УДК 004.519

Компан С.В.[1], аспірант

Особливості побудови запитів у об'єктних базах даних на прикладі об'єктної СУБД db4o

[1]Київський національний університет імені Тараса Шевченка, 03680, м. Київ, пр-т. Глушкова 4д

e-mail: robin_2005@mail.ru

S.V. Kompan[1], postgraduate

Peculiarities of query construction in object databases on the example of DBMS db4o

[1]Taras Shevchenko National University of Kyiv, 03680, Kyiv, Glushkova st., 4d

e-mail: robin_2005@mail.ru

Мова запитів є невід'ємною частиною будь-якої системи управління базами даних (СУБД). У статті розглядається типізація та реалізація запитів до об'єктної СУБД db4o. Зроблений аналіз існуючих типів запитів у реляційних СУБД та відповідна їх реалізація в об'єктних СУБД на прикладі СУБД db4o. Для об'єктної СУБД db4o окремо розглянуто використання таких типів запитів як Query By Example (QBE), Native Query (NQ), Simple Object Database Access (S.O.D.A). По кожному типу запиту наведена практична реалізація за допомогою мови програмування Java v.8. Розглянуто SQL оператори та їх аналоги в об'єктних БД на прикладі СУБД db4o. Практично показаний життєвий шлях об'єкта, починаючи від створення його в мові програмування та закінчуючи знищенням з бази даних. Проведена аналогія між мовами запитів реляційних СУБД та об'єктних, виділені спільні їх риси. Практично показана можливість реалізації одного і того ж запиту в об'єктних базах даних різними шляхами.

Ключові слова: об'єктні бази даних, запити в об'єктних базах даних, об'єктна база даних db4o.

Data query language is an integral part of any database management system (DBMS) performance. In this article query typing and implementation of queries to object DBMS db4o are analyzed. The analysis of types of queries existing in relational DBMS and their implementation in object DBMS on the example of db4o have been made. As for object DBMS db4o, the usage of such query types as Query By Example (QBE), Native Query (NQ), Simple Object Database Access (S.O.D.A) have been examined. Practical implementation of every query type has been introduced with the help of Java v.8. software language. SQL operators and their analogues in object DBMS have been examined on the example of DBMS db4o. Object life history is shown starting from its posting on software language and ending up with database destroying. A comparison between relational DBMS and object DBMS query languages has been made and their common features have been singled out. Practical possibilities of implementation of one and the same query in object databases in a variety of ways have been presented.

Key Words: object database, queries in object database, object database db4o.

Статтю представив д.ф.-м.н., професор Буй Д.Б.

Вступ. Одною із важливих функцій систем управління базами даних (СУБД) є зберігання та обробка інформації (даних), що знаходяться в базі даних (БД). Зберігання даних у БД бере на себе безпосередньо СУБД, а можливість оброблювати ці дані СУБД надає користувачу через відповідні мови запитів. Так, для реляційних СУБД існує мова запитів SQL (Structured Query Language), яка застосовується для створення, модифікації та управління даними в реляційних БД. Реалізацій реляційних СУБД існує багато. Це призвело до виникнення багатьох діалектів цієї мови, що спонукало розробників СУБД описати мову SQL на рівні стандарту. Такий підхід дав можливість переносити реляційні БД, в яких зберігаються SQL запити (збережені процедури) з однієї СУБД до іншої. На відміну від реляційних БД (РБД), де інформація зберігається у вигляді рядків таблиць, в об'єктно-орієнтованих базах даних (ООБД) інформація зберігається у вигляді об'єктів, якій є аналогами рядків таблиць в РБД. Тому, для об'єктних СУБД повинен існувати механізм, який дозволяє маніпулювати даними, які зберігаються у таких типах СУБД.

Мову SQL можна умовно розділити на дві складові: мова визначення даних (DDL – Data Definition Language), яка, між іншим, дозволяє описувати схему, визначати тип даних атрибутів таблиць та мова маніпулювання даними (DML – Data Manipulation Language), яка надає можливість

зберігати, редагувати, знищувати, а також здійснювати пошук рядків в БД. За аналогією з реляційними СУБД в об'єктних існує мова опису об'єктів ODL (Object Definition Language), яка дозволяє описати структуру даних та схему БД для об'єктної моделі ODMG (Object Database Management Group) [1], та мова OQL (Object Query Language), яка дозволяє здійснювати пошук об'єктів у БД. Про мову ODL докладно викладено в статті [2]. З іншого боку, існують такі об'єктні СУБД (NeoDatis, db4o, objectdb, тощо), в яких створення схеми класу та екземплярів класу покладається на мову програмування (Java, C++, тощо). Такі СУБД надають користувачу для використання бібліотеки (класи), які дають можливість працювати з СУБД: записувати, зчитувати, знищувати об'єкти з БД, а також здійснювати вибірку даних з БД. На відміну від реляційних СУБД, в яких існує єдиний підхід до реалізації вибірки даних (мова SQL), в об'єктних СУБД існує декілька шляхів до реалізації вибірки даних (QBE, S.O.D.A, тощо). Реалізацію вибірки даних обирає безпосередньо користувач ООБД.

Мета статті. Метою статті є аналіз існуючих типів запитів у об'єктних СУБД на прикладі об'єктної СУБД db4o [3], проведення аналогії між SQL запитами та їхньою реалізацією в об'єктних СУБД. Зазначимо, що використання того чи іншого типу запиту визначається моделлю даних, які зберігаються у відповідних типах СУБД. Проста реалізація запиту в одному типі СУБД в іншому типі СУБД може бути складною.

Приклади запитів будемо демонструвати на основі об'єктної СУБД db4o.

Постановка задачі. Розглянемо предметну область School для реляційної СУБД. Для цього побудуємо ER-діаграму предметної області (рис. 1).

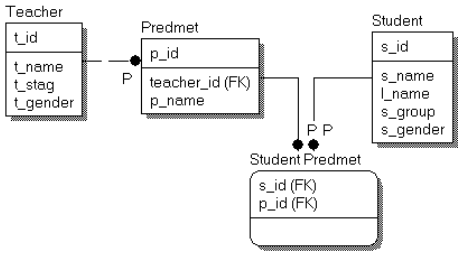

Рис. 1. ER-діаграма БД School

Як видно з рис. 1, відношення багато до багатьох (M:N) між рядками таблиць "Predmet" і "Student" реалізується шляхом створення нової таблиці "Student Predmet", атрибути якої (s_id(FK), p_id(FK)) є зовнішніми ключами (для таблиць "Student" та "Predmet" відповідно).

Можна розглянути предметну область School для об'єктно-орієнтованої парадигми. В цьому випадку таблицям "Teacher", "Predmet", "Student" буде відповідати відповідно клас Teacher, Predmet, Students. Наведемо UML діаграму класів БД School.

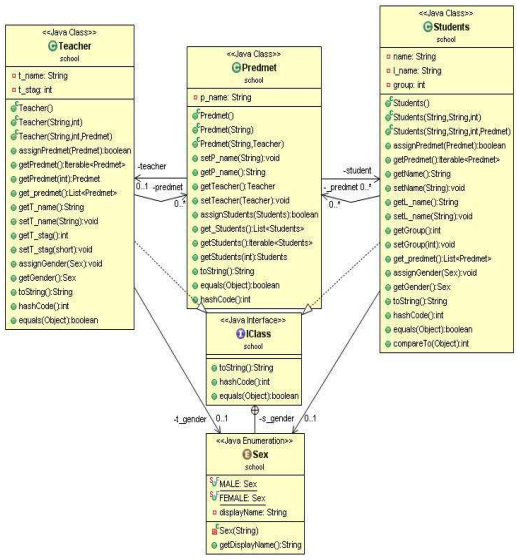

Рис. 2. UML–діаграма класів БД School

Для більш чіткого розуміння інтерпретації запитів у об'єктних СУБД наведемо специфікації класів Teacher, Predmet, Students (без set-, get-методів) на мові програмування Java v.8.0.

```
public class Teacher { private String t_name;
private int t_stag;
/* Для можливості зберігати в об'єкті класу Teacher об'єкти класу Predmet ми описуємо атрибут predmet, який є динамічним масивом типу даних Predmet. */
private final List<Predmet> predmet = new
ArrayList<Predmet>();
Teacher(String name, int stag){
this.t_name=name; this.t_stag=stag; }
public Teacher(String name, int stag, Predmet obj) {
this.t_name = name; this.t_stag = stag;
this.assignPredmet(obj); }
public boolean assignPredmet(Predmet obj){return
predmet.add(obj);} }
public class Predmet {
private String p_name; private Teacher teacher;
/* Для можливості зберігати в об'єкті класу Predmet об'єкти класу Students ми описуємо атрибут student, який є динамічним масивом типу даних Students.*/
private List<Students> student = new
ArrayList<Students>();
public Predmet(String name){ this.p_name=name; }
public Predmet(String name,Teacher obj){
this.p_name = name; this.teacher = obj;
this.student = new ArrayList<Students>(); }
public boolean assignStudents(Students obj){ return
student.add(obj); } }
public class Students {
private String name; private String l_name;
private int group;
```

```
/* Для можливості зберігати в об'єкті класу Students об'єкти класу Predmet ми описуємо атрибут _predmet, який є динамічним масивом типу даних Predmet.*/
private final List<Predmet> _predmet = new ArrayList<Predmet>();
Students(String _name,String lname, int group){
this.name=_name; this.l_name=lname; this.group=group; }
Students(String _name,String lname, int group, Predmet obj){ this.name=_name; this.l_name=lname;
this.group=group; this.assignPredmet(obj); }
public boolean assignPredmet(Predmet obj){
return _predmet.add(obj);} }
```

SQL оператори та їх аналоги в об'єктних БД на прикладі об'єктної СУБД db4o. Для початку роботи в реляційних СУБД ми повинні створити БД. Для створення БД ми використовуємо SQL оператор CREATE DATABASE. На відміну від реляційних СУБД в об'єктних немає чітко визначеного оператора для створення БД. Як правило, СУБД надає користувачу набір методів, які реалізуються Application Programming Interface (API) та дозволяють це реалізувати. Приклади створення БД, як правило, описуються в документації. В об'єктних СУБД БД представляється у вигляді контейнера. Продемонструємо створення контейнера (БД):

```
ObjectContainer container =
Db4oEmbedded.openFile("school.yap");
```

Системний об'єкт container, який має тип ObjectContainer, містить методи (container.store(Object arg0), object.remove(Object arg0), object.delete(Object arg0), тощо), які дають можливість працювати з об'єктами в БД.

Важливою характеристикою будь-якої СУБД є можливість здійснювати збереження інформації у БД, зокрема, для об'єктних СУБД можливість працювати з так званими persistent objects (збереженими об'єктами). Для реляційних СУБД вставка рядка в таблицю реалізується за допомогою SQL оператора INSERT. Збереження рядка таблиці у БД покладається безпосередньо на СУБД і для користувача така дія виконується прозоро. Так, вставка нового рядка в таблицю "Teacher" в реляційних СУБД реалізовується наступним чином:

```
INSERT INTO Teacher VALUES("Ivanov",7);
```

Цю ж саму операцію в об'єктній СУБД ми повинні реалізувати в декілька кроків. Спочатку треба створити новий об'єкт класу Teacher і надати відповідним атрибутам об'єкта значення, реалізувавши тим самим його ініціалізацію. Після цього створений об'єкт за допомогою методів бібліотеки класів СУБД записуємо у БД, яка представляє собою контейнер. Для об'єктної СУБД db4o операція збереження об'єкта в БД реалізовується наступним чином:

```
//створюємо контейнер та відкриваємо його
ObjectContainer container =
Db4oEmbedded.openFile("school.yap");
//створюємо об'єкт
Teacher teach = new Teacher("Ivanov",(byte)7);
//зберігаємо об'єкт у контейнері (БД)
container.store(teach);
//закриваємо доступ до контейнера
container.close();
```

Слід також зазначити, що існують різні способи зберігання об'єктів в БД. Так, в мові програмування Java v.8, починаючи з 5-ї версії, існує набір класів для роботи з об'єктами – JPA (Java persistence API), JDO (Java Data Objects) та EJB (Enterprise JavaBeans).

Будь-яка сучасна СУБД надає можливість оновлювати інформацію, що знаходиться в БД. Для реляційних СУБД існує SQL оператор UPDATE. Так, SQL вираз

```
UPDATE Teacher Set t_stag="33" WHERE t_name="Petrov";
```

дозволяє змінити значення стажу для вчителя з прізвищем Petrov на 33, якщо такий вчитель існує в БД. Як видно, реалізацію пошуку і запису після зміни рядка таблиці бере на себе СУБД. Для об'єктних БД операція оновлення інформації дещо відрізняється. Вона повинна бути проведена у декілька кроків. Спочатку ми повинні знайти в БД об'єкт, який задовольняє критерію пошуку (деякому предикату). Наступним кроком ми оновлюємо відповідні атрибути знайденого об'єкта. Після чого, записуємо оновлений об'єкт до БД. Продемонструємо це на прикладі.

```
// Пошук об'єкта за кретерієм прізвище = "Petrov"
ObjectSet<Teacher> result =
container.queryByExample(new Teacher("Petrov",0));
// якщо такий об'єкт знайдений, оновлюємо інформацію про стаж вчителя
if (!result.isEmpty()){ Teacher found = result.next();
found.setT_stag((byte)33); }
// записуємо оновлений об'єкт до БД
container.store(found);
//закриваємо доступ до контейнера
container.close();
```

Якщо потрібно змінити стаж у всіх викладачів на 15 років, то для реляційних СУБД це буде досягатися наступним SQL запитом:

```
UPDATE Teacher SET t_stag=15;
```

В об'єктних же СУБД користувач повинен самостійно визначати реалізацію вищенаведеної задачі. Ця реалізація залежить від механізмів, які надає об'єктна СУБД та мова програмування, на якій здійснюється доступ до об'єктної БД, з якою працює користувач. Продемонструємо це на прикладі.

```
public static void updateallstag(ObjectContainer db) {
List<Teacher> result = db.queryByExample(new Teacher(null,0));
```

```
ListIterator<Teacher> litr = result.listIterator(result.size());
while(litr.hasPrevious()) { Teacher teach = litr.previous();
teach.setT_stag((byte)15); db.store(teach);} }
```

Як видно з наведеного прикладу, реалізація зміни значення атрибута у всіх об'єктів об'єктної СУБД складніша, ніж в реляційних СУБД, проте поставлена задача може бути реалізована в об'єктних БД.

Для видалення рядка з таблиці, що задовольняє певному критерію, існує SQL оператор DELETE. Наведемо приклад знищення рядка таблиці Teacher, у якого атрибут t_name="Petrov".

```
DELETE FROM Teacher WHERE t_name="Petrov";
```

Для об'єктних СУБД видалення об'єкта з БД реалізується у декілька кроків. Спочатку реалізуємо пошук об'єкта в БД за певним критерієм, а потім знайдений об'єкт знищуємо з БД.

```
// Пошук за критерієм
ObjectSet<Teacher> result =  db.queryByExample(new
Teacher("Petrov",0));
// Знищення знайденого об'єкта з БД
if (!result.isEmpty()) db.delete(result.next());
```

Наступною важливою характеристикою будь-якої СУБД є можливість здійснювати вибірку даних з БД. Так, для вибірки даних в реляційних СУБД використовується мова SQL, зокрема, її оператор SELECT. В об'єктних СУБД, на відміну від реляційних, немає єдиного механізму реалізації вибірки даних. Серед загальноприйнятих шляхів вибірки даних слід виділити наступні: Object SQL (OQL), Java Persistence Query Language (JPQA), Query-By-Example (QBE), Native Queries, Simple Object Database Access (S.O.D.A) та інші. Це пов'язано з тим, що немає загальноприйнятого стандарту до роботи об'єктних СУБД і кожен розробник самостійно вирішує, якими механізмами буде відбуватися вибірка даних.

Тип запитів Query By Example (QBE). Для об'єктної СУБД db4o самим найпростішим з точки зору практичної реалізації є мова запитів типу Query By Example (QBE – пошук за шаблоном). Ця мова запитів за своїм синтаксисом і семантикою є аналогічною до графічної мови запитів QBE реляційних СУБД. Синтаксис мови QBE для реляційних СУБД базується на побудові рядка, в якому атрибутам надаються певні значення, які описують критерій пошуку. В процесі пошуку кожен рядок таблиці співставляється з рядком-шаблоном (по атрибутам, визначеним у рядку-шаблоні). Рядки, у яких атрибути рівні відповідним атрибутам рядка-шаблону, задовольняють умові пошуку. Інтерпретація мови запитів QBE для об'єктних СУБД аналогічна до мови запитів QBE для реляційних СУБД, тільки в якості рядка-шаблону ми будуємо об'єкт-шаблон. Отже, для побудови запиту типу QBE потрібно побудувати шаблон, згідно з яким буде здійснюватися пошук всіх об'єктів, які йому задовольняють. Цей шаблон будується в якості об'єкта, в якому описується критерій пошуку у вигляді предиката. Всі об'єкти, на яких предикат буде істинний, підпадають під критерій пошуку та зберігаються в колекції типу List. В якості значень атрибутів об'єкта-шаблона можна використовувати null, 0, 0.0 та "" (порожній рядок). Значення 0 використовується у шаблоні пошуку для заміщення атрибутів, що мають цілочисельне значення (тип int), 0.0 для заміщення атрибутів, що мають значення з плаваючою комою (тип float/double), "" – для заміщення атрибутів типу String і null – для заміщення атрибутів інших типів даних. Ці значення не використовуються для порівняння значень атрибутів, а лише вказують на те, що значення такого атрибута є довільним. Так, якщо потрібно зпівставити з шаблоном всі об'єкти класу Teacher, то шаблон (об'єкт), побудований за допомогою конструктора класу Teacher, повинен містити в значеннях своїх атрибутів null, 0, 0.0 або "", в залежності від типу даних атрибута.

```
Teacher tch = new Teacher(null, 0);
```

Мова запитів типу QBE не завжди може використовуватися через обмеження, які накладаються на цю мову [4]. Ці обмеження накладаються внаслідок можливості використання у запиті атрибутів із значенням null, 0 та "". На використання мови запитів QBE існують обмеження:

- не можна будувати запити, які містять умовний вираз;
- не можна будувати складних запитів, які містіть логічні операції AND, OR, NOT;
- не можна побудувати запит для пошуку об'єктів, атрибути яких приймають значення null, 0 та "";
- не можна побудувати запит, критерієм пошуку якого є діапазон значень атрибута.

Для побудови QBE запиту потрібна наявність конструктора по замовченню для створення об'єктів без ініціалізації атрибутів (приклад такого конструктора: public Teacher(){}). Такий конструктор створює об'єкт-шаблон, згідно якого будуть знайдені всі відповідні екземпляри класу.

Тип запитів Native Query (NQ). Розглянемо тип запиту Native Query. Запит типу NQ може бути застосований у випадку, коли не можна використати запит типу QBE. Запит типу NQ полягає в тому, що в якості параметра метода db.query() ми передаємо анонімний клас (в нашому прикладі new Predicate<Teacher>() {}). Анонімний

клас не має імені та є підкласом класу, в якому він оголошений [5]. В анонімному класі описуємо метод match(), в якому задаємо логічний вираз. В залежності від того, яке логічне значення (true або false) повертає метод match(), об'єкти класу, над яким виконується запит, додаються до результату запиту або ні. Для побудови логічного виразу, в тілі якого присутні атрибути класу, необхідна наявність методів доступу[1] (принаймні get-методів) до атрибутів класу, що присутні в логічному виразі. Наведемо приклад NQ запиту:

```
List<Teacher> result = db.query(new Predicate<Teacher>()
{ @Override public boolean match(Teacher teach) {
return teach.getT_stag()>9; } } );
```

В запитах типу NQ є можливість використовувати складні умовні вирази, в яких присутні логічні оператори && (AND), || (OR) та | (Not). Наведемо приклад NQ запиту, який здійснює пошук всіх вчителів, у яких стаж роботи не дорівнює 9 рокам:

```
List<Teacher> result = db.query(new Predicate<Teacher>()
{ @Override public boolean match(Teacher teach) {
return (teach.getT_stag() > 9) || (teach.getT_stag() < 9);}});
```

Тип запитів Simple Object Database Access (S.O.D.A.) Наступний тип запиту, який підтримується в СУБД db4o, є S.O.D.A [6]. Запити типу S.O.D.A використовують API (набір методів), що реалізовують низькорівневий доступ до вузлів графа запиту. Застосування запитів типу S.O.D.A надає користувачеві гнучкість в побудові динамічних запитів, хоча для більшості рутинних операцій більш поширеним є запити типу Native. Концепція запиту типу S.O.D.A полягає в тому, що будується граф запиту (query graph), в вузлах якого містяться класи та обмеження на екземпляри класу. В якості прикладу знайдемо всіх вчителів, у яких прізвище Ivanov. Запит типу S.O.D.A буде наступним:

```
Query query = db.query(); query.constrain(Teacher.class);
query.descend("t_name").constraints("Ivanov");
ObjectSet<Teacher> result = query.execute();
```

Граф, згідно якому виконується запит, представлений на рис. 3 [7].

Використовуючи S.O.D.A запити, можна будувати більш складні запити, які містять логічні операції (AND, OR, NOT).

Наведемо S.O.D.A запит, який виводить всіх вчителів, хто читає предмет Muzik та має стаж більше 5 років.

```
Predmet predmet = new Predmet();
predmet.setP_name("Muzik"); Query query = db.query();
query.constrain(Teacher.class);
query.descend("t_stag").constrain(5).greater().and(query.de
scend("predmet").constrain(predmet));
ObjectSet<Teacher> result = query.execute();
```

Як ми вже казали, реалізація запитів залежить від користувача, можливостей об'єктної СУБД та мови програмування. В СУБД db4o існують декілька шляхів реалізації вибірки об'єктів. Розглянемо докладніше типи запитів на прикладі об'єктної СУБД db4o. Даний запит виконується згідно такого S.O.D.A графа запиту (рис. 4).

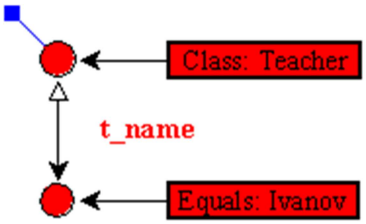

Рис. 3. Приклад графу S.O.D.A запиту

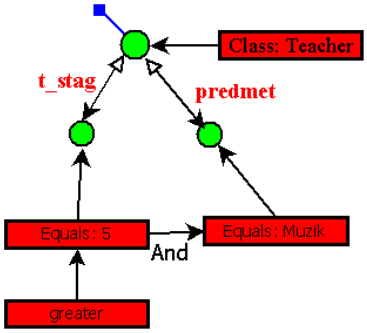

Рис. 4. S.O.D.A граф складного запиту

Поставимо задачу вивести всі рядки (об'єкти) таблиці (класу) "Teacher". Для реляційних СУБД це реалізується за допомогою наступного SQL запиту: SELECT * FROM Teacher. Запит типу QBE для нашого завдання буде виглядати наступним чином:

```
Teacher tch = new Teacher(null, 0);
// або Teacher tch = new Teacher();
List<Teacher> result = container.queryByExample(tch);
```

Результат вибірки даних отримується у вигляді колекції, яку можна потім вивести, наприклад, на екран.

Розглянемо реалізацію SQL запиту SELECT * FROM Teacher за допомогою запиту типу Native Query. Цей запит для нашого завдання такий:

```
List<Teacher> result = container.query(new
Predicate<Teacher>() {@Override
public boolean match(Teacher teach) {
return teach.getT_name()!=null; }}
```

Розглянемо реалізацію SQL запиту SELECT * FROM Teacher за допомогою запиту типу S.O.D.A.

```
Query query = container.query();
query.constrain(Teacher.class);
ObjectSet<Teacher> result1 = query.execute();
```

Знайдемо в БД викладача з прізвищем Ivanov. Для реляційних СУБД це завдання можна

[1] У випадку інкапсуляції.

вирішити за допомогою SQL запиту: SELECT * FROM Teacher WHERE name=”Ivanov”.

Звернемо увагу, на те, що ми можемо в даному SQL запиті вказати тільки частину атрибутів таблиці Teacher. В результатом такого запиту отримаємо таблицю з новою схемою R', відмінної від схеми R початкової таблиці, причому $T' \subseteq T$ $(R' \subseteq R)$. На відміну від реляційних СУБД, де в результаті запиту ми можемо отримати частину таблиці, яку потім при необхідності можна зберегти в БД, в об’єктних СУБД отримати в результаті запиту частину об’єкта не можливо, так як отримуємо з БД в результаті запиту об’єкт з усіма його атрибутами. Але, при необхідності, ми можемо вивести значення тільки частини атрибутів.

Для об’єктної СУБД db4o запит реалізується шляхом створення нового об’єкту типу Teacher, у якого атрибутам, по яким не здійснюємо пошук надаємо значення Null або 0[2]; інші атрибути, по яким здійснюється пошук, мають свої вихідні значення. В нашому випадку побудова об’єкта буде виглядати наступним чином:

```
//створюємо об’єкт-шаблон
Teacher tch = new Teacher(“Ivanov”, 0);
//результат отримуємо у вигляді колекції List, елементи якої мають тип Teacher
List<Teacher> result = container.queryByExample(tch);
```

В колекцію result потраплять всі об’єкти класу Teacher, які “співпадають” з об’єктом-шаблоном tch. Як видно, перевагою використання типу запитів QBE є відносна простота реалізації.

Реалізуємо поставлену задачу за допомогою запиту типу Native.

```
List<Teacher> result = db.query(new Predicate<Teacher>()
{@Override public boolean match(Teacher teach) { return
teach.getT_name().contains("Ivanov"); } });
```

Реалізація поставленої задачи за допомогою запиту типу S.O.D.A була наведена вище.

Оператори мови SQL IN, BETWEEN, LIKE можуть бути реалізованими програмістом в СУБД db4o в запитах типу Native та S.O.D.A. Так, наша реалізація запиту типу Native, який є аналогом SQL запиту:

```
SELECT * FROM Teacher WHERE t_stag IN (3, 7);
```

буде мати вигляд:

```
final HashSet<Integer> set = new HashSet<Integer>
(Arrays.asList(3, 7));
List<Teacher> result = db.query(new Predicate
<Teacher>() { @Override public boolean match(Teacher
teach) { return (set.contains(teach.getT_stag())); } });
```

Реалізація запиту типу S.O.D.A буде виглядати наступним чином:

```
Query query = db.query(); query.constrain(Teacher.class);
query.descend("t_stag").constrain((int)3).or(query.descend(
"t_stag").constrain((int)7));
ObjectSet<Teacher> result = query.execute();
```

Реалізуємо запити типу Native та S.O.D.A для такого SQL запиту:

```
SELECT * FROM Teacher WHERE t_stag BETWEEN 3
AND 7;
```

Реалізація запиту типу Native буде мати вигляд:

```
List<Teacher> result = db.query(new Predicate<Teacher>()
{@Override public boolean match(Teacher teach) {return
(teach.getT_stag()>=3)&&(teach.getT_stag()<=7); } });
```

Реалізація запиту типу S.O.D.A буде мати вигляд:

```
Query query = db.query(); query.constrain(Teacher.class);
query.descend("t_stag").constrain((int)3).greater().like().an
d(query.descend"t_stag").constrain((int)7).smaller().like());
ObjectSet<Teacher> result = query.execute();
```

Об’єктна СУБД db4o не надає в розпорядження користувачу методи, які аналогічні за семантикою до агрегатних функцій мови запитів SQL. Натомість, користувач може самостійно створювати методи за допомогою мови програмування Java v.8.0, в яких реалізована логіка роботи агрегатних функцій.

Покажемо це на прикладі підрахування середнього стажу всіх вчителів школи. Для реляційних СУБД це завдання буде описуватися за допомогою такого SQL запиту:

```
SELECT AVG(t_stag) FROM Teacher;
```

Визначення середнього стажу вчителів в СУБД db4o можемо реалізувати наступним чином [7]:

```
public avg() { int sum = 0; Query query = db.query();
query.constrain(Teacher.class);
query.descend("t_name").constraints();
ObjectSet<Teacher> result = query.execute();
while(result.hasNext()){sum+=result.next().getT_stag();}
System.out.println(“AVG: ”+ (result.size() == 0 ? 0 : sum /
result.size())); }
```

Агрегатні функції MIN, MAX та інші можуть бути реалізовані аналогічно.

Можна припустити, що агрегатні функції, а також оператори в мові SQL представляють собою функції (методи), реалізація яких прихована від користувача. Тому відкритість реалізацій аналогів агрегатних функцій та операторів мови SQL в об’єктних СУБД дає можливість уточнювати в разі необхідності цю реалізацію, що дає можливість більш гнучко оперувати даними, зокрема, проводити оптимізацію.

В об’єктних СУБД, як і в реляційних, існує можливість здійснювати вибірку даних шляхом їх групування. Для цього в мові SQL існує конструкція GROUP BY. Вона дає можливість згрупувати рядки таблиці, в найпростішому випадку, за значеннями одного або декількох атрибутів. Результатом запиту може бути

[2] Для числових даних надаємо значення 0, для текстових даних – Null.

узагальнене значення атрибутів, точно так само, як і значення одного атрибута. Це робиться за допомогою стандартних агрегатних функцій SQL: COUNT, SUM, AVG, MAX, MIN [8].

Як ми вже казали, програмна реалізація SQL конструкції GROUP BY в об'єктній СУБД db4o повністю покладається на користувача. Це, з одного боку, дає можливість користувачу, виходячи з конкретної предметної області або моделі БД, самостійно задати реалізацію цією конструкції, що, в свою чергу, може призвести до оптимізації запиту. (group by по даним з 2 таблиць – можна обирати з 2-х класів, а можно обирати з одного). З іншого боку, на власну реалізацію користувачем конструкції мови SQL впливає рівень користувача, як програміста.

Таблиця 1

Вміст таблиці Student

s_id	s_name	l_name	s_group	s_gender
1	Борис	Ткач	11	MALE
2	Юлия	Гришко	21	FEMALE
3	Светлана	Іванова	21	FEMALE
4	Павел	Іванов	12	MALE

Таблиця 2

Результат виконання запиту з оператором GROUP BY

Група	Кількість
11	1
12	1
21	2

Продемонструємо це на прикладі. Нехай маємо таблицю Student (Табл. 1). Якщо потрібно вивести кількість студентів в кожній групі, то ця задача в реляційних СУБД розв'язується за допомогою такого SQL запиту:

```
SELECT s_group AS “Група”, COUNT(l_name) AS “Кількість” FROM Student GROUP BY s_group;
```

Результатом запиту буде таблиця (Табл. 2).

В об'єктній СУБД db4o цю задачу можна розв'язати декількома способами.

Спосіб 1. Відсортуємо об'єкти класу Students за значенням атрибута s_group. Очевидно, що таким чином ми згрупували студентів за групами, де вони навчаються. Потім підрахуємо кількість студентів в кожній такій групі. Наведемо програмний код.

```
Query query = db.query();
query.constrain(Students.class);
query.descend("s_name").constraints();
query.descend("s_group").orderDescending();
ObjectSet<Students> result = query.execute();
System.out.println("Group"+"\t"+"Count"+"\t");
int sum=0, d1=0, d2=0, size = result.size();
ListIterator<Students> iter = result.listIterator(result.size());
while(iter.hasPrevious()) { Students stud =  iter.previous();
size = size-1; if (d1==0) {
d1=stud.getGroup(); sum=sum+1; continue; }
d2=d1; d1=stud.getGroup();
if ((d1==d2) && (size!=0)) {sum=sum+1;}
else switch(size) { case 0: { sum=sum+1;
System.out.println(d2+"\t"+sum); break; } default:
System.out.println(d2+"\t"+sum);sum=1;; }}
```

Спосіб 2. Використовуємо хеш-таблиці (HashMap), які реалізує інтерфейс Map. Дані, які зберігаються в хеш-таблиці, мають структуру ключ\значення. В якості ключа візьмемо номер групи (атрибут s_group), а значенням відповідного ключа буде виступати масив об'єктів типу Students. Всіх студентів, які мають відповідний ключ (номер групи) у хеш-таблиці, ми будемо додавати до відповідного масиву. Продемонструємо практичну реалізацію[3]:

```
public static void CountStudGroup(ObjectContainer db) {
Map<Integer, ArrayList<Students>> map = new
HashMap<Integer, ArrayList<Students>>();
Query query = db.query(); query.constrain(Students.class);
query.descend("name").constraints(); ObjectSet<Students>
result = query.execute();
System.out.println("Group"+"\t"+"Count"+"\t"+
"Students"); for (Students student : result){ if
(map.containsKey(student.getGroup())) {
ArrayList<Students> values = map.get(student.getGroup());
if (values.isEmpty()) {
values = new ArrayList<Students>();}
map.put(student.getGroup(),values); values.add(student);}
else { ArrayList<Students> st = new
ArrayList<Students>(); st.add(student);
map.put(student.getGroup(),st); } }
Set<Entry<Integer, ArrayList<Students>>> set =
map.entrySet(); Iterator<Entry<Integer,
ArrayList<Students>>> i = set.iterator(); while(i.hasNext())
{ Map.Entry map = (Map.Entry)i.next();
System.out.println(map.getKey() + ": ");
ArrayList<Students> a = (ArrayList<Students>)
map.getValue(); System.out.println("\t"+a.size()); for
(Students stud: a){
System.out.println("\t\t"+stud.getName() +" " +
stud.getL_name());} } }
```

Отже, ми показали, що існують різні способи програмної реалізації SQL операторів на прикладі конструкції GROUP BY.

[3] В прикладі демонструємо заповнення і вивід хеш-таблиці.

```
SELECT * FROM Student ORDER BY s_group DESC;
```

Конструкція ORDER BY може впорядковувати рядки за одним атрибутом або декількома (по суті лексикографічне впорядкування).

В об'єктній СУБД db4o в запитах типу Native та S.O.D.A є можливість впорядковувати об'єкти за атрибутом чи групою атрибутів. Продемонструємо це. Запит типу S.O.D.A:

```
Query query = db.query(); query.constrain(Students.class);
query.descend("s_group").orderAscending();
ObjectSet<Students> result = query.execute();
```

Для реалізації впорядкування в запиту типу Native ми повинні перевизначити в інтерфейсі Comparator метод compare(Object obj1, Object obj2), в якому опишемо правило порівняння об'єктів Students за атрибутом s_group. Наведемо приклад запиту типу Native, в якому реалізоване впорядкування об'єктів Students по атрибуту s_group:

```
List<Students> result = db.query(new
Predicate<Students>() {@Override
public boolean match(Students stud) {
return stud.getName() != null; } }, new
Comparator<Students>() { @Override
public int compare(Students o1, Students o2){
return o1.getGroup()<o2.getGroup()? -1 :
o1.getGroup()==o2.getGroup()? 0 : 1;} });
```

В даному прикладі ми перевизначаємо за допомогою анотації @Override метод compare, в якому описуємо механізм порівняння двох об'єктів класу Students по атрибуту s_group, доступ до якого відбувається за допомогою get-метода getGroup().

Висновки. В реляційних СУБД існує одна мова запитів SQL, яка при переході від одної реляційної СУБД до іншої може бути відмінною своїми діалектами. В об'єктних СУБД, на відміну від реляційних, немає єдиного механізму реалізації вибірки даних. Можна виділити об'єктну мову запитів (Object Query Language), яка є аналогом мови SQL, а також декілька типів запитів, які реалізуються за допомогою мови програмування. Серед таких типів запитів можна виділити QBE, NQ, S.O.D.A, тощо. Саме типи запитів в об'єктних СУБД надають користувачеві гнучкість при написанні запитів. Зазначимо, що якщо для реляційних СУБД результатом запиту є таблиця, то для об'єктних в загальному випадку результатом вибірки є колекція об'єктів.

Список використаних джерел

1. Object Data Management Group [Online]. – Available from: http://www.odbms.org/odmg/
2. *Кузнецов С.Д.* Три манифеста баз данных: ретроспектива и перспективы / *С.Д. Кузнецов* [Online]. – Available from: http://citforum.ru/database/articles/manifests/
3. Object database Db4o [Online]. – Available from: http://en.wikipedia.org/wiki/Db4o
4. Object-Oriented Database Development using db4o [Online]. – Available from: http://www.pdfport.com/view/755759-object-oriented-database-development-using-db4o.html
5. Anonyms class [Online]. – Available from: http://javatutor.net/articles/anonymyous-classes-in-java
6. S.O.D.A – Simple Object Database Access. [Online]. – Available from: http://sourceforge.net/projects/sodaquery .
7. The Definitive Guide to db4o [Online]. – Available from: http://ontwerpen1.khlim.be/~lrutten/cursussen/inf8/db4o.pdf
8. *Копейкин М.В.* Базы данных. Основы SQL реляционных баз данных: Учебное пособие. / М.В. Копейкин, В.В. Спиридонов, Шумова Е.О. – СПб.: СЗТУ, 2005. – 160 с.

References

1. *Object Data Management Group* [Online]. – Available from: http://www.odbms.org/odmg/
2. KUZNETSOV, S.D. (2003) *Three manifest database: Retrospect and Prospect* [Online]. Available from: http://citforum.ru/database/articles/manifests/
3. *Object database Db4o* [Online]. – Available from: http://en.wikipedia.org/wiki/Db4o
4. *Object-Oriented Database Development using db4o* [Online]. – Available from: http://www.pdfport.com/view/755759-object-oriented-database-development-using-db4o.html
5. *Anonyms class* [Online]. – Available from: http://javatutor.net/articles/anonymyous-classes-in-java
6. *S.O.D.A – Simple Object Database Access.* [Online]. – Available from: http://sourceforge.net/projects/sodaquery .
7. *The Definitive Guide to db4o* [Online]. – Available from: http://ontwerpen1.khlim.be/~lrutten/cursussen/inf8/db4o.pdf
8. KOPEIKIN, M.V., SPIRIDONOV, V.V., SHUMOVA, H.O. (2005) *Databases. SQL-based relational databases: Tutorial*. St. Petersburg.

Надійшла до редколегії 31.10.14