УДК 681.3.062

ОБ'ЄКТНО-ОРІЄНТОВАНА БД NEODATIS: РОЗГОРТАННЯ, ПРОГРАМУВАННЯ ЗАПИТІВ, ПОРІВНЯННЯ З РЕЛЯЦІЙНИМИ БАЗАМИ ДАНИХ

Д.Б. Буй, С.В. Компан

Київський національний університет імені Тараса Шевченка, МСП, 01601, проспект АкадемікаГлушкова, 2, корп. 6, Київ, Україна, факс/тел. (044)521 3345, buy@unicyb.kiev.ua
Кіровоградський державний педагогічний університет імені Володимира Винниченка, 25006, вул. Шевченка 1, Кіровоград, Україна, факс/тел. (0522)24-89-01; skompan@kspu.kr.ua

Розглядаються об'єктно-орієнтована БД Neodatis (представляє собою набір класів) та реляційна СУБД Postgresql, що порівнюються на прикладі модельної бази даних "Школа". Розглядаються приклади класів та методів для створення об'єктів у БД Neodatis. Продемонстровано підходи до створення запитів у БД Neodatis з використанням мови програмування Java та в СУБД Postgresql з використанням мови SQL.

The object-oriented data base (represented as a different variety of classes) and relational DBMS Postgresql, that are compared on the basis of model base "Scale (dial)" are viewed. The examples of the classes and methods to form the objects in DB Neodatis are investigated. The ways of creating of queries in BD Neodatis with the usage of the programming Java language and in DBMS Postgresql with the usage of SQL language have been demonstrated.

Загальні зауваження

У час стрімкого розвитку інформатизації суспільства виникає необхідність зберігати та обробляти велику кількість інформації. Тому широке поширення набули реляційні бази даних (БД), побудовані на основі реляційної моделі даних, яка запропонована Є. Кодом [1, с. 790]. Одним із недоліків таких БД є обмежений та фіксований набір типів даних, а також обмежений набір операцій над цими типами. Іноді це не дозволяє повністю та чітко описати адекватну інформаційну модель даних. Розвиток об'єктно-орієнтовних мов програмування (ООМП) дав можливість перенести основні ідеї цих мов на реляційні СУБД. Появилися об'єктно-реляційні БД (ґрунтуються на об'єктно-реляційної моделі даних), у яких здійснюється підтримка типів користувача та які мають властивості, притаманні ООМП. Слід зауважити, що класи об'єктів в об'єктно-реляційних БД (ОРБД) відповідають таблицям, причому для кожного класу, що наслідується, створюється окрема таблиця, яка пов'язана з таблицею базового класу по первинному ключу відношенням – зв'язком один до одного. Об'єктам БД відповідають окремі записи в таблицях. Первинним ключем для таблиці базового класу може бути автоматичне нумероване поле цілочислового типу (так зване автоінкрементне поле). Тоді таблиці, які наслідують даний клас, можна зв'язати за допомогою поля цілочислового типу. Але такий підхід іноді не забезпечує унікальності об'єкта, тому що первинний ключ може бути унікальним у межах певного відношення, але не для всієї системи в цілому. Зазвичай первинний ключ вибирається на основі атрибутів відношення, це призводить до його залежності від поточного стану об'єкта. Ці проблеми намагаються усунути розробники ОРБД за допомогою використання для кожного об'єкта так званого OID-ідентифікатора (Object ID), який дає змогу зробити кожен об'єкт унікальним у межах всієї системи. Причому OID-ідентифікатор ні в якому разі не залежить від даних, які знаходяться в даному об'єкті. Ця властивість дозволяє змінювати значення кожного атрибута об'єкта, але навіть при зміні своїх атрибутів об'єкт матиме той самий OID-ідентифікатор. Розрізняють логічні OID-ідентифікатори, які не залежать від фізичного розташування об'єкта на зовнішньому носієві, а також фізичні OID-ідентифікатори, в яких закодовано розташування об'єкта [2, с. 844]. Система ООБД повинна мати властивості, притаманні СУБД, а також мати характеристики об'єктно-орієнтовної системи [3].

Порівняння ООБД NeoDatis та реляційної СУБД Postgresql

Існують багато різних ООБД, що вільно розповсюджуються, або є комерційними. Серед них поширеними є ODBMS, db4o та інші. У статті будемо розглядати об'єктно-орієнтовну базу класів **NeoDatis [**4**],** на основі якої можна створювати ООБД. Ця база класів містить методи, які дозволяють створювати файл БД, додавати об'єкт до БД, вилучати його, реалізовувати запити тощо. Базу класів NeoDatis можна підключати в середовищі програмування до проекту (наприклад, оператором include) та засобами мови програмування здійснювати доступ до відповідних класів, змінних, методів цієї бази класів.

Розробники позиціонують **NeoDatis** як просту, але в той же час повністю ООБД, яка може працювати на декількох платформах – Java, Net, Google Android, Groovy та Scala. За секунду може зберігати більше ніж 30000 об'єктів, причому під час роботи підтримує ACID (Atomicity, Consistency, Isolation, Durability) транзакцій, що дає можливість гарантувати надійність операціям, які виконуються в базі. БД, побудована на основі **NeoDatis**, складається з одного файлу, в який поміщаються всі об'єкти БД: мета-модель, об'єкти, індекси. При необхідності дані з ООБД можуть бути експортовані в формат XML та імпортовані за допомогою API або через Object Explorer. **NeoDatis** можна використовувати вільно, але технічна підтримка використання **NeoDatis** здійснюється на комерційній основі [4].

Будемо здійснювати доступ до **NeoDatis** через Java-платформу. Для цього на робочій станції встановлюється програмне забезпечення Java (TM) Platform Standard Edition Runtime Environment. Як середовище розробки для Java будемо використовувати **Eclipse**.

Підключивши **NeoDatis**, маємо можливість використовувати традиційні функції БД в мові програмування (в даному випадку Java). Щоб використовувати **NeoDatis** упроекті, потрібно добавити до проекту бібліотеку neodatis-odb.

З демонстраційною метою будемо створювати БД Школа. З одного боку створимо її в реляційній СУБД PostgreSQL [5], з іншого – як ООБД за допомогою бібліотеки класів **NeoDatis**. Мета – порівняти роботу цих двох БД та продемонструвати основні можливості ООБД **NeoDatis**.

Створимо БД Школа в ООБД **Neodatis**. Для цього створимо об'єкт Student з атрибутами: прізвище (name), клас (nomer), оцінка з математики (value_m), оцінка з хімії (value_h).

Реалізуємо його в мові програмування Java як клас:

```
public class Student {
private String name;
private String nomer;
private Room nomer_room;
private int value_m;
private int value_h;
public Student(String name, Room nomer, int value_m, int value_h);
public String getName();
}
```

У блоці є метод (конструктор) для ініціалізації об'єкта Student. Повністю опис цього конструктора такий:

```
public Student(String name, Room nomer, int value_m, int value_h) {
this.name = name;
this.nomer_room = nomer;
this.value_m = value_m;
this.value_h = value_h;
}
```

Вираз this.name означає пряме посилання на значення змінної об'єкта. Атрибут nomer_room – це об'єкт типу Room, який є окремим класом. Метод, який дозволяє здійснити доступ до атрибуту nomer (який private),

```
public String getName() {
return nomer;
}
```

Розглянемо об'єкт Room з атрибутами: номер класу (nomer), прізвище класного керівника (teacher). Створимо для нього клас:

```
public class Room {
private String nomer;
private String teacher;
public Room(String nomer,String teacher);
public String getName();
}
```

У блоці є методи для ініціалізації об'єкта Room та доступу до атрибуту об'єкта name. Повністю опис цього конструктора такий:

```
public Room(String nomer,String teacher) {
this.nomer = nomer;
this.teacher = teacher;
}
public String getName() {
return nomer;
}
```

Створимо третій клас main_class, у якому буде міститись функція main. Загальний вигляд функції:

```
public static void main(String[] args) throws Exception;
```

У цієї функції реалізовані такі дії:

1. Виклик метода (з бібліотеки класів NeoDatis), який дозволяє знищити існуючий файл БД:
   `IOUtil.deleteFile(ODB_NAME);`
2. Створення нового об'єкта класу main_class:
   `main_class main_class = new main_class();`
3. Створення нового об'єкта класу Room та його ініціалізація:
   `Room room1 = new Room("10-a","Petrova");`
4. Створення нового об'єкту класу Student та його ініціалізація:
   `Student stud1 = new Student("Ivanov",room1,4,3);`
5. За допомогою методу add_student_todb класу main_class занесення отриманого об'єкта в БД
   `main_class.add_student_todb(stud1);`

Аналогічно додаються інші об'єкти до БД.

Внаслідок отримаємо БД, в якій існує дві таблиці, що містять по три об’єкти кожна (рис. 1).

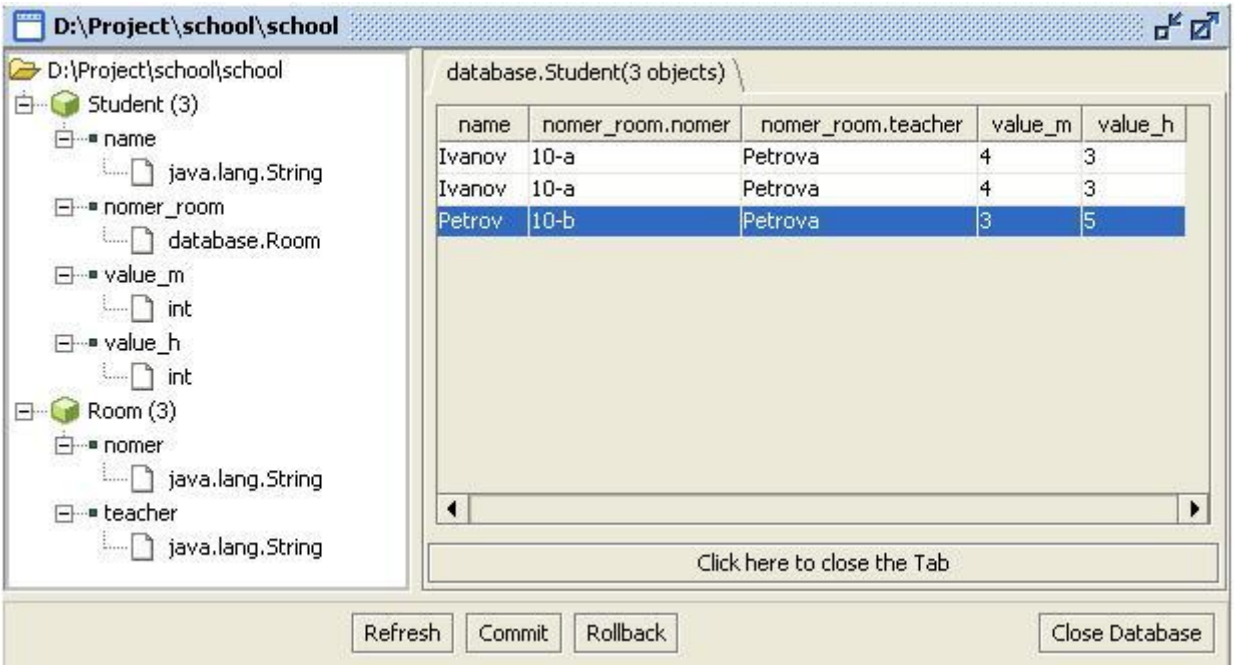

Рис. 1. Таблиця Student, яка містить об’єкти класу Room

Наведена таблиця має два однакових об’єкта, але цілісність не порушується через різні значення OID для кожного об’єкта. Кожний об’єкт у своєму класі має свій унікальний OID (рис. 2).

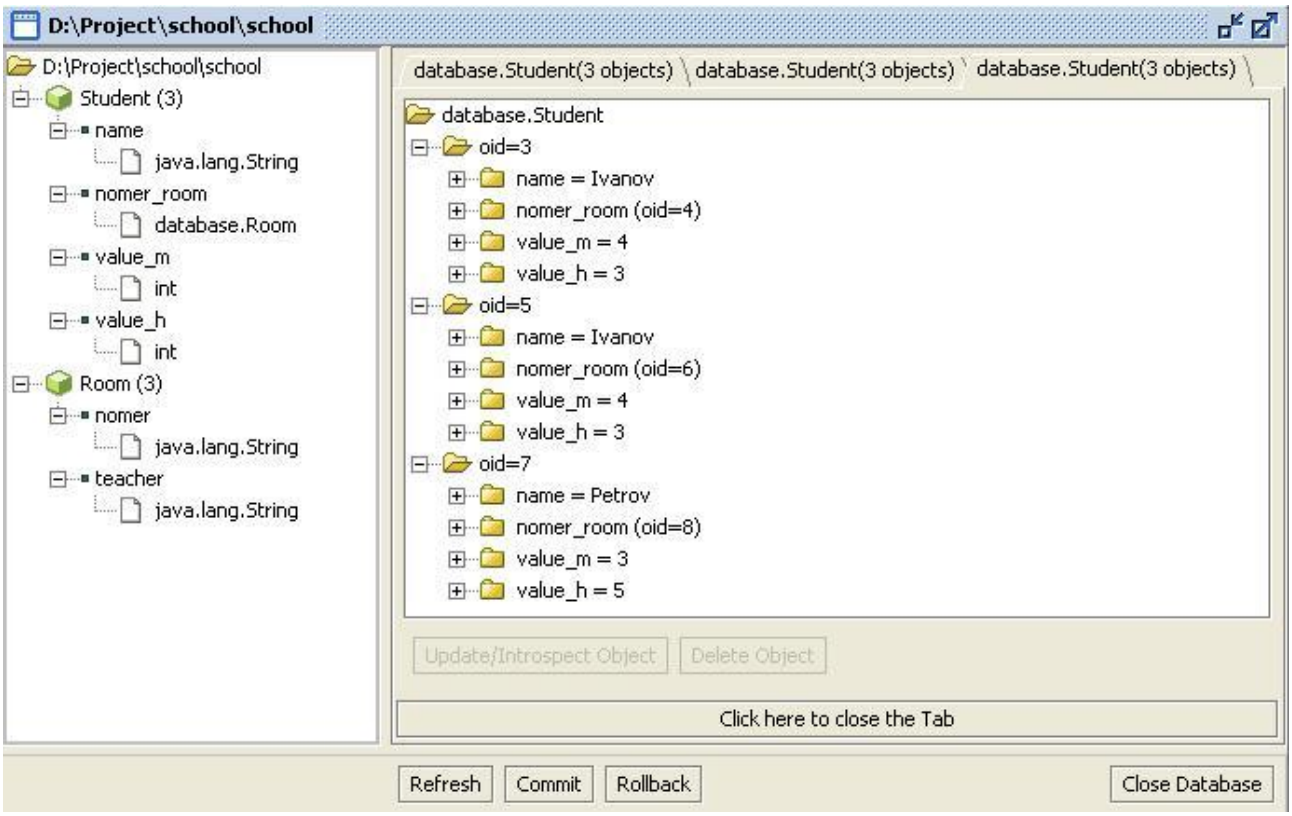

Рис. 2. OID об’єктів в БД NeoDatis

Тепер створимо цю ж саму БД Школа засобами СУБД Postgresql. Треба зауважити, що на основі схеми БД для ООБД **NeoDatis** не можна побудувати реляційну модель БД для Postgresqle, тому що в реляційній БД не може бути двох однакових записів, як це було в ООБД. Тому потрібно змінити схему для реляційної БД, додаючи ключові поля в таблицю.

Схема для таблиці Student (будемо вважати, що в класі не має однакових прізвищ):

name character (15)
value_m int
value_h int
nomer character (5)

Схема для таблиці Room:

nomer character (5) ключове
teacher character (15)

На відміну від ООБД, в таблиці Room маємо тільки один запис, який відповідає класу з номером '10-a', через те, що це поле є ключовим.

Для роботи в Postgresql будемо використовувати pgAdmin III – програма, яка надає можливість працювати з базами даних у СУБД Postgresql через графічний інтерфейс [6]. Для створення БД побудуємо SQL-запит:

```
CREATE DATABASE school;
```

Надамо необхідні права доступу до БД користувачам. У даному випадку користувачеві postgres надаються права адміністратора на базу school.

```
GRANT ALL ON DATABASE school TO postgres;
```

Створимо таблиці Student, Room:

```
CREATE TABLE "Student" (
"name" character(15),
value_m integer NOT NULL DEFAULT 2,
value_h integer NOT NULL DEFAULT 2,
nomer character(5)
);
GRANT ALL ON TABLE "Student" TO postgres;
CREATE TABLE "Room" (
teacher character(15),
nomer character(5) NOT NULL,
CONSTRAINT nomer PRIMARY KEY (nomer)
);
GRANT ALL ON TABLE "Room" TO postgres;
```

Ініціалізуємо наші таблиці даними за допомогою SQL виразів:

```
INSERT INTO "Student"("name", value_m, value_h, nomer) VALUES ('Ivanov', 3, 4, '10-a');
INSERT INTO "Room"("nomer", "teacher") VALUES ('10-a','Petrova');
```

Аналогічно додаємо до БД інші записи.

Вкажемо відмінність між розглянутими двома БД. У реляційній БД у створеній таблиці не завжди можна поміняти тип атрибуту, наприклад, з текстового на числовий. Такі зміни виконуються шляхом знищення атрибуту текстового типу та створенням атрибуту іншого типу. У випадку ООБД змінювати схему можемо легко через те, що схема є класом. Для цього засобами мови програмування потрібно внести відповідні зміни, змінивши тип поля класу.

Наведемо функцію додавання об’єкта до ООБД. На початку оголосимо зміну ODB_NAME:

```
public static final String ODB_NAME = "school";
public void add_student_todb(Student stud) {
ODB odb = null;
try {
odb = ODBFactory.open(ODB_NAME);
odb.store(stud);
} finally {if (odb != null) {odb.close();}
}
}
```

Для використання в ній методів ODBFactory.open (відкриття файлу), odb.store (запису об’єкта до БД), odb.close() (закриття файлу) необхідно добавити клас import org.neodatis.odb.ODB;

Продемонструємо побудову запитів в ООБД NeoDatis та РБД Postgresql.

Виведемо студентів, які мають прізвище 'Ivanov' та оцінку по хімії 3.

Для **NeoDatis**:

```
public void query1() throws Exception {
ODB odb = null;
try {
odb = ODBFactory.open(ODB_NAME);
System.out.println("\nЗапит 1 : Студенти, які мають прізвище 'Ivanov' та оцінку по хімії 3");
IQuery query = new CriteriaQuery(Student.class, Where.and().add(Where.equal ("value_h", 3)).
add(Where.equal("name", 'Ivanov')));
Objects <Student>stud = odb.getObjects(query);
```

```
int i = 1;
while (stud.hasNext()) {
System.out.println((i++) + "\t: " + stud.next().getName());
}
} finally {
if (odb != null) {
odb.close();
}
}
}
```

В **Postgresql**:

```
SELECT   "Student"."name" FROM   shcool."Student"
WHERE   "Student"."name" = 'Ivanov' AND "Student".value_h = 3;
```

Як бачимо доступ до об'єктів БД в Postgresql здійснюється легше ніж в **NeoDatis**.

Виведемо студентів, які мають оцінку по хімії 3 або 5.

Для **NeoDatis**:

```
public void query2() throws Exception {
ODB odb = null;
try {
odb = ODBFactory.open(ODB_NAME);
System.out.println("\n3апит 2 : Студенти, які мають оцінку по хімії 3 або 5");
IQuery query = new CriteriaQuery(Student.class, Where.or().add(Where.equal("value_h", 3)).add
(Where.equal("value_h", 5)));
Objects <Student>stud = odb.getObjects(query);
int i = 1;
while (stud.hasNext()) {
System.out.println((i++) + "\t: " + stud.next().getName());
}
} finally {
if (odb != null) {
odb.close();
} } }
```

В **Postgresql**:

```
SELECT  "Student"."name" FROM shcool."Student"
WHERE  "Student".value_h = 5 OR "Student".value_h = 3;
```

Виведемо студентів, прізвища яких Ivanov та які навчаються в 10-а класі.

Для **NeoDatis**:

```
public void query3() throws Exception {
ODB odb = null;
try {
odb = ODBFactory.open(ODB_NAME);
IQuery query = new CriteriaQuery(Student.class, Where.and().add(Where.equal("name",
'Ivanov')).add(Where.equal("nomer_room.nomer", '10-a')));
Objects<Student> stud = odb.getObjects(query);
System.out.println("\n3апит 3 : Студенти, прізвища яких 'Ivanov' та які навчаються в '10-a' класі ");
int i = 1;
while (stud.hasNext()) {
System.out.println((i++) + "\t: " + stud.next().getName());
}
} finally {
if (odb != null) {
odb.close();
} } }
```

Результат виконання: 2 записи. Отримано такий результат завдяки тому, що при побудові класів використовується агрегація, тобто атрибут nomer_room з класу Student має тип класу Room.

В **Postgresql**:

```
SELECT   "Student"."name",  "Room".teacher, "Room".nomer
FROM   shcool."Student", shcool."Room"
WHERE  "Student".nomer = "Room".nomer AND "Student"."name" = 'Ivanov'
AND   "Room".nomer = '10-a';
```

Виведемо студентів, які навчаються в 10-х класах.

Для **NeoDatis**:

```
public void query4() throws Exception {
ODB odb = null;
try {
odb = ODBFactory.open(ODB_NAME);
IQuery query = new CriteriaQuery(Student.class, Where.like("nomer_room.nomer", '10%'));
Objects<Student> stud = odb.getObjects(query);
System.out.println("\nЗапит 4 : Студенти, які навчаються в 10-х класах");
int i = 1;
while (stud.hasNext()) {
System.out.println((i++) + "\t: " + stud.next().getName());
}
} finally {
if (odb != null) {
odb.close();
}
}
}
```

В **Postgresql**:

```
SELECT  "Student"."name",  "Student".nomer FROM  shcool."Student"
WHERE "Student".nomer LIKE '10%';
```

Основні результати та висновки

1. На конкретних прикладах розглянуто роботу реляційної та ОО БД.
2. Показано, що зміна схеми легше відбувається у ООБД ніж в реляційних.
3. Реалізація запитів в ООБД, на відміну від реляційних БД, як правило, здійснюється через мову програмування.
4. ООБД повинна задовольняти властивостям, притаманним СУБД, та мати можливості об’єктно-орієнтовних мов програмування.
5. Для об’єднання двох таблиць в ООБД використовують агрегацію, яка дозволяє включати об’єкти одного класу в інший. У РБД існує інший підхід: можна об’єднати записи з двох таблиць через поля одного і того ж типу, які, як правило, є ключовими. Декартовий добуток двох таблиць є стандартною операцією у РБД, але його не можна реалізувати в ООБД з огляду на іншу ідеологію побудови РБД.
6. Показано, що модель ООБД не можна повністю перенести на РБД.

1. *Дейт К. Дж.* Введение в системы баз данных; 8-е изд. / К. Дж. Дейт. – Киев: М.; СПб.: Издательский дом “Вильямс”, 2005. – 1315 с.
2. *Конолли Т.* Базы данных: проектирование, реализация и сопровождение. Теория и практика / Т. Конолли, К. Бегг, А. Страчан – М.: Издательский дом “Вильямс”, 2001. – 1101 с.
3. *Date C.J.* Foundation for Object-Relational Databases: The Third Manifesto, 1998 [Електронний ресурс] / C.J. Date, Hugh Darwen. Addison-Wesley точка доступу: http://www.citforum.ru/database/classics/oo_manifesto
4. [Електронний ресурс] точка доступу: http://www.neodatis.org/overview
5. [Електронний ресурс] точка доступу: http://www.postgresql.org/about/
6. [Електронний ресурс] точка доступу: http://www.pgadmin.org/