, dar nu poate avea acces la obiectul Y, în timp ce utilizatorul B poate accesa obiectul Y, dar nu poate accesa obiectul X.

Schemele de control diiscreţionar sunt foarte flexibile. Se pot combina drepturi care se atribuie utilizatorilor şi obiectelor în funcţie de necesităţi.

În cazul controlului obligatoriu, fiecare obiect de dată este asociat unui anumit nivel de clasificare şi fiecare utilizator are un anumit nivel de permisiuni. Un anumit obiect de dată poate fi accesat doar de către utilizatorii care au permisiunile corespunzătoare. Schemele obligatorii sunt ierarhice, motiv pentru care sunt mult mai rigide decât cele discreţionare.

În funcţie de tipul de schemă de securitate care se foloseşte, toate deciziile referitoare la drepturile pe care le au diverşi utilizatori asupra obiectelor din baza de date sunt decizii ce depind de domeniul de care aparţin şi nu de decizii tehnice.

Pentru a decide care sunt constrângerile de securitate aplicabile unei cereri de acces, sistemul trebuie să poată recunoaşte sursa cererii. Cu alte cuvinte, trebuie să recunoască utilizatorul care a lansat în execuţie aplicaţia şi trebuie să verifice drepturile pe care la are acesta.

9.1.3 Studiu de caz referitor la securitatea unei baze de date

În cadrul acestei secţiuni vom analiza felul în care DB2, serverul principal de baze de date de la IBM, rezolvă aceste probleme de securitate folosind propriile mecanisme de securitate. Mecanismele de securitate de la DB2 au în vedere două operaţii principale: autorizarea şi autentificarea, care sunt folosite împreună pentru a asigura securitatea accesului la resursele DB2.

9.1.3.1. Autentificarea

Autentificarea este prima acţiune efectuată în vederea conectării cu succes la o bază de date DB2. Autentificarea reprezintă procesul prin care utilizatorii sunt identificaţi de către sistemul de gestiune al bazei de date, demonstrându-şi identitatea pentru a obţine accesul la baza de date.

Validarea identităţii utilizatorilor şi a grupurilor de utilizatori se obţine cu ajutorul unor facilităţi de securitate care se află în afara sistemului de gestiune a bazei de date, ceea ce înseamnă faptul că acestea acţionează ca parte a sistemului de operare sau prin utilizarea unor aplicaţii separate cum ar fi Kerberos sau Lightweight Directory Access Protocol (LDAP). Autentificarea unui utilizator necesită două elemente identificatorul utilizatorului şi simbolul de autentificare.

Identificatorul utilizatorului permite componentei de securitate să identifice utilizatorul şi, prin introducerea corectă a simbolului de autentificare (de obicei o parolă cunoscută doar de către utilizator şi de către componenta de securitate), să verifice identitatea acestuia. Uneori, dacă este necesară o flexibilitate mai mare, se pot crea module de securitate personalizate care să fie folosite de către DB2.

După autentificarea cu succes a unui utilizator, identificatorul utilizatorului se asociază unui identificator de autorizare. Această corespondenţă este determinată prin intermediul unui modul de autentificare. Dacă se foloseşte modulul de securitate implicit pus la dispoziţie de către IBM, atunci vom avea de a face cu doi identificatori de autorizare derivaţi: identificatorii de sistem şi de sesiune. În acest caz ambii identificatori de autorizare sunt derivaţi în acelaşi fel pe baza identificatorului utilizatorului şi sunt identici. Identificatorul de autorizare al sistemului se foloseşte la verificarea privilegiilor de conectare pentru a realiza conectarea. Identificatorul de autorizare al sesiunii este identificatorul primar al coenxiunii următoare.

9.1.3.2. Autorizare

După autentificarea utilizatorului, este necesar să se determine dacă utilizatorul respectiv are dreptul să utilizeze anumite date sau resurse. Autorizarea este procesul de acordare a privilegiilor care permit unui subiect să obţină accesul legitim la un sistem sau la un obiect al acestuia. Definiţia autorizării conţine termenii de subiect şi obiect. Subiectul se referă la un utilizator sau program, iar obiectul se adresează unui tabel, vedere, aplicaţie, procedură sau orice alt obiect ce poate fi creat în cadrul sistemului.

Controlul autorizării poate fi implementat cu ajutorul unor elemente software şi poate reglementa atât sistemele cât şi obiectele la care utilizatorul are acces precizând ce poate face utilizatorul cu acestea. Din acest motiv, autorizarea se mai numeşte şi controlul accesului. De exemplu, un utilizator poate fi autorizat să citească înregistrări din baza de date, dar nu poate modifica sau insera o înregistrare nouă.

Regulile de autorizare sunt elemente de control încorporate în cadrul SGBD care restricţionează acţiunile pe care un utilizator le poate întreprinde la accesarea datelor. Pentru a înregistra permisiunile asociate fiecărui utiliaztor în parte, în DB2 se folosesc tabele şi fişierele de configurare.

Atunci când un utilizator autentificat încearcă să acceseze date, numele de autorizare al acestuia precum şi setul de privilegii asociate, acordate direct sau indirect printr-un grup sau rol, sunt comparate cu permisiunile înregistrate. Rezultatul comparaţiei se folsoeşte pentru a stabili dacă accesul este permis sau respins.

Pentru un identificator de autorizare, există mai multe surse de permisiuni. Mai întâi, există

permisiuni acordate în mod direct identificatorului de autorizare. Apoi, există permisiuni acordate grupurilor şi/sau rolurilor al căror membru este utilizatorul. Permisiunile publice sunt acele permisiuni acordate grupuluii PUBLIC în timp ce permisiunile ce depind de context sunt cele acordate unui anumit rol sigur.

Pentru a putea efectua diverse activităţi, SGBD-ul cere fiecărui utilizator să fie autorizat în mod specific, implicit, sau explicit. Privilegiile explicite sunt acordate unui utilizator, unui grup sau unui rol, în timp ce privilegiile implicite sunt cele obţinute prin intermediul grupurilor de care aparţine utilizatorul sau prin intermediul rolurilor asociate.

DB2 lucrează cu trei forme de autorizări înregistrate: autoritatea administrativă, privilegii şi credenţiale Label-Based Access Control (LBAC – controlul accesului bazat pe etichete). Un utilizator, un grup, sau un rol poate avea unul sau mai multe dintre aceste tipuri de autorizări.

9.1.3.3 Autoritatea administrativă

Autoritatea administrativă conferă unei persoane dreptul de a controla baza de date şi de a răspounde pentru integritatea datelor. Autoritatea administrativă în DB2 este ierarhică. Pe nivelul cel mai înalt se află SYSADM. Sub SYSADM se află două tipuri de autorităţi: la nivel de instanţă şi la nivel de bază de date. Nivelele de autoritate oferă o metodă de grupare a diverselor privilegii.

Autoritatea pe instanţa DB2 se aplică tuturor bazelor de date din cadrul instanţei şi este acordată prin apartenenţa la grup. Numele grupului ce are acest nivel de autoritate se păstrează în baza de date, în fişierele de configurare ale managerului bazei de date corespunzătoare fiecărei instanţe.

La acest nivel, există patru tipuri de autorităţi de instanţă DB2:

- SYSADM (administrator de sistem) controlează toate resursele create şi păstrate de către managerul bazei de date. Utilizatorii care au autoritatea SYSADM pot efectua următoarele acţiuni: migrare de baze de date, modificarea fişierelor de configurare ale managerului baze de date şi ale bazei de date, efectuarea de copii de siguranţă ale bazei de date şi fişierelor jurnal şi restaurări ale bazei de date şi obiectelor acesteia, cum ar fi spaţii pentru tabele, acordarea de drepturi şi privilegii utilizatorilor, grupurilor sau rolurilor, control complet al instanţelor precum şi auditul controlului la nivel de instanţă.
- SYSCTRL (controlul sistemului) oferă controlul asupra operaţiilor care afectează resursele sistemului. Un astfel de utilizator poate crea, actualiza, porni, sau opri o bază de date. De asemenea, poate porni sau opri o instanţă, dar nu poate accesa date din tabele.
- SYSMAINT (mentenanţa sistemului) permite efectuarea de operaţii de mentenanţă pe toate bazele de date dintr-o instanţă. Aceste operaţii sunt cele de actualizare a configuraţiei bazei de date, de realizare a copiilor de siguranţă ale bazei de date sau ale spaţiilor pentru tabele, de restaurare a unei baze de date sau de monitorizare a acesteia. SYSMAINT nu are acces la date.

- SYSMON (monitorizarea sistemului) poate opera la nivel de instanţă. Mai concret, are autoritatea necesară pentru a utiliza monitorul sistemului de baze de date.

Autorităţile bazelor de date DB2 se leagă de o anumită bază de date din cadrul unei instaţe DB2. Acestea sunt:

- DBADM (administratorul bazei de date) se foloseşte la nivel de bază de date şi oferă autoritatea administrativă pe o singură bază de date. Administratorul bazei de date are toate privilegiile necesare pentru a crea obiecte, a executa comenzi, şi de a accesa toate datele. El poate, de asemenea, să acorde sau să retragă privilegii în mod individual Administratorul bazei de date poate crea fişiere de jurnalizare, tabelele catalog ale sistemului, actualizarea fişierelor de jurnalizare, reorganizarea tabelelor bazei de date sau obţinerea de statistici din cataloage.
- SECADM (administrator de securitate) poate crea, elimina, aproba sau retrage autorizări sau privilegii, cu transferul drepturilor de proprietate pe obiectele de securitate (de exemplu, roluri şi etichete LBAC). Nu are dreptul de a accesa date din tabele.
- CONNECT permite utilizatorului să se conecteze la o bază de date.
- BINDADD permite utilizatorului să creeze pachete noi în cadrul bazei de date.
- CREATETAB permite crearea de tabele noi în baza de date.
- CREATE_EXTERNAL_ROUTINE permite crearea unei proceduri utilizate în cadrul unei aplicaţii sau de către alţi utilizatori.
- CREATE_NOT_FENCED_ROUTINE permite crearea unei funcţii definite de utilizator sau a unei proceduri.
- IMPLICIT_SCHEMA permite crearea unei scheme sau a unui obiect cu ajutorul comenzii CREATE. În acest caz, SYSIBM devine proprietarul schemei create în mod implicit, iar rolul PUBLIC are dreptul de a crea obiecte în cadrul acestei scheme.

9.1.3.4 Privilegii

Privilegiile sunt autorităţi atribuite utilizatorilor, grupurilor, sau rolurilor, ce le permit efectuarea diverselor activităţi cu obiectele bazei de date. *Figura 9.2*, prezintă o serie dintre aceste privilegii existente în DB2 [9.1].

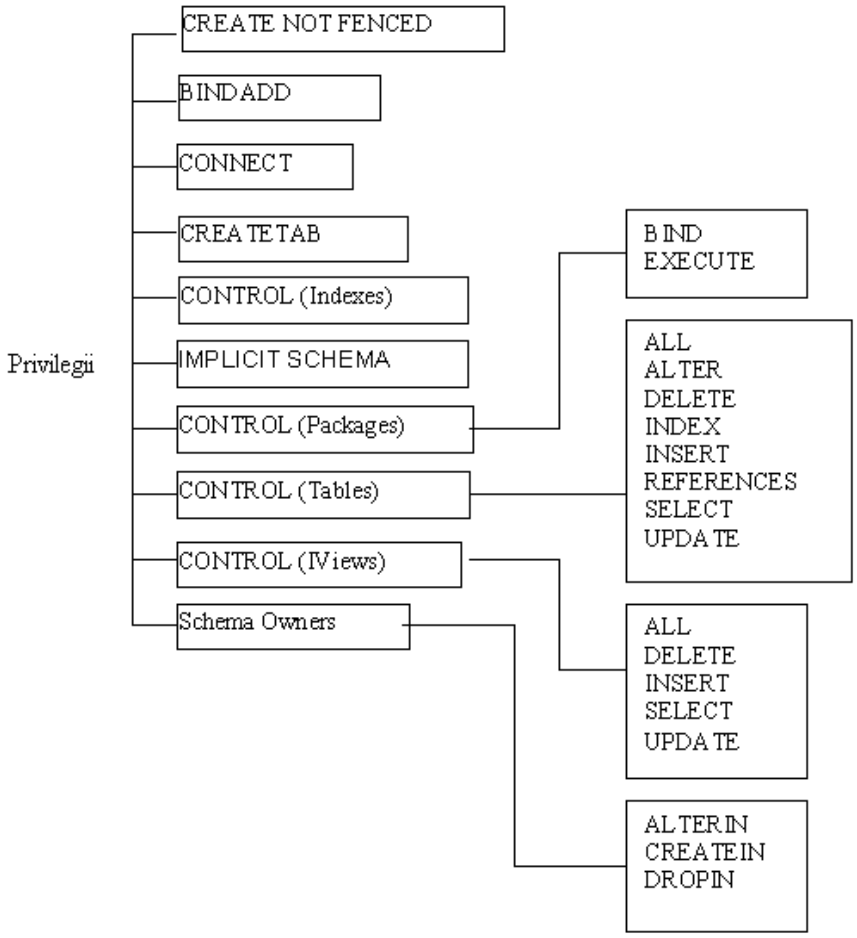


Figura 9.2 – Lista unor privilegii din DB2

Privilegiile stabilesc activităţile ce pot fi efectuate de către utilizator cu obiectele bazei de date şi pot fi acordate utilizatorilor individuali, grupurilor, roluilor sau rolului PUBLIC.

PUBLIC este un grup special căruia i se pot acorda sau retrage anumite autorităţi sau privilegii în cadrul unei baze de date. Acest lucru se poate face de către utilizatorul care s-a autentificat cu succes pe o instanţă DB2. După crearea unei baze de date se acordă implicit următoarele privilegii rolului PUBLIC: CONNECT, CREATETAB, BINDADD, IMPLICIT_SCHEMA, SELECT, UPDATE, EXECUTE, USE.

Utilizatorii sau grupurile care primesc privilegiile CONTROL pot acorda la rândul lor privilegiul altor utilizatori sau grupuri.

9.1.3.5 Controlul accesului bazat pe etichete

Label Based Access Control (LBAC) este o implementare flexibilă a controlului obligatoriu al accesului (mandatory access control - MAC). LBAC acţionează atât la nivel de rând cât

şi la nivel de coloană şi completează controlul discreţionar al accesului (discretionary access control - DAC).

Cele două nivele de protecţie se pot folosi separat sau împreună. LBAC poate fi configurat să îndeplinească cerinţele de securitate ale fiecărui mediu. Toate configuraţiile sunt realizate de către administratorul de securitate prin crearea politicilor de securitate care descriu criteriile ce vor fi folosite pentru a decide cine are acces la ce date.

Odată stabilită politica de securitate, administratorul de securitate poate crea obiecte numite etichete de securitate ca parte a acestei politici. După crearea etichetelor de securitate, acestea pot fi asociate coloanelor şi rândurilor din cadrul tabelelor pentru protecţia datelor. Datele protejate cu ajutorul unei etihcte de securitate sunt numite date protejate. Administratorul de securitate permite accesul utilizatorilor la datele protejate prin acordarea de etichete de securitate. Atunci când un utilizator încearcă să acceseze date protejate, eticheta pe care o deţine este comparată cu eticheta acordată datelor. În acest fel se determină dacă utilizatorul are dreptul să acceseze datele la nivel de coloană, de rând sau ambele, sau îi este interzis accesul la date. Administratorul de securitate poate folosi excepţii pentru a permite accesul acestora la date la care altfel nu ar avea acces. Etichetele de securitate şi excepţiile sunt credenţiale LBAC care se păstrează în cataloagele bazei de date.

Principallul avantaj al folosirii LBAC pentru protecţia datelor importante este acela că nici o altă autoritate (SYSDBA, DBADM, şi SECADM) nu mai are privilegii implicite de accesare a datelor utilizatorului.

9.1.3.6 Roluri

În practică există situaţii în care mai mulţi utilizatori trebuie să posede acelaşi grup de privilegii. În acest caz, cel mai bine este ca să se gestioneze acest set de privilegii în comun şi nu individual, introducându-se astfel noţiunea de rol.

Un rol dintr-o bază de date este un obiect care grupează la un loc unul sau mai multe privilegii sau autorităţi ale bazei de date. Se poate asocia utilizatorilor, grupurilor, rolului PUBLIC sau altor roluri folosind comanda GRANT. De exemplu, se poate defini rolul de programator, permiţând acestuia să introducă, actualizeze sau elimine date dintr-un grup de tabele.

Prin atribuirea de roluri utilizatorilor, aceştia moştenesc toate privilegiile rolului respectiv, suplimentar privilegiilor pe care le au deja.

Un rol poate controla accesul la o bază de date la nivelul de abstractizare corespunzător structurii organizaţiei, deoarece se pot crea roluri corespunzătoare celor existente în organizaţia respectivă. În acest caz utilizatorii sunt ataşaţi rolurilor pe baza responsabilităţilor pe care le au în organizaţie. Atunci când responsabilităţile utilizatorilor se schimbă, aceştia pot fi trecuţi în cadrul altui rol. Rolurile pot fi actualizate fără a fi nevoie să se realizeze acest lucru la nivelul fiecărui utilizator. Rolurile sunt gestionate în cadrul bazei de date, iar DB2 poate stabili când se modifică autorizarea, acţionând corespunzător.

9.1.3.7 Contexte de încredere

Într-o arhitectură a modelului organizată pe trei nivele, în care se află un server Web, o aplicaţie server şi un server de baze de date, nivelul de mijloc, sau serverul aplicaţiei răspunde de autentificarea utilizatorilor care rulează aplicaţia client şi gestionează interacţiunile cu serverul bazei de date. Identificatorul de autorizare al serverului aplicaţiei trebuie să posede toate privilegiile atribuite utilizatorilor pentru a putea efectua operaţiile de care are nevoie. În acest fel modelul de aplicaţii pe trei nivele prezintă o serie de avantaje, prin realizarea tuturor interacţiunilor cu serverul bazei de date (aşa cum cere utilizatorul) în funcţie de identificatorul de autorizare al serverului aplicaţiei. Aceasta însă, ridică o serie de probleme de securitate, cum ar fi de exemplu, faptul că nu se va putea determina care client a efectuat o interogare în momentul apariţiei unei erori.

Contextele de încredere stabilesc o relaţie de încredere între DB2 şi o entitate externă, cum ar fi un server de aplicaţie sau un alt server DB2. Această relaţie de încredere se bazează pe următoarele atribute: autorizarea sistemului, adresa IP sau numele de domeniu şi fluxul de date criptat.

După conectarea la baza de date atributele conexiunii respective sunt comparate cu definiţia fiecărui obiect de tip context de încredere din cadrul bazei de date. Dacă aceste atribute corespund definiţiei unui obiect de tip context de încredere, atunci conexiunea se numeşte conexiune de încredere. Acest tip de conexiune permite iniţiatorului său să obţină facilităţi suplimentare care nu sunt valabile decât în cadrul acelei conexiuni de încredere.

Se poate lucra cu două tipuri de conexiuni de încredere: explicite sau implicite. O conexiune de încredere explicită este o conexiune de încredere care se cere în mod explicit şi care permite iniţiatorului său să treacă de identificatorul curent din cadrul conexiunii la un alt identificator cu sau fără autentificare, şi să obţină privilegii suplimentare care este posibil să nu mai fie valabile în afara conexiunii de încredere.

O conexiune de încredere implicită este o conexiune de încredere care nu este cerută în mod explicit şi care permite doar facilitatea de a obţine privilegii suplimentare.

Pentru a stabili o conexiune de încredere trebuie definite următoarele atribute:

- Un identificator de autorizare al sistemului care este identificatorul ce trebuie folosit de către o conexiune pentru a deveni de încredere.
- O listă de adrese IP ce reprezintă adresele IP de la care trebuie să provină conexiunea pentru a fi declarată de încredere.
- Un flux de date criptat ce reprezintă nivelul criptării ce trebuie folosit de către o conexiune pentru a fi declarată de încredere.

9.1.4 Vederi

Alături de procesul de autorizare, vederile reprezintă o componentă importantă a mecanismului de securitate oferită de către un SGBD relaţional. Vederile permit utilizatorului să vadă informaţiile care îi sunt permise în timp ce alte informaţii, la care acesta nu are acces, să rămână ascunse.

O vedere este un rezultat dinamic al uneia sau mai multor operaţii relaţionale care se aplică pe unul sau mai multe tabele pentru a produce un alt tabel. O vedere se bazează întotdeauna pe datele curente existente în tabelele de bază din care se obţin.

Avantajul unei vederi este acela că poate fi creată pentru a arăta doar datele la care utilizatorul are acces, împiedicând vizualizarea altor date ce pot avea caracter privat sau confidenţial.

Utilizatorul poate primi dreptul de a accesa o vedere, dar nu şi tabelele de bază din care provin.

9.1.5 Controlul integrităţii

Scopul controlului integrităţii este acela de protejare a datelor împotriva utilizării neautorizate, prin restrângerea variabilelor admise şi a operaţiilor ce pot fi efectuate pe date. Controlul integrităţii poate conduce şi la declanşarea execuţiei unei proceduri prestabilite, cum ar fi introducerea de date în cadrul unui jurnal în care se păstrează istoricul lucrului cu date din partea utilizatorilor. Există multe forme de control al integrităţii.

Prima formă pe care o vom discuta este integritatea domeniului. Un domeniu poate fi văzut ca o modalitate de creare a unui tip de dată definit de utilizator. Odată ce se creează un domeniu, acesta poate fi folosit la declararea unei coloane în cadrul unui tabel. La crearea unui domeniu se pot folosi constrângeri (de exemplu, constrângerea CHECK) pentru a restricţiona valorile folosite pentru a satisface condiţia impusă. Un avantaj important al unui domeniu este acela că dacă trebuie modificat, atunci modificarea va avea loc într-un singur loc – în definiţia domeniului.

Aserţiunile reprezintă de asemenea o constrângere foarte puternică prin care se impun anumite condiţii în baza de date. Acestea sunt verificate în mod automat de către SGBD la efectuarea de tranzacţii care implică tabele sau câmpuri care au astfel de aserţiuni. Dacă condiţia nu este adevărată, SGBD-ul va genera un mesaj de eroare.

Din motive de securitate, în celaşi scop, se mai pot folosi declanşatori. Declanşatorii sunt blocuri de cod procedural păstrat în baza de date şi care se execută numai la efectuarea unei comenzi de tip INSERT, UPDATE sau DELETE. Un declanşator, care conţine un eveniment, o condiţie şi o acţiune poate fi mult mai complex decât o aserţiune. Poate interzice efectuarea de acţiuni nepotrivite, poate controla modul de execuţie a anumitor proceduri, sau poate permite scrierea unui rând în cadrul fişierului de jurnalizare pentru a păstra informaţii importante despre utilizator sau despre tranzacţiile efectuate asupra datelor critice din baza de date.

9.1.6 Criptarea datelor

Datele cu caracter critic sau personal păstrate în cadrul tabelelor, precum şi datele critice transmise prin intermediul reţelei, cum ar fi conturile utilizatorilor (ID şi parolă), sunt vulnerabile şi trebuie protejate împotriva atacatorilor.

Criptarea este procesul de codificare a datelor cu ajutorul unui algoritm special, care face imposibilă citirea datelor de către program fără a dispune de o cheie de decriptare. De

obicei, criptarea protejează datele transmise pe liniile de comunicaţie. Există foarte multe tehnici de criptare a datelor, dintre care unele sunt reversibile, în timp ce altele sunt ireversibile. Tehnicile de criptare ireversibile nu permit cunoaşterea datelor originale, dar pot fi folosite pentru a obţine informaţii statistice.

Orice sistem care oferă facilităţi de criptare poate să ofere şi tehnicile corespunzătoare de decriptare, cod ce trebuie protejat folosind mijloace specifice de securitate.

9.2 Politici şi proceduri de securitate

Este obligatoriu să se stabilească setul de politici şi proceduri administrative pentru a crea un context pentru implementarea efectivă a măsurilor de securitate. Există două tipuri de politici şi proceduri de securitate: controlul personalului şi controlul accesului fizic.

9.2.1 Controlul personalului

De obicei, cele mai mari pericole ce pândesc securitatea datelor sunt cele interne, astfel încât trebuie stabilite şi urmate o serie de controale ce privesc personalul ce lucrează cu date. Procedurile de autorizare şi autentificare trebuie impuse cu ajutorul altor proceduri care să ajute la procesul de selecţie a personalului prin care să se valideze potenţialul acestuia în ceea ce priveşte pregătirea şi trăsăturile de caracter urmărite. Angajaţii trebuie pregătiţi în acele aspecte de securitate relevante postului pe care sunt angajaţi fiind atenţionaţi şi încurajaţi să urmeze măsurile standard de securitate. Dacă un angajat părăseşte organizaţia trebuie stabilite o serie de proceduri prin care să se elimine toate datele referitoare la autorizarea şi autentificarea acestuia, precum şi transmiterea de notificări către ceilalţi angajaţi referitoare la aceste modificări.

9.2.2 Controlul accesului fizic

Un control al accesului fizic important se referă la limitarea accesului la anumite zone din cadrul clădirii. În acest scop se poate folosi un card de proximitate a accesului pentru a obţine accesul în acele zone. În acest caz, fiecare acces poate fi înregistrat într-o bază de date. Dacă este nevoie, vizitatorii trebuie să primească ecusoane şi trebuie escortaţi în interiorul acestor zone.

Echipamentele sensibile pot fi amplasate în zone de securitate. Celelelte echipamente pot fi închise în birouri sau pot avea alarme ataşate. Mediile pe care se fac copii de siguranţă trebuie păstrate în zone ferite de foc sau în alte locaţii din afară. Procedurile trebuie să stabilească explicit un plan de transfer al datelor pe alte suporturi media, etichetându-se şi indexându-se.

În ultimul timp, tot mai multe organizaţii au tendinţa să-şi desfăşoare activitatea pe medii mobile pentru a răspunde mai bine cerinţelor pieţii. Laptop-urile sunt astfel de medii mobile, fiind mai sensibile la atacuri, ceea ce face ca datele păstrate pe aceste dispozitive să fie supuse riscurilor de mai mare amploare. Criptarea şi autentificările multiple pot ajuta la protectţia acestor date. Dispozitivele antifurt, cum ar fi cablurile de securitate sau cipurile de urmărire geografică, pot identifica pericolul şi recupera rapid laptop-urile pe care se păstrează date critice.

9.3 Rezumat

Acest capitol acoperă aspecte generale legate de securitatea bazelor de date.

Capitolul începe prin descrierea necesităţii protejării datelor şi a mediului în care se află, precum şi pericolele ce pot afecta datele existente în bazele de date, dar şi întreaga organizaţie.

Prima măsură de securitate se referă la controlul accesului, care poate fi discreţionar sau obligatoriu. Au fost prezentate diverse situaţii de control al accesului ce pot fi implementate în DB2, cum ar fi folosirea mecanismelor de autentificare şi autorizare, a privilegiilor şi a rolurilor, a controlului accesului bazat pe etichete, dar şi a folosirii contextelor de încredere.

O modalitate simplă şi flexibilă de a ascunde părţi mari dintr-o bază de date o constituie utilizarea vederilor.

Scopul controlului de integritate este acela de a proteja datele de accesul neautorizat, prin restricţionarea valorilor ce pot fi atribuite precum şi a operaţiilor ce pot fi efectuate pe o bază de date. Din aceste motive, se definesc domenii, aserţiuni şi declanşatori.

Datele cu caracter critic transmise prin intermediul reţelei, precum şi datele cu caracter personal trebuie protejate folosind tehnica criptării.

Măsurile prezentate în cadrul acestui capitol s-ar putea să nu stopeze toate atacurile accidentale sau rău intenţionate, sau modificarea datelor. Din acest motiv, este necesar să se stabilească o serie de politici şi proceduri cu caracter administrativ pentru a crea un context eficient de implementare a acestor măsuri. Cele mai utilizate politici şi proceduri de securitate folosite se referă la controlul personalului şi la controlul fizic al accesului.

9.4 Exerciţii

Citiţi capitolul 10, "Securitatea bazelor de date" din cartea gratuită în format electronic Getting started with DB2 Express-C şi creaţi doi utilizatori în cadrul sistemului de operare.

Pot cei doi utilizatori crea o bază de date? De ce?

Pot cei doi utilizatori să se conecteze la o bază de date? De ce?

Pot cei doi utilizatori să folosească vederea SYSCAT.TABLES? De ce?

9.5 Întrebări recapitulative

1. De ce este necesar să se protejeze datele din cadrul unei baze de date?
2. Este suficient să se protejeze doar datele din cadrul bazei de date?
3. Care sunt câteva dintre pericolele ce trebuie luate în considerare atunci când se elaborează un plan complet de securitate?
4. Care este definiţia controlului discreţionar al accesului?
5. Care sunt formele de autorizare în DB2?

6. Ce sunt privilegiile?
7. Cum se folosesc contextele de siguranţă în DB2?
8. Ce este o vedere şi de ce este considerată o componentă importantă a mecanismului de securitate?
9. Cum poate fi asigurat controlul integrităţii?
10. Care sunt cele mai folosite poliitci şi proceduri de securitate?

10

Capitolul 10 – Tendinţe în tehnologie şi în bazele de date

Un sondaj efectuat de către IBM în anul 2010 scoate la vedere o serie de tendinţe noi în tehnologie ce pot fi evidenţiate până în anul 2015. Sondajul foloseşte răspunsurile a mai mult de 2,000 profesionişti în domeniul IT din întreaga lume ce au expertiză în domenii cum ar fi testarea software, administrarea reţelei şi a sistemelor, arhitectura software, precum şi elaborarea de aplicaţii Web sau de tip organizaţie. Două aspecte principale se desprind în urma acestui sondaj:

- Cloud computing va prelua conducerea în domeniul achiziţionării de resurse IT de către organizaţii.
- Elaborarea de aplicaţii pentru dispozitivele mobile cum ar fi iPhone şi Android, şi chiar a tabletelor, cum ar fi iPad şi PlayBook, va fi superioară elaborării de aplicaţii pentru alte platforme.

Acest capitol se va referi la Cloud computing, la elaborarea de aplicaţii mobile, precum şi la alte tendinţe existente în tehnologie, explicând rolul bazelor de date în cadrul acestora. Suplimentar, capitolul mai prezintă şi un scenariu real în care se folosesc astfel de tehnologii. După parcurgerea acestui capitol ar trebui să aveţi o înţelegere mai bună referitor la aceste tehnologii, pentru a beneficia de avantajul folosirii lor în soluţiile adoptate în viitor.

10.1 Ce este Cloud computing?

Cloud computing nu este o tehnologie nouă, ci un model nou de furnizare a resurselor IT. Acesta oferă iluzia faptului că persoanele pot avea acces la un număr infinit de resurse de calcul disponibile la cerere. Prin folosirea Cloud computing, se poate închiria putere de calcul fără a avea nici un angajament. Nu este nevoie să cumperi un server, doar plăteşti ce foloseşti. Acest nou model este comparat, de multe ori, cu felul în care oamenii procedează cu plata utilităţilor. De exemplu, se plăteşte doar apa sau electricitatea consumate pe o anumită perioadă de timp.

Cloud computing a modificat drastic modalitatea de obţinere a resurselor de calcul, permiţând aproape oricui, de la companii ale persoanelor fizice, până la companii guvernamentale să lucreze la proiecte ce nu puteau fi altfel abordate anterior.

Tabelul 10.1 compară modelul IT tradiţional al unei întreprinderi cu modelul Cloud computing.

Modelul IT tradiţional	Modelul Cloud computing
Este necesară bugetarea	Parte a cheltuielilor de operare
Investiţii mari în avans	Începe de la 2 cenţi / oră
Se planifică încărcarea maximă	Scalabil la cerere
120 zile necesare pentru pornirea unui proiect	Mai puţin de 2 ore pentru a avea un sistem funcţional.

Tabelul 10.1 – Compararea modelului tradiţional IT cu modelul Cloud computing.

În timp ce în modelul tradiţional IT este nevoie de obţinerea unui buget pentru achiziţionarea de suport hardware, fiind necesară investirea unei mari sume de bani, în modelul Cloud computing cheltuielile sunt considerate cheltuieli de operare necesare desfăşurării activităţii; se plăteşte la cerere o sumă mică pe oră pentru aceleaşi resurse.

În modelul IT tradiţional, cererea bugetului, achiziţionarea de hardware şi software, instalarea acestora în cadrul unui laborator sau centru de date, precum şi operaţia de configurare necesită un mare consum de timp. În medie se poate spune că un proiect poate dura 120 de zile sau mai mult pentru a putea începe. Folosind modelul Cloud computing, se poate obţine un sistem configurat şi gata de lucru în mai puţin de 2 ore!

Companiile care se bazează pe modelul IT tradiţional trebuie să planifice încărcarea maximă. De exemplu, încărcarea medie a unei companii este de 3 servere pentru 25 de zile ale unei luni, dar mai are nevoie de alte 2 servere pentru ultimele 5 zile ale lunii, motiv pentru care compania trebuie să achiziţioneze 5 servere, nu 3. Folosind modelul Cloud computing aceeaşi companie poate investi doar în 3 servere şi închiria 2 servere pentru ultimele 5 zile ale lunii.

10.1.1 Caracteristicile unui Cloud

Cloud Computing se bazează pe trei caracteristici simple:

- Standardizarea.

  Standardizarea oferă posibilitatea de obţinere a unui set important de resurse IT omogene pe baza unor componente oarecare. Standardizarea este opusă personalizării.

- Virtualizarea

  Virtualizarea oferă o metodă de partiţionare a unui grup mare de resurse IT şi alocarea lor la cerere. După utilizare, resursele pot fi returnate grupului pentru fi folosite de către alţii.

- Automatizarea

Automatizarea permite utilizatorilor unui Cloud să obţină controlul asupra resurselor folosite fără a fi nevoie de intervenţia administratorului care să gestioneze cererile. Acest lucru este foarte important într-un mediu Cloud de mari dimensiuni.

Dacă vă gândiţi bine, probabil că aţi folosit anterior o serie de servicii Cloud. Facebook, Yahoo, şi Gmail, de exemplu, furnizează servicii standardizate, virtuale şi automate. Pentru utilizarea acestor servicii trebuie create conturi.

10.1.2 Modele de servicii Cloud computing

Există trei modele de servicii Cloud computing:

- Infrastructură sub formă de servicii (IaaS)
- Platformă sub formă de servicii (PaaS)
- Software sub formă de servicii (SaaS)

Furnizorii de servicii de tip infrastructură pun la dispoziţie infrastructura de care este nevoie la un moment dat (centru de date, hardware, sistem de operare) astfel încât nu mai trebuie să mai fim preocupaţi de astfel de aspecte.

Furnizorii de servicii de tip platformă pun la dispoziţie infrastrctura de care este nevoie pentru o aplicaţie sau pentru aplicaţiile de tip middleware. De exemplu, în cazul produselor middleware de la IBM, PaaS oferă serverul DB2, serverul de aplicaţii WebSphere application server, ş.a.m.d.

Furnizorii de servicii de tip software pun la dispoziţie aplicaţia necesară în cadrul activităţii. Ne putem gândi la aceşti furnizori ca la nişte depozite de aplicaţii din care se pot închiria aplicaţiile dorite cu ora. Un exemplu tipic de SaaS este Salesforce.com.

10.1.3 Furnizorii de servicii Cloud

Astăzi, există pe piaţă o serie de furnizori de servicii de tip Cloud computing, cum ar fi IBM cu oferta "Smart Business Development and Test on the IBM Cloud", Amazon cu oferta Amazon Web Services (AWS), Rackspace, ş.a.m.d.

10.1.3.1 IBM Smart Business Development and Test on the IBM Cloud

IBM Smart Business Development and Test on the IBM Cloud, sau ***IBM developer cloud,*** oferă servicii create special pentru dezvoltare şi testare. Acest Cloud îşi are centrele de date situate în Statele Unite şi Europa şi pemite folosirea de instanţe bazate pe Intel pe 32 şi 64 de biţi folosind sistemele de operare RedHat sau SUSE Linux, sau Microsoft Windows. O instanţă este de fapt un server virtual.

IBM Developer Cloud poate fi accesat la adresa https://www-147.ibm.com/cloud/enterprise

Figura 10.1 prezintă ecranul IBM Developer Cloud. Se prezintă trei instanţe diferite care rulează, sistemul de operare folosit şi adresa IP.

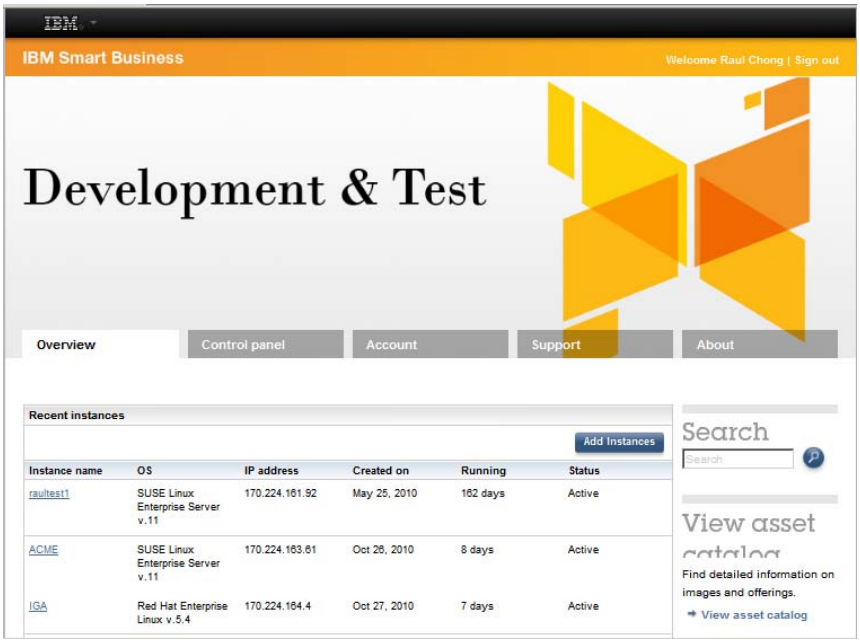


Figura 10.1 - Ecranul IBM Developer Cloud

Încărcarea şi rularea unui sistem se face în doar câteva minute prin apăsarea butonului "Add instances". Apoi se alege din multitudinea de instaţe disponibile cea de care este nevoie, aşa cum se vede în *Figura 10.2*.

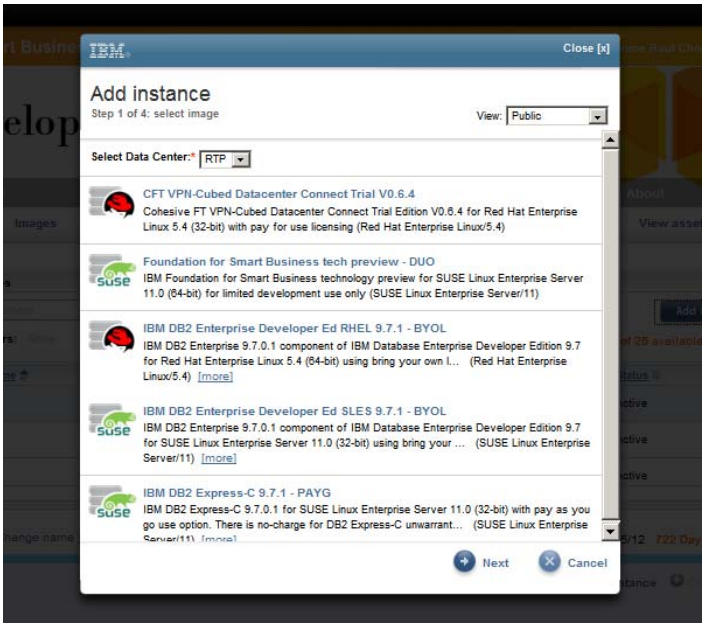


Figura 10.2 – Alegerea instanţei care se foloseşte în IBM Developer Cloud

De exemplu, dacă se doreşte să se lucreze cu IBM DB2 Express-C 9.7.1 - PAYG (Pay as you go – plăteşti în funcţie de cât foloseşti) pe sistemul de operare SUSE Linux, se alege pur şi simplu opţiunea respectivă. După selectare, se apasă *Next*, şi se va ajunge la ecranul din *Figura 10.3*

În acest ecran se poate alege tipul de instanţă dorită (pe 32 de biţi, sau pe 64 de biţi) precum şi numărul de procesoare necesare în funcţie de categorie (cupru, bronz, argint, aur). De exemplu, cupru pe 32 de biţi oferă, de obicei, 1 CPU cu frecvenţa de 1.25GHz, 2GB de memorie şi 175GB spaţiu pe disc.

Figura 10.3 prezintă ceea ce este necesar să se introducă pentru a crea o cheie de care este nevoie pentru descărcarea pe calculatorul personal astfel încât ulterior, prin intermediul SSH să se obţină accesul la instanţa creată, asemănător folosirii *putty*.

Figura 10.3 – Configurarea instanţei

După ce se apasă *Next*, instanţa va fi disponibilă, ceea ce înseamnă faptul că sunt alocate procesoarele şi memoria, este instalat sistemul de operare, iar software-ul din imagine (serverul de baze de date DB2 Express-C în acest exemplu) este şi acesta instalat. Procesul poate dura câteva minute. Ulterior, se poate adăuga spaţiu suplimentar de stocare pe disc.

10.1.3.2 Amazon Web Services

Amazon Web Services, sau ***AWS,*** este liderul de piaţă la furnizarea de infrastructuri de tip cloud public. AWS are centre de date situate in patru regiuni: US-East, US-West, Europa şi

Asia Pacific. Fiecare regiune are mai multe zone de disponibilitate pentru îmbunătăţirea activităţii.

Asemănător IBM developer cloud, prin folosirea AWS se pot alege servere virtuale numite **Elastic Cloud compute (EC2) instances.** Aceste instanţe se bazează pe Intel (32 şi 64-biţi) şi pot rula sistemele de operare Windows sau Linux (diverse distribuţii). Se alege din tipurile predefinite de instanţe în funcţie de cerinţele de processor, memorie şi stocare locală. *Figura 10.4* prezintă diverse tipuri de instanţe AWS EC2. Fiecare tip are un preţ diferit pe oră aşa cum se arată în documentaţia de la aws.amazon.com.

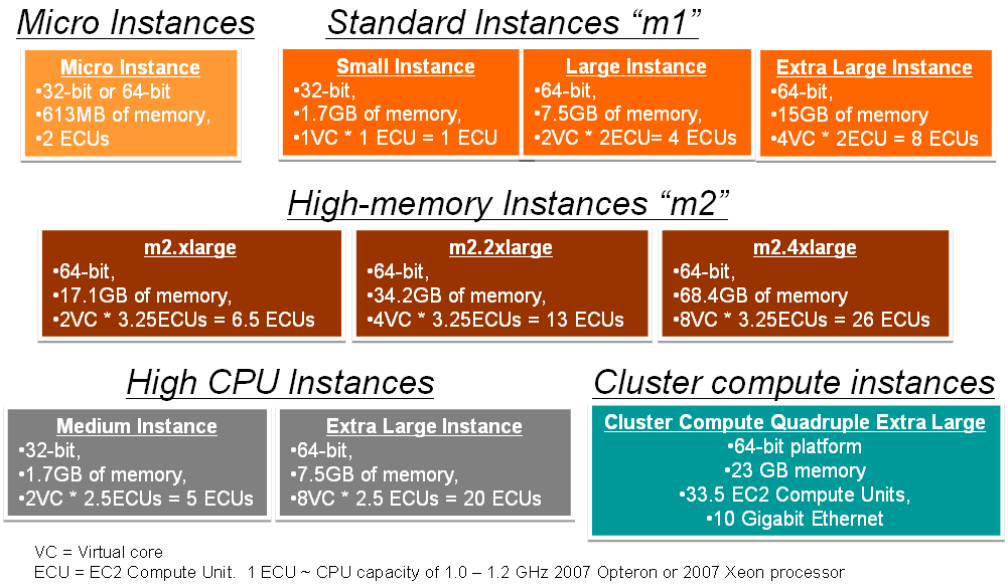


Figura 10.4 – Tipuri de instanţe AWS EC2

De asemenea se poate adăuga spaţiu suplimentar de stocare pentru instanţă. AWS oferă trei variante:

- Stocare pe instanţă:

  Această variantă este inclusă în instanţa aleasă fără costuri suplimentare; totuşi, datele nu sunt persistente, ceea ce înseamnă faptul că acestea vor dispare dacă instanţa se distruge sau se încheie.

- Simple Storage Service (S3).

  S3 se comportă asemănător unui mediu de stocare bazat pe fişiere şi organizat pe foldere. Se poate interacţiona cu acesta folosind cereri http de tip put sau get.

- Elastic Block Storage (EBS).

  Volumele EBS pot fi tratate asemănător discurilor din cadul unui sistem de calcul. Acesta permite stocarea persistentă, fiind ideal pentru lucrul cu baze de date.

10.1.4 Gestionarea securităţii pe Cloud

Securitatea este motivul principal pentru care companiile evită să folosească un cloud public. Faptul că datele confidenţiale ale companiei sunt păstrate la un terţ este văzut de către companii ca riscuri de securitate şi de păstrare a caracterului privat al datelor.

Deşi aceste temeri sunt îndreptăţite, totuşi tehnologia Cloud computing evoluează în continuu. Norii de calcul privaţi (private cloud) oferă o modalitate de reasigurare a utilizatorului că datele sale sunt păstrate în siguranţă. Produsele software, de tipul IBM Cloudburst™ şi IBM WebSphere Cloudburst Appliance lucrează împreună pentru a permite companiilor să-şi dezvolte propriile sisteme de tip cloud.

Companii precum Amazon şi IBM oferă servicii de creeare a norilor de calcul privaţi virtuali (VPC) în care serverele continuă să se afle în centrele de date ale furnizorilor, nefiind accesibile pe Internet; securitatea poate fi gestionată complet prin intermediul infrastructurii de securitate şi de procesele proprii companiei.

Companiile pot folosi şi ***nori de calcul hibrizi*** păstrând datele critice în norii de calcul privaţi, în timp ce datele folosite pentru dezvoltare şi testare se păstrează pe norii de calcul publici.

10.1.5 Baze de date şi Cloud Computing

Cloud Computing este o metodă nouă de furnizare a resurselor IT, inclusiv a bazelor de date. Serverul de date IBM DB2 este pregătit pentru a lucra pe Cloud, atât din punct de vedere al licenţierii cât şi al caracteristicilor.

Pe AWS sunt disponibile diverse ediţii de DB2 pentru producţie, dezvoltare şi testare folosind imagini IBM developer cloud. Imaginile DB2 sunt disponibile şi pentru norii de calcul privaţi folosind VMware sau maşina WebSphere Cloudburst. *Figura 10.5* prezintă imaginile DB2 disponibile pe nori privaţi, publici şi hibrizi.

Figura 10.5 – Imagini DB2 disponibile pe nori privaţi, hibrizi şi publici

Pentru licenţiere se pot folosi metodele:

- Bring your own license (BYOL) permite utilizarea licenţelor DB2 existente în cloud.
- Pay as you go (PAYG) permite plata doar a ceea ce se foloseşte

Întotdeauna se poate folosi DB2 Express-C în Cloud fără să se plătească, dar s-ar putea să se ceară să se plătească pentru infrastructura folosită.

În ceea ce priveşte caracteristicile DB2 care prezintă avantaje particulare în mediul Cloud amintim:

- Database Partitioning Feature (DPF) – caracteristica de partiţionare a bazelor de date
- High Availability Disaster Recovery (HADR) – disponibilitate ridicată la refacerea datelor după dezastre
- Comprimare

DPF se bazează pe arhitectura de lucru individual care se potriveşte foarte bine în modelul Cloud. DPF este ideală pentru depozite mari de date în care trebuie regăsite date pentru efectuarea de rapoarte. Folosind DPF, interogările sunt în mod automat paralelizate pe un grup de servere, acest lucru fiind transparent pentru utilizator. Se pot adăuga servere din cadrul Cloud-ului la cerere care au propriile procesoare, memorii şi spaţii de stocare. DB2 va rebalansa apoi automat datele, performanţele obţinute fiind îmbunătăţite aporape liniar.

HADR este o caracteristică robustă a DB2 având mai mult de 10 ani de existenţă. HADR foloseşte două servere, unul primar şi altul în aşteptare. Atunci când ambele servere se află în aceeaşi locaţie, această caracteristică oferă o *disponibilitate ridicată*. Atunci când unul dintre servere este plasat într-o locaţie, iar celălalt se află într-o altă locaţie (de obicei într-un alt judeţ sau ţară), HADR oferă caracteristica de *refacere după dezastre*. Disaster Recovery (DR – refacerea după dezastre) reprezintă unul dintre cele mai importante şi costisitoare domenii din IT. Este costisitor deoarece companiile trebuie să plătească pentru spaţiul ocupat în alte locaţii, dar şi pentru resursele IT (servere, medii de stocare, reţea) necesare. Cloud Computing răspunde foarte bine la aceste cerinţe. Se poate închiria un spaţiu în cadrul unui Cloud dintr-un centru de date distribuit, fără a se plăti pentru spaţiul ocupat, electricitatea consumată, răcirea necesară, securitate ş.a.m.d. În acelaşi timp, se pot obţine toate resursele IT necesare fără a avea un buget destinat special acestui lucru. Accesarea “site-ului DR” de pe Cloud poate fi realizată din orice locaţie în mod sigur, doar prin folosirea unui browser si a ssh. Având HADR se poate plasa serverul primar după cum este nevoie, iar celălalt server pe Cloud. Pentru a avea o securitate mai bună, cel de-al doilea server se poate păstra pe un cloud privat virtual. HADR permite sistemului să-şi reia activitatea în mai puţin de 15 secunde de la căderea serverului primar.

Se mai poate folosi şi caracteristica de comprimare a DB2 pe Cloud pentru economisire de spaţiu. Comprimarea realizată de DB2 depinde de gradul de încărcare, astfel încât pot fi îmbunătăţite performanţele generale ale sistemului.

Mai există şi o serie de alte caracteristici din DB2 care nu au fost prezentate în acest capitol, dar care se pretează foarte bine unei astfel de abordări.

10.2 Elaborarea de aplicaţii mobile

Aplicaţiile mobile sunt produse software care se execută pe dispozitive mobile. Aplicaţiile mobile obişnuite folosesc tehnologia de calcul mobil pentru a obţine informaţiile păstrate pe un server în orice loc şi moment. În ultimul timp, aplicaţiile mobile care folosesc un mediu Cloud devin din ce în ce mai populare.

Proiectarea şi implementarea aplicaţiilor mobile nu este la fel de simplă ca elaborarea aplicaţiilor de tip desktop. Este foarte important ca programatorii de aplicaţii mobile să aibă în vedere contextul în care se folosesc astfel de aplicaţii. Din punct de vedere al afacerii, este foarte dificil pentru managerii de proiect să estimeze riscurile unui proiect de aplicaţii mobile datorită dificultăţilor ce apar la compararea diverselor caracteristici de pe platformele mobile existente.

Astăzi, platformelor consacrate, cum ar fi Symbian, Microsoft Windows Mobile, Linux şi BlackBerry OS li s-au alăturat Apple's OS X iPhone, Android şi mai recent Palm's Web OS, ceea ce contribuie la creşterea complexităţii în programare. *Figura 10.6* prezintă o arhitectură tipică folosită la elaborarea unei aplicaţii mobile.

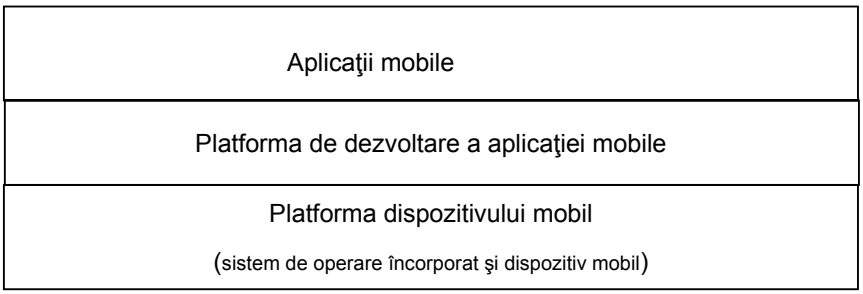


Figura 10.6 – Elaborarea unei aplicaţii mobile: vedere de ansamblu

În *Figura 10.6*, se observă faptul că aplicaţiile mobile se află deasupra a două straturi:

Platforma dispozitivului mobil se referă la dispozitivul mobil şi sistemul său de operare şi/sau produsul software integrat. De exemplu, sistemul de operare Microsoft Windows Mobile 5.0 este instalat pe un dispozitiv Dell AXIM X51v.

Platforma de elaborare a aplicaţiei mobile se referă la combinaţia de limbaje de programare, API-uri generale de dezvoltare şi medii de execuţie specifice limbajului pe dispozitivul mobil. Platforma de elaborare a aplicaţiilor mobile rulează deasupra platformei dispozitivului mobil şi oferă un strat suplimentar sub forma unui produs software ca barieră de protecţie împotriva produselor software ce au erori sau a celor rău intenţionate. De exemplu, mediul obişnuit de elaborare a aplicaţiilor mobile pentru Windows Mobile 6.0 este Microsoft Visual Studio 2005/2008 şi Windows Mobile 6.0 SDK.

Deoarece există diverse tipuri de platforme pentru dispozitivele mobile şi platforme pentru elaborarea de aplicaţii mobile, alegerea uneia dintre acestea se poate face prin răspunsul la întrebarea: se produce o aplicaţie pentru un anumit dispozitiv, sau o platformă de elaborare a aplicaţiilor mobile?

10.2.1 Elaborarea aplicaţiilor pentru un anumit dispozitiv mobil

Atunci când se scrie o aplicaţie pentru un anumit dispozitiv (sau grup de dispozitive) sau sistem de operare al unui dispozitiv, furnizorul dispozitivului respectiv pune la dispoziţie un SDK, un emulator, şi alte instrumente ce pot ajuta programatorul să scrie aplicaţii specifice acelei platforme.

Tabelul 10.2 prezintă unele dintre platformele de dispozitive mobile precum şi ce anume oferă acestea din punct de vedere al SDK-urilor, instrumentelor ş.a.m.d.

Sistem de operare (OS)	Limbaj/ opţiuni OS	SDK-uri/instrumente	Site Web pentru programator
Symbian OS	C++, Java, Ruby, Python, Perl, OPL, Flash Lite, .NET	Carbide C++, IDE(Nokia) – C++, şi Java SDK, pe dispozitiv	developer.symbian.com
Windows Mobile	Java, C++	Visual Studio, Platform Builder, Embedded Visual C++ (eVC), Free Pascal, şi Lazarus	Microsoft Windows Embedded Developer Center
iPhone OS	Objective-C	Xcode	www.apple.com/iphone
Android	Java	Eclipse, Android SDK, Android Development Tool Plugin, JDK 5	www.android.com
Palm OS	C, C++, Java	Palm OS SDK, Java development cu IBM Websphere EveryPlace Micro Environment, Palm Windows Mobile SDK pentru produse Palm pe platforme Windows Mobile	www.palm.com/us/developer

Tabelul 10.2 – Instrumente şi platforme pentru dispozitive

10.2.2 Elaborarea aplicaţiilor mobile pentru o anumită platformă de aplicaţii

Fiecare programator are încredere într-un anumit limbaj de programare şi un anumit mediu de dezvoltare. De ce se preferă anumite limbaje de programare şi medii de dezvoltare?

Elaborarea aplicaţiilor mobile redeschide aceste întrebări, fiind necesară o re-evaluare a răspunsurilor în lumina elaborării de aplicaţi mobile. Java, .NET (C#, VB), C, C++ şi alte limbaje de programare sunt toate opţiuni valabile în elaborarea de aplicaţii mobile. Într-o anumită privinţă, aceleaşi criterii se folosesc şi la evaluarea limbajelor de programare folosite pentru aplicaţiile de tip desktop Cu toate acestea, elaborarea aplicaţiilor mobile aduce în discuţie consideraţii care modifică modul vechi de elaborare a aplicaţiilor de tip desktop/server.

De exemplu, puterea limbajului de programare Java va fi întotdeauna portabilitatea sa. Java este o platformă disponibilă pe scară largă şi se poate folosi pe cele mai multe dintre dispozitivele mobile. Totuşi, Java nu este prezent pe toate aceste dispozitive. Pentru ca Java să corespundă diferitelor stiluri şi platforme, arhitectura Java a fost descompusă într-o serie de configuraţii şi profile. Configuraţiile şi profilele diferă în funcţie de dispozitiv şi de caracteristicile acestuia. Din acest motiv, s-ar putea ca scrierea pe o anumită platformă să nu răspundă cerinţelor clienţilor.

Prin urmare cum trebuie căutate platformele software destinate aplicaţiilor mobile? Următoarele două secţiuni prezintă cele mai importante platforme pentru dispozitivele mobile precum şi a platformelor de elaborare a aplicaţiilor mobile, prezentând modalităţi de compare a acestora.

10.2.3 Platforme pentru dispozitive mobile

În continuare vor fi prezentate unele dintre cele mai folosite platforme pentru dispozitivele mobile.

10.2.3.1 Windows Mobile

Windows Mobile, provine din Windows CE (apărut în 1996), şi oferă un subgrup de API-uri Microsoft Windows pe un sistem de operare în timp real compact destinat dispozitivelor de calcul mobil şi terminalelor wireless. După câteva încercări nereuşite pe PDA-uri, Windows Mobile a reuşit în sfârşit să obţină succesul aşteptat pe aceste dispozitive şi pe handset-uri, probabil cele mai notabile fiind pe Motorola Q şi pe anumite dispozitive Treo.

Microsoft oferă tehnologia ActiveSync pentru sincronizarea utilizatorilor. Versiunea curentă de Microsoft Windows Mobile este o platformă versatilă pentru programatori, care permite elaborarea de aplicaţii scrise în limbajele de programare C sau C++ , suportând în acelaşi timp şi codul folosit în mediul .NET Compact Framework.

10.2.3.2 Symbian OS

Symbian OS este un sistem de operare proprietar destinat dispozitivelor mobile care are biblioteci asociate, interfaţă utilizator, framework-uri şi implementări ce fac legătura cu instrumentele uzuale dezvoltate de către Symbian Ltd. Symbian a început sub denumirea de EPOC Release 5. Symbian este o platformă foarte puternică care foloseşte limbajul C++ în cadrul unui framework pus la dispoziţie de către Symbian pentru elaborarea de aplicaţii şi servicii. Platforma Symbian rulează Java ME, astfel că telefoanele Symbian pot rula atât aplicaţiile Symbian native cât şi aplicaţiile Java ME.

10.2.3.3 iPhone

iPhone este proiectat şi vândut de către Apple Inc. Este o platformă închisă, care permite folosirea aplicaţiilor mobile doar sub un browser Web Safari. Deoarece interfaţa minimală hardware nu are tastatură fizică, ecranul multi-touch prezintă la nevoie o tastaură virtuală.

iPhone şi iPod Touch SDK folosesc limbajul Objective C, bazat pe C. Mediul de dezvoltare integrat folosit la elaborarea aplicaţiilor iPhone se numeşte Xcode, şi este alcătuit dintr-o suită de instrumente de programare pentru elaborarea de software pe Mac OS X de la Apple.

10.2.3.4 Android

Android este un produs software deschis şi gratuit destinat dispozitivelor mobile care conţine sistem de operare, middleware, şi alte aplicaţii cheie. Limbajul de programare pentru Android este Java.

Programarea aplicaţiilor Android se face exclusiv în Java. Pentru aceasta este nevoie de un anumit SDK Java pentru Android care conţine un set de instrumente de dezvoltare. Printre acestea putem enumera un depanator, biblioteci, emulator handset (bazat pe QEMU), documentaţie, exemple de cod şi tutoriale. Platformele de dezvoltare suportate în prezent conţin arhitecturi de calcul bazate pe x86 ce rulează Linux (orice distribuţie recentă de Linux pentru desktop), Mac OS X 10.4.8 sau mai nou, Windows XP sau Vista. Mai sunt necesare Java Development Kit, Apache Ant şi Python 2.2 sau mai nou. Mediul integrat de dezvoltare suportat în mod oficial este Eclipse (3.2 sau mai nou) ce foloseşte Plug-in-ul Android Development Tools (ADT), prin care programatorii pot folosi orice editor de texte pentru editatea fişierelor Java şi XML după care apelează un instrument în linie de comandă pentru crearea, compilarea şi depanarea aplicaţiilor Android.

10.2.4 Platforme de dezvoltare a aplicaţiilor mobile

În această secţiune se vor prezenta platformele de elaborare a aplicaţiilor folosite la implementarea acestora pe dispozitivele mobile.

10.2.4.1 Java Micro Edition (Java ME)

Java ME, numită anterior ***Java 2 Micro Edition*** (***J2ME***), este o platformă ce împacă dispozitivele mobile uşoare cu alte dispozitive de calcul netradiţionale, cum ar fi dispozitivele media. Lozinca promisă odată programatorilor care spunea “scrie odată, ruelază oriunde” continuă să se aplice şi pentru Java ME, aplicaţiile fiind disponibile pe diverse dispozitive, deoarece limbaje cum ar fi C sau C++ pe alte platforme necesită cunoştinţe solide despre API-urile specifice acesteia, cum sunt cele de la Windows Mobile sau Symbian OS.

De fapt, există numeroase implementări Java pentru diverse dispozitive suportate pe Java ME.

10.2.4.2 .NET Compact Framework

Această platformă este o versiune a.NET Framework folosită pe platformele Windows CE. Aceasta este un subset standard .NET framework, dar conţine şi alte câteva clase

suplimentare ce răspund cerinţelor specifice dispozitivelor mobile.

Actualmente, sunt disponibile .NET Compact Framework V1.0, V2.0 şi V3.5. Ce versiune de .NET Compact Framework trebuie aleasă? "Ultima" poate fi răspunsul normal, dar, ca multe alte aspecte ce privesc dispozitivele mobile, nu este chiar aşa simplu! Ca programator de aplicaţii mobile, trebuie să alegeţi versiunea de .NET Compact Framework pe care să vă dezvoltaţi aplicaţia. Dacă se alege versiunea 1.0, puteţi fi sigur că aplicaţia va rula pe toate dispozitivele, deoarece versiunile începând cu 2.0 rulează aplicaţii care au fost create să ruleze pe versiuni anterioare. Totuşi, dacă se scrie cod ce utilizează caracteristici valabile doar în versiunea .NET Compact Framework 2.0, trebuie ca mediul de execuţie corespunzător să fie instalat pe dispozitivul pe care se va folosi aplicaţia.

10.2.4.3 Native C++

Aplicaţiile C++ pot rula nativ pe dispozitiv, făcând în acest fel ca acestea să fie mai puţin portabile, dar aceasta înseamnă, de obicei, o viteză sporită precum şi un control mai bun asupra dispozitivului.

De obicei, programele C++ native au nevoie de instrumente cum ar fi Microsoft Visual C++ 6/.NET, Eclipse IDE alături de C++ SDK, sau Borland C++ Builder. În mod deosebit, pentru Symbian OS, se poate alege şi Metrowerks CodeWarrior Studio care cere la rândul său un SDK, Symbian C++ SDK de la Nokia.

10.2.4.4 Mediu binar de execuţie pentru Wireless

Binary Runtime Environment for Wireless (BREW - mediul binar de execuţie pentru Wireless) este o platformă de programare dezvoltată de către Qualcomm pentru telefoanele bazate pe CDMA. BREW oferă SDK şi emulator pentru elaborarea şi testarea aplicaţiilor scrise în C sau C++.

Pentru a lucra cu BREW, este nevoie de instrumente de programare cum ar fi BREW SDK, instrumente de compilare ARM RealView, Visual C++ etc.

10.2.5 Valul următor de aplicaţii mobile

Un spectru larg de aplicaţii mobile şi servicii din generaţia următoare a ieşit la suprafaţă în ultimul timp. Noile aplicaţii mobile au în spate furnizorii de servicii de telefonie care caută aplicaţii şi servicii pentru sistemele 3G şi alte infrastructuri de date wireless. Aplicaţiile mobile tradiţionale, cum sunt cele de voce sau serviciile simple de date care oferă navigare pe Internet nu mai sunt suficiente astăzi; consumatorii cautând să beneficieze din plin de avantajele mobilităţii la putere redusă, de folosirea de dispozitive wireless mobile pentru distracţie, accesarea de informaţie de oriunde, dar şi efectuarea de operaţii flexibile în domeniul afacerilor.

Multe pieţe mari sunt astăzi inundate de dispozitivele mobile, cum ar fi sistemele telematice, servicii m-commerce şi m-enterprise, servicii de streaming multimedia pe dispozitive mobile, mesageria mobilă, localizarea bazată pe calcul mobil, ş.a.m.d.

Cu siguranţă, în următorii câţiva ani, vom fi martorii multor descoperiri interesante şi inovaţii tehnologice în domeniul calculului mobil.

10.2.6 DB2 Everyplace

Ediţia pentru servicii mobile a DB2 se numeşte DB2 Everyplace. Caracteristicile DB2 Everyplace, o bază de date relaţională de mici dimensiuni şi de performanţă ridicată, permit folosirea aplicaţiilor de tip organizaţie, dar şi a datelor, într-un mediu sigur, pe dispozitive mobile.

Obs:

Pentru a afla mai multe despre elaborarea aplicaţiilor mobile, vedeţi cartea în format electronic, *Getting started with mobile application development*, ce face parte din seria de cărţi a programului DB2 on Campus.

10.3 Prelucrarea datelor cu caracter economic

Bazele de date se folosesc de obicei pentru a efectua tranzacţii online, cum ar fi tranzacţiile bancare, dar mai pot fi folosite şi în scopul prelucrării datelor cu caracter economic pentru a observa tendinţele şi modelele ce ajută la luarea deciziilor corecte. Astăzi, companiile culeg zilnic cantităţi uriaşe de date, care sunt păstrate în depozite de date de mari dimensiuni în scopul creării de rapoarte cu ajutorul unor aplicaţii cum ar fi IBM Cognos®.

Companiile care luptă din greu pentru întreţinerea depozitelor de date de mari dimensiuni sunt din ce în ce mai atrase de ideea achiziţiei de maşini care să lucreze cu asttfel de depozite. O astfel de maşină este alcătuită din componente hardware şi software care sunt integrate şi testate pentru a dezvolta anumite funcţionalităţi. ***IBM Smart Analytics System*** este una dintre aceste maşini folosite la lucrul cu depozite mari de date cu caracter economic.

Obs:

Pentru a afla mai multe amănunte despre depozitele mari de date şi datele cu caracter economic, vedeţi cartea în format electronic, *Getting started with data warehousing*, ce face parte din seria de cărţi a programului DB2 on Campus.

10.4 db2university.com: Implementarea unei aplicaţii în Cloud (studiu de caz)

Progresele obţinute în tehnologie permit progrese în alte domenii. Datorită Internetului, multe universităţi oferă diplome online. Se acordă creditele corespunzătoare studenţilor care se înregistrează şi participă la cursurile online. Aceste cursuri constau deseori din seminarii înregistrate, sesiuni live, forumuri şi altele. Pundits spune că în viitorul apropiat oricine are acces la Internet va putea să primească o educaţie excelentă fără a fi nevoie să urmeze fizic o universitate.

Educaţia online dă posibilitatea învăţării în ritmul propriu, dispunând în acelaşi timp de confortul de acasă. Datorită dispozitivelor mobile se pot apela resurse educaţionale chiar şi pe perioada deplasărilor.

În această secţiune se va prezenta **db2university.com**, un site Web educaţional care a fost implementat folosind multe dintre tehnologiile descrise în acest capitol. *Figura 10.7* prezintă aspectul acestui site la momentul scrierii cărţii.

Figura 10.7 – Pagina de pornire a site-ului db2university.com

10.4.1 Moodle - sistem open source de management al cursurilor

O serie de materiale, de la materialele video la prezentări media distribuite prin intermediul Internetului, dar şi cărţi în format electronic, sunt disponibile permanent pe Internet pentru studiu individual, dar de cele mai multe ori, acestea nu sunt organizate într-o modalitate care să permită utilizarea lor cu uşurinţă. Pe de altă parte, o serie dintre informaţiile obţinute pe această cale s-ar putea să nu mai fie de actualitate. ***Sistemele de management al cursurilor*** permit instructorilor să creeze şi să aranjeze materialele existente fiind astfel mai uşor de urmărit. La **db2university.com** se foloseşte sistemul open source de management al cursurilor, numit ***Moodle***.

Moodle, scris în PHP, este unul dintre cele mai bune, dacă nu cel mai bun sistem de acest tip de pe piaţă. Martin Dougiamas de la Curtin University of Technology din Perth, Australia a elaborat prima versiune a sistemului în anul 2001. Astăzi, sunt peste 52,000 de site-uri Moodle înregistrate şi peste 950,000 de utilizatori înregtistraţi la moodle.org din peste 175 de ţări din întreaga lume.

Contactul cu Moodle a început la sfârşitul anului 2009. Studenţii de la California State University, Long Beach (CSULB) au fost desemnaţi, ca parte a unuia dintre cursurile pe care le aveau, să facă ca versiunea Moodle 1.9.6 să lucreze împreună cu DB2 Express-C. Contribuţia acestora pentru comunitate poate fi văzută la adresa http://moodle.org/mod/data/view.php?d=13&rid=3100&filter=1.

La începutul lui 2010 am făcut ca versiunea Moodle 2.0 să folosească DB2 Express-C.

Moodle 2.0 reprezintă o revizuire majoră a aplicaţiei Moodle, prezentând modificări fundamentale faţă de versiunile precedente prin felul în care realizează conexiunile cu bazele de date. La momentul scrierii acestei cărţi, este disponibilă pentru public versiunea Moodle 2.0 Release Candidate 1. Aceasta este versiunea pe care am făcut-o să se conecteze cu DB2 Express-C, şi cea care rulează pentru **db2university.com**. La momentul în care versiunea Moodle 2.0 va fi disponibilă, se va actualiza site-ul db2university.com şi vom marca această contribuţie pentru comunitatea Moodle.

Figura 10.8 prezintă o serie de cursuri disponibile la **db2university.com**. La această pagină se poate ajunge prin alegerea filei *Learn* aflată pe pagina de pornire. Fila Learn prezintă implementarea de Moodle aleasă. S-a creat o temă Moodle specifică pentru db2university.

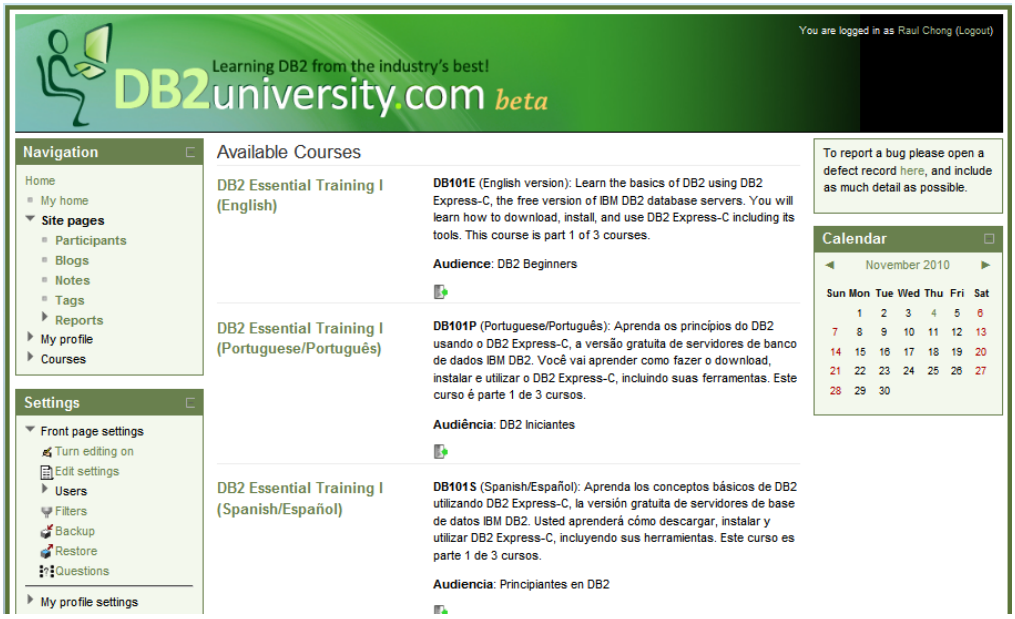


Figura 10.8 – Fila Learn conduce către implementarea Moodle în db2university

Cursurile de la **db2university.com** nu sunt doar despre DB2. Se pot urmări şi alte subiecte, cum ar fi cele despre PHP sau Ruby on Rails, dar fiecare lecţie din cadrul cursurilor respective care are nevoie de o interacţiune cu o bază de date, foloseşte DB2. Instructorii cursurilor sunt membrii ai comunităţii DB2 care include cadre didactice universitare, specialişti, angajaţi IBM şi studenţi.

Există două categorii de cursuri la acest site: *Free courses* şi *Paid courses*. Pentru *Paid courses* se foloseşte plug-in-ul Paypal inclus în Moodle.

Figura 10.9 şi *10.10* oferă mai multe detalii despre cursul "DB2 Essential Training I". Am luat acest curs drept exemplu a ceea ce se poate face în această implementare de Moodle. Se observă faptul că acest curs are multe prezentări video, texte (fişiere PDF), şi link-uri către cartea folosită ca referinţă. Nu se prezintă în figură link-urile către forumul cursului, paginile Web, evaluări şi examene.

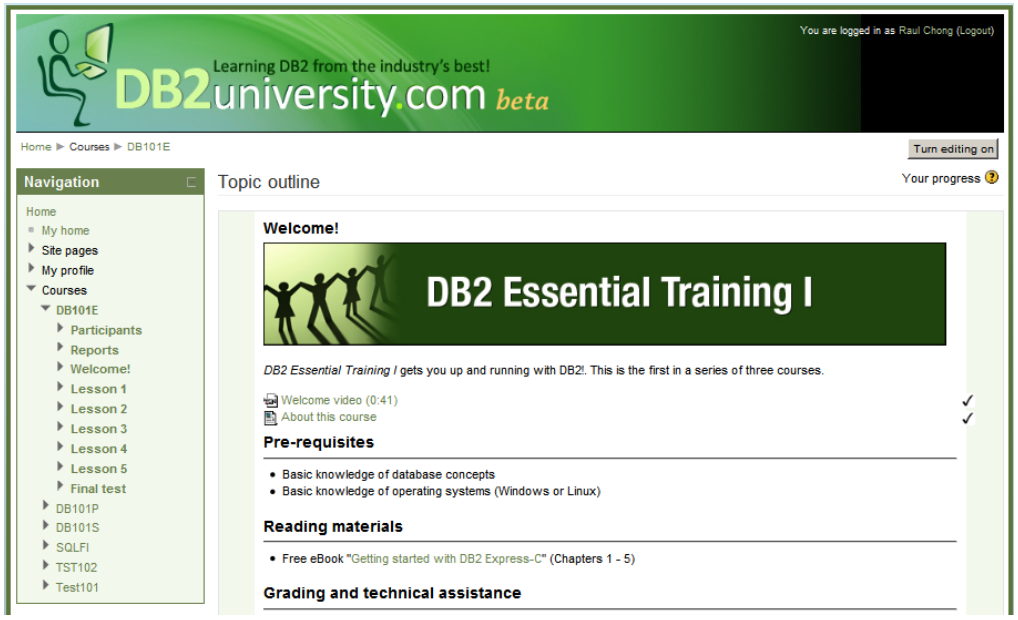


Figura 10.9 – Conţinuturile cursului DB2 Essential Training I

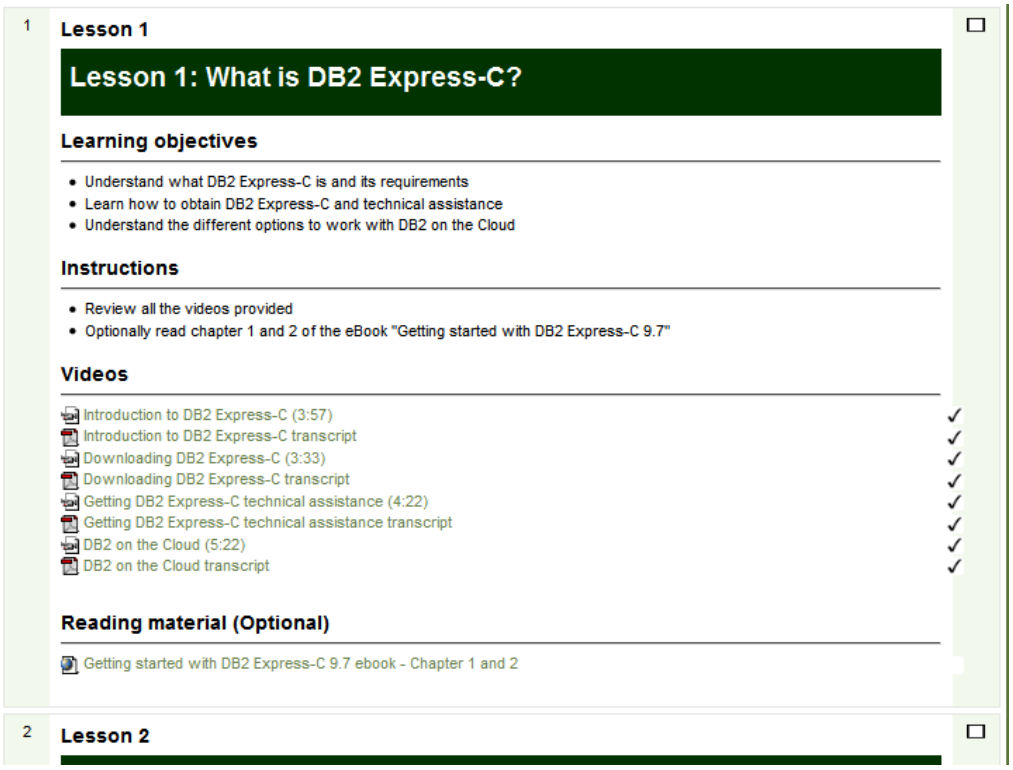


Figura 10.10 - Conţinuturile cursului DB2 Essential Training I (continuare)

La adresa IBM developerWorks® este disponibil un tutorial care vă învaţă cum să creaţi un curs în db2university.com.

Obs:

Pentru a afla mai multe despre ce este nevoie pentru a elabora produse software de tip open source, vedeţi cartea în format electronic *Getting started with open source development*, ce face parte din seria de cărţi sin cadrul programului DB2 on Campus.

10.4.2 Activarea contului de înregistrare openID

Cercetările arată faptul că persoanele care ajung pe o pagină Web care are o procedură complicată sau lungă de înregistrare au tendinţa de a renunţa rapid la acel site. Pentru a încuraja utilizatorii să se înregistreze la **db2university.com** am activat contul de înregistrare openID. Aceasta înseamnă faptul că persoanele care au deja un cont pe Facebook, Google, Yahoo, AOL, ChannelDB2 şi altele, nu trebuie să introducă informaţii de înregistrare. db2university va cere în schimb celorlalţi furnizori - cu consimţământul dumneavoastră - să folosească informaţiile existente acolo. *Figura 10.11* prezintă acest proces.

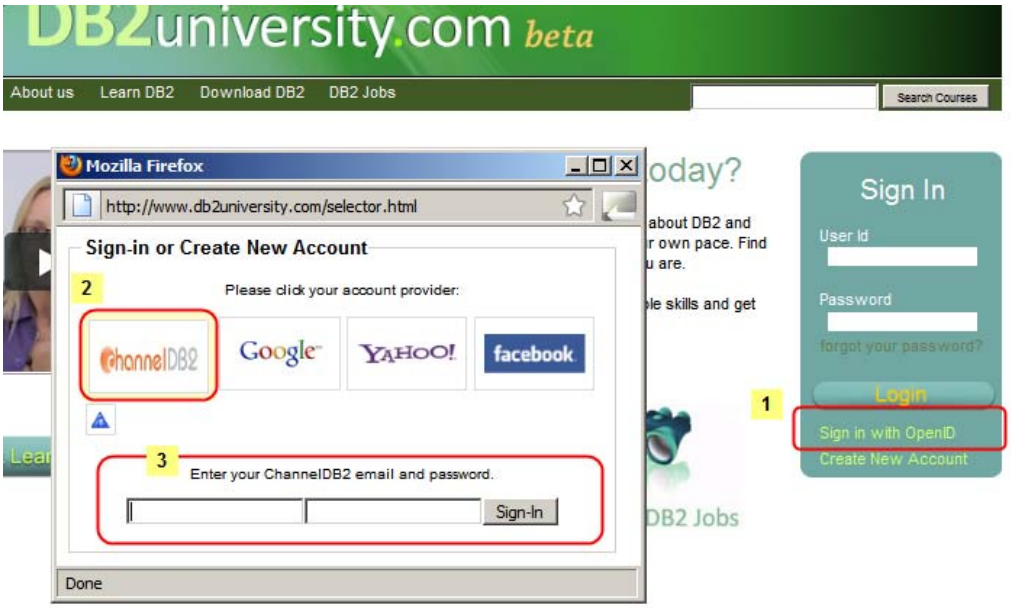


Figure 10.11 – Procesul de înregistrare folosind openID

Mai întâi se apasă pe link-ul *Sign in with OpenID*. Va apare o fereastră ce prezintă furnizorii de informaţii pentru contul openID. Se va alege unul dintre aceşti furnizori. În exemplul de mai sus, s-a ales *ChannelDB2*. În sfârşit, se va introduce contul corespunzător pentru ChannelDB2. Prima dată când se parcurge acest proces va apare o pagină de la furnizorul de informaţii prin care se cere acceptul de trimitere a acestora. Dacă se acceptă, informaţia se transmite, fiind automat înregistrat la db2university!. De fiecare dată când se

doreşte accesul la **db2university.com** se vor urma aceeaşi paşi.

La adresa IBM developerWorks se găseşte un articol care explică felul în care se realizează activarea cu ajutorul contului openID.

10.4.3 Folosirea Amazon Cloud

db2university.com este găzduit de către cloud-ul de la Amazon. Aceasta permite preluarea tuturor avantajelor oferite prin intermediul Cloud computing, inclusiv setarea caracteristicii HADR din DB2 pentru refacerea în caz de dezastre în Cloud. Pentru cloud-ul AWS folosim:

- O instanţă mare EC2 standard (64-bit, 4 ECUs, 7.5 GB de memorie, 850GB capacitate de stocare pe instanţă)
- Volume EBS de 50 GB
- Grupuri S3 cu dimensiuni între 3 şi 1 GB
- CloudFront

Pentru regiunea US-East au fost prevăzute trei instanţe EC2. Una dintre acestea, se foloseşte pentru a rula aplicaţia Moodle application şi serverul Web, iar cealaltă pentru a rula serverul de baze de date DB2 (serverul primar DB2 HADR). Serverul secundar DB2 foloseşte cea de-a treia instanţă EC2 într-o altă zonă de disponibilitate din regiunea US-East ca site de refacere în caz de dezastre.

În ceea ce priveşte stocarea, folosim stocare pe instanţă, volumele S3 şi EBS. *Figura 10.12* prezintă detaliile.

Figura 10.12 – Stocarea DB2 pe cloud-ul Amazon

În figură:

- Volumele EBS se folosesc pentru stocarea datelor DB2 şi a jurnalelor de refacere

DB2. Am ales 50GB pentru aceasta. Opţional, luăm în considerare folosirea RAID.

- Un volum EBS diferit, mai mic, de 2GB se foloseşte pentru păstrarea codului şi a fişierelor de configurare DB2.
- Spaţiul de stocare al instanţei se foloseşte pentru a stoca temporar copii de siguranţă ale DB2. Ulterior vom muta aceste copii pe S3 pentru a creşte perioada de păstrare a acestora.
- Mediul de stocare S3 se foloseşte pentru păstrarea copiilor de siguranţă şi a arhivelor jurnalelor pentru o perioadă de timp mai lungă. Imaginile volumului ESB se fac repetat şi se păstrează pe S3.

AWS CloudFront se foloseşte pentru a permite utilizatorilor să descarce materiale pentru cursuri de la un server Amazon care este mai apropiat locaţiei la care se află utilizatorul. Fişierele care se descarcă sunt mai întâi copiate pe un grup S3, fiind apoi replicate pe aceste servere.

Din punct de vedere software se folosesc:

- Ubuntu Linux 10.04.1 LTS
- DB2 Express 9.7.2 for Linux 64-bit
- Moodle 2.0 Release Candidate 1
- PHP 5.3.3

Se foloseşte DB2 Express şi nu DB2 Express-C, deoarece se foloseşte caracteristica DB2 HADR, care este disponibilă doar pe ediţia DB2 Express.

RightScale este ideal pentru managementul resurselor AWS. RightScale este partener IBM, iar produsele sale software permit crearea rapidă de medii prin folosirea şabloanelor şi a scripturilor.

Un articol aflat la adresa IBM developerWorks explică detaliie AWS folosite la db2university.

10.4.4 Utilizarea unui telefon Android pentru a obţine notele de la curs

Moodle permite studenţilor să-şi vadă notele de la evaluări şi examene pe Web; suplimentar, a fost elaborată o aplicaţie pentru telefoanele Android folosind App Inventor.

Această aplicaţie simplă este prezentată în *Figura 10.13*. Mai întâi trebuie introdus un *cod de acces pentru mobil* care este un cuvânt cheie unic pentru fiecare student ce poate fi obţinut pe baza profilului studentului din Moodle. Apoi, se poate alege cursul respectiv pentru care se afişează notele.

Pentru ca această aplicaţie să funcţioneze, este necesar un cod pe parte de client scris cu App Inventor, şi unul pe parte de server (folosind un serviciu Web creat cu Ruby/Sinatra). Articolul de la IBM developerWorks explică acest lucru.

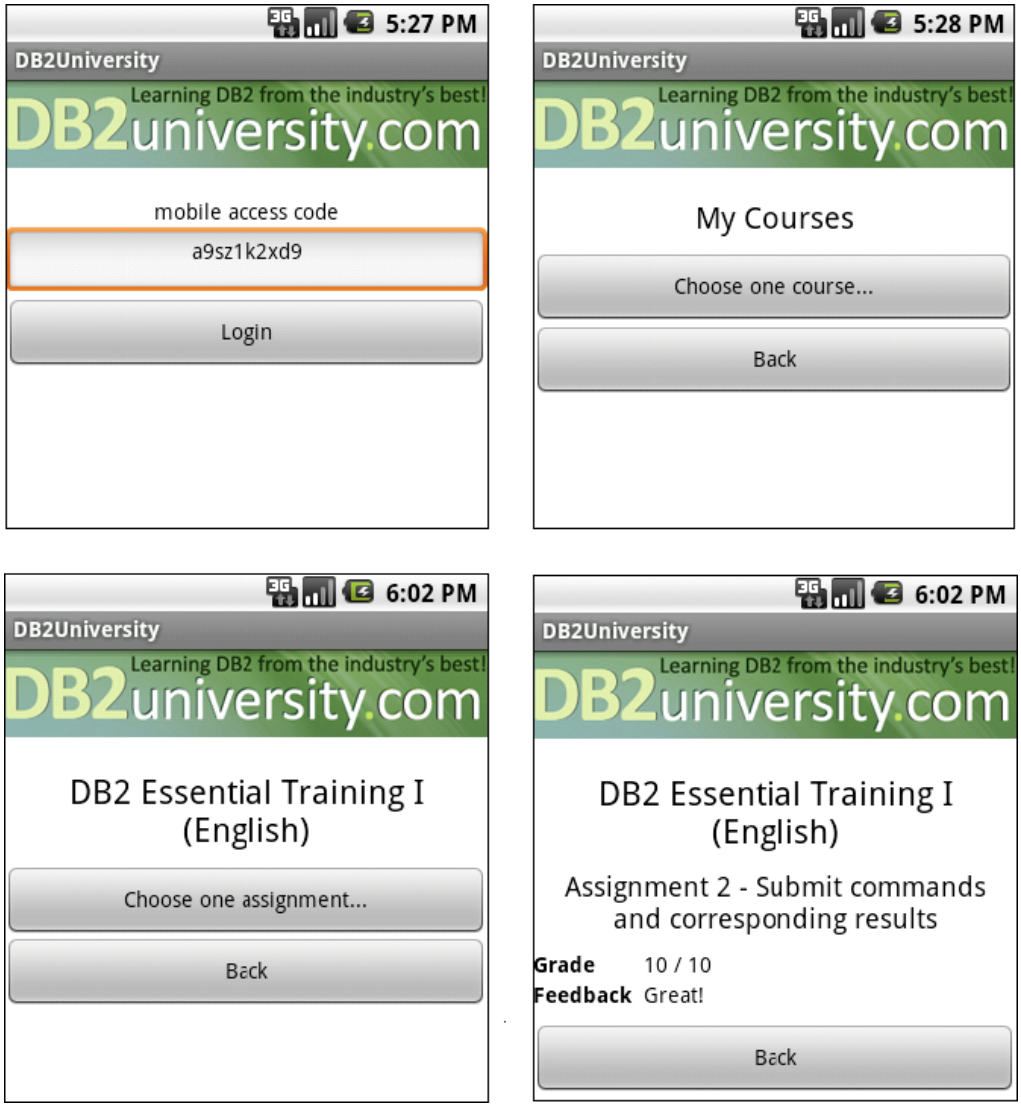


Figura 10.13 - App Inventor pentru aplicaţia Android folosit la accesarea db2university

10.5 Rezumat

Acest capitol a luat în discuţie tendinţele importante manifestate în tehnologie ce vor fi disponibile până în anul 2015, subliniind rolul bazelor de date în cadrul acestor tehnologii. Cloud computing se află în vârful acestei liste, fiind actualmente cea mai discutată temă. Cloud computing este o metodă nouă de furnizare a resurselor IT prin care se permite companiilor şi persoanelor fizice accesul la practic orice fel de resurse de calculator la cerere. Cloud computing este foarte eficientă din punct de vedere financiar deoarece se

plăteşte doar ce se consumă.

În continuare s-a discutat despre aplicaţiile mobile. Aplicaţiile mobile reprezintă un alt domeniu de interes în mare creştere în ultimul timp. Capitolul de faţă a făcut o prezentare a diferitelor platforme de dispozitive mobile, precum şi a platformelor de dezvoltare.

După aceea, s-au prezentat pe scurt, modul de prelucrare a datelor cu caracter economic şi a maşinilor folosite în acest scop. Companiile au interes să obţină informaţii cu caracter economic pe baza datelor pe care le deţin pentru a lua cele mai bune decizii în desfăşurarea activităţii. În acelaşi timp, acestea nu doresc ca departamentul IT să consume prea mult timp cu setarea şi configurarea depozitelor mari de date. Pentru rezolvarea unei astfel de stiuaţii se pot folosi depozite mari de date pe baza cărora se fac prelucrări de date cu caracter economic folosind maşini configurate special în acest scop. Astfel de soluţii sunt oferite cu ajutorul sistemului IBM Smart Analytics.

Capitolul se încheie cu o discuţie despre **db2university.com** ca studiu de caz ce foloseşte multe dintre tehnologiile descrise aici.

A

Anexa A – Răspunsuri la întrebările recapitulative

Capitolul 1

1. O bază de date este un depozit de date proiectat să suporte stocarea eficientă a datelor, extragerea şi mentenanţa acestora.
2. Un sistem de gestiune al bazelor de date, pe scurt SGBD, este un grup de instrumente software care controlează accesul, organizarea, stocarea, obţinerea şi mentenanţa datelor dintr-o bază de date.
3. Modelul informaţiilor este o reprezentare abstractă şi formală a entităţilor împreună cu proprietăţile, relaţiile şi operaţiile ce pot fi efectuate asupra lor. Modelele de date, pe de altă parte, sunt definite la un nivel mult mai concret şi conţin mai multe detalii. Modelele de date sunt mult mai specifice şi servesc la proiectarea sistemelor de baze de date.
4. Principalul avantaj este independenţa datelor (nu este specific înmagazinării fizice).
5. Instalarea şi testarea noilor versiuni de sisteme de gestiune a bazelor de date (SGBD), cu permisiuni şi privilegii de control al accesului
6. Un model pureXML
7. C. Performanţă
8. E. Nici unul dintre enumerate
9. D. Integrare
10. B. Folosirea pureXML este principala diferenţă dintre DB2 şi celelalte sisteme relaţionale de gestiune a bazelor de date, pentru Cloud principala diferenţă este caracteristica Database Partitioning Feature (DPF). Aceasta permite scalabilitatea în cloud atunci când sunt oferite doar instanţe standard.

Capitolul 2

1. Constrângerea de integritate a entităţii sau constrângerea de unicitate împreună cu cea nenulă puse pe numărul de identificare al furnizorului, valoare nenulă pentru numele furnizorului, domeniul de valori şi valoarea nenulă pentru discountul

furnizorului.

2. Intersecţia: $R1 \cap R2 = R1-(R1-R2)$.
   Joncţiunea: $R1_{join\_condition} \blacktriangleright\blacktriangleleft R2 = \sigma_{join\_condition}(R1 X R2)$
   Împărţirea relaţiilor R1(A,B) şi R2(A):

$$R1 \div R2 = \pi_B(R1) - \pi_B((R2 X \pi_B(R1)) - R1)$$

3. $\pi_{Name}(\sigma_{Address='New\ York' \text{ AND } Discount>0.05}(R))$

4. RANGE OF SUPPLIERS IS SUPPLIERS.Name WHERE (SUPPLIERS.Address ='New York' AND SUPPLIERS.Discount>0.05)

5. NameX WHERE ∃ (DiscountX >0.05 AND SUPPLIERS (Name:NameX, Discount:DiscountX, Address:'New York')

6. A. Date din lumea reală modelate în baza de date.

   C. Caracteristică a datelor.

7. B. Într-o relaţie nu există tupluri duplicat.

   C. Atributele au valori atomice.

8. B. Instanţa.

   D. Gradul.

9. A. O cheie primară este în acelaşi timp şi cheie candidat.

10. D. Cascade.

Capitolul 3

1. D.
2. A.
3. C.
4. A.
5. B.
6. D.
7. B.
8. D.
9. C.
10. B.

Capitolul 4

1. Vedeţi exemplul din secţiunea 4.6.1 descompunere fără pierdere de informaţie
2. C
3. B
4. E
5. B
6. C

Capitolul 5

1. D.
2. D.
3. A, C.
4. D.
5. D.
6. A.
7. B.
8. D.
9. D.
10. B.

Capitolul 6

1. C
2. C
3. E
4. C
5. C
6. D
7. D
8. D
9. A
10. A

Capitolul 7

1. B.
2. A.

3. B.
4. A.
5. B.
6. B.
7. B.
8. A.
9. B.
10. B.

Capitolul 8

1. D.
2. B.
3. B.
4. D.
5. A.
6. A.
7. C.
8. A.
9. A.

Capitolul 9

1. Dezvoltarea tehnologiei informaţiei conduce la colectarea unor mari volume de date de către organizaţii. Aceste date acoperă domenii vaste de activitate şi se pot afla în spatele luării de decizii importante. Pentru a păstra secretul datelor, coerenţa şi disponibilitatea acestora trebuie luate măsuri de securitate.
2. Nu. Doar concentrarea pe securitatea bazelor de date nu va asigura siguranţa acestora. Toate părţile unui sistem trebuie să fie sigure: baza de date, reţeaua, sistemul de operare, clădirea în care se află baza de date precum şi persoanele care au posibilitatea să acceseze sistemul.
3. Într-un plan de securitate complet, există mai multe pericole ce trebuie avute în vedere. Acestea sunt: furtul şi frauda, pierderea caracterului privat şi al confidenţialităţii datelor, pierderea integrităţii datelor, pierderea disponibilităţii şi pierderea accidentală a datelor.
4. Controlul discreţionar al accesului este o metodă ce permite utilizatorilor să acceseze şi să efectueze diverse operaţii pe date în funcţie de drepturile de acces sau a privilegiilor pe obiectele bazei de date.

5. DB2 lucrează cu trei forme de autorizare: autoritatea administrativă, privilegii şi controlul accesului pe baza etichetelor
6. Privilegiile sunt autorităţi atribuite utilizatorilor, grupurilor sau rolurilor care permit acestora să efectueze diverse activităţi cu obiectele din baza de date.
7. Contextele de încredere stabilesc relaţii de încredere între DB2 şi o entitate externă, cum ar fi un server Web sau un server de aplicaţie. Această relaţie se bazează pe autorizarea sistemului, adresa IP şi fluxul de date criptat şi se defineşte cu ajutorul unui obiect de tip context de încredere în cadrul bazei de date.
8. O vedere este un tabel virtual obţinut ca rezultat dinamic al efectuării uneia sau mai multor operaţii relaţionale pe tabelele de bază. O vedere se poate crea pentru a prezenta doar datele pentru care utilizatorul primeşte acceptul să le vadă, fără a afişa datele care au caracter privat sau confidenţial.
9. Controlul integrităţii are ca scop protecţia datelor la accesul neautorizat şi la actualizare prin restricţionarea valorilor ce pot fi luate de către acestea în momentul efectuării unor operaţii. În acest scop, se pot folosi: definiţii de domenii, aserţiuni şi declanşatori.
10. Cele mai folosite politici şi proceduri de securitate sunt: controalele personalului şi controalele de acces fizic.

B

Anexa B – Instalarea şi rularea DB2

Această anexă introduce o serie de aspecte de bază necesare lucrului cu DB2. Anexa vine în sprijinul celor care doresc să instaleze şi să ruleze rapid şi uşor DB2.

În cadrul acestei anexe veţi învăţa despre:

- Pachetele DB2
- Instalarea DB2
- Instrumentele DB2
- Mediul DB2
- Configuraţia DB2
- Conectarea la o bază de date
- Exemple de programe de bază
- Documentaţia DB2

Obs:

Pentru mai multe informaţii despre DB2, vedeţi cartea în format electronic *Getting Started with DB2 Express-C* ce face parte din această serie de cărţi.

B.1 DB2: vedere de ansamblu

DB2 este un server de date ce permite păstrarea şi obţinerea în siguranţă a datelor. Comenzile DB2, XQuery şi SQL sunt folosite la interacţiunea utilizatorului cu serverul DB2 permiţând crearea de obiecte şi manipularea datelor într-un mediu sigur. Pentru a introduce aceste comenzi se folosec o serie de instrumente aşa cum se vede din *Figura B.1*. Această figură prezintă o vedere de ansamblu asupra DB2 şi provine din cartea în format electronic *Getting Started with DB2 Express-C*.

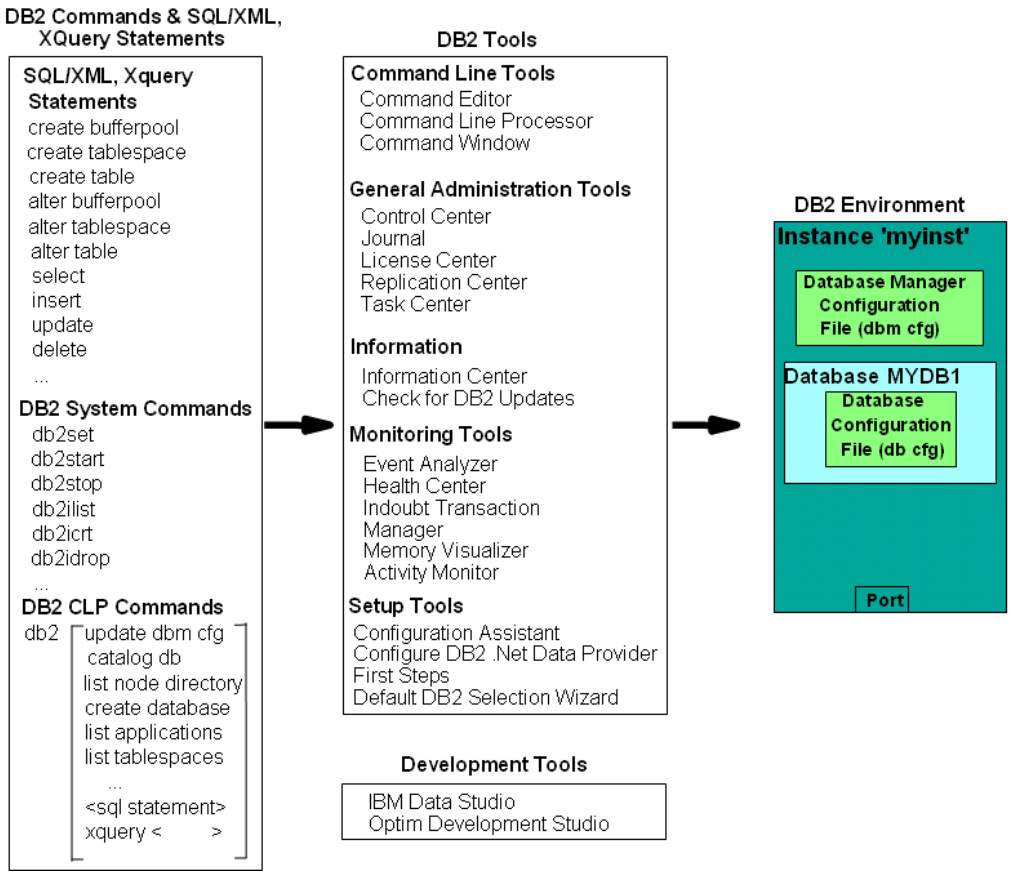


Figura B.1 - DB2 – Vedere de ansamblu

În partea stângă a figurii, se prezintă exemple de comenzi ce pot fi folosite de către utilizatori. În centrul figurii, se prezintă câteva instrumente în care se pot introduce aceste comenzi, iar în partea dreaptă a figurii se poate vedea mediul DB2 în care se păstrează baza de date. În secţiunile următoare vom discuta în detaliu despre unele dintre elementele ce apar în această figură.

B.2 Pachetele DB2

Serverele, clienţii şi driverele DB2 se creează folosind aceleaşi componente de bază, după care se împachetează într-o modalitate care permite utilizatorilor să aleagă funcţiile de care au nevoie la preţul corect. Această secţiune descrie diversele ediţii de DB2 sau pachete de produse disponibile.

B.2.1 Servere DB2

Figura B.2 prezintă o vedere de ansamblu a diferitelor servere de date DB2.

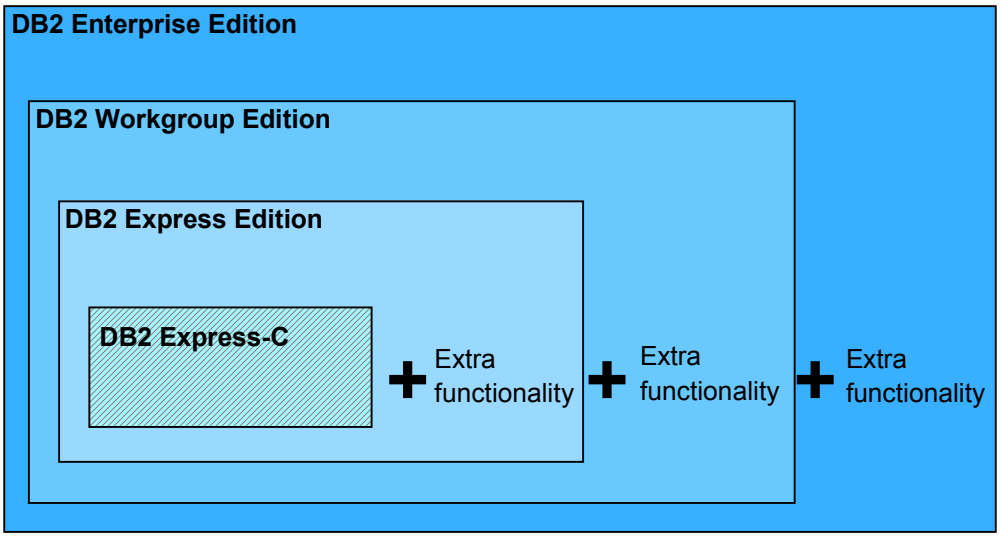


Figure B.2 – Pachetele de server DB2

Aşa cum se vede din *Figura B.2*, toate ediţiile de servere DB2 sunt construite unul pe baza celuilalt. DB2 Express-C este versiunea gratuită de DB2 şi este componenta de bază a tuturor produselor DB2. Prin adăugarea de funcţionalităţi suplimentare, aceasta devine DB2 Express. Prin adăugarea de funcţionalităţi suplimentare la DB2 Express, se obţine DB2 Workgroup ş.a.m.d. *Figura B.2* prezintă motivul pentru care este foarte uşor să se treacă de la DB2 Express-C la orice alt server DB2: *Toate ediţiile de servere DB2 sunt construite pornind de la DB2 Express-C*.

De asemenea, aplicaţiile create pentru DB2 Express-C sunt valabile în toate celelalte ediţii. Aplicaţiile create vor funcţiona fără a fi necesară nici un fel de modificare!

B.2.2 Clienţii si driverele DB2

La instalarea serverului DB2 se instalează şi o componentă de client DB2. Dacă se doreşte doar instalarea clientului, se poate alege între IBM Data Server Client, sau IBM Data Server Runtime Client. *Figura B.3* prezintă cei doi clienţi.

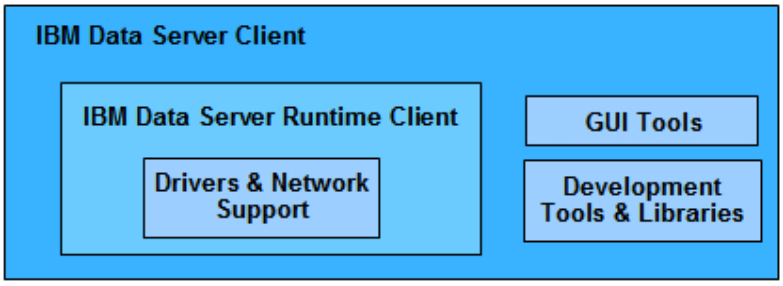


Figure B.3 – Clienţi DB2

Din figura de mai sus, se poate observa clientul IBM Data Server Runtime care are toate

componentele necesare (suport pentru driver şi reţea) pentru a se conecta şi a lucra cu serverul de date DB2. Clientul IBM Data Server are acelaşi suport, incluzând şi instrumentele de interfaţă grafică utilizator şi bibliotecile pentru elaborarea de aplicaţii.

Suplimentar, se mai oferă şi următorii clienţi şi drivere:

- DB2 Runtime Client Merge Modules pentru Windows: se foloseşte în principal pentru a încorpora un client DB2 ca parte a instalării unei aplicaţii Windows
- IBM Data Server Driver for JDBC and SQLJ: permite aplicaţiilor Java să se conecteze la serverele DB2 fără a fi necesar să se instaleze clientul
- IBM Data Server Driver for ODBC and CLI: permite aplicaţiilor ODBC şi CLI să se conecteze la serverul DB2 fără a fi necesar să se instaleze clientul
- IBM Data Server Driver Package: conţie un driver specific Windows cu suport pentru mediile .NET suplimentar ODBC, CLI şi open source. Acest driver se numea anterior IBM Data Server Driver for ODBC, CLI and .NET.

Clienţii şi driverele DB sunt gratuite.

B.3 Instalarea DB2

În această secţiune se arată cum se instalează DB2 folosind programul ajutător existent.

B.3.1 Instalare pe Windows

Instalarea DB2 pe Windows este simplă şi preuspune parcurgerea umătoarelor etape:

1. Se va folosi un cont local sau utilizator de domeniu care este parte a grupului Administrator de pe serverul pe care se instalează DB2.
2. După descărcarea şi dezarhivarea ediţiei DB2 Express-C for Windows de la adresa ibm.com/db2/express, se caută fişierul `setup.exe`, care se execută.
3. Se urmăresc instrucţiunile explicative din cadrul programului ajutător. Se aleg valorile implicite, ceea ce este suficient pentru început.
4. În timpul instalării se efectuează următoarele:
   - DB2 se instalează în `C:\Program Files\IBM\SQLLIB`
   - Se creează grupurile DB2ADMNS şi DB2USERS în cadrul sistemului de operare Windows.
   - Se creează instaţa **`DB2`** în `C:\Program Files\IBM\SQLLIB\DB2`
   - Se creează DB2 Administration Server (DAS)
   - Jurnalele de instalare se găsesc în:
     `My Documents\DB2LOG\db2.log`
     `My Documents\DB2LOG\db2wi.log`

- Se creează câteva servicii Windows.

B.3.2 Instalare pe Linux

Instalarea DB2 pe Linux este simplă şi preuspune parcurgerea umătoarelor etape:

Logarea ca utilizator Root pentru instalarea DB2.

1. După descărcarea DB2 Express-C for Linux de la adresa ibm.com/db2/express, se caută fişierul `db2setup`, care se execută: **`./db2setup`**
2. Se urmează instrucţiunile explicative din programul ajutător. Se aleg valorile implicite.
3. Pe parcursul instalării se efectuează următoarele în mod implicit:
   - Se instalează DB2 în `/opt/ibm/db2/V9.7`
   - Se creează trei identificatori de utilizatori cu valorile implicite:
     db2inst1 (proprietarul instanţei)
     db2fenc1 (utilizatori protejaţi, rutine protejate)
     dasusr1 (utilizatorDAS)
   - Se creează trei grupuri de utilizatori corespunzători identificatorilor anteriori:
     db2iadm1
     db2fadm1
     dasadm1
   - Se creează instanţa ***db2inst1***
   - Se creează ***dasusr1***
   - Jurnalele de instalare se găsesc în:
     `/tmp/db2setup.his`
     `/tmp/db2setup.log`
     `/tmp/db2setup.err`

B.4 Instrumente DB2

Există o serie de instrumente incluse în serverul de date DB2 cum ar fi DB2 Control Center, DB2 Command Editor ş.a.m.d. Începând cu versiunea DB2 9.7 aceste instrumente nu mai sunt de actualitate (adică, ele sunt suportate în continuare, dar nu mai sunt îmbunătăţite) folosindu-se în schimb IBM Data Studio. IBM Data Studio este oferit ca pachet suplimentar nefiind inclus în DB2. Vedeţi cartea în format electronic *Getting started with IBM Data Studio for DB2* pentru a obţine mai multe detalii.

B.4.1 Control Center

Înainte de versiunea DB2 9.7, instrumentul principal al DB2 pentru administrarea bazelor de date era Control Center, aşa cum se vede din *Figura B.4*. Acest instrument nu mai este de actualitate, dar există încă în serverele DB2.

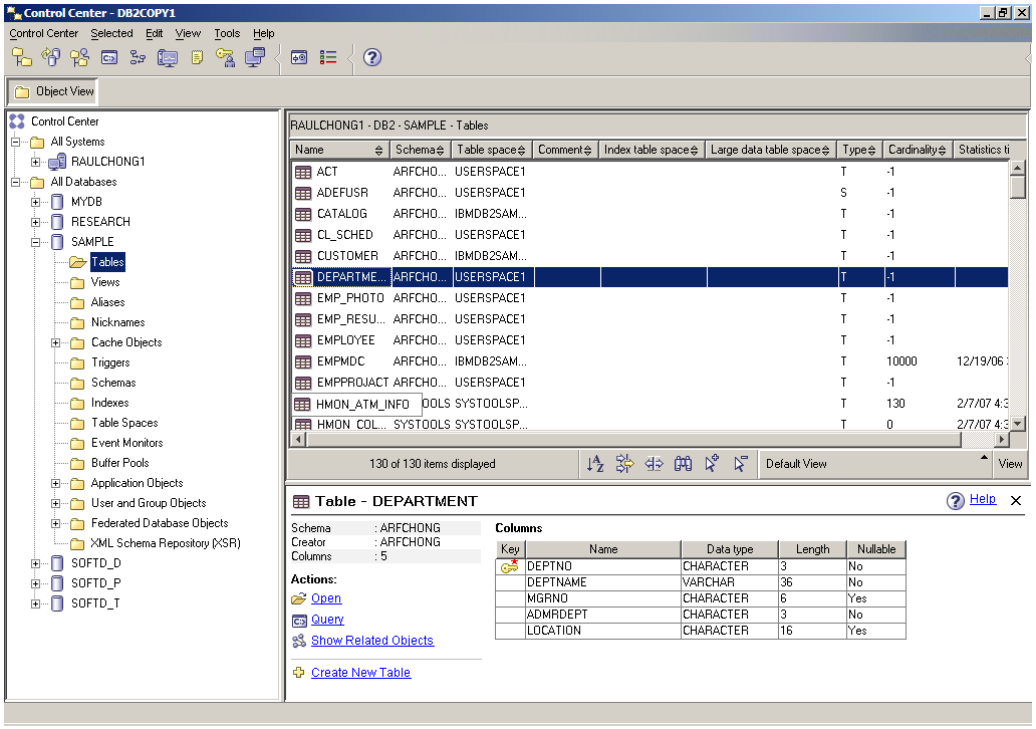


Figura B.4 - DB2 Control Center

Pentru a porni Control Center pe Windows se foloseşte *Start -> Programs -> IBM DB2 -> DB2COPY1 (Default) -> General Administration Tools -> Control Center* sau, se scrie comanda **`db2cc`** în Windows Command Prompt sau în Linux.

Control Center este un instrument centralizat de administrare ce permite:

- Urmărirea sistemului, instanţelor, bazelor de date şi a obiectelor din cadrul bazelor de date.
- Crearea, modificarea şi gestionarea bazelor de date şi a obiectelor acestora
- Pornirea altor instrumente grafice DB2

Fereastra din partea stângă prezintă ierarhia obiectelor bazei de date din cadrul sistemului, prezentând câte un folder pentru tabele, vederi etc. Atunci când se apasă pe un folder (de exemplu folderul Tables, aşa cum se vede din *Figura B.5*), fereastra din partea dreaptă sus va prezenta toate obiectele corespunzătoare, în acest caz toate tabelele existente în baza de date **`SAMPLE`**. Dacă se alege un anumit tabel din fereastra din dreapta sus, fereastra din dreapta jos va afişa o serie de informaţii despre tabel.

Prin apăsarea butonului din dreapta al mouse-ului, atunci când acesta este poziţionat pe foldere sau obiecte în arborele Object vor apare meniurile disponibile pentru acel folder sau obiect. De exemplu, prin apăsarea pe o instanţă şi alegerea comenzii *Configure parameters* se va permite vizualizarea şi actualizarea parametrilor la nivel de instanţă. La

fel se procedează pentru bazele de date.

B.4.2 Instrumente în linie de comandă

Există trei tipuri de instrumente în linie de comandă:

- DB2 Command Window (doar pe Windows)
- DB2 Command Line Processor (DB2 CLP)
- DB2 Command Editor (o interfaţă grafică utilizator, care nu se mai foloseşte)

Aceste instrumente sunt prezentate pe larg în secţiunile următoare.

B.4.2.1 DB2 Command Window

DB2 Command Window este disponibil doar pe sistemele de operare Windows şi se confundă deseori cu Windows Command Prompt. Deşi arată la fel, DB2 Command Window, iniţializează mediul de lucru cu DB2. Pentru pornirea acestui instrument, se foloseşte *Start -> Programs -> IBM DB2 -> DB2COPY1 (Default) -> Command Line Tools -> Command Window* sau se scrie comanda **`db2cmd`** din Windows Command Prompt pentru a se deschide într-o altă fereastră. *Figura B.5* prezintă DB2 Command Window.

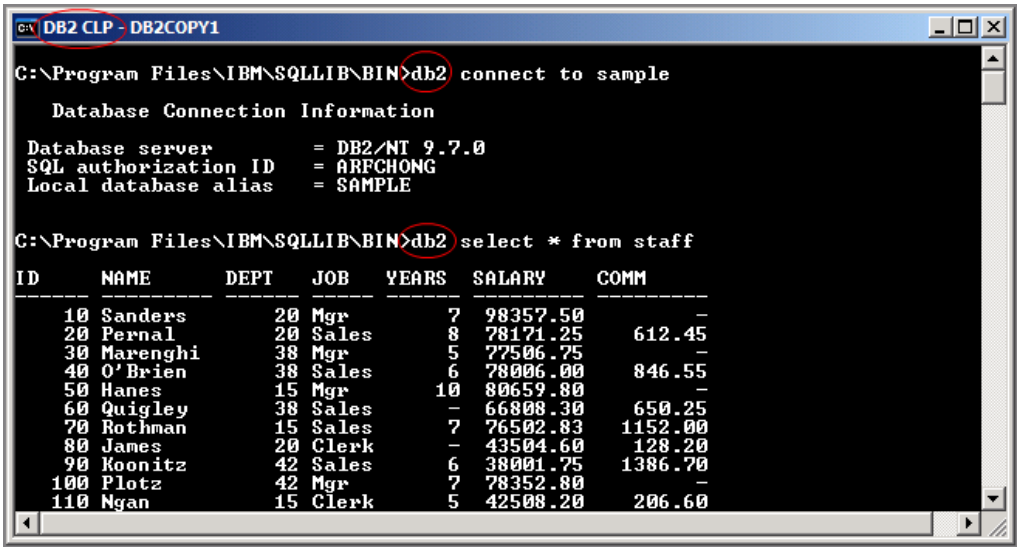


Figura B.5 - DB2 Command Window

Se poate afla cu uşurinţă că se lucrează cu DB2 Command Window dacă se citeşte titlul ferestrei, care întotdeauna conţine cuvintele *DB2 CLP* aşa cum se vede din figură. Din DB2 Command Window, toate comenzile trebuie scrise cu prefixul **`db2`**. De exemplu, în figura de mai sus, se văd două comenzi:

```
db2 connect to sample
db2 select * from staff
```

În Linux, echivalentul pentru DB2 Command Window este mediul Linux (aplicaţia Terminal) în care mediul DB2 se configurează prin folosirea fişierului `db2profile`. Acest filşier se creează implicit şi se adaugă fişierului `.login` pentru proprietarul instanţei DB2. Implicit, proprietarul instanţei este ***db2inst1***.

B.4.2.2 DB2 Command Line Processor

DB2 Command Line Processor (CLP) este asemănător cu DB2 Command Window, cu singura excepţie că prompterul este **db2=>** şi nu prompterul sistemului de operare. Pentru a porni DB2 Command Line Processor pe Windows, se folsoeşte *Start -> Programs -> IBM DB2 -> DB2COPY1 (Default) -> Command Line Tools -> Command Line Processor* sau se scrie în DB2 Command Window sau în Linux **db2** şi se apasă *Enter*. Prompterul se va schimba în **db2** aşa cum se vede în *Figura B.6*.

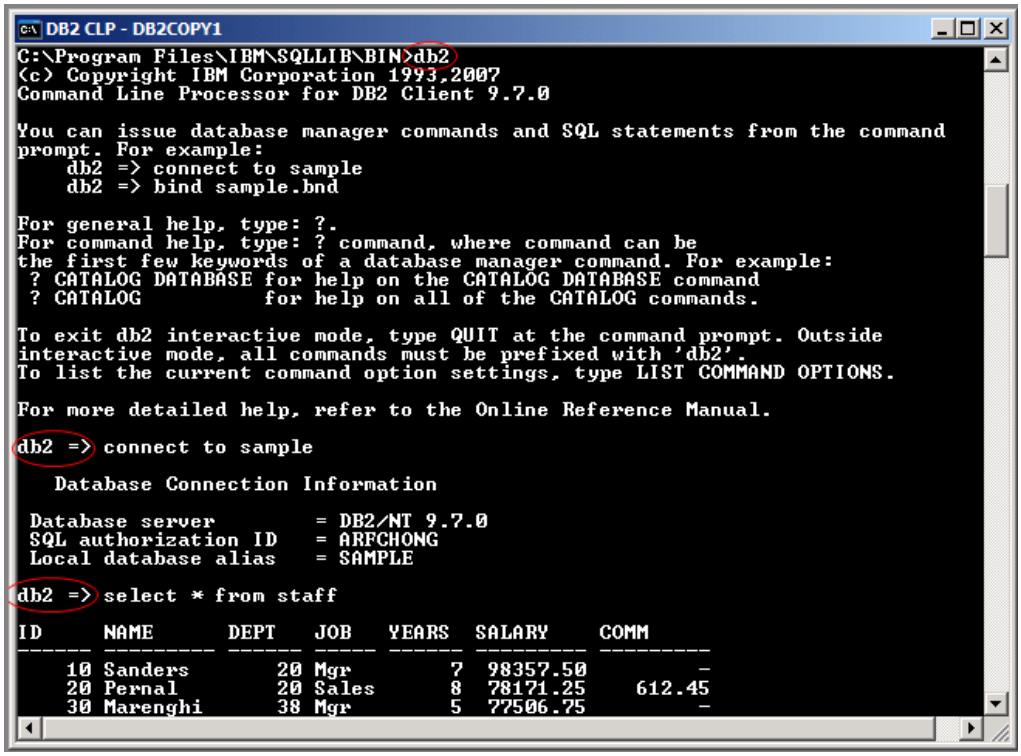


Figura B.6 - DB2 Command Line Processor (CLP)

Se observă şi faptul că *Figura B.6* mai arată şi că dacă se lucrează în CLP, nu mai trebuie prefixate comenzile cu DB2. Pentru a ieşi din CLP, se introduce comanda **quit**.

B.4.2.3 DB2 Command Editor

DB2 Command Editor este o versiune grafică utilizator a DB2 Command Window sau DB2 Command Line Processor aşa cum se vede din *Figura B.7*. Acest instrument nu se mai foloseşte în versiunea DB2 9.7.

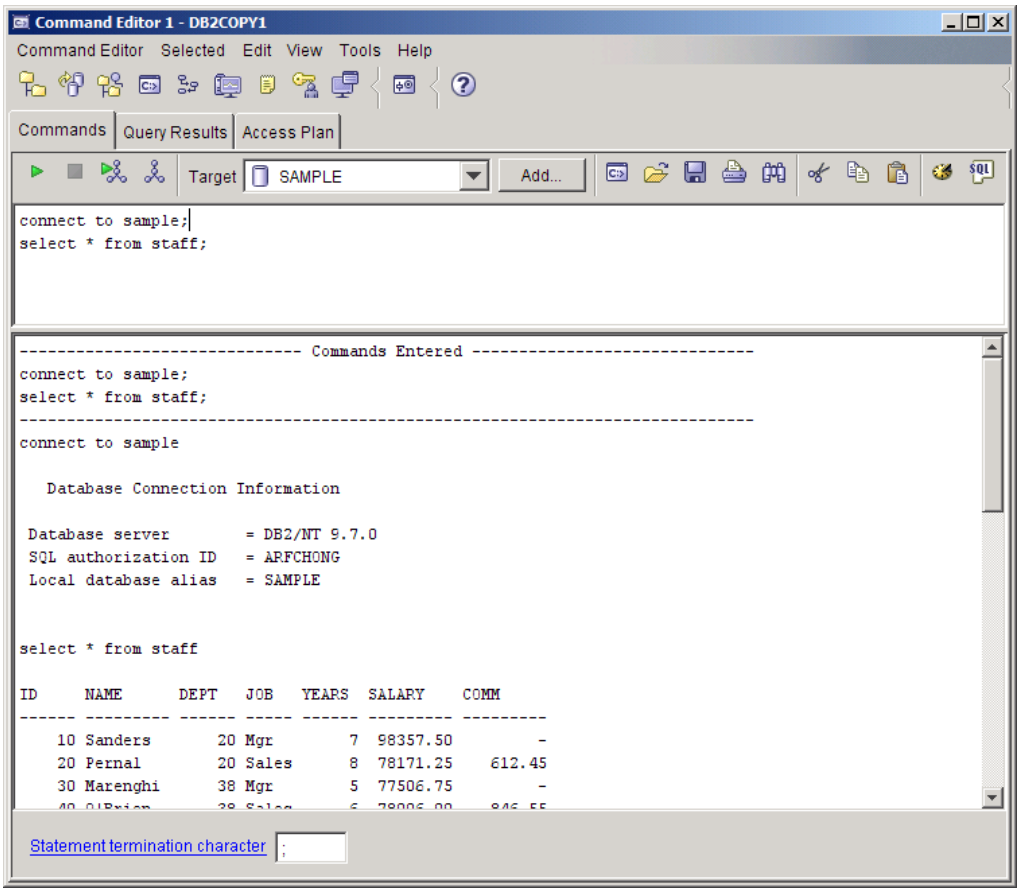


Figura B.7 - DB2 Command Editor

B.5 Mediul DB2

Figura B.8 oferă o vedere de ansamblu asupra mediului DB2.

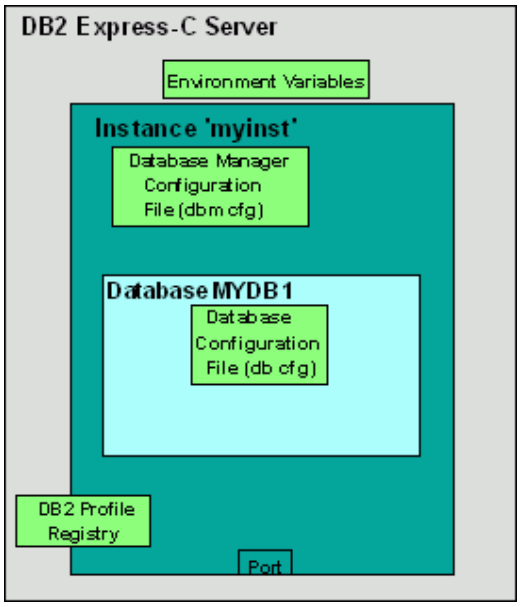


Figura B.8 - Mediul DB2

Figura prezintă instalat un server DB2 Express-C. Cutiile mai mici în verde deschis (Environment Variables, Database Manager Configuration File, Database Configuration File, DB2 Profile Registry) sunt zonele diferite în care poate fi configurat serverul DB2, acestea fiind explicate detaliat în secţiunea următoare. Cutia mai mare în verde închis reprezintă o instanţă care în exemplul de faţă are numele ***`myinst`***.

O ***instanţă*** este mediul în care se creează obiectele bazei de date. Pe acelaşi server se pot crea mai multe instanţe, fiecare dintre acestea fiind tratată separat. De exemplu, se poate folosi o instanţă pentru dezvoltare, alta pentru testare şi alta pentru producţie. *Tabelul B.1* prezintă câteva comenzi utile ce pot fi folosite la nivel de instanţă. Comenzile prezentate în cadrul acestei secţiuni se pot folosi şi în cadrul instrumentelor grafice DB2.

Comanda	Descrierea
db2start	Porneşte instanţa curentă
db2stop	Opreşte instanţa curentă
db2icrt <instance_name>	Crează o instanţă nouă
db2idrop <instance_name>	Elimină o instanţă
db2ilist	Afişează instanţele existente pe sistem

db2 get instance	Afişează instanţele active existente

Tabelul B.1 – Instanţe utile – comenzi la nivel de DB2

În cadrul unei instanţe se pot crea mai multe baze de date. O ***bază de date*** este o colecţie de obiecte, cum ar fi tabele, vederi, indecşi ş.a.m.d. De exemplu, în *Figura B.8*, baza de date ***`MYDB1`*** a fost creată în cadrul instanţei ***`myinst`***. *Tabelul B.2* prezintă câteva comenzi ce pot fi folosite la nivel de bază de date.

Comandă	**Descriere**
create database <nume_baza_de_date>	Creează o bază de date nouă
drop database <nume_baza_de_date>	Elimină o bază de date
connect to <nume_baza_de_date>	Se realizează conexiunea la o bază de date
create table/create view/create index	Comenzi SQL pentru crearea tabelelor, vederilor şi, respectiv, a indecşilor.

Tabelul B.2 - Comenzi la nivel de baze de date

B.6 Configurarea DB2

Parametrii DB2 pot fi configuraţi cu ajutorul instrumentului Configuration Advisor GUI. Configuration Advisor poate fi accesat prin intermediul Control Center apăsând butonul din dreapta al mouse-ului atunci când acesta se află deasupra unei baze de date şi alegând *Configuration Advisor*. În funcţie de răspunsurile la întrebări referitoare la resursele sistemului şi încărcare, se vor pune la dispoziţie o serie de parametrii DB2 care vor opera optimal prin intermediul valorilor sugerate. Dacă doriţi mai multe informaţii despre configurarea DB2, citiţi în continuare, altfel folosiţi Configuration Advisor şi totul este pregătit pentru lucrul cu DB2!

Serverul DB2 poate fi configurat la patru nivele diferite, aşa cum aţi văzut în *Figura B.8*:

- ***Environment variables*** sunt variabilele la nivel de sistem de operare. Principala variabilă de mediu care vă interesează în mod deosebit este DB2INSTANCE. Această variabilă arată instanţa curentă cu care lucraţi şi la ce comenzi DB2 se aplică.
- ***Database Manager Configuration File (dbm cfg)*** conţine parametrii care afectează instanţa şi toate bazele de date existente acolo. *Tabelul B.3* prezintă câteva comenzi utile pentru gestionarea dbm cfg.

Comanda	**Descriere**
get dbm cfg	Obţine informaţii despre dbm cfg

update dbm cfg using <nume_parametru> <valoare>	Actualizează valoarea parametrului dbm cfg

Tabelul B.3 – Comenzi pentru controlul dbm cfg

- ***Database Configuration File (db cfg)*** conţine parametrii care afectează o anumită bază de date. *Tabelul B.4* prezintă câteva comenzi utile pentru controlul db cfg.

Comanda	Descriere
get db cfg for <nume_baza_de_date>	Obţine informaţii despre db cfg pentru o anumită bază de date
update db cfg for <nume_baza_de_date> using <nume_parametru> <valoare>	Actualizează valoarea parametrului db cfg

Tabelul B.4 – Comenzi pentru controlul db cfg

- ***DB2 Profile Registry variables*** conţine parametri ce pot fi specifici platformei şi care pot fi configuraţi global (afectează toate instanţele), sau la nivel de instanţă (afectează doar o anumită instanţă). *Tabelul B.5* prezintă câteva comenzi care controlează registrul profilului DB2.

Comanda	Descriere
db2set -all	Afişează toate variabilele registrului de profil DB2 care pot fi configurate
db2set <parametru>=<valoare>	Introduce o valoare în cadrul unui parametru

Tabelul B.5 - Comenzi pentru controlul registrului de profil DB2

B.7 Conectarea la o bază de date

Dacă baza de date este locală, adică aceasta se află pe acelaşi sistem pe care se efectuează operaţii cu baza de date, setarea conexiunii se realizează automat la crearea bazei de date. Mai trebuie doar executată comanda `connect to nume_baza_de_date` pentru a vă conecta la baza de date respectivă.

Dacă baza de date se află la distanţă, cea mai simplă metodă pentru realizarea conexiunii la baza de date este folosirea instrumentului Configuration Assistant, urmând paşii:

1. Se porneşte Configuration Assistant de pe sistemul în care se doreşte conexiunea la baza de date. Pentru a porni acest instrument, se foloseşte comanda **`db2ca`** din Windows command prompt sau Linux. *Figura B.9* prezintă instrumentul Configuration Assistant.

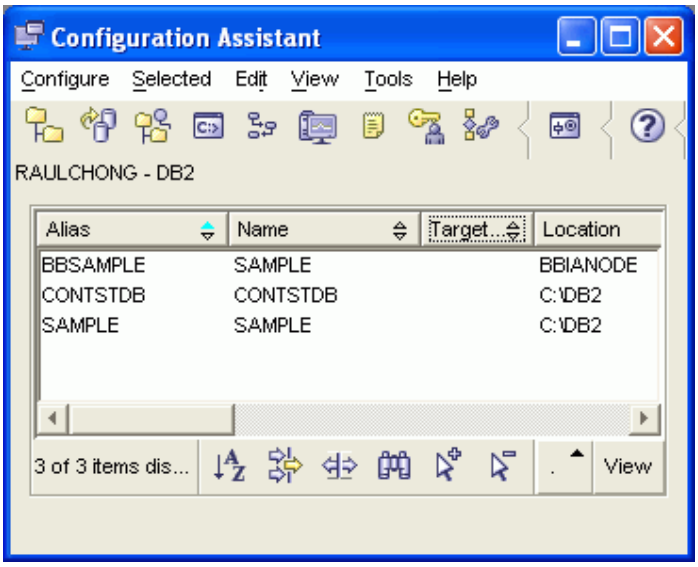


Figura B.9 – Instrumentul DB2 Configuration Assistant

2. Din Configuration Assistant, se alege meniul *Selected --> Add database using Wizard*
3. Din *Select how you want to set up a connection* window, se poate folosi *Search the network* dacă reţeaua este mică şi are mai multe dispozitive de conectare. Dacă se cunoaşte numele serverului pe care se află DB2, se alege *Known systems* şi se caută baza de date la care se doreşte să se realizeze conexiunea. Se foloseşte programul ajutător cu valori implicite. Dacă nu se cunoaşte numele sistemului se alege *Other systems* (*Search the network*). De remarcat că în acest caz este nevoie de mai mult timp dacă reţeaua este mare.
4. Dacă *Search the network* nu funcţionează, vă întoarceţi la fereastra *Select how you want to set up a* şi alegeţi *Manually configure a connection to a database*. Alegeţi TCP/IP şi apăsaţi *Next*. Introduceţi *hostname* sau *IP address* acolo unde se află serverul DB2. Introduceţi fie *numele serviciului* fie *numărul portului*.
5. Continuaţi să folosiţi programul ajutător şi folosiţi valorile implicite.
6. După terminarea configuraţiei, va apare o fereastră care vă întreabă dacă doriţi să se facă testarea conexiunii cu baza de date. Puteţi testa conexiunea şi după terminarea configurării, prin apăsarea butonului din dreapta al mouse-ului atunci când acesta se află deasupra bazei de date şi alegerea *Test Connection*.

B.8 Exemple de programe de bază

În funcţie de limbajul de programare folosit, se folosesc sintaxe diferite la conectarea cu baza de date, pentru a putea efectua operaţiile dorite. În continuare se prezintă o serie de link-uri către exemple de programe de bază care ajută la conectarea cu o bază de date pentru a putea obţine date. Vă sugerăm să descărcaţi toate aceste programe de la adresa

ftp://ftp.software.ibm.com/software/data/db2/udb/db2express/samples.zip)

Produsul CLI

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0401chong/index.html#scenario1

Produsul ODBC

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0401chong/index.html#scenario2

Produsul C cu codul SQL încorporat

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0401chong/index.html#scenario3

Produsul JDBC cu folosirea driverului Type 2 Universal (JCC)

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0401chong/index.html#scenario6

Produsul JDBC cu folosirea driverului Type 4 Universal (JCC)

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0401chong/index.html#scenario8

Produsul Visual Basic and C++ ADO – Folosirea provider-ului IBM OLE DB pentru DB2 (IBMDADB2)

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0402chong2/index.html#scenario1

Produsul Visual Basic and C++ ADO - Folosirea provider-ului OLE DB Provider pentru ODBC (MSDASQL)

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0402chong2/index.html#scenario2

Visual Basic and C# ADO.Net folosind IBM DB2 .NET Data Provider

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0402chong2/index.html#scenario3

Visual Basic and C# ADO.Net folosind Microsoft OLE DB .NET Data Provider

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0402chong2/index.html#scenario4

Visual Basic and C# ADO.Net folosind Microsoft ODBC .NET Data Provider

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0402chong2/index.html#scenario5

B.9 Documentaţia DB2

DB2 Information Center oferă online cele mai recente informaţii despre DB2. DB2 Information Center este o aplicaţie Web ce poate fi accesată la adresa (http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp), sau se poate descărca şi instala local. Adresa pentru DB2 Information Center, dar şi pentru versiunile ce pot fi descărcate este:

http://www.ibm.com/software/data/db2/9/download.html?S_TACT=download&S_CMP=expcsite

Resurse

Site-uri Web

1. **DB2 Express-C**

   ibm.com/db2/express

   Aceasta este pagina de pornire pentru DB2 Express-C. Pe această pagină se pot găsi link-uri pentru descărcarea DB2 Express-C.

2. **IBM Data Studio**

   http://www-01.ibm.com/software/data/optim/data-studio/

   Aceasta este pagina de pornire pentru IBM Data Studio, un instrument Eclipse ce poate fi folosit împreună cu DB2.

3. **Descărcarea DB2 Express-C şi a IBM Data Studio (gratuite)**

   http://www.ibm.com/db2/express/download.html?S_CMP=ECDDWW01&S_TACT=DOCBOOK01

4. **InfoSphere Data Architect**

   http://www-01.ibm.com/software/data/optim/data-architect/

Cărţi

1. Carte gratuită în format electronic: Getting started with DB2 Express-C (3rd Edition)

   Raul F. Chong et all - June 2009

   http://www.db2university.com

2. Carte gratuită în format electronic: Getting started with IBM Data Studio for DB2

   Debra Eaton et all - Dec 2009

   http://www.db2university.com

3. DB2 9 pureXML® Guide

   Whei-Jen Chen, Art Sammartino, Dobromir Goutev, Felicity Hendricks, Ippei Komi, Ming-Pang Wei, Rav Ahuja

   August 2007 - SG24-7315-01

   http://www.redbooks.ibm.com/abstracts/sg247315.html

4. Free Redbook®: DB2 Security and Compliance Solutions for Linux, UNIX, and Windows, Whei-Jen Chen, Ivo Rytir, Paul Read, Rafat Odeh, March 2008, SG 24-7555-00

   http://www.redbooks.ibm.com/abstracts/sg247555.html?Open

Bibliografie

[1.1] CODD, E.F. *A relational model of data for large shared data banks*, CACM 13, NO 6, 1970

[2.1] DATE, C.J. *An introduction to database* systems, Addison-Wesley Publishing Company, 1986

[2.2] MITEA, A.C. *Relational and object-oriented databases*, “Lucian Blaga” University Publishing Company, 2002

[2.3] CODD, E.F. *Relational completeness on data base sublanguage*, Data Base Systems, Courant Computer Science Symposia Series, Vol.6 Englewood Cliffs, N.J, Prentice-Hall, 1972

[2.4] KUHNS, J.L. *Answering questions by computer: A logical study*, Report RM-5428-PR, Rand Corporation, Santa Monica, California, 1967

[2.5] CODD, E.F. *A data base sublanguage founded on the relational calculus*, Proceedings ACM SIGFIDET Workshop on Data Description, Access and Control, 1971

[2.6]LACROIX, M., PIROTTE, A. *Domain oriented relational languages*, Proceedings 3rd International Conference on Very Large Data Bases, 1977

[2.7]LACROIX, M., PIROTTE, A. *Architecture and models in data base management systems*, G.M. Nijssen Publishing company, North-Holland, 1977

[3.1] IBM Rational Data Architect Evaluation Guide

[3.2] Connolly, T., Begg, C., Strachan, A. – Database Systems – A Practical Approach to Design, Implementation and Management, Addison Wesley Longman Limited 1995, 1998

[3.3] IBM InfoSphere Data Architect – Information Center

[3.4] http://www.ibm.com/developerworks/data/bestpractices/

[3.5] 03_dev475_ex_workbook_main.pdf, IBM Rational Software, Section 1: Course Registration Requirements, Copyright IBM Corp. 2004

[4.1] Codd, E. F. The Relational Model for Database Management

[4.2] Codd, E.F. "Further Normalization of the Data Base Relational Model."

[4.3] Date, C. J. "What First Normal Form Really Means"

[4.4] Silberschatz, Korth, Sudershan - Database System Concepts

[4.5] William Kent - A Simple Guide to Five Normal Forms in Relational Database Theory

[4.6] Raghu Ramakrishnan, Johannes Gehrke - Database management systems

[4.7] Vincent, M.W. and B. Srinivasan. "A Note on Relation Schemes Which Are in 3NF But Not in BCNF."

[4.8] C. J Date : An Introduction to Database Systems 8th Edition

[4.9] William Kent: A simple guide to five normal forms in relational database theory

http://www.bkent.net/Doc/simple5.htm

[4.10] Ronald Fagin, C J Date: Simple conditions for guaranteeing higher normal forms in relational databases

http://portal.acm.org/citation.cfm?id=132274

[4.11] Ronald Fagin: A Normal Form for Relational Databases That Is Based on Domains and Keys

http://www.almaden.ibm.com/cs/people/fagin/tods81.pdf

[4.12] C. J. Date, Hugh Darwen, Nikos A. Lorentzos: Temporal data and the relational model p172

[4.13] C J Date: Logic and databases, Appendix –C

[5.1] Differences between SQL procedures and External procedures

http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.apsg/db2z_differencesqlprocexternalproc.htm

[5.2] SQL Reference Guide

http://www.ibm.com/developerworks/data/library/techarticle/0206sqlref/0206sqlref.html

[6.1]
http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.apsg/db2z_differencesqlprocexternalproc.htm

Date de contact

Contact email:

Căsuţa poştală a programului DB2 on Campus: db2univ@ca.ibm.com

Parcurgerea cărţii Baze de date - Fundamente s-ar putea să nu fie uşoară. Citiţi această carte pentru a:

- **Afla la ce se folosesc bazele de date**
- **Înţelege modelele relaţional, de informaţii şi conceptual**
- **Învăţa cum se proiectează bazele de date**
- **Scrie comenzi SQL, funcţii şi proceduri în bazele de date**
- **Afla cum se pot integra datele XML şi cele relaţionale folosind DB2 pureXML**
- **Înţelege securitatea bazelor de date**
- **Efectua o serie de exerciţii**

Datele tind să devină cele mai importante bunuri critice ale unei afaceri. Prin pierderea datelor există riscul de pierdere a afacerii. Bazele de date sunt folosite peste tot, de la cele mai simple aplicaţii care păstrează informaţii individuale, până la cele mai complexe care păstrează informaţii despre milioane de utilizatori.

Această carte vă introduce în fascinanta lume a bazelor de date. Se prezintă elementele fundamentale ale sistemelor de gestiune a bazelor de date făcându-se referire la sistemele IBM DB2. Folosind DB2 Express-C, versiunea gratuită a DB2, puteţi trece în revistă toate elementele constitutive ale sistemelor de baze de date, ale bazelor de date, ale limbajelor SQL şi XML.

Cartea conţine exemple şi exerciţii ce vă ajută să vă îmbogăţiţi experienţa şi vă permit să aplicaţi conceptele bazelor de date.

Pentru a obţine mai multe informaţii despre fundamentele bazelor de date şi a subiectelor referitoare la gestiunea informaţiilor, vizitaţi:
ibm.com/developerworks/data/

Pentru a obţine mai multe informaţii sau pentru a descărca DB2 Express-C, vizitaţi:
ibm.com/db2/express

Pentru a socializa şi pentru a urmări materialele video vizitaţi:
channelDB2.com

Această carte face parte din seria de cărţi DB2 on Campus, cărţi în format electronic, gratuite. Aflaţi mai multe la adresa: db2university.com

9 780986 628375

Preţ: 24.99USD