JDBC I ORACLE BAZA

Zlatko Sirotić, univ.spec.inf.
ISTRA TECH d.o.o.
Pula

ISTRA TECH d.o.o. & me

- ISTRA TECH je novo ime (od 2015.) poduzeća **Istra informatički inženjering**, osnovanog 1990. godine.
- Radim na informatičkim poslovima od 1984. godine.
- Oracle softverske alate (baza, Designer CASE, Forms 4GL, Reports, Java) koristim oko 25 godina.
- Objavljivao sam stručne radove na kongresima / konferencijama HrOUG, JavaCro, CASE, KOM, "Hotelska kuća", te u časopisima "Mreža", "InfoTrend" i "UT".
- Neka moja programska rješenja objavljivana su na web stranicama firmi Oracle i Quest.
- Od 2012. sam vanjski suradnik na Fakultetu informatike Pula.

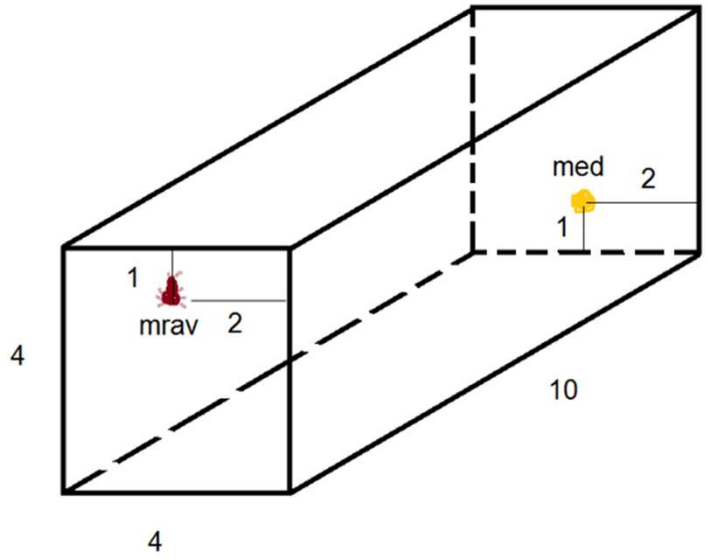
HrOUG i JavaCro & me

- Prvi put predavač na HrOUG 2002.
  Sudjelovao sam 18 puta i održao 24 predavanja.
- Prvi put predavač na JavaCro 2014.
  Sudjelovao sam 4 puta i održao 4 predavanja.
- "Odijelo ne čini čovjeka"
  i "Naslov ne čini prezentaciju", ali možda ipak pomaže:
  2004. Kako spriječiti "začarani krug"
  (rješavanje određenog tipa poslovnih pravila u Oracle BP)
  2007. Objektno-relacijske baze podataka – postoje li?
  2013. Što poslije Pascala? Pa … Scala!
  2014. Trebaju li nam distribuirane baze u vrijeme oblaka?
  2015. Višestruko nasljeđivanje - san ili Java 8?
  2020. Postoji li samo jedna "ispravna" arhitektura
  web poslovnih aplikacija

Mrav i med na prizmi (i valjku)

- Na HrOUG 2015. prvi put sam spomenuo zadatak Mrav i med na prizmi (verzija 0), unutar predavanja Povratak u Prolog.

  Na HrOUG 2019. sam najavio prezentaciju (verzija 1).



- Na stranici http://www.istratech.hr/category/blog/

nalazi se najnovija verzija (2), uz još neka predavanja:

- Mrav i med na prizmi - verzija 2
- Povratak u Prolog - verzija 2
- Strukturna složenost algoritama

Teme

- Java Database Connectivity (JDBC)
- Result set (interface ResultSet) – primjer P1ResultSet
- Prepared statement – primjer P2PreparedStatement
- Callable statement – primjer P3CallableStatement
- Korištenje ref cursora – primjer P4RefCursor
- Korištenje meta podataka – primjer P5MetaData
- Updatable result set – primjer P6UpdateResultSet
- Konekcije i connection pool – primjer P7UCP
- Connection pool i proxy user – primjer P8UCPProxy
- Row set (interface RowSet)
- Cached row set – primjer P9CachedRowSet

Java Database Connectivity (JDBC)

- JDBC je aplikacijsko programsko sučelje (API) za programski jezik Java, koje definira kako klijent može koristiti bazu podataka. JDBC je dio Java Standard Edition (JSE) platforme, koja je sada vlasništvo Oracle korporacije.
- JDBC se prvi put pojavio 1997. godine, kada je Sun Microsystem izbacio drugu verziju Jave, tj. Java Development Kit (JDK) 1.1. Od tada nadalje, JDBC je dio JSE. Trenutačno je zadnja verzija JDBC 4.3, koja je uključena u Java SE 9.
- JDBC je od početka podržavao tzv. result set (klasa ResultSet), pomoću kojeg u JDBC-u klijent pristupa podacima dobivenim iz baze. Kroz specifikaciju JSR 114, uveden je i tzv. row set, koji je nadograđen nad result setom, i koji pruža dodatne mogućnosti manipulacije podacima kroz JDBC.

Result set (interface ResultSet)

```
/* 1. primjer: ResultSet sa SELECT naredbom */ ...
public class P1ResultSet {
public static void main (String[] args) throws SQLException {
  String url = "jdbc:oracle:thin:@localhost:1521:ORCL";
  String shema = "obuka";
  String lozinka = "obuka";
  String query =
    "select naziv from m_artikli order by naziv";
  try (Connection conn =
          DriverManager.getConnection(url, shema, lozinka);
       Statement stmt = conn.createStatement();
       ResultSet rs = stmt.executeQuery(query);
      ) // try-with-resources, postoji od Jave 7
  {while (rs.next()) System.out.println(rs.getString(1));}
} // main
} // class P1ResultSet
```

Result set – Prepared statement


```
/* 2. primjer: ResultSet - PreparedStatement */ ...
public class P2PreparedStatement {
public static void main (String[] args) throws SQLException {
  String url = "jdbc:oracle:thin:@localhost:1521:ORCL";
  String shema = "obuka"; String lozinka = "obuka";
  String query =
    "select sifra, naziv from m_artikli where sifra = ?";
  try {
    Connection conn = DriverManager.getConnection
      (url, shema, lozinka);
    PreparedStatement stmt = conn.prepareStatement(query);
    stmt.setString(1, "1");
    ResultSet rs = stmt.executeQuery();
    while (rs.next())
      System.out.println(rs.getString(1) +" "+ s.getString(2));
  } catch (Exception e) {System.out.println(e.toString());}
} // main
}
```

Callable statement


```
/* 3. primjer: CallableStatement - procedura na bazi */ ...
public class P3CallableStatement {
public static void main (String[] args) throws SQLException {
  String url = "jdbc:oracle:thin:@localhost:1521:ORCL";
  String schema = "obuka"; String password = "obuka";
  String sql = "{call test1(?, ?)}";
  try {
    Connection conn =
      driverManager.getConnection(url, schema, password);
    CallableStatement cstmt = conn.prepareCall(sql);
    cstmt.setInt(1, 100);
    cstmt.registerOutParameter(2, Types.INTEGER);
    cstmt.execute();
    int newValue = cstmt.getInt(2);
    System.out.println("b = " + newValue);
  } catch (Exception e) {System.out.println(e.toString());}
} // main
}
```

Korištenje ref cursora


```
/* 4. primjer: korištenje ref cursora */ ...
public class P4RefCursor {
public static void main( String[] args ) throws SQLException {
  String url = "jdbc:oracle:thin:@localhost:1521:ORCL";
  String shema = "obuka"; String lozinka = "obuka";
  String sql = "{? = call proba_rc()}";
  try (Connection conn =
         DriverManager.getConnection(url, shema, lozinka);
       CallableStatement cstmt = conn.prepareCall(sql);)
  { cstmt.registerOutParameter(1, OracleTypes.CURSOR);
    cstmt.execute();
    ResultSet rs = (ResultSet)cstmt.getObject(1);
    while (rs.next()) {
      String sifra = rs.getString("sifra");
      String naziv = rs.getString("naziv");
      System.out.println(sifra + " " + naziv);
    }
  }
}}
```

Korištenje MetaData


```
/* 5. primjer: korištenje MetaData */ ...
public class P5MetaData {
public static void main (String[] args) throws SQLException {
  String url = "jdbc:oracle:thin:@localhost:1521:ORCL";
  String user = "obuka"; String pass = "obuka";
  String query =
    "select sifra sif, naziv naz from m_artikli order by 1";
  try (Connection conn =
         DriverManager.getConnection(url, user, pass);
       Statement stmt = conn.createStatement();
       ResultSet rs = stmt.executeQuery(query);)
  { ResultSetMetaData rsmd = rs.getMetaData();
    int columnCount = rsmd.getColumnCount();
    for (int i= 1; i <= columnCount; i++) {
      System.out.println(
        "Index: " + i + " Name: " + rsmd.getColumnName(i) +
        " Label: " + rsmd.getColumnLabel(i) );
    }
  } catch (Exception e) {System.out.println(e.toString());}}}
```

Updatable result set

- Result set ne mora služiti samo za čitanje, već i za izmjene.
- Prvo, postoje tri tipa result set kursora:

TYPE_FORWARD_ONLY (default): kursor ide samo naprijed;

TYPE_SCROLL_INSENSITIVE: može se micati naprijed ili nazad, ili na određenu poziciju; no, takav kursor ne vidi promjene na bazi – otuda riječ INSENSITIVE;

TYPE_SCROLL_SENSITIVE: može se micati naprijed, nazad, ili na određenu poziciju; dodatno, promjene na bazi (ali ne sve) reflektiraju se na result set, tj. kursor ih vidi - otuda riječ SENSITIVE; rijetki su JDBC driveri koji ga implementiraju.

Updatable result set

Result Set Type	Visibility of	Internal Delete	Internal Update	Internal Insert	External Delete	External Update	External Insert
Forward-only		No	Yes	No	No	No	No
Scroll-insensitive		Yes	Yes	No	No	No	No
Scroll-sensitive		Yes	Yes	No	No	Yes	No

- Dalje, što se tiče ažuriranja, postoje dva tipa result seta:

CONCUR_READ_ONLY (default);

CONCUR_UPDATABLE: result set može se mijenjati.

Updatable result set

- U 6. primjeru Java programa koristi se result set koji je TYPE_SCROLL_SENSITIVE i CONCUR_UPDATABLE.

- Primijetimo dvije važne stvari:

- jako je bitno da se na početku transakcije kaže **conn.setAutoCommit(false);**
  ako to ne kažemo, JDBC default ponašanje je da nakon svake DML naredbe radi automatski COMMIT;
- kad napravimo **rs.updateRow();**
  izmjena retka (UPDATE; isto vrijedi za DELETE i INSERT) **odmah se šalje na bazu**, što kao posljedicu ima i to da je za drugu sesiju baze podataka taj redak odmah zaključan, i ostaje zaključan do kraja transakcije, tj. do COMMIT ili ROLLBACK.

Updatable result set

```
/* 6. primjer: ResultSet sa UPDATE */ ...
public class P6UpdateResultSet {

public static void main (String[] args) throws SQLException {
  String url = "jdbc:oracle:thin:@localhost:1521:ORCL";
  String shema = "obuka";
  String lozinka = "obuka";
  String query =
    "select sifra, naziv from m_artikli order by naziv";
  try (Connection conn =
         DriverManager.getConnection(url, shema, lozinka);
       Statement stmt =
         conn.createStatement
          (ResultSet.TYPE_SCROLL_SENSITIVE,
           ResultSet.CONCUR_UPDATABLE);
       ResultSet rs = stmt.executeQuery(query);
      )
```

Updatable result set

```
      conn.setAutoCommit(false);
      while (rs.next()) {
        System.out.println
          ("Prije UPDATE " + rs.getString("naziv"));
        rs.updateString("naziv", "XXX");
        rs.updateRow(); // nakon ovoga, "vanjski UPDATE čeka"
        System.out.println
          ("Nakon UPDATE " + rs.getString("naziv"));
        Thread.sleep(5_000);
      }
      conn.rollback();
    } catch (Exception e) {
      System.out.println(e.toString());
    }
  } // main

} // class P6UpdateResultSet
```

Konekcije i connection pool

- Konekcije baze podataka u današnjim web i mobilnim aplikacijama rijetko se kreiraju i zatvaraju kod svake upotrebe. Razlog je taj što kreiranje konekcije traje neko vrijeme, koje nije zanemarivo (reda je npr. stotinjak milisekundi, ili više).
- Zbog toga se, za razliku od dosadašnjih primjera, konekcije često koriste kroz connection pool. Nakon upotrebe, konekcije se stvarno ne zatvaraju, već se vraćaju u connection pool (zatvaraju se logički, a ne i fizički).
- 7. primjer Java programa pokazuje korištenje connection poola (u konkretnom slučaju je to Oracle Universal Connection Pool, raspoloživ od Oracle baze 11g). U primjeru se kreira UCP koji ima maksimalno 10 konekcije, a inicijalno se kreiraju sve konekcije.

Konekcije i connection pool


```
/* 7. primjer: Universal Connection Pool (UCP) */ ...
public class P7UCP {
public static void main(String[] args) {
  try{
    PoolDataSource pds =
      PoolDataSourceFactory.getPoolDataSource();
    pds.setConnectionFactoryClassName
      ("oracle.jdbc.pool.OracleDataSource");
    pds.setURL("jdbc:oracle:thin:@localhost:1521:ORCL");
    pds.setUser("u1"); pds.setPassword("u1");
    pds.setInitialPoolSize(10); pds.setMaxPoolSize(10);
    pds.setValidateConnectionOnBorrow(true);
    System.out.println("Pripremljen CP"); Thread.sleep(5_000);

    Connection conn1 = pds.getConnection("u1", "u1");
    printStatistics(pds); Thread.sleep(10_000);
    // Available Conn: 9   // Borrowed Conn: 1
    // Closed Conn: 0      // 10 * U1
```

Konekcije i connection pool


```
Connection conn2 = pds.getConnection("u1", "u1");
printStatistics(pds); Thread.sleep(10_000);
// Available Conn: 8  // Borrowed Conn: 2
// Closed Conn: 0     // 10 * U1
conn2.close();        conn2 = null;
printStatistics(pds); Thread.sleep(10_000);
// Available Conn: 9  // Borrowed Conn: 1
// Closed Conn: 0     // 10 * U1

Connection conn2a = pds.getConnection("u1", "u1");
printStatistics(pds); Thread.sleep(10_000);
// Available Conn: 8  // Borrowed Conn: 2
// Closed Conn: 0     // 10 * U1
((ValidConnection) conn2a).setInvalid();
conn2a.close();       conn2a = null;
printStatistics(pds); Thread.sleep(10_000);
// Available Conn: 8  // Borrowed Conn: 1
// Closed Conn: 1     // 9 * U1
```

Konekcije i connection pool


```
    Connection conn2b = pds.getConnection("u1", "u1");
    printStatistics(pds);  Thread.sleep(10_000);
    // Available Conn: 7    // Borrowed Conn: 2
    // Closed Conn: 1       // 9 * U1
  } catch (Exception e) {System.out.println(e.toString());}
}

public static void printStatistics(final PoolDataSource pds) {
  JDBCConnectionPoolStatistics statistics =
    pds.getStatistics();
  System.out.println("Available Conn: " +
    statistics.getAvailableConnectionsCount());
  System.out.println("Borrowed Conn: "  +
    statistics.getBorrowedConnectionsCount());
  System.out.println("Closed Conn: "    +
    statistics.getConnectionsClosedCount());
  System.out.println("");
}
} // class P7UCP
```

Konekcije, connection pool i proxy user

- U 8. primjeru se koristi i tzv. proxy user. Naime, problem koji se može pojaviti kod connection poola jeste taj da svaki korisnik baze podataka dobiva svoj connection pool.
- To obično nije problem, jer današnje web aplikacije najčešće rade kroz samo jednog korisnika baze podataka (iako to nije baš najbolje za sigurnost baze podataka).
- Ako u nekoj aplikaciji želimo zadržati (npr. zbog sigurnosti) odnos jedna osoba = jedan korisnik na bazi, tada svakako želimo izbjeći kreiranje connection poola za svakog korisnika. Uobičajeno se to radi kroz proxy usera. On nam služi kao "vlasnik" connection poola, a "pravi" korisnici baze ulaze preko proxy usera u bazu.

Konekcije, connection pool i proxy user

- Primijetimo da (barem u Oracle bazi) **konekcija baze podataka i sesija baze podataka nisu istovrsni pojmovi!**
- **Jedna konekcija baze podataka može imati nula, jednu ili više sesija baze podataka**. Istina, najčešće se kroz jednu konekciju koristi samo jedna sesija.
- No, baš u 8. primjeru to nije tako. Vidimo da se npr. na početku otvore tri konekcije i tri sesije za proxy usera, uproxy. Međutim, kada kroz takvu konekciju uđemo kao pravi korisnik u1 (kroz proxy usera), vidimo da broj konekcija ostaje isti, a broj sesija se povećava za jedan, tj. kroz jednu konekciju sada idu dvije sesije, za korisnike uproxy i u1.
- Istina, korisnička sesija uproxy je u ovom slučaju nepristupačna, ali u nekim drugim slučajevima zaista možemo imati dvije aktivne sesije kroz istu konekciju.

Konekcije, connection pool i proxy user

```
/* 8. primjer: Universal Connection Pool i proxy user */ ...
public class P8UCPProxy {
public static void main(String args[])
  throws SQLException, Exception {
  try {
    //Create pool-enabled data source instance.
    PoolDataSource pds =
      PoolDataSourceFactory.getPoolDataSource();
    //set the connection properties on the data source.
    pds.setConnectionFactoryClassName
      ("oracle.jdbc.pool.OracleDataSource");
    pds.setURL("jdbc:oracle:thin:@localhost:1521:ORCL");
    pds.setUser("uproxy");
    pds.setPassword("uproxy");
    //Override any pool properties.
    pds.setInitialPoolSize(3);
    pds.setMaxPoolSize(4);
```

Konekcije, connection pool i proxy user

```
    System.out.println("Prije otvaranja logicke konekcije\n");
    Thread.sleep (5_000);
    //Get a database connection from the datasource.
    Connection conn1 = pds.getConnection();
... // vide se tri konekcija uproxy
    Properties prop1 = new Properties();
    prop1.put(OracleConnection.PROXY_USER_NAME, "u1");
    prop1.put(OracleConnection.PROXY_USER_PASSWORD, "u1");
    ((OracleConnection)conn1).openProxySession
      (OracleConnection.PROXYTYPE_USER_NAME, prop1);
... // vide se tri sesije uproxy i jedna u1,
    // ali samo su tri procesa (konekcije)
    Connection conn2 = pds.getConnection();
    ((OracleConnection)conn2).openProxySession
      (OracleConnection.PROXYTYPE_USER_NAME, prop1);
... // vide se tri sesije uproxy i dvije u1,
    // ali samo su tri procesa (konekcije)
```

Konekcije, connection pool i proxy user

```
    Connection conn3 = pds.getConnection();
    ((OracleConnection)conn3).openProxySession
      (OracleConnection.PROXYTYPE_USER_NAME, prop1);
... // vide se tri sesije uproxy i tri u1,
    // ali samo su tri procesa (konekcije)
    Connection conn4 = pds.getConnection();
    ((OracleConnection)conn4).openProxySession
      (OracleConnection.PROXYTYPE_USER_NAME, prop1);
... // vide se ČETIRI sesije uproxy i četiri u1,
    // ali samo su četiri procesa (konekcije)
    (OracleConnection)conn4).
      close(OracleConnection.PROXY_SESSION);
... // ostaju četiri uproxy, tri u1
... (OracleConnection)conn1).
      close(OracleConnection.PROXY_SESSION);
... // ostaju četiri uproxy, nula u1
```

Konekcije, connection pool i proxy user

```
    ((OracleConnection)conn1).openProxySession
      (OracleConnection.PROXYTYPE_USER_NAME, prop1);
... // ostaju četiri uproxy, opet jedan u1
    prop1.put(OracleConnection.PROXY_USER_NAME, "u2");
    prop1.put(OracleConnection.PROXY_USER_PASSWORD, "u2");
    ((OracleConnection)conn2).openProxySession
      (OracleConnection.PROXYTYPE_USER_NAME, prop1);
... // ostaju četiri uproxy, jedan u1 i jedan u2
  } catch (Exception e) {System.out.println(e.toString());}
} // main
public static void printStatistics(final PoolDataSource pds) {
 JDBCConnectionPoolStatistics statistics = pds.getStatistics();
 System.out.println(statistics.getAvailableConnectionsCount());
 System.out.println(statistics.getBorrowedConnectionsCount());
 System.out.println(statistics.getConnectionsClosedCount());
} // printStatistics
} // class P8UCPProxy
```

Row set (interface RowSet)

- JDBC row set nije nova stvar. Zapravo, pojavio se već u JDBC 2.0 (kada su se pojavili npr. i scrollable result set i updatable result set), ali kao opcionalni paket.
- Implementacija za row set standardizirana je kroz JDBC RowSet Implementations Specification (u JSR-114), kao ne-opcionalni paket, od Java SE 5.0 dalje. Row set se naslanja na result set, tj. RowSet sučelje nasljeđuje ResultSet sučelje.
- Neke prednosti row seta u odnosu na result set su sljedeće:

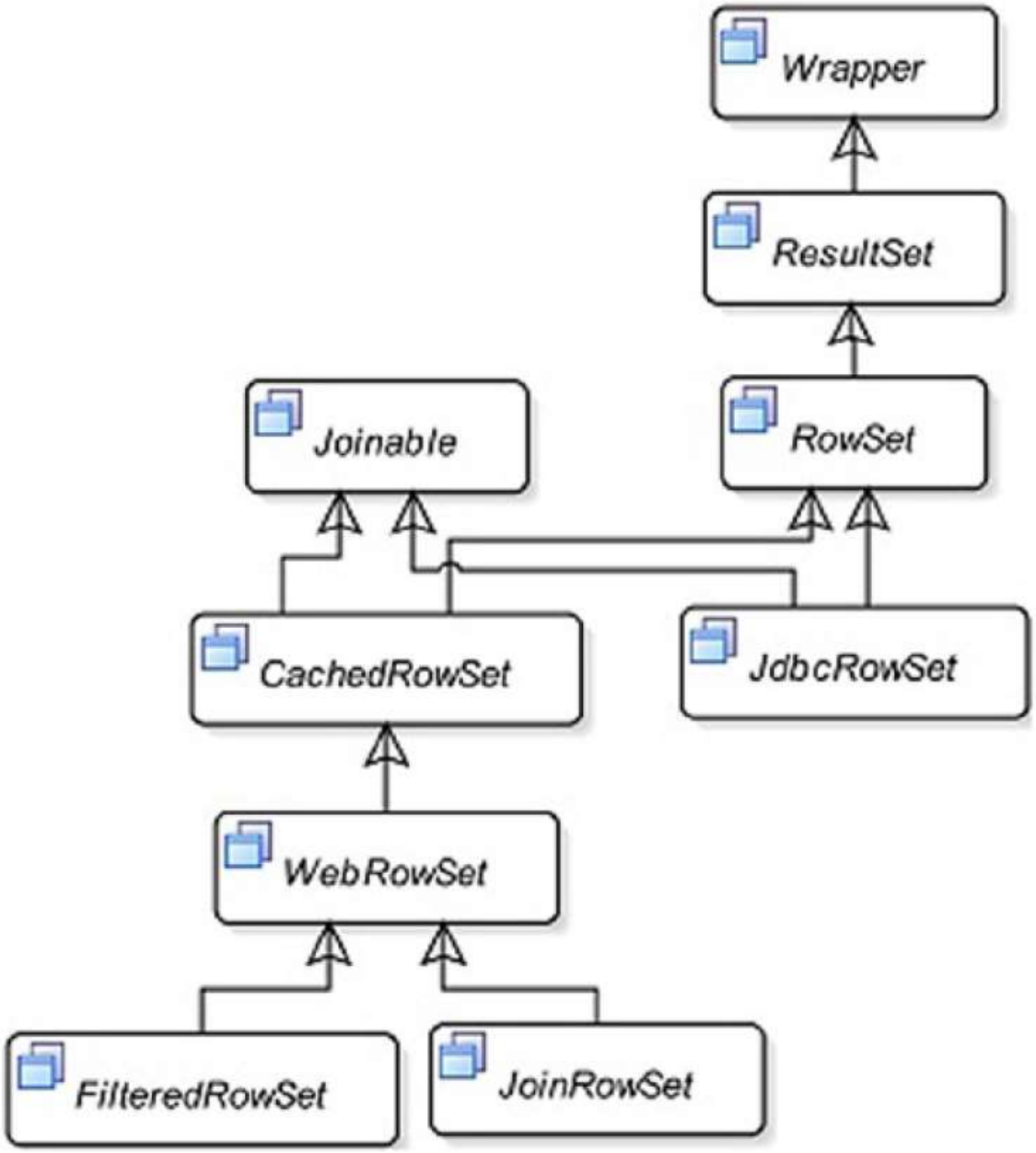
- programiranje sa row setom je jednostavnije; kada se koristi result set, mora se eksplicitno raditi s konekcijama (Connection) i naredbama (Statement), koji su kod row seta skriveni od programera;
- row set je serijabilan (Serializable; result set nije), pa **može biti poslan preko mreže, ili snimljen na disk za kasniju upotrebu;**

Row set (interface RowSet)

- result set mora uvijek biti spojen na izvor podataka, dok row set može biti i odspojen; **row set se može spojiti na bazu podataka npr. samo onda kada čita ili piše u bazu;**
- result set koristi bazu podataka kao izvor podataka; row set može koristiti i druge izvore podataka koji vraćaju podatke u tabularnom obliku;
- **serijabilnost row seta i mogućnost da radi i kada nije spojen na izvor podataka, čine row set vrlo korisnim za web ili mobilne aplikacije.**

 Row set može imati i nedostataka. Budući da se kod nekih row set klasa podaci keširaju u memoriji klijenta, može se javiti veliko zauzeće memorije (ako se na klijent učita velika količina podataka). Dalje, javlja se veća vjerojatnost nekonzistencije između podataka na klijentu i na izvoru.

Row set (interface RowSet)



Cached row set (interface CachedRowSet)

RowSet Type	Serializable	Connected to Database	Movable Across JVMs	Synchronization of data to database	Presence of JDBC Drivers
JDBC	Yes	Yes	No	No	Yes
Cached	Yes	No	Yes	Yes	No

- Kako je prikazano prethodnoj slici, CachedRowSet ima i podklase (zapravo, sučelja):
- **WebRowSet**: omogućava čitanje i pisanje podataka / metapodataka kao XML dokumenta;
- **FilteredRowSet**: omogućava filtriranje podataka na klijentskoj strani;
- **JoinRowSet**: omogućava spajanje (join) dva ili više row seta u jedan row set, na klijentskoj strani.

Cached row set (interface CachedRowSet)

- Sljedeći Java program pokazuje korištenje sučelja CachedRowSet.
- **Nakon čitanja podataka sa baze (nakon crs.execute();), klijent se automatski odspaja.**
- Dok je klijent odspojen, nad lokalnim podacima radi se
  UPDATE - crs.updateRow();
  DELETE - crs.deleteRow();
  INSERT - (crs.insertRow();
- **Nakon što se promjene pošalju na bazu sa crs.acceptChanges();**
  (a pritom se klijent automatski konektira na bazu), stvarno se ažurira baza podataka.

Cached row set (interface CachedRowSet)

- Kako je označeno u programskom kodu (kao komentar), može se desiti sljedeće:

- ako netko u međuvremenu na bazi mijenja ili izbriše (i napravi COMMIT) redak koji se lokalno mijenjao, lokalna izmjena neće imati efekta, ali se na klijentu neće javiti greška;
- ako netko u međuvremenu na bazi izbriše (i napravi COMMIT) redak koji se lokalno brisao, na klijentu se neće javiti greška;
- ako netko u međuvremenu na bazi unese (i napravi COMMIT) redak koji se lokalno unio, na bazi će se možda javiti greška - ako je zbog toga npr. narušen UK (unique key).

Cached row set (interface CachedRowSet)


```
/* 9. primjer: Cached Row Set sa UPDATE */
public class P9CachedRowSetTest {

public static void main (String[] args) throws SQLException {
  String url = "jdbc:oracle:thin:@localhost:1521:ORCL";
  String shema = "obuka";
  String lozinka = "obuka";
  String query =
    "select sifra, naziv from m_artikli order by naziv";
  try {
    OracleCachedRowSet crs = new OracleCachedRowSet();
    crs.setUrl(url);
    crs.setUsername(shema);
    crs.setPassword(lozinka);
    crs.setCommand(query);
    crs.execute();
    crs.next(); // idemo na 1.redak
```

Cached row set (interface CachedRowSet)


```
System.out.println("Prije UPDATE:"+crs.getString("naziv"));
Thread.sleep(5_000);
crs.updateString("naziv", "XXX");
crs.updateRow();
System.out.println("Nakon UPDATE:"+crs.getString("naziv"));
Thread.sleep(5_000);
// ako se izvana promijeni redak i da commit,
// ova izmjena se neće napraviti, ali ne desi se greška

crs.absolute(3);
System.out.println("Prije DELETE:"+crs.getString("naziv"));
Thread.sleep(5_000);
crs.deleteRow(); // brišemo 3. redak
System.out.println("Nakon DELETE ");
Thread.sleep(5_000);
// ako se izvana izbriše redak i da commit,
// ne desi se greška
```

Cached row set (interface CachedRowSet)


```
      crs.moveToInsertRow();
      crs.updateString("sifra", "9");
      crs.updateString("naziv", "A9");
      crs.insertRow();
      crs.moveToCurrentRow();
      System.out.println("Nakon INSERT i prije accept");
      Thread.sleep(5_000);
      // ako se izvana unese redak sa istim PK / UK i da commit,
      // desi se greška na bazi (nakon crs.acceptChanges())

      crs.acceptChanges(); // odmah je commit
      System.out.println("Nakon acceptChanges");
      Thread.sleep(5_000);
    } catch (Exception e) {System.out.println(e.toString());}
  } // main

} // class P9CachedRowSetTest
```

Cached row set (interface CachedRowSet)

- Kako je već rečeno, kod cached row seta mogu se javiti konflikti između lokalnih ažuriranja i ažuriranja koje je netko drugi u međuvremenu napravio u bazi podataka (ili drugom izvoru podataka).
- Kada se detektira konflikt kod izvršavanja metode acceptChanges(), dolazi do **SyncProviderException** iznimke. Tada se može koristiti synchronization resolver objekt (instanca od SyncResolver) za rješavanje konflikta, kako prikazuje sljedeći isječak koda:

```
catch (SyncProviderException spe) {
  // Kada acceptChanges() detektira neki konflikt
  SyncResolver resolver = spe.getSyncResolver();
  // Procedura ispisuje detalje o konfliktu
  printConflicts(resolver, cachedRs);
}
```

Cached row set (interface CachedRowSet)

- SyncResolver objekt omogućava navigaciju kroz sve konflikte, te omogućava da izmijenimo retke u row setu:

```
public static void printConflicts ...
  try {
    while (resolver.nextConflict()) {
      int status = resolver.getStatus();
      String operation = "None";
      if (status == INSERT_ROW_CONFLICT) operation = "insert";
...   // Čitamo person ID sa baze
      Object oldPerId = resolver.getConflictValue("per_id");
      // Čitamo person ID iz cached row seta
      int row = resolver.getRow();
      cachedRs.absolute(row);
      Object newPersonId = cachedRs.getObject("per_id");
      // Koristimo setResolvedValue() metodu
      // da postavimo resolved value za stupac
      // resolver.setResolvedValue(columnName, resolvedValue);
...
```

Zaključak

- JDBC je aplikacijsko programsko sučelje (API) za Javu, koje definira kako klijent može koristiti bazu podataka.
- Obično koristimo result set, i za čitanje i za ažuriranje. Kod korištenja result seta, moramo uvijek biti spojeni na izvor podataka. Izmjene (DML) se odmah šalju na bazu.
- Za razliku od toga, row set se može spojiti na bazu podataka npr. samo onda kada čita ili piše podatke u bazu. Row set je serijabilan (result set nije), pa može biti poslan preko mreže, ili snimljen na disk za kasniju upotrebu. To čini row set vrlo korisnim za web ili mobilne aplikacije.
- Kod web aplikacija u pravilu se koristi connection pool, npr. Oracle Universal Connection Pool (UCP).
- Kod korištenja connection poola u Oracle bazi, ipak je moguće zadržati odnos jedan korisnik = jedna shema, pomoću proxy usera.

Literatura

- Boyarsky, J., Selikoff, S. (2015): OCP: Oracle Certified Professional Java SE 8 Programmer II Study Guide: Exam 1Z0-809, Sybex
- Menon, R., M. (2005): Expert Oracle JDBC Programming, Apress
- Sharan, K. (2018): Java APIs, Extensions and Libraries: With JavaFX, JDBC, jmod, jlink, Networking, and the Process API, Apress
- Sierra K., Bates B. (2015): OCA/OCP Java SE 7 Programmer I & II Study Guide, McGraw-Hill Education
- Oracle priručnik (2021): Oracle Database JDBC Developer's Guide 19c (E96471-09)
- Oracle priručnik (2021): Oracle Universal Connection Pool for JDBC Developer's Guide 19c (E96473-04)