Relational and Object-Oriented Methodology in Data Bases Systems

Marian CRISTESCU, Gabriel SOFONEA, Eugen COJOCARIU
Economic Informatics Department – "Lucian Blaga" University of Sibiu

Database programming languages integrate concepts of databases and programming languages to provide both implementation tools for data-intensive applications and high-level user interfaces to databases. Frequently, database programs contain a large amount of application knowledge which is hidden in the procedural code and thus difficult to maintain with changing data and user views. This paper presents a first attempt to improve the situation by supporting the integrated definition and management of data and rules based on a set-oriented and predicative approach. The use of database technology for integrated fact and rule base management is shown to have some important advantages in terms of fact and rule integrity, question-answering, and explanation of results.
Keywords*: data bases management, object-oriented data bases, integrity and scalability of the data bases components, object-oriented rules, object fragmentation.*

Introducere

Lumea reală este complexă şi orice încercare de a o reprezenta în modele simple cu ajutorul tehnologiei informaţiei de cele mai multe ori eşuează. Modelele tradiţionale de date, în special cele relaţionale, sunt relaţiv usor de implementat. Problema apare în momentul în care o aplicaţie încearcă să transpună datele reale într-un model de tip relaţional. Rezultatul este o mulţime de tabele care interacţionează în cele mai complexe moduri şi simulează aproximativ relaţiile cu adevarat existente dar care va creşte în dimensiune foarte mult şi repde. În plus, conexiunile dintre aceste tabele sunt deseori ascunse în programele de aplicaţie, în afara bazei de date, acolo unde ar putea fi mai usor administrate. Nu este nici un secret faptul ca modelul relaţional de date este unul dintre cele mai mari impedimente pentru o funcţionare eficientă a aplicaţiilor.

În schimb o bază de date post-relaţională combină viteza şi scalabilitatea unui model tranzacţional, multidimensional cu puterea şi flexibilitatea tehnologiei de programare orientată pe obiecte. Datorită caracteristicilor sale unice, o bază de date post-relaţională este unul din cele mai performante medii de dezvoltare a unor aplicaţii tranzacţionale de înaltă performanţă.

Modelele de date multidimensionale uşurează procesul de modelare, deoarece pot include structuri reale complexe potrivind relaţiile într-un model tehnologic mai accesibil. Pe scurt, bazele de date relaţionale transpun realitatea în două dimensiuni, în timp ce bazele postrelaţionale permit producerea unei imagini reale a lumii atât de complexe a datelor.

De exemplu dacă o aplicaţie de tipul unui magazin virtual pe Internet nu are în stoc marimea, culoarea sau stilul unui articol dorit de un client, magazinul va pierde un client şi prin urmare o vânzare. Deci inventarul trebuie administrat corect. Cu un model de date relaţional, este nevoie de construirea mai multor tabele. Într-un tabel se vor urmări, de exemplu, culorile, stilul şi dimensiunile articolelor din gestiune. În mai multe tabele se va ţine o evidenţă a preţurilor produselor, a costurilor, a clienţilor şi multe altele. Aceste tabele vor trebui actualizate la zi după fiecare operaţiune efectuată (de exemplu o vânzare). În această situatie baza de date va creşte foarte repede în marime şi va deveni supraaglomerată, deci toate operaţiile cu date se vor desfăşura lent.

În cazul multidimensional, datele sunt stocate într-un "cub", care are tot atâtea feţe câte sunt necesare pentru a descrie complet şi simultan toate datele. Accesul la o astfel de baza de date este extrem de rapid şi, fiindcă informaţia redundantă a fost eliminată, baza este compactă. Prin urmare, bazele multidimensionale nu doar se comportă mai bine,

însă sunt şi mai economice din punct de vedere al spaţiului şi timpului de acces. Putem spune că acestea înglobează toate atributele unui SGBD modern şi extrem de eficient.

Facilităţi şi restricţii la nivel relaţional şi post-relaţional

Un exemplu de S.G.B.D. care lucrează cu baze de date post-relaţionale este sistemul Caché [ZIEM02]. O aplicaţie reprezentativă pentru acesta este cea în care o agenţie de turism caută zborurile disponibile şi face rezervările necesare. Un sistem de rezervare a locurilor în traficul aerian foloseşte date complexe, se adresează unui numar mare de utilizatori şi se potriveşte perfect la capacitatea de lucru cu obiecte a sistemului Caché. Orice programator fie că a lucrat mai mult sau mai puţin cu baze de date este conştient de faptul că în ziua de azi destul de puţine structuri care să nu fie orientate obiect se mai pot concepe.

Administratorii de reţea îşi doresc aplicaţii care lucrează rapid, eficient şi care să fie scalabile pentru reţelele lor, rareori omogene. Tehnologia bazelor de date post-relaţionale are şi unele probleme legate în special de procesarea tranzacţiilor într-un mediu on-line, cu mulţi utilizatori, de viteză redusă şi interoperabilitatea limitată cu standardele şi produsele cele mai populare. Soluţia acestor probleme este sistemul Caché. Acesta a reuşit sa rezolve problema de viteză redusă în mediul online cu mulţi utilizatori.

Pentru a demonstra performanţele şi eficienţa tehnologiei Caché, cei de la InterSystems au facut mai multe sudii de caz printre care şi cel prezentat în continuare [ZIEM02]. Astfel s-a folosit o aplicaţie din domeniul sănătaţii care a fost supusă unui test pentru a compara durata de raspuns la unele interogări a unei baze de date post-relaţionale (Caché) şi durata de răspuns pentru interogarea unei baze de date relaţionale, ambele folosindu-se de SQL. Astfel folosind date ale pacienţilor (7 tabele şi peste 6,5 milioane de întregistrări), un set de 8 interogări au fost executate simultan de către 30, 60, 90 şi 120 de utilizatori.

Tabelul 1 prezintă rezultatele la care au ajuns cei de la InterSystems în urma acesor teste la care a fost supusă aplicaţia. Datele sunt luate din studiul de caz efectuat [ZIEM02]. Timpii de execuţie a interogărilor sunt exprimaţi în milisecunde pentru ambele baze de date.

Tabelul 1. Rezultatele studiului (Case Studies in Performance, InterSystems Corporation - [ZIEM02])

Utilizatori concurenţi	**Timp de răspuns pentru 8 interogări**		**Performanţa relativă**
	RDBMS	Caché	Caché:RDBMS
30	375.125 ms	59.125 ms	6.3:1
60	637.25 ms	137.75 ms	4.6:1
90	915.625 ms	206.875 ms	4.2:1
120	1146.375 ms	290.125 ms	3.9:1

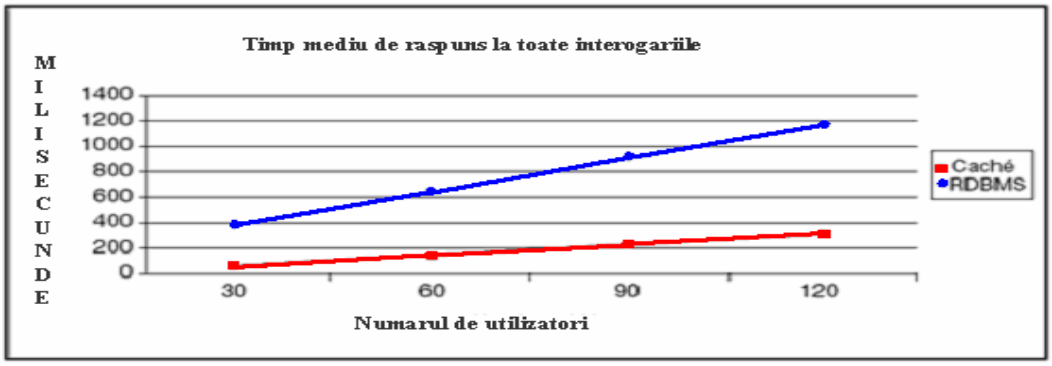

Fig.1. Diferenţa dintre post-relaţional şi relaţional (Case Studies in Performance InterSystems Corporation - [ZIEM02])

Ambele au fost rapide dar Caché a avut în medie timpi de răspuns de 4 - 6 ori mai rapizi. După cum este reprezentat şi pe grafic Caché este mai scalabil. De exemplu când se trece la testarea de la 90 la 120 de utilizatori în cazul Caché timpul de răspuns scade cu 84 milisecunde pe când în cazul bazei de date relaţionale scade cu 231 milisecunde adică cu 2,7 mai mult.

Una dintre cele mai importante limitări a modelului relaţional rezolvată de către cel obiectual este cea legată de abilitatea limitată a bazelor de date relaţionale de a lucra cu date de tip BLOB (Binary Large OBjects). Acestea sunt tipuri de date complexe care conţin documente, mesaje e-mail, structuri ale directoarelor şi altele. La nivelul de bază datele sunt secvenţe de biti ('1' sau '0'). Tradiţional bazele de date sunt create pentru a suporta şiruri mici de biţi ce reprezintă date de tip numeric sau caracter (de lungime limitată). Un stream de biţi ce reprezintă o dată, este un atom ce nu poate fi rupt în bucaţi mai mici. Tipurile de date BLOB sunt date mari ne-atomice, acestea au părţi şi subpărţi şi nu sunt reprezentate uşor în bazele de date relaţionale. O bază de date relaţională poate stoca date BLOB dar acestea le stochează în afara bazei de date iar acestea sunt referite cu ajutorul pointerilor. Pointerii permit bazelor de date relaţionale să caute datele BLOB, acestea fiind manipulate de fişiere de intrare/ieşire. Bazele de date orientate–obiect oferă suport pentru tipurile BLOB .

Tabelul 2. Analiza comparativă a limitărilor şi avantajelor

Baza de date relaţională	**Baza de date post-relaţională**
Eficienţă scazută faţă de bazele de date post-relaţionale	Eficienţă sporită în dezvoltarea aplicaţiilor
Simplitate. Uşor de implementat	Complexitate. Sunt mai puţin cunoscute şi utilizate
Limitări în lucrul cu tipul de date BLOB.	Suport deplin în lucrul cu date BLOB.
Viteză şi scalabilitate	Viteza marită şi scalabilitate + puterea şi flexibilitatea tehnologiei de programare orientată pe obiecte.
Toate funcţiile şi procedurile folosite trebuie declarate ca proceduri stocate	Posibilitatea încapsulării operaţiilor şi faptul că obiectele oferă o perspectivă de întreg asupra datelor deci sunt mai uşor de reprezentat şi gestionat.
Transpunerea unui eveniment complex conduce la o mulţime de tabele care interacţionează în cele mai complexe moduri şi simulează aproximativ relaţiile cu adevărat existente dar care va creşte în dimensiune foarte mult şi repede şi de cele mai multe ori conexiunile sunt făcute în cadrul codului aplicaţiilor unde sunt mai greu de administrat şi verificat	Transpune uşor, rapid şi transparent evenimentele complexe din lumea reală în obiecte. Obiectele îşi implementează metodele şi funcţiile lor proprii care trebuie doar apelate în interiorul funcţiilor şi porocedurilor stocate sau interogărilor.

Baze de date distribuite

O bază de date distribuită reprezintă un set de baze de date aflate pe mai multe calculatoare care este văzut de către aplicaţie ca fiind o singură bază de date (aflată pe un singur calculator) – adică baza de date vazută de către aplicaţie este fragmentată şi împărtită pe mai multe calculatoare din reţea. Fragmentele pot fi replicate. O bază de date se spune că este distribuită dacă diferitele componente ale acesteia sunt memorate în staţiile şi/sau serverul reţelei. O bază de date distribuită este o bază de date (o colecţie ordonată de date) care se află sub controlul unui sistem de management al bazelor de date (DBMS – DataBase Management System). Aceasta poate fi stocată pe mai multe calculatoare aflate în aceeaşi locaţie fizică sau poate fi distribuită în reţea pe staţiile componente. Unele din cele mai cunoscute sisteme de management al bazelor de date sunt DB2, Microsoft SQL Server, MySQL, Oracle şi PostgreSQL. Colecţiile de date pot fi distribuite pe mai multe locaţii fizice.

Când se vorbeşte despre o bază de date distribuită trebuie avute în vedere următoarele caracteristici:

- Distribuţia este transparentă – utilizatorii trebuie să poată interacţiona cu sistemul ca şi când acesta ar fi unul logic. Această caracteristică se aplică, printre altele, la performanţele sistemului şi metodele de acces;
- Tranzacţiile sunt transparente – fiecare tranzacţie trebuie să menţină integritatea bazei de date peste baze de date multiple. Tranzacţiile trebuie şi ele să fie împărţite, fragmentate în subtranzacţii, fiecare subtranzacţie afectând o bază de date.

Adevăratul sens al atributului "distribuit“ în contextul SGBD-urilor corespunde nu faptului că sistemul permite accesarea datelor de la distanţă (prin reţea), ci a acelor implementări care îngăduie aplicaţiilor şi utilizatorilor să trateze baza de date ca pe un singur depozit logic chiar dacă datele constituente sunt repartizate în mai multe locaţii ale reţelei (transparenţa completă a localizării datelor).

Ţinând cont că de obicei volumul şi complexitatea datelor elimină idealul "un computer foarte performant care cumulează întreaga bază de date şi deserveşte toţi utilizatorii organizaţiei“ trebuie găsită o soluţie de compromis. Devine foarte interesantă colecţia de criterii practice de distribuire a datelor. În cazul fiecarei implementări trebuie avut în vedere:

- nu trebuie niciodată pierdut din vedere dezideratul vitezei;
- limita de stocare şi puterea calculatoarelor gazdă;
- limita de transfer a reţelei;
- preferabil ca fiecare aplicaţie să acceseze uzual un singur depozit al bazei de date (fară a împiedica accesarea cu frecvenţă redusă a celorlalte noduri ale reţelei);
- folosirea funcţiilor *two-phase-commit* existente pentru a asigura integritatea datelor actualizate în mod distribuit;
- planificarea, controlul şi minimizarea duplicării de obiecte ale bazei de date;
- corelarea organizării cu facilităţile de optimizare distribuită ale SGBD-ului.

Chris Date a enunţat cele 12 cerinţe cărora trebuie să se supună bazele de date distribuite,[IESC1]:

1. Autonomia locală: datele locale sunt deţinute şi administrate local - nici un post nu depinde de altele pentru a funcţiona;
2. Toate posturile sunt egale: nici un post nu se bazează pe o staţie centrală. Funcţionare neîntreruptă: nu trebuie să fie necesară o oprire planificată (instalările/ştergerile efectuate la un post nu afectează funcţionarea celorlalte). Fiecare staţie are propriul sau dictionar de date;
3. Transparenţa amplasării: utilizatorii nu sunt obligaţi să ştie unde sunt amplasate datele pentru a le extrage.
4. Transparenţa fragmentării: relaţiile dintre componentele bazei de date pot fi fragmentate pentru stocare, dar acest lucru rămâne transparent pentru utilizator. Se poate utiliza atât fragmentarea pe verticală, cea pe orizontală cât şi cea mixtă;
5. Transparenţa duplicării: relaţiile şi fragmentele pot fi reprezentate fizic prin copii multiple stocate separat, dar transparent pentru utilizator;
6. Prelucrarea interogărilor distribuite: operaţiile de citire/scriere se pot desfăşura la mai multe posturi, permiţând optimizarea locală şi globală a interogărilor;
7. O tranzacţie poate să facă update, să insereze sau să şteargă date din mai multe baze de date. Tehnologia Oracle asigură integritatea tranzacţiilor distribuite;
8. Actualizările distribuite: tranzacţiile singulare pot executa codul la mai multe posturi;
9. Independenţa de hardware: toate calculatoarele participă ca membrii egali;
10. Independenţa de sistemul de operare: sunt suportate mai multe sisteme de operare conectabile la reţea. Standardul Oracle funcţionează pe majoritatea platformelor;
11. Independenţa de reţea: sunt suportate mai multe reţele prin protocoale comune. Independenţa de bazele de date: se asigură accesul uniform (interfaţare unică) pentru datele provenind din SGBD-uri diferite. Softul Oracle SQL*Net suportă majoritatea sistemelor de operare pentru reţea. Multi protoco-

lul Oracle permite aplicaţiilor să comunice cu bazele de date prin intermediul multiplelor reţele;
12. Independenţa sistemului de managemement al bazelor de date (DBMS) : aceasta este reprezntată de abilitatea de a integra baze de date diferite. Tehnologia Open Gateway de la Oracle suportă conexiuni şi la baze de date care nu sunt Oracle.

Avantajele oferite de bazele de date distribuite:

1. Capacitatea şi creşterea continuă – este un avantaj al bazelor de date distribuite reprezentat prin faptul că pe măsură ce informaţia creşte în volum noi fragmente ale bazei de date pot fi adăugate pe noi staţii din reţea. În cazul unui sistem centralizat această crştere a bazei de date ar avea ca urmări creşterea cerinţelor hardware şi software pentru serverul pe care se află;
2. Reflectă structura organizatorică – fragmentele bazei de date sunt plasate în departamentele cu care sunt înrudite;
3. Autonomia locală – un departament local poate controla datele;
4. Interfaţa prietenoasă pentru utilizatori;
5. Un alt avantaj este acela că fiecare utilizator aflat la o staţie din reţea poate accesa datele aflate pe o alta staţie şi în acelaşi timp să menţină controlul asupra datelor stocate pe staţia lor de lucru;
6. Disponibilitate sporită – o eroare într-o bază de date va afecta doar un fragment şi nu întreaga bază de date. Folosind replicarea datelor în cazul în care în baza de date aflată pe o staţie care nu funcţionează este permis accesul la datele replicate (copia de siguranţă) aflate pe altă staţie. Restul staţiilor continuă să funcţioneze în mod normal;
7. Performanţa sporită – datele sunt localizate în apropierea locurilor cu cele mai multe cereri şi sistemele de baze de date sunt "paralizate" permiţând astfel ca informaţiile din baza de date să circule între servere şi prin urmare să asigure accesul mai rapid la date şi procesarea mai rapidă;
8. Costuri reduse (economii) din punctul de vedere hardware al staţiilor din reţea – costă mai puţin să construieşti o reţea de calculatoare cu capacităţi mai reduse decât să achiziţionezi un calculator puternic.
9. Modularitatea – sistemele pot fi modificate, adăugate sau înlăturate din cadrul bazei de date distribuite fără a afecta alte module (sisteme).

Dezavantaje ale bazelor de date distribuite:

1. Complexitatea – necesită muncă în plus din partea administratorului bazei de date pentru a asigura transparenţa sistemului. Munca în plus trebuie deasemenea depusă pentru a menţine fragmentele bazei de date împrăştiate pe reţea faţă de efortul de a menţine o singură bază de date mare. O complexitate sporită şi o reţea întinsă înseamnă costuri şi muncă în plus. De asemenea erorile sunt mai greu de evitat;
2. Dificultatea de a menţine integritatea – în cazul unei baze de date menţinerea integrităţii într-o reţea poate duce la un consum prea mare al resurselor reţelei;
3. Lipsa de experienţă – lucrul cu bazele de date distribuite este dificil, acestea fac parte dintr-un domeniu nou care încă nu este destul de cunoscut existând o experienţă scazută în acest domeniu;
4. Mentenanţa – întreţinerea bazelor de date este dificilă datorită fragmentării sale şi răspândirea pe mai multe staţii ale reţelei.

Utilizatorii bazelor de date distribuite trebuie să ştie că acestea sunt împărţite, partiţionate, stocate pe mai multe staţii ale reţelei. Teoretic un fragment poate fi considerat în cazul bazelor de date relaţionale un întreg tabel sau în cazul celor post-relaţional un obiect. Fragmentarea datelor ne permite să "rupem" un obiect în două sau mai multe segmente sau fragmente. Informaţiile despre fragmentarea datelor sunt stocate în catalogul de date distribuite (DDC), de unde sunt accesate de către procesorul de tranzacţii(TP) pentru a procesa cererile utilizatoriilor.

Există trei moduri de fragmentare: orizontală, verticală sau mixtă.

Fragmentarea orizontală se referă la fragmentarea obiectului, a unui tabel pe tuple (rânduri). Fiecare fragment este stocat într-un nod diferit şi fiecare fragment are rânduri diferite. Fiecare fragment reprezintă echiva-

lentul a unei interogari SELECT cu clauza WHERE setată pe un singur atribut.
Fragmentarea verticală se referă la divizarea pe baza atributelor (coloanelor). Fiecare fragment va conține coloane unice, cu excepția cheii primare, şi va fi stocat în noduri diferite. Aceasta este echivalentă cu clauza PROJECT.
Fragmentarea mixtă este o combinație între fragmentarea orizontală şi cea verticală. Astfel un obiect (de exemplu un tabel) poate fi divizat în mai multe subseturi orizontale (rânduri), fiecare având un subset de atribute (coloane).
Un punct forte al bazelor de date distribuite îl constituie *replicarea datelor*. Aceasta se referă la copiile datelor stocate pe diferite stații din rețea. Astfel toate datele sunt replicate (copiate) pe diferite calculatoare din rețea .

- o copie a unui fragment din baza de date distribuită este stocat pe mai multe stații din rețea;
- existența acestor copii ale fragmentelor de date pe diferite stații ale rețelei sporesc disponibilitatea datelor în rețea şi scad timpul de răspuns al interogărilor.

Fragmentele de date obținute în urma partiționării bazei de date distribuite vor fi alocate, distribuite, pe stații individuale din rețea ținând cont în alocarea lor de stațiile care le vor solicita cel mai des. Deciziile cu privire la modul de distribuire a fragmentelor pe diferitele stații din rețea se bazează pe costurile şi beneficiile pe care le implică modul de a avea un fragment de date pe o anumită stație unde:

- beneficiul se referă la numărul de cereri formulate pentru acel fragment de date;
- costurile se referă la numărul de schimbări ca rezultat al unei tranzacții de la o altă stație (costurile cresc o dată cu numărul de tranzacții).

Maparea relațional-obiectuală

Maparea relațional-obiectuală sau obiectual-relațională sau O/RM este o *tehnică de programare* care leagă bazele de date de limbajele de programare orientate-obiect, creând o bază virtuală de obiecte. Există pachete comerciale şi gratuite disponibile pe piață care efectuează maparea între obiect şi relațional, deşi unii programatori preferă să-şi realizeze şi utilizeze propriul cod pentru mapare.
În programarea orientată-obiect, obiectele reprezintă entități din lumea reală. Problema intervine la realizarea de *obiecte persistente*, atunci când este necesară translatarea acelor obiecte sub forme care să poată fi stocate în fişiere sau în baze de date, şi, ulterior, *obținerea obiectelor inițiale din fişiere sau baze de date*, cu toate proprietățile şi relațiile existente. Din punct de vedere istoric, au existat câteva soluții pentru rezolvarea problemei, dintre care cele mai importante sunt : programele de mapare relațional-obiectual, sistemele de baze de date obiectuale.
Bazele de date relaționale tradiționale nu permit stocarea obiectelor. Folosirea acestora este totuşi posibilă, dar duce la un efort dublu de programare : programatorii trebuie să asigure funcționarea programelor de aplicații cu baze de date la două nivele diferite : procesarea datelor s-ar face într-o manieră orientată-obiect, pe de o parte, iar aceleaşi trebuie stocate într-o bază de date după modelul relațional, pe de altă parte. Necesitatea acestei conversii constante a aceloraşi date între cele două forme nu numai că are efecte asupra performanțelor programelor, dar aduce şi greutăți d.p.d.v. al programării programatorilor, întrucât modelul obiectual şi cel relațional implică limitări unul asupra celuilalt. De exemplu, bazele de date relaționale transpun relațiile complexe între entități într-o manieră destul de complicată şi dificilă şi pot fi cu greu mapate în modelul obiectual, neputând să implementeze tipuri definite de utilizator.
În bazele de date relaționale un obiect din lumea reală este deseori stocat pe părți în mai multe tabele, iar obținerea obiectului respectiv în vederea procesării de către programele de aplicații necesită reunirea acestor date din toate tabelele. În modelul obiectual relațiile între componentele unui obiect sunt mult mai clare şi mai uşor de reprezentat, se poate lua exemplul unei persoane dintr-o agendă telefonică care are un număr de telefon ce poate fi stocat în cadrul obiectului persoană. Acest

lucru nu etse valabil pentru modelul relaţional în care numărul de telefon al persoanei este stocat într-un tabel separat, tabelele nu au nici o idee de modul în care sunt relaţionate cu alte tabele la un nivel fundamental. Utilizatorul trebuie, în schimb, să emită o interogare cu join şi să specifice modul de relaţionare al tabelelor pentru a colecta informaţiile, pentru că tabelele ca atare în baza de date nu pot să se relaţioneze între ele. De fapt, la nivelul bazei de date se cunosc într-o anumită măsură relaţiile dintre tabele prin intermediul restricţiilor, dar o interogare SQL este independentă de acestea.

Modelul relaţional poate fi translatat într-un model obiectual dacă se au în vedere următoarele aspecte :

- o entitate din modelul relaţional poate fi replicată sub forma unei entităţi din modelul obiectual, ambele fiind descrise prin intermediul unor atribute ;
- o instanţă a unei entităţi este reprezentată în modellu relaţional printr-o înregistrare într-o tabelă, iar în modelul obiectual printr-un obiect ;
- relaţiile între entităţi sunt reprezentate în acelaşi mod în ambele modele, doar că sunt implementate în mod diferit ;

Unele sisteme de mapare O/R menţin obiectul încărcat în memorie în legătură constantă cu baza de date. Acest lucru este posibil astfel : datele obţinute prin interogare din baza de date sunt copiate în câmpurile unui obiect de tipul corespunzător ; odată obţinut obiectul, la fiecare modificare a atributelor sale, acesta trebuie să realizeze procesul invers de scriere a modificărilor înapoi în baza de date.

Având în vedere cele două lumi foarte diferite, codul (orientat-obiect) necesar pentru a lucra cu bazele de date relaţionale tinde să devină foarte complex şi predispus la erori. Dezvoltatorii de soft pentru aplicaţii cu baze de date au căutat soluţii mai bune pentru a obţine persistenţa obiectelor lor.

Au fost dezvoltate o serie de pachete comerciale care realizează maparea între relaţional şi obiectual în locul programatorului. Aceste sisteme oferă biblioteci de clase care efectuează automat maparea. Având o listă de tabele dintr-o bază de date şi o listă de clase de obiecte într-un pogram, sistemele mapează automat cererile dintre acestea. De exemplu, obţinerea numerelor de telefon ale unei persoane se înseamnă crearea şi trimiterea interogării corespunzătoare serverului de baze de date, recepţionarea rezultatelor şi transpunerea lor automată în obiecte de tipul numărului de telefon în cadrul programului de aplicaţii. Din punctul de vedere al programatorului, obiectele par persistente. Programatorul poate crea obiecte şi poate lucra cu ele ca şi când ar fi obiecte obişnuite iar în final, acestea vor ajunge automat în baza de date, fără alte bătăi de cap.

În realitate, lucrurile nu stau tot timpul aşa. Toate sistemele O/RM se fac vizibile într-o oarecare măsură, reuşind mai mult sau mai puţin să ignore existenţa bazei de date. În unele cazuri, operaţia automată de conversie se dovedeşte a fi lentă, mare consumatoare de memorie şi ineficientă, fiind mai convenabilă scrierea de cod propriu pentru rezolvarea problemei.

Numeroase sisteme de mapare relaţional-obiectuale au fost create de-a lungul timpului şi acestea s-au impus mai mult sau mai puţin pe piaţă. Unul dintre cele mai bune astfel de sisteme a fost considerat *Enterprise Objects Framework* al companiei NeXT. Succesul lui nu a durat însă prea mult, asta în principal din cauză că era integrat şi strâns legat de întregul pachet OpenStep. Un alt sistem de acets gen, mult mai performant este Caché. Acesta este de fapt un sistem de gestiune a bazelor de date post-relaţionale care include utilităţile uzuale ale unui sistem de mapare O/R în software-ul pentru baza de date, pentru a maximiza performanţa. Nu în ultimul rând, merită amintit tot în acestă categorie un sistem mai recent, *Java Data Objects (JDO)* , care a devenit un standard şi este disponibil în mai multe versiuni.

Soluţia ideală pentru nepotrivirea dintre cele două abordări – relaţională şi obiectuală – a datelor este un sistem de management al bazelor de date orientat-obiect (OODBMS), care, după cum arată numele, este un sistem proiectat special pentru lucrul cu date şi programme din perspectiva obiectuală. Folosirea unui OODBMS elimină necesitatea de a

converti datele în şi din forma relaţională, întrucât acestea sunt stocate în baza de date direct sub formă de obiecte. Această tehnică este într-adevăr convenabilă şi de mare ajutor în multe situaţii. Una din limitările ei este că, trecerea de la un sistem relaţional la unul pur obiectual implică pierderea capacităţii de a folosi interogări SQL. Din această cauză, unii programatori rămân la sistemele relaţionale de baze de date şi la un sistem de mapare. Totuşi, multe OODBMS sunt capabile să proceseze interogări SQL cu un grad de complexitate limitat.

Concluzii

De-a lungul anilor s-au evidenţiat mai multe tehnologii pentru realizarea aplicaţiilor cu baze de date. Sistemele de baze de date relaţionale au constituit o revoluţie în domeniu. Ele au fost şi sunt în continuare folosite pe scară largă. Ulterior asistăm la extinderea tipurilor de date stocate in baza de date, prin implementarea modelului obiectual-relational şi obiectual.

Bazele de date post-relaţionale reprezintă un pas înainte pentru dezvoltarea aplicaţiilor cu baze de date deoarece asigură programatorilor o serie de avantaje care nu se regăsesc la celelalte tehnologii. Printre cele mai importante avantaje oferite de acestă tehnologie se numără:

- Manevrează date şi reguli complexe prin *definirea de tipuri şi metode proprii* şi *stocarea de obiecte în baza de date*;
- Beneficiază de *avantajele modelului relaţional* : limbaj standard de interogare (SQL), asigurarea integrităţii prin definirea de restricţii în baza de date, suport pentru tranzacţii, vizualizarea datelor în diverse forme şi actualizarea datelor prin tabele virtuale, etc. ;
- Beneficiază de *avantajele programării orientate obiect* : uşurinţă în înţelegere prin reprezentarea datelor într-un mod natural, refolosirea codului, extensibilitate prin compunere şi moştenire de tipuri, supraîncărcare de funcţii şi operatori, productivitate ridicată, flexibilitate, adaptarea cu uşurinţă la modificări etc ;
- Obiectele pot conţine *referinţe* la alte obiecte, fiind posibilă astfel *navigarea între obiecte* ;
- Concordanţa cu modelul de gestionarea a datelor în programele de aplicaţii, care de regulă este un model obiectual, scutind programatorul de efortul necesar pentru maparea relaţional-obiectuală;
- Eleganţă: programare orientată pe obiect, cod curat, modularizat, simplu;
- Extensibilitate şi flexibilitate în lucru.

Bibliografie

[ABRA01] - Abraham P. *„Building a 3-Tier Application using ASP.NET”* , 17-decembrie – 2001 http://www.c-sharpcorner.com/Tutorials/Building3TierAppPA.asp;

[HUPE02] - Hupe, Matthes, Schmidt *„ECommerce – Concepts and Technologies”*, 2002 *http://www.sts.tu-harburg.de/teaching* > Ecommerce;

[IESC05] - http://iesc.unitbv.ro/pdf/admitere2005/material_bsi_masterat_2005.pdf ;

[JADA05] - Jagadish C. *„ Oracle Database Interaction Using ODP.NET and ASP.NET: All Possible Ways To Get Connected”*, sep., 22, 2005;

[MARK01] - Markus V. *„3-Tier Pattern Language”*, 2001, markus.voelter@mathema.de;

[PORT02] - Tom Portfolio, John Russell, *„PL/SQL User’s Guide and Reference”*, Release 9.0.1, Oracle Documentation;

[RANS03] – Ranson J. B. *„Applying ADO.NET in a Multi-Tier Environement”*, http://www.elca.ch

[RENE01] - René S. „*Connecting to Oracle 9 using the Microsoft .NET Framework”* , 2001, http://www.akadia.com/services/dotnet_odbc.html ;

[ZIEM02] – Ziemer S. *„An Architecture for Web Applications”*, 28 nov., 2002.