1 Cassandra

Cassandra jest składem kolumnowym opartym na koncepcjach pochodzących z BigTable and DynamoDB.

Różnice pomiędzy relacyjnymi bazami danych i składem kolumnowym Cassandra:

- brak złączeń (ang. joins),
- brak podzapytań (ang. subqeries),
- brak więzów integralności,
- brak normalizacji,
- projektowanie ukierunkowane na zapytania,
- projektowanie ukierunkowane na optymalne przechowywanie,
- sposób posortowania danych jest ustalany podczas projektowania.

Modelowanie danych:

- Koncepcyjne modelowanie danych, które polega na stworzeniu modelu domeny. Można w tym celu wykorzystać diagram związków encji (ang. entity–relationship).
- Definiowanie zapytań aplikacji, które mogą zostać przedstawione w kontekście przepływu pracy (ang. workflow) aplikacji.
- Logiczne modelowanie danych, które polega na utworzeniu tabeli dla każdego zapytania, wykorzystując encje i związki z modelu koncepcyjnego.
- Fizyczne modelowanie danych, które polega przypisaniu typu do każdego elementu znajdującego się modelu logicznym. Możne do tego celu wykorzystać podstawowe typy danych, kolekcji i typów zdefiniowane przez użytkownika.
- Ocena i udoskonalanie modeli danych.
- Definiowanie schematu bazy danych.

1.1 Instalacja

Instalacja:

- Ze strony `https://cassandra.apache.org/download/` należy pobrać stabilną wersję (poniższe przykłady były uruchamiane dla wariantu nie wymagającej instalacji apache-cassandra-3.11.6-bin.tar.gz). Istnieje również wariant z instalatorem.
- Należy rozpakować pobrane archiwum do wybranego katalog (nazwijmy go `CASSANDRA_HOME`).
- Należy ustawić zmienną środowiskową `JAVA_HOME`. Można to zrobić z poziomu systemu operacyjnego lub w początkowej części pliku:
  `CASSANDRA_HOME\bin\cassandra.bat`
  Przykładowo jeżeli JDK jest zainstalowane w `C:\java\jdk1.8.0_202` to instrukcja miałby postać:
  `SET JAVA_HOME=C:\java\jdk1.8.0_202`

Uruchomienie serwera wykorzystując CMD:

`CASSANDRA_HOME\bin\cassandra.bat`

Uruchomienie serwera wykorzystując PowerShell:

`CASSANDRA_HOME\bin\cassandra.ps1`

1.2 CQL

Opis języka Cassandra Query Language (CQL) można znaleźć pod adresem:

`https://cassandra.apache.org/doc/latest/cql/`.

Z tej strony pochodzą definicje wszystkich instrukcji języka CQL zaprezentowane poniżej.

CQL jest o tyle podobny do SQL, że dane również przechowywane są w tabelach zawierających wiersze i kolumny.

Powłoka CQL nie jest niezbędna do realizacji laboratorium, ale jeżeli chcielibyśmy z niej korzystać, to należy:

- pobrać ze strony `https://www.python.org/downloads/` i zainstalować Python. W poniższych przykładach wykorzystano ostatnie wydanie Python 2, czyli Python 2.7.18 `https://www.python.org/downloads/release/python-2718/`
- Należy zmodyfikować zmienną środowiskową `PATH`. Można to zrobić z poziomu systemu operacyjnego lub w początkowej części pliku:

  `CASSANDRA_HOME\bin\cqlsh.bat`

  Przykładowo jeżeli Python jest zainstalowany w `C:\Python27` to instrukcja miałby postać:

  `SET PATH=C:\Python27;%PATH%`

Uruchomienie powłoki CQL:

`CASSANDRA_HOME\bin\cqlsh.bat`

Podstawowe operacje dotyczą:

- przestrzenie kluczy: CREATE KEYSPACE, USE, ALTER KEYSPACE, DROP KEYSPACE,
- tabele: CREATE TABLE, ALTER TABLE, DROP TABLE, TRUNCATE,
- typy zdefiniowane przez użytkownika UDT (ang. User-Defined Types): CREATE TYPE, ALTER TYPE, DROP TYPE,
- manipulowania danymi: SELECT, INSERT, UPDATE, DELETE, BATCH,
- indeksów wtórnych: CREATE INDEX, DROP INDEX,
- widoków zmaterializowanych: CREATE MATERIALIZED VIEW, ALTER MATERIALIZED VIEW, DROP MATERIALIZED VIEW,
- wyzwalaczy: CREATE TRIGGER, DROP TRIGGER.

Typy danych:

- podstawowe typy danych: ASCII, BIGINT, BLOB, BOOLEAN, COUNTER, DATE, DECIMAL, DOUBLE, DURATION, FLOAT, INET, INT, SMALLINT, TEXT, TIME, TIMESTAMP, TIMEUUID, TINYINT, UUID, VARCHAR, VARINT,
- kolekcje: mapy, zbiory, listy,
- typy zdefiniowane przez użytkownika UDT,
- krotki (ang. tuples) można je traktować jako anonimowe UDT z anonimowymi polami.

Do podłączenia do składu Cassandra wykorzystano "Datastax Java Driver for Apache Cassandra" w wersji 4.6. Dokumentacja jest dostępna pod adresem:

`https://docs.datastax.com/en/developer/java-driver/4.6/`

Przykład "simple" można uruchomić poprzez execSimple.bat lub poprzez wykonanie w katalogu Lab-Cassandra instrukcji:

```
1 mvn clean compile exec:java -Dexec.mainClass="pl.kielce.tu.cassandra.builder.TestCassandraBuilder"
```

Przykład "builder" można uruchomić poprzez execBuilder.bat lub poprzez wykonanie w katalogu Lab-Cassandra instrukcji:

```
mvn clean compile exec:java -Dexec.mainClass="pl.kielce.tu.cassandra.builder.TestCassandraBuilder"
```

Przykład "mapper" można uruchomić poprzez execMapper.bat lub poprzez wykonanie w katalogu Lab-Cassandra instrukcji:

```
mvn clean compile exec:java -Dexec.mainClass="pl.kielce.tu.cassandra.mapper.TestCassandraMapper"
```

Tworzenie sesji:

```
CqlSession session = CqlSession.builder().build();

CqlSession session = CqlSessionbuilder()
    .addContactPoint(new InetSocketAddress("127.0.0.1", 9042))
    .withLocalDatacenter("localdatacenter")
    .withKeyspace(CqlIdentifier.fromCql("localkeyspace"))
    .build();
```

1.2.1 Przestrzenie kluczy

Tworzenie przestrzeni kluczy:

```
create_keyspace_statement ::= CREATE KEYSPACE [ IF NOT EXISTS ] keyspace_name WITH options
```

Opcje dotyczą:

- replication - strategii replikacji (opcja obowiązkowa), możliwe są następujące klasy strategii replikacji:
  - SimpleStrategy - podany współczynnik replikacji używany jest do replikowanie danych w całym klastrze,
  - NetworkTopologyStrategy - pozwala ustawić współczynnik replikacji niezależnie dla każdego centrum danych.
- durable_writes - określa, czy używać dziennika zmian (ang. commit log) podczas aktualizacji (domyślna wartość true).

Tworzenie przestrzeni:

```
CREATE KEYSPACE IF NOT EXISTS university
WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 1};
```

Tworzenie przestrzeni kluczy przy użyciu SchemaBuilder:

```
CreateKeyspace create = SchemaBuilder.createKeyspace(keyspaceName).withSimpleStrategy(1);
session.execute(create.build());
```

Używanie przestrzeni kluczy:

```
use_statement ::= USE keyspace_name
```

Służy do ustawienia aktualnej przestrzeni kluczy. Będzie ona wykorzystana jako domyślna przestrzeń kluczy, w sytuacji gdy przestrzeń kluczy nie zostanie podana jawnie w instrukcji.

Przykład:

```
USE university;
```

Modyfikacja przestrzeni kluczy:

```
alter_keyspace_statement ::= ALTER KEYSPACE keyspace_name WITH options
```

Usuwanie przestrzeni:

```
drop_keyspace_statement ::= DROP KEYSPACE [ IF EXISTS ] keyspace_name
```

Usuwanie przestrzeni:

```
DROP KEYSPACE IF EXISTS university;
```

Usuwanie przestrzeni kluczy przy użyciu SchemaBuilder:

```
Drop drop = SchemaBuilder.dropKeyspace(keyspaceName).ifExists();
session.execute(drop.build());
```

Tworzenie tabeli:

```
create_table_statement ::= CREATE TABLE [ IF NOT EXISTS ] table_name
                           '('
                               column_definition
                               ( ',' column_definition )*
                               [ ',' PRIMARY KEY '(' primary_key ')' ]
                           ')' [ WITH table_options ]
column_definition ::= column_name cql_type [ STATIC ] [ PRIMARY KEY]
primary_key ::= partition_key [ ',' clustering_columns ]
partition_key ::= column_name
                           | '(' column_name ( ',' column_name )* ')'
clustering_columns ::= column_name ( ',' column_name )*
table_options ::= COMPACT STORAGE [ AND table_options ]
                           | CLUSTERING ORDER BY '(' clustering_order ')' [ AND table_options ]
                           | options
clustering_order ::= column_name (ASC | DESC) ( ',' column_name (ASC | DESC) )*
```

W języku CQL klucz główny (ang. primary key) składa się z dwóch części:

- klucza partycji - jest to pierwsza część definicji klucza głównego, może to być pojedyncza kolumna lub kilka kolumn ujętych w nawiasy półokrągłe (),
- kolumny klasteryzacji (ang. clustering columns) - kolumny następujące po pierwszej części definicji klucza głównego, porządek tych kolumn definiuje porządek klasteryzacji.

Klucz partycji określa, które wiersze będą przechowywane razem. Należy w taki sposób dobierać klucze partycji żeby wiersze, które będą pobierane razem znajdowały się na jednej partycji. Dzięki temu minimalizujemy liczbę węzłów, z którymi należy się skontaktować podczas pobierania.

Podczas zapisywania danych wszystkie aktualizacje dotyczące jednej partycji są wykonywane atomowo i niepodzielnie, w przeciwieństwie do sytuacji gdy zapisy dotyczą wielu partycji.

Kolumny klasteryzacji nie są obowiązkowe, ale jeżeli nie występują, to klucz główny jest równy kluczowi partycji. Powoduje to, że partycje składają się tylko z pojedynczych wierszy.

Przykład:

```
CREATE TABLE Vector3D (
    id int PRIMARY KEY,
    int x, int y, int z
);
CREATE TABLE Vector3D (
    id int,
    int x, int y, int z,
    PRIMARY KEY (id)
);
CREATE TABLE Vector3D (
    id int,
    int x, int y, int z,
    PRIMARY KEY ((id), x, y, z)
);
```

Kolumny statyczne

Wszystkie wiersze należące do tej samej partycji (posiadające ten sam klucz partycji) mają taką samą wartości kolumn statycznych.

Tworzenie tabeli:

```
CREATE TABLE students (
    id int PRIMARY KEY,
    names list<text>,
    age int,
    address frozen<address>,
    awards set<text>,
    marks map<text, double>
);
```

Tworzenie tabeli przy użyciu SchemaBuilder:

```
CreateTable createTable = SchemaBuilder.createTable("students")
    .withPartitionKey("id", DataTypes.INT)
    .withColumn("names", DataTypes.listOf(DataTypes.TEXT))
    .withColumn("age", DataTypes.INT)
    .withColumn("address", QueryBuilder.udt("address"))
    .withColumn("awards", DataTypes.setOf(DataTypes.TEXT))
    .withColumn("marks", DataTypes.mapOf(DataTypes.TEXT, DataTypes.DOUBLE));
session.execute(createTable.build());
```

Używanie tabeli:

```
alter_table_statement ::= ALTER TABLE table_name alter_table_instruction
alter_table_instruction ::= ADD column_name cql_type ( ',' column_name cql_type )*
                          | DROP column_name ( column_name )*
                          | WITH options
```

Usuwanie tabeli:

```
drop_table_statement ::= DROP TABLE [ IF EXISTS ] table_name
```

Usuwanie tabeli:

```
DROP TABLE students;
```

Usuwanie tabeli przy użyciu SchemaBuilder:

```
Drop drop = SchemaBuilder.dropTable("students");
session.execute(drop.build());
```

Usuwanie zawartości tabeli:

```
truncate_statement ::= TRUNCATE [ TABLE ] table_name
```

1.2.2 UDT

Tworzenie UDT:

```
create_type_statement ::= CREATE TYPE [ IF NOT EXISTS ] udt_name
                          '(' field_definition ( ',' field_definition )* ')'
field_definition ::= identifier cql_type
```

Tworzenie UDT:

```
CREATE TYPE address (
street text,
houseNumber int,
apartmentNumber int
);
```

Tworzenie UDT przy użyciu SchemaBuilder:

```
CreateType createType = SchemaBuilder.createType("address")
    .withField("street", DataTypes.TEXT)
    .withField("houseNumber", DataTypes.INT)
    .withField("apartmentNumber", DataTypes.INT);
session.execute(createType.build());
```

Modyfikacja UDT:

```
alter_type_statement ::= ALTER TYPE udt_name alter_type_modification
alter_type_modification ::= ADD field_definition
                          | RENAME identifier TO identifier ( identifier TO identifier )*
```

Usuwanie UDT:

```
drop_type_statement ::= DROP TYPE [ IF EXISTS ] udt_name;
```

Pobieranie danych:

```
select_statement ::= SELECT [ JSON | DISTINCT ] ( select_clause | '*' )
                     FROM table_name
                     [ WHERE where_clause ]
                     [ GROUP BY group_by_clause ]
                     [ ORDER BY ordering_clause ]
                     [ PER PARTITION LIMIT (integer | bind_marker) ]
                     [ LIMIT (integer | bind_marker) ]
                     [ ALLOW FILTERING ]
select_clause ::= selector [ AS identifier ] ( ',' selector [ AS identifier ] )
selector ::= column_name
                     | term
                     | CAST '(' selector AS cql_type ')'
                     | function_name '(' [ selector ( ',' selector )* ] ')'
                     | COUNT '(' '*' ')'
where_clause ::= relation ( AND relation )*
relation ::= column_name operator term
                     '(' column_name ( ',' column_name )* ')' operator tuple_literal
                     TOKEN '(' column_name ( ',' column_name )* ')' operator term
operator ::= '=' | '<' | '>' | '<=' | '>=' | '!=' | IN | CONTAINS | CONTAINS KEY
group_by_clause ::= column_name ( ',' column_name )*
ordering_clause ::= column_name [ ASC | DESC ] ( ',' column_name [ ASC | DESC ] )*
```

We frazie WHERE można używać kolumn, które są częścią klucza głównego lub zdefiniowano dla nich indeks wtórny.

Relacje inne niż równości lub operator IN nie są obsługiwane dla klucza partycji (ale można wykorzystać metodę TOKEN, aby wykonać tego typu zapytania na kluczu partycji).

We frazie GROUP BY możliwe jest grupowanie wierszy tylko na poziomie klucza partycji lub na poziomie kolumny klasteryzacji.

Sposoby sortowania są ograniczone porządkiem klasteryzacji zdefiniowanym w tabeli:

- jeśli tabela została zdefiniowana bez opcji CLUSTERING ORDER, to dozwolone są następujące sposoby sortowania: zgodnie oraz odwrotnie do sposobu określonego przez kolumny klastryzacji,
- w przeciwnym przypadku dozwolone są następujące sposoby sortowania: zgodnie oraz odwrotnie do sposobu określonego w opcji CLUSTERING ORDER.

Domyślnie CQL dopuszcza tylko zapytania, które nie wymagają filtrowania po stronie serwera, można to zmienić używając frazy ALLOW FILTERING.

Pobieranie danych:

```
String statement = "SELECT * FROM students;";
ResultSet resultSet = session.execute(statement);
for (Row row : resultSet) {
    System.out.print("student: ");
    System.out.print(row.getInt("id") + ", ");
    System.out.print(row.getList("names", String.class) + ", ");
    System.out.print(row.getInt("age") + ", ");
    UdtValue address = row.getUdtValue("address");
    System.out.print("{" + address.getString("street") + ", " + address.getInt("houseNumber") + ", " +
          address.getInt("apartmentNumber") + "}" + ", ");
```

```
    System.out.print(row.getSet("awards", String.class) + ", ");
    System.out.print(row.getMap("marks", String.class, Double.class) + ", ");
    System.out.println();
}
```

Pobieranie danych przy użyciu SchemaBuilder:

```
Select query = QueryBuilder.selectFrom("students").all();
SimpleStatement statement = query.build();
ResultSet resultSet = session.execute(statement);
for (Row row : resultSet) {
    System.out.print("student: ");
    System.out.print(row.getInt("id") + ", ");
    System.out.print(row.getList("names", String.class) + ", ");
    System.out.print(row.getInt("age") + ", ");
    UdtValue address = row.getUdtValue("address");
    System.out.print("{" + address.getString("street") + ", " + address.getInt("houseNumber") + ", " +
        address.getInt("apartmentNumber") + "}" + ", ");
    System.out.print(row.getSet("awards", String.class) + ", ");
    System.out.print(row.getMap("marks", String.class, Double.class) + ", ");
    System.out.println();
}
```

Wprowadzanie danych:

```
insert_statement ::= INSERT INTO table_name ( names_values | json_clause )
                     [ IF NOT EXISTS ]
                     [ USING update_parameter ( AND update_parameter )* ]
names_values ::= names VALUES tuple_literal
json_clause ::= JSON string [ DEFAULT ( NULL | UNSET ) ]
names ::= '(' column_name ( ',' column_name )* ')'
```

Wprowadzanie danych:

```
INSERT INTO students (id, names, age, address, awards, marks)
VALUES (1, ['Jan', 'Simple', 'Kowalski'], 0, {street : 'Kielecka', houseNumber : 1, apartmentNumber : 1}, {'The best student award',
    'Nobel Prize'}, {'physics':3.0, 'chemistry':5.0});
```

Wprowadzanie danych przy użyciu QueryBuilder:

```
Insert insert = QueryBuilder.insertInto("university", "students")
    .value("id", QueryBuilder.raw("1"))
    .value("names", QueryBuilder.raw("['Jan', 'Builder', 'Kowalski']"))
    .value("age", QueryBuilder.raw("0"))
    .value("address", QueryBuilder.raw("{street : 'Kielecka', houseNumber : 1, apartmentNumber : 1}"))
    .value("awards", QueryBuilder.raw("{'The best student award', 'Nobel Prize'}"))
    .value("marks", QueryBuilder.raw("{'physics':3.0, 'chemistry':5.0}"));
session.execute(insert);
```

Usuwanie danych:

```
delete_statement ::= DELETE [ simple_selection ( ',' simple_selection ) ]
                     FROM table_name
                     [ USING update_parameter ( AND update_parameter )* ]
                     WHERE where_clause
                     [ IF ( EXISTS | condition ( AND condition )*) ]
```

Usuwanie danych:

```
DELETE FROM students WHERE id = 1;
```

Usuwanie danych przy użyciu QueryBuilder:

```
Delete delete = QueryBuilder.deleteFrom("students").whereColumn("id").isEqualTo(QueryBuilder.literal(1));
session.execute(delete);
```

Przetwarzanie wsadowe:

```
batch_statement ::= BEGIN [ UNLOGGED | COUNTER ] BATCH
                          [ USING update_parameter ( AND update_parameter )* ]
                          modification_statement ( ';' modification_statement )*
                          APPLY BATCH
modification_statement ::= insert_statement | update_statement | delete_statement
```

1.3 Mapowanie tabel Cassandry na klasy Javy

Opis mapowania tabel Cassandry na klasy Javy można znaleźć pod adresem:

`https://docs.datastax.com/en/developer/java-driver/4.6/manual/mapper/`

Tabela:

```
CREATE TABLE student (
     id int PRIMARY KEY,
     names list<text>,
     age int,
     address frozen<address>,
     awards set<text>,
     marks map<text, double>
);
```

Encja:

```
@Entity
public class Student {

    @PartitionKey
    private Integer id;

    private Integer age;

    private List<String> names;

    private Address address;

    private Set<String> awards;

    private Map<String, Double> marks;

    public Student() {
    }

    public Student(Integer id, List<String> names, Integer age, Address address, Set<String> awards, Map<String, Double>
          marks) {
        this.id = id;
        this.names = names;
        this.age = age;
        this.address = address;
        this.awards = awards;
        this.marks = marks;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    public List<String> getNames() {
        return names;
    }

    public void setNames(List<String> names) {
        this.names = names;
    }

    public Address getAddress() {
        return address;
    }

    public void setAddress(Address address) {
        this.address = address;
    }

    public Set<String> getAwards() {
```

```
        return awards;
    }

    public void setAwards(Set<String> awards) {
        this.awards = awards;
    }

    public Map<String, Double> getMarks() {
        return marks;
    }

    public void setMarks(Map<String, Double> marks) {
        this.marks = marks;
    }

    @Override
    public String toString() {
        StringBuilder builder = new StringBuilder()
            .append("student {")
            .append(id + ",")
            .append(names + ",")
            .append(age + ",")
            .append(address + ",")
            .append(awards + ",")
            .append(marks + ",")
            .append("}");
        return builder.toString();
    }
}
```

Dao:

```
@Dao
public interface StudentDao {
  @Select
  Student findById(Integer id);

  @Insert
  void save(Student student);

  @Update
  void update(Student student);

  @Delete
  void delete(Student student);
}
```

Mapper:

```
@Mapper
public interface StudentMapper {
  @DaoFactory
  StudentDao studentDao(@DaoKeyspace CqlIdentifier keyspace);
}
```

Procesor adnotacji:

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>1.8</source> <!-- (or higher) -->
        <target>1.8</target> <!-- (or higher) -->
        <annotationProcessorPaths>
          <path>
            <groupId>com.datastax.oss</groupId>
            <artifactId>java-driver-mapper-processor</artifactId>
            <version>${java-driver.version}</version>
          </path>
        </annotationProcessorPaths>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Test:

```java
public class TestCassandraMapper {
    public static void main(String[] args) {
        try (CqlSession session = CqlSession.builder().build()) {
            KeyspaceManager keyspaceManager = new KeyspaceManager(session, "university");
            keyspaceManager.dropKeyspace();
            keyspaceManager.selectKeyspaces();
            keyspaceManager.createKeyspace();
            keyspaceManager.useKeyspace();

            StudentsTableSimpleManager tableManager = new StudentsTableSimpleManager(session);
            tableManager.createTable();

            CodecManager codecManager = new CodecManager(session);
            codecManager.registerAddressCodec();

            StudentMapper studentMapper = new StudentMapperBuilder(session).build();
            StudentDao dao = studentMapper.studentDao(CqlIdentifier.fromCql("university"));

            Student student = createStudent();
            dao.save(student);

            student.setAge(21);
            dao.update(student);

            Student s = dao.findById(1);
            System.out.println(s);

            dao.delete(student);
        }
    }
```

Mapowanie UDT na klasy Javy można zrealizować poprzez stworzenia niestandardowego kodeka. Opis jak to zrobić znajduje się pod adresem:

`https://docs.datastax.com/en/developer/java-driver/4.6/manual/core/custom_codecs/`

UDT Address:

```
CREATE TYPE address (
     street text,
     houseNumber int,
     apartmentNumber int
);
```

AddressCodec:

```java
public class AddressCodec extends MappingCodec<UdtValue, Address> {
    public AddressCodec(@NonNull TypeCodec<UdtValue> innerCodec) {
        super(innerCodec, GenericType.of(Address.class));
    }

    @NonNull
    @Override
    public UserDefinedType getCqlType() {
        return (UserDefinedType) super.getCqlType();
    }

    @Nullable
    @Override
    protected Address innerToOuter(@Nullable UdtValue address) {
        return address == null ? null : new Address(address.getString("street"), address.getInt("houseNumber"),
              address.getInt("apartmentNumber"));
    }

    @Nullable
    @Override
    protected UdtValue outerToInner(@Nullable Address address) {
        return address == null ? null : getCqlType().newValue().setString("street",
              address.getStreet()).setInt("houseNumber", address.getHouseNumber()).setInt("apartmentNumber",
              address.getApartmentNumber());
    }
}
```

Rejestracja kodeka:

```java
public class CodecManager extends SimpleManager{
```

```
2
3       public CodecManager(CqlSession session) {
4           super(session);
5       }
6
7       public void registerAddressCodec() {
8           CodecRegistry codecRegistry = session.getContext().getCodecRegistry();
9           UserDefinedType coordinatesUdt =
10              session
11                  .getMetadata()
12                  .getKeyspace("university")
13                  .flatMap(ks -> ks.getUserDefinedType("address"))
14                  .orElseThrow(IllegalStateException::new);
15          TypeCodec<UdtValue> typeCodec = codecRegistry.codecFor(coordinatesUdt);
16          AddressCodec addressCodec = new AddressCodec(typeCodec);
17          ((MutableCodecRegistry) codecRegistry).register(addressCodec);
18      }
19
20  }
```

Test:

```
1   public class TestCassandraMapper {
2       public static void main(String[] args) {
3           try (CqlSession session = CqlSession.builder().build()) {
4               KeyspaceManager keyspaceManager = new KeyspaceManager(session, "university");
5               keyspaceManager.dropKeyspace();
6               keyspaceManager.selectKeyspaces();
7               keyspaceManager.createKeyspace();
8               keyspaceManager.useKeyspace();
9
10              StudentsTableSimpleManager tableManager = new StudentsTableSimpleManager(session);
11              tableManager.createTable();
12
13              CodecManager codecManager = new CodecManager(session);
14              codecManager.registerAddressCodec();
15
16              StudentMapper studentMapper = new StudentMapperBuilder(session).build();
17              StudentDao dao = studentMapper.studentDao(CqlIdentifier.fromCql("university"));
18
19              Student student = createStudent();
20              dao.save(student);
21
22              student.setAge(21);
23              dao.update(student);
24
25              Student s = dao.findById(1);
26              System.out.println(s);
27
28              dao.delete(student);
29          }
30      }
```

Wszystkie przykłady dotyczące Cassandry znajdują się w katalogu LabCassandra.

Przykład ”simple” można uruchomić poprzez execSimple.bat lub poprzez wykonanie w katalogu LabCassandra instrukcji:

```
1   mvn clean compile exec:java -Dexec.mainClass="pl.kielce.tu.cassandra.simple.TestCassandraSimple"
```

Przykład ”builder” można uruchomić poprzez execBuilder.bat lub poprzez wykonanie w katalogu LabCassandra instrukcji:

```
1   mvn clean compile exec:java -Dexec.mainClass="pl.kielce.tu.cassandra.builder.TestCassandraBuilder"
```

Przykład ”mapper” można uruchomić poprzez execMapper.bat lub poprzez wykonanie w katalogu LabCassandra instrukcji:

```
1   mvn clean compile exec:java -Dexec.mainClass="pl.kielce.tu.cassandra.mapper.TestCassandraMapper"
```

Literatura

[1] Brewer E. A.,”Towards Robust Distributed Systems”, PODC Keynote, July 19, 2000

[2] http://bsonspec.org/

[3] Cattell R. A.,"Scalable SQL and NoSQL Data Stores",ACM SIGMOD Vol. 39 Issue 4, 2010

[4] Chan K. Haggarty J.,"Web Service Tutorials for WTP 1.5",http://www.eclipse.org/webtools/jst/components/ws/1.5/tutorials/index.html

[5] Connolly T., Begg C.,"Systemy baz danych",Wydawnictwo RM, 2004

[6] Coulouris G. F.,"Systemy rozproszone. Podstawy i projektowanie",WNT, 1999

[7] Creswell D.,"Getting started with JINI(TM) 2.x",http://www.dancres.org/cottage/starting_jini.html

[8] Crockford D.,"JSON: Wprowadzenie", "How JavaScript",https://www.json.org/json-pl.html

[9] Eskicioglu M.,"A Comprehensive Bibliography of Distributed Shared Memory",In Operating Systems Review, Vol. 30, No. 1, pp. 71–96, 1996.

[10] Garcia-Molina H,Salem K."Main Memory Database Systems: An Overview",IEEE Transactions on Knowledge and Data Engineering, Vol. 4, No. 6, pp. 509–516, 1992

[11] Garcia-Molina H, Ullman J. D., Widom J."Systemy baz danych, Pełny wykład",WNT, 2006

[12] Garcia-Molina H, Ullman J. D., Widom J."Systemy baz danych, Kompletny podręcznik", Wydanie II,Helion, 2011

[13] Hall M., Brown L.,"Core Java Servlet i JavaServer Pages", Tom I,Helion, 2006

[14] Hall M., Brown L.,"Core Java Servlet i JavaServer Pages", Tom II,Helion, 2009

[15] Horstmann C. S., Cornell G.,"Java 2 Core. Java Techniki zaawansowane",Helion, 2008

[16] "IDL to Java Language Mapping, Version 1.3", 2008,http://www.omg.org/spec/I2JAV/1.3/PDF

[17] Jain A.,"COMPSCI 455/555: Distributed Systems",http://cs.boisestate.edu/ amit/teaching/555/cs455-555.html

[18] "The Java™ Tutorials",http://docs.oracle.com/javase/tutorial/

[19] "Java to IDL Language Mapping, Version 1.4", 2008,http://www.omg.org/spec/JAV2I/1.4/PDF

[20] "Oracle® Database JDBC Developer's Guide 12c Release 1 (12.1) E17657-14", http://docs.oracle.com/cd/E16655_01/java.121/e17657/toc.htm

[21] "JDBC Basics", http://docs.oracle.com/javase/tutorial/jdbc/basics/index.html

[22] "The Java EE 6 Tutorial",http://docs.oracle.com/javaee/6/tutorial/doc/bnadf.html

[23] "The JNDI Tutorial Building Directory-Enabled Java Applications",http://docs.oracle.com/javase/jndi/tutorial/

[24] "DNS Service Provider for the Java Naming Directory Interface™ (JNDI)", http://docs.oracle.com/javase/1.5.0/docs/guide/jndi/jndi-dns.html

[25] Praca zbiorowa pod red. A. Karbowski i E. Niewiadomska-Szynkiewicz ,"Programowanie równoległe i rozproszone", Wydawnictwo: OWPW, 2009

[26] Praca zbiorowa pod red. A. Karbowski i E. Niewiadomska-Szynkiewicz , "Obliczenia Równoległe i Rozproszone", 2001, wyczerpana, niewznawiana

[27] Karwin B.,"Antywzorce języka SQL. Jak unikać pułapek podczas programowania baz danych",Helion, 2012

[28] Krakowiak S."Middleware Architecture with Patterns and Frameworks",February 27, 2009

[29] Krzyzanowski P., van Steen M.,"Introduction to programming with Sun/ONC RPC",http://www.cs.rutgers.edu/ pxk/rutgers/notes/rpc/

[30] "LDAP for Rocket Scientists",http://www.zytrax.com/books/ldap/

[31] May P.,"Jini and JavaSpaces",http://www.softwarematters.org/jini-intro.html

[32] Meijer, Bierman G.,"A Co-Relational Model of Data for Large Shared Data Banks",CACM, Volume 54 Issue 4, April 2011 pp. 49–58

[33] Newmarch J., "Jan Newmarch's Guide to Jini Technologies", Version 4.03 23 March 2006, http://jan.newmarch.name/java/jini/tutorial/Jini.html

[34] Özsu M. T., Valduriez P.,"Distributed database systems: where are we now?", IEEE Computer, Vol. 24, Number 8, pp. 68–78, 1991

[35] Price J.,"Oracle Database 11g i SQL Programowanie",Helion, 2009

[36] "User Guide - Basic River Services ",http://river.apache.org/user-guide-basic-river-services.html

[37] Sadalage P. J., Fowler M.,"NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence", Addison-Wesley Professional, 2012

[38] "Simple Object Access Protocol (SOAP)",http://www.w3.org/TR/SOAP

[39] Sobolewski M., "Exertion Oriented Programming", Computer Science, Texas Tech University SORCER Research Group, http://sorcer.cs.ttu.edu

[40] Sommers F., "Call on extensible RMI An introduction to JERI", http://www.javaworld.com/article/2073858/soa/call-on-extensible-rmi.html

[41] Sommers F., "Object mobility in the Jini environment, Reduce the complexity of installation and configuration of complex Jini applications with mobile code", http://www.javaworld.com/article/2076045/soa/object-mobility-in-the-jini-environment.html

[42] Stonebraker M., Cattell R.,"10 Rules for scalable Performance in 'simple operation' Datastores",CACM, Vol. 54 No. 6, pp. 72–80

[43] Michael Stonebraker,"New Sql: An Alternative to Nosql and Old Sql For New Oltp Apps", June 16, 2011

[44] Subieta K.,"Słownik terminów z zakresu obiektowości", Akademicka Oficyna Wydawnicza PLJ, 1999

[45] Tanenbaum A. S., van Steen M.,"Systemy rozproszone. Zasady i paradygmaty",WNT, Warszawa 2006

[46] "Universal Description Discovery and Integration (UDDI)",http://uddi.xml.org/, http://www.oasis-open.org/

[47] Wrembel R., Bębel B.,"Oracle. Projektowanie rozproszonych baz danych",Helion, 2003

[48] "Web Services Description Language (WSDL)", http://www.w3.org/TR/wsdl

[49] Litwin W.,"Linear hashing: A new tool for file and table addressing",In Proceedings of Very Large Data Bases (VLDB), pp. 212–223, 1980

[50] Litwin W., Neimat M.–A., Schneider D. A. ,"$LH*$ – linear hashing for distributed files",In Proceedings of ACM SIGMOD Conference, pp. 327–336, 1993

[51] Litwin W., Neimat M.A., Schneider D.A. , $RP*$: A Family of Order Preserving Scalable Distributed Data Structures, In Proceedings of Very Large Data Bases (VLDB), pp. 342–353, 1994

[52] Litwin W., Neimat M.–A., Schneider D. A., "$LH*$ – A Scalable, Distributed Data Structure", In ACM Transactions on Database Systems (TODS), Vol. 21, No. 4, pp. 480–525, 1996

[53] Devine R.,"Design and Implementation of DDH: A Distributed Dynamic Hashing Algorithm",In Proceedings of International Conference on Foundations of Data Organization and Algorithms (FODO), Vol. 730, pp. 101–114, 1993

[54] Vingralek R., Breitbart Y., Weikum G.,"Distributed file organization with scalable cost/performance", In Proceedings of the ACM International Conference on Management of Data (SIGMOD), pp. 253–264, 1994

[55] Werner Vogels "Eventually Consistent", Communications of the ACM, 2009

[56] Sapiecha K., Łukawski G., ,"Scalable Distributed Two-Layer Data Structures (SD2DS),International Journal of Distributed Systems and Technologies (IJDST), Tom: 4, Strony: 15-30, 2013

Materiały do przedmiotu dostępne są na stronie: http://weaii-moodle.tu.kielce.pl/