JDBC – aplikacje bazodanowe

Relacyjne bazy danych

- **Baza danych** (ang. DB – Database) to grupa danych powiązanych ze sobą pewnymi relacjami.
- **System bazodanowy** (ang. DBMS – Database Management System) zarządza przechowywanymi danymi, i użytkownikami tych danych.
- **System zarządzania relacyjnymi bazami danych** (ang. RDBMS – Relational Database Management System) to system bazodanowy, w którym organizacja danych bazuje na pojęciu relacji z matematycznej teorii mnogości.
- Historia RDBMS:
  - Twórcą teorii relacyjnych baz danych jest Edgar Frank Coode, który w 1970 roku po raz pierwszy opublikował postulaty relacyjnego modelu danych w pracy *A Relational Model of Data for Large Shared Data Banks*.
  - Pierwszy komercyjny relacyjny system zarządzania bazą danych wypuściła na rynek w 1979 roku firma *Relational Software* (później *Oracle*).

Relacyjne bazy danych

- Dane w bazie są zorganizowane w postaci tabel, widoków i schematów.
- **Tabela** (ang. table) składa się z wierszy (**rekordów**) i kolumn (**pól**).
- **Widok** (ang. view) to zakres widocznych wierszy i kolumn z jednej lub kilku tabel.
- Tabela zajmuje fizyczne miejsce w pamięci, a widok to jakby tabela wirtualna.
- **Schemat** (ang. scheme) to logiczne uporządkowanie tabel i widoków oraz relacji między nimi.
- Baza danych jest więc zestawem schematów oraz tabel i widoków, pomiędzy którymi zachodzą pewne relacje.

Relacyjne bazy danych

- System zarządzania relacyjną bazą danych zapewnia spójność danych poprzez:
  - więzy jednoznaczności (ang. unique constraint) – **klucz główny** (ang. primary key);
  - spójność referencyjną (ang. referential constraint) – **klucze obce** (ang. foreign keys);
  - warunki na wartości kolumn (ang. check constraint) – **zawężenia** wartości pól;
  - **wyzwalacze** (ang. triggers) – procedury automatycznie uruchamiane w procesie modyfikacji danych w bazie.

Relacyjne bazy danych

- **Indeksy** to dodatkowa struktura związana z tabelą, która przyspiesza dostęp do rekordów w tabeli. Indeksy, podobnie jak klucze, mogą odnosić się do jednej lub kilku kolumn w tabeli. Indeksy powodują nieznaczny spadek wydajności maszyny bazodanowej.
- **Klastry**, podobnie jak indeksy, przyspieszają dostęp do danych. Klastry to sposób przechowywania powiązanych ze sobą danych w tym samym miejscu na dysku. Klastry powodują znaczny spadek wydajności maszyny bazodanowej w przypadku modyfikacji danych.

Relacyjne bazy danych

- Ochrona danych (ang. security) to aspekty związane z dostępem do danych przez poszczególnych użytkowników. Jest ona realizowana przez system bazodanowy poprzez:
  - **przywileje** (ang. privileges) można użytkownikowi nadawać albo odbierać; przywileje dotyczą określonych operacji na rzecz określonych danych;
  - **role** (ang. roles) to zbiory przywilejów; rolę można przypisać do konkretnego użytkownika lub do innej roli.
- **Transakcje** (ang. transactions) gwarantują integralność danych w bazie. Transakcja to grupa operacji wykonywanych przez maszynę bazodanową, która tworzy logiczną całość (albo wszystkie operacje wykonają się poprawnie albo żadna nie dojdzie do skutku).

Relacyjne bazy danych

- **SQL** (ang. Structured Query Language) to język, w którym można rozmawiać z systemem bazodanowym.
- SQL nie jest językiem proceduralnym.
- Za pomocą SQLa można:
  - utworzyć bazę danych;
  - zarządzać danymi w bazie (dodawać, usuwać i modyfikować dane);
  - pobierać dane z bazy;
  - zarządzać systemem bazodanowym.
- Kurs SQL: http://www.sqlcourse.com/index.html

JDBC

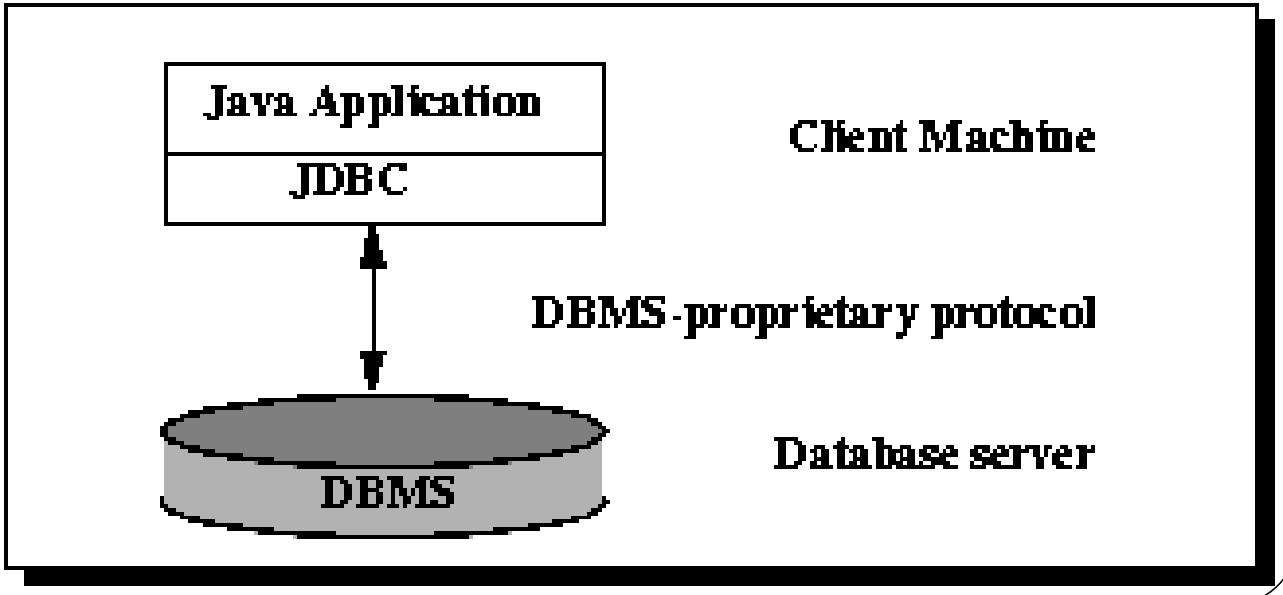
- Profesjonalne systemy bazodanowe udostępniają interfejsy programistyczne (API), dzięki którym można uzyskiwać dostęp do baz danych. Interfejsy takie są zdefiniowane dla różnych popularnych języków programowania na różne platformy systemowe.
- Programistyczny interfejs dostępu do baz danych z poziomu Javy to JDBC (ang. Java Database Connectivity API). JDBC jest uniwersalny i nie zależny od:
  - maszyny bazodanowej,
  - platformy sprzętowej,
  - systemu operacyjnego.

JDBC

- JDBC to zestaw klas i interfejsów umieszczonych w pakiecie `java.sql`, który umożliwia:
  - połączenie z relacyjną bazą danych,
  - wykonywanie instrukcji SQL na bazie,
  - przetwarzanie wyników instrukcji SQL (w tym tabel wynikowych).
- Obecna wersja JDBC to 4.2 API.

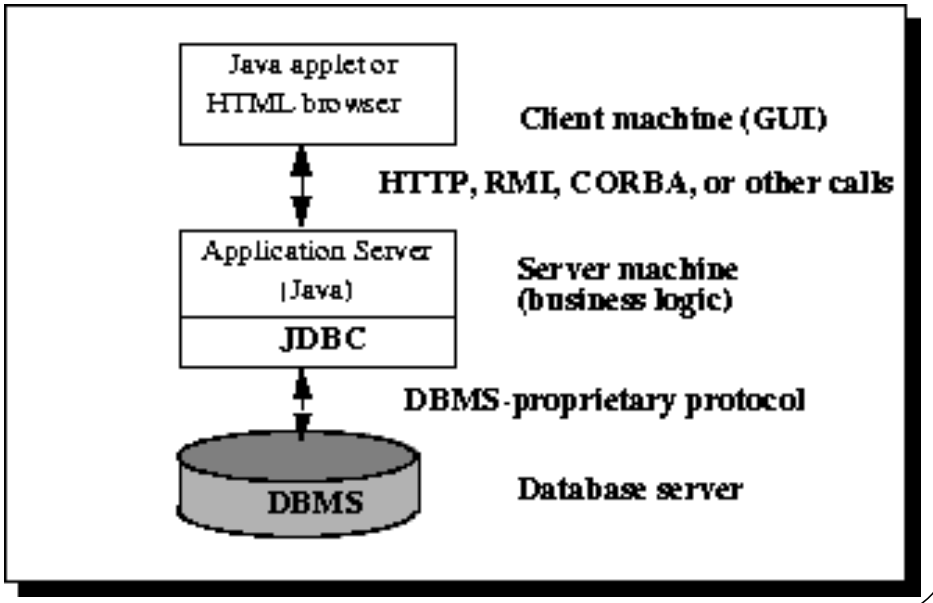
Architektura aplikacji bazodanowych

- Architektura dwuwarstwowa:
  komputer klienta (aplikacja Javy, JDBC)
  <- protokół dostępu do bazy ->
  serwer bazodanowy (RDBMS)



Architektura aplikacji bazodanowych

- Architektura trójwarstwowa:
  komputer klienta (aplikacja Javy, przeglądarka WWW)
  <- protokół komunikacji z serwerem (HTTP, RMI) ->
  maszyna serwera (serwer aplikacji, JDBC)
  <- protokół dostępu do bazy ->
  serwer bazodanowy (RDBMS)



Struktura JDBC

- Sterownik JDBC do określonej bazy danych to zestaw klas, które implementują interfejsy w pakiecie `java.sql`.
- Niezależność bazodanowa osiągnięta dzięki zestawowi interfejsów w pakiecie `java.sql`, implementowanych przez różnych producentów.
- Zadaniem JDBC jest ukrycie przed programistą wszelkich specyficznych właściwości określonej bazy danych.

Struktura JDBC

- `java.sql.Driver` to interfejs odpowiedzialny za nawiązanie połączenia z bazą danych oraz za tłumaczenie zapytań SQLowych na język określonej bazy.
- `java.sql.DriverManager` to klasa, która odpowiada za zarządzanie listą dostępnych sterowników do baz danych i udostępnienie aplikacji tego jednego, który jest przez nią wymagany.
- `java.sql.Connection` to interfejs, który reprezentuje pojedynczą transakcję bazodanową.
- `java.sql.Statement` jest to interfejs reprezentujący zapytanie SQLowe.
- `java.sql.ResultSet` to interfejs reprezentujący zbiór rekordów, będących wynikiem zapytania SQLowego; wynik zapytania znajduje się po stronie bazy danych.

Sterowniki JDBC

- JDBC-ODBC Bridge Driver – to sterownik pomostowy, czyli wykorzystujący inny ogólny mechanizm dostępu do baz danych (na przykład ODBC).
- Native API Partly Java Driver – to sterowniki napisane w Javie, które wykorzystują do uzyskania dostępu do bazy danych biblioteki napisane w innych językach (na przykład C++).
- Net-Protocol Pure Java Driver – to sterownik, który uzyskuje dostęp do bazy danych na poziomie serwera, używając do tego celu protokołu sieciowego (dostęp pośredni przez serwer).
- Native-Protocol Pure Java Driver – to sterowniki, które wykorzystują protokoły sieciowe wbudowane w maszyny bazodanowe (dostęp bezpośredni do systemu bazodanowego).

URL dla bazy danych

- URL dla bazy danych umożliwia identyfikację bazy, załadowanie odpowiedniego sterownika i uzyskanie połączenia z bazą.
- Składnia URLa używanego w JDBC:
  `jdbc:<subprtokół>:<subnazwa>`
  gdzie `<subprtokół>` to nazwa sterownika lub mechanizmu obsługi połączenia z bazą danych a `<subnazwa>` to identyfikator bazy danych.

URL dla bazy danych

- Przykład URLa dla połączenia z bazą zarejestowaną w ODBC:
  `jdbc:odbc:moja_baza`
- Przykłady URLi dla różnych systemów bazodanowych:
  - MySQL: `jdbc:mysql://localhost:3306/mydb`
  - MS-SQL: `jdbc:sqlserver://127.0.0.1:1433/mydb`
  - PostgreSQL: `jdbc:postgresql://serv:5432/mydb`
  - Mini-SQL: `jdbc:msql://dbserver:1234/mydb`
  - Oracle: `jdbc:oracle:thin:@localhost:1521:orcl`
  - Java-DB: `jdbc:derby:newdb;create=true`

Pliki JAR zawierające sterownik do RDBMS

- Większość producentów systemów bazodanowych dostarcza sterowniki javowe do swoich baz w postaci plików JAR.
- Przed uruchomieniem programu korzystającego z bazy danych musimy zlokalizować taki sterownik w swoim systemie (na przykład dla bazy Apache Derby będzie to plik `derbyclient.jar`).
- Uruchamiając program z wiersza poleceń trzeba określić położenie tego sterownika:
  `java -cp .:sterownik.jar Program`

Pliki JAR zawierające sterownik do RDBMS

- Program w javie musi zarejestrować klasę sterownika JDBC.
- Automatyczna rejestracja jest wymagana dla sterowników zgodnych z JDBC4.
- W starszych wersjach trzeba ręcznie dokonać rejestracji:
  - poprzez załadownie sterownika do pamięci:
    `Class.forName(”org.postgresql.Driver”);`
  - albo przez skonfigurowanie właściwości `jdbc.drivers` przy uruchamianiu programu:
    ```
    java –Djdbc.drivers=org.postgresql.Driver
        Program
    ```
  - sama aplikacja też może skonfigurować tą właściwość:
    ```
    System.setProperty(”jdbc.drivers”,
        ”org.postgresql.Driver”);
    ```

Metainformacje o bazie danych

```
Connection conn; // połączenie z bazą
DatabaseMetaData md; // metadane
// należy połączyć się z bazą danych…
md = conn.getMetaData();
// …
md.getDatabaseProductName();
md.getDatabaseProductVersion();
md.getDriverName();
md.getURL();
md.getUserName();
md.supportsAlterTableWithAddColumn();
md.supportsAlterTableWithDropColumn();
md.supportsBatchUpdates();
md.supportsPositionedDelete();
md.supportsPositionedUpdate();
md.supportsTransactions();
md.supportsResultSetType(ResultSet.TYPE_SCROLL_INSENSITIVE);
md.supportsResultSetType(ResultSet.TYPE_SCROLL_SENSITIVE);
```

Łączenie się z bazą danych

- Połączenie z bazą uzyskujemy za pomocą klasy `DriverManager`:

```
public Connection getConnection (String urlDB)
throws SQLException
{
    Connection conn = null;
    Properties connectionProps = new Properties();
    connectionProps.put("user", this.userName);
    connectionProps.put("password", this.password);
    conn = DriverManager.getConnection(
        urlDB,  connectionProps);
    System.out.println("Connected to database");
    return conn;
}
```

- Połączenie z bazą należy na końcu zamknąć:

```
try {
    if (conn!=null) conn.close();
} catch (SQLException ex) {/*…*/}
```

Wykonywanie zapytań do bazy

- Najprostszym sposobem wykonania zapytania do bazy jest użycie obiektu `Statement`.
- Obiekt `Statement` nie tworzymy bezpośrednio, tylko używamy metody `createStatement` obiektu `Connection`:
```
Statement stm = conn.createStatement();
```
- Zapytanie SQLowe `SELECT`, które da nam w wyniku tabelę z rekordami możemy wykonać metodą `executeQuery`:
```
String query = "SELECT * FROM tabela;";
ResultSet rs = stm.executeQuery(query);
```
- Obiekt `ResultSet` reprezentuje tabelę rekordów z danymi; poruszanie się po tych rekordach jest realizowane za pomocą metod `next` i `previous`.
- Obiekt `ResultSet` jest powiązany ze swym macierzystym obiektem `Statement`; jeśli macierzysty obiektem `Statement` użyjemy do wykonania kolejnego zapytania, to obiekt `ResultSet` zostanie automatycznie zamknięty.

Wyjątki SQL

- Często zdarza się, że jakaś operacja na bazie danych zakończy się niepowodzeniem – wtedy zgłaszany jest wyjątek `SQLException`.
- W czasie operacji na bazie może zostać zgłoszonych kilka wyjątków – wyjątek `SQLException` potrafi się kolejkować.
- Łapanie wyjątku `SQLException` na przykładzie:

```
try {
    // operacje na bazie danych
}
catch (SQLException ex) {
    do {
        System.err.println(ex.getMessage());
    } while ((ex=ex.getNextException())!=null);
}
```

- Czasami pojawiają się tylko ostrzeżenia w postaci wyjątków `SQLWarning` dziedziczących po `SQLException`.

Przykład prostego użycia JDBC

```
String db = "jdbc:default:connection";
String login = "itsme";
String pass = "*****";
Connection conn = null;
Statement stm = null;
String query = "SELECT a, b, c FROM table";
try {
  conn = DriverManager.getConnection(db,login,pass);
  stm = conn.createStatement();
  ResultSet rs = stm.executeQuery(query);
  while (rs.next()) {
    int x = rs.getInt("a");
    String s = rs.getString("b");
    float f = rs.getFloat("c");
    // do something with x, s, f
  }
}
finally {
  try { if (stm != null) stm.close();
  } catch (Exception ex) {}
  try { if (conn != null) conn.close();
  } catch (Exception ex) {}
}
```

Modyfikowanie danych w bazie

- Do modyfikowania danych w tabelach korzystamy z poleceń SQLowych `UPDATE`, `INSERT` i `DELETE`; polecenia te nie zwracają wyniku w postaci tabeli rekordów, tylko liczbę określającą ilość dokonanych modyfikacji.
- Do modyfikowania danych w tabelach używamy metody `executeUpdate`:

```
Statement stm = con.createStatement();
String query = "DELETE FROM os WHERE wiek<18";
int cnt = stm.executeUpdate(query);
con.close();
//…
```

Polecenia SQLowe do bazy

- W sytuacji, gdy nie wiemy czy polecenie SQLowe będzie wyciągać rekordy z danymi z bazy, czy będzie modyfikować dane, używamy polecenia `execute`,
- Polecenie `execute` zwraca wartość typu `boolean`, która jest `true`, gdy baza odpowiada tabelą rekordów (obiekt `ResultSet`) albo `false`, gdy otrzymujemy liczbę zmodyfikowanych rekordów.
- Po wykonaniu polecenia metodą `execute` można otrzymać referencję do obiektu `ResultSet`:
  ```
  ResultSet rs = stm.getResultSet();
  ```
  albo liczbę zmian dokonanych w tabeli:
  ```
  int cnt = stm.getUpdateCount();
  ```

Obsługa pól z wartością NULL

- Odczytując wartość pola numerycznego wskazywanego przez `ResultSet` metodą `getInt` możemy otrzymać wartość 0 lub -1 – wtedy nie wiemy, czy jest to wartość zapisana w tym polu czy może efekt przekształcenia `null` do typu `int`.
- Rozwiązaniem tego problemu jest metoda `wasNull`, która zwraca `boolean` i mówi nam czy ostatnio przeczytana wartość była `null`:

```
int wiek = rs.getInt("wiek");
if (rs.wasNull()) {/*…*/}
```

- Innym rozwiązaniem jest użycie metody `getObject`:

```
Integer wiek = (Integer)rs.getObject("wiek");
if (wiek==null) {/*…*/}
```

Jak zarejestrować bazę w ODBC (Windows 10)

- *Panel sterowania* → *Narzędzia administracyjne* → *Źródła danych (ODBC)*
- Okno *Administrator źródeł danych ODBC* : w zakładce *DSN użytkownika* naciśnij przycisk *Dodaj*
- Okno *Tworzenie nowego źródła danych*: z listy sterowników wybierz odpowiedni typ (np. *Driver do Microsoft Excel .xls*) i naciśnij przycisk *Zakończ*
- Okno *Ustawienia dla programu … (np. Microsoft Excel)* : w polu nazwa źródła danych wpisz swoją nazwę (np. *kontakty*), w polu opis możesz dodać komentarz do bazy, dalej w ramce *Baza danych* wybierz wersję programu (systemu bazodanowego) i wskaż samą bazę przyciskiem *Wybierz… (np. skoroszyt)*, na koniec naciśnij przycisk *OK.*

- Uwaga 1: w przypadku danych zapisanych w Excelu nazwa tabeli to nazwa arkusza (np. dla arkusza *abc* nazwa tabeli to *[abc$]*).
- Uwaga 2: w jednym arkuszu można zapisać jedną tabelę; w pierwszym wierszu są zapisane nazwy kolumn.
- Uwaga 3: nie należy używać polskich znaków diakrytycznych w nazwach arkuszy i w nazwach kolumn.

Obsługa transakcji

- Obsługa transakcji jest w JDBC sterowana obiektem `Connection`.
- Domyślnie nowe połączenie z bazą jest typu auto-commit – każde polecenie do bazy jest pojedynczą transakcją.
- Aby samodzielnie sterować transakcjami trzeba właściwość auto-commit ustawić na `false`:
  `conn.setAutoCommit(false);`
- Po wykonaniu kilku poleceń na bazie możemy je zatwierdzić poleceniem `commit()` albo anulować poleceniem `rollback()`.
- Istnieje też metoda `getAutoCommit()`, która informuje o trybie auto-commit obiektu `Connection`.

Obiekt `ResultSet`

- Obiekt `ResultSet` reprezentuje zbiór uporządkowanych danych (tabela z danymi).
- Dane z obiektu `ResultSet` odczytuje się, podobnie jak z pliku, za pomocą kursora, który przesuwa się nad kolejnymi rekordami w tabeli.
- `ResultSet` jest tworzony przez obiekt `Statement` po wykonaniu metody `executeQuery` albo `execute` z poleceniem *SELECT*.
- Zasoby z danymi, do których odwołuje się `ResultSet` znajdują się po stronie systemu bazodanowego.
- Obiekt `Statement` utworzony za pomocą:
  `Statement stm = conn.createStatement();`
  może wygenerować obiekt `ResultSet` typu *forward-only*.

Obiekt `ResultSet`

- Przy tworzeniu obiektu `Statement` można określić jak będzie się zachowywać obiekt `ResultSet`:
  ```
  Statement stm =
      conn.createStatement(rsType, rsCon);
  ```
  przy czym oba argumenty `createStatement` są typu `int` a odpowiadające im stałe są zdefiniowane w klasie `ResultSet`.
- Argument `rsType` określa możliwości poruszania kursorem.
- Argument `rsCon` określa możliwość równoczesnego wprowadzania zmian w tabelach.

Obiekt `ResultSet`

- Argument `rsType` może przyjmować następujące wartości:
  - `ResultSet.TYPE_FORWARD_ONLY`
    kursor można przesuwać tylko do przodu po jednym rekordzie;
  - `ResultSet.TYPE_SCROLL_INSENSITIVE`
    kursor można dowolnie przesuwać, dane w tabeli nie są wrażliwe na równoczesne zmiany;
  - `ResultSet.TYPE_SCROLL_SENSITIVE`
    kursor można dowolnie przesuwać, dane w tabeli są wrażliwe na równoczesne zmiany.
- Aby można było przemieszczać kursor w dowolnie wybrane pozycje należy wybrać jeden z typów `TYPE_SCROLL_INSENSITIVE` lub `TYPE_SCROLL_SENSITIVE`.
- Przykład obiektu `Statement`, który może utworzyć `ResultSet`, po którym będzie można dowolnie przemieszczać kursor:

```
Statement stm = conn.createStatement(
    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_READ_ONLY);
```

Obiekt `ResultSet`

- Metody przemieszczające kursor w obiekcie `ResultSet`:
  - `boolean next ()`
  - `boolean previous ()`
  - `boolean first ()`
  - `boolean last ()`
  - `boolean beforeFirst ()`
  - `boolean afterLast ()`
  - `boolean absolute (int n)`
  - `boolean realtive (int n)`
- Metody przemieszczające kursor w obiekcie `ResultSet` zwracają wartość boolowską `true`, gdy kursor ustawi się na jakimś rekordzie w tabeli albo `false`, gdy kursor wyskoczy poza tabelę.
- Rekordy w obiekcie `ResultSet` są numerowane od 1.

Obiekt `ResultSet`

- Kursor, który można dowolnie przemieszczać po tabeli z rekordami może być albo nie być wrażliwy na zmiany dokonywane równolegle przez innych użytkowników w bazie.
- Aby kursor był nie wrażliwy na zmiany w bazie należy wybrać typ kursora jako `TYPE_SCROLL_INSENSITIVE` (dane nie będą się zmieniały w tabeli od momentu ich odczytania).
- Aby kursor był wrażliwy na zmiany w bazie należy wybrać typ kursora jako `TYPE_SCROLL_SENSITIVE` (w momencie ustawienia kursora na danym rekordzie z bazy danych ściągane są aktualne wartości danych przypisanych do pól w bieżącym rekordzie).
- Przykład obiektu `Statement`, który może utworzyć `ResultSet`, po którym będzie można dowolnie przemieszczać kursor i który będzie wrażliwy na zmiany w bazie:

```
Statement stm = conn.createStatement(
    ResultSet.TYPE_SCROLL_SENSITIVE,
    ResultSet.CONCUR_READ_ONLY);
```

Obiekt `ResultSet`

- Argument `rsCon` decyduje o możliwości modyfikowania danych w bazie i może przyjmować następujące wartości:
  - `ResultSet.CONCUR_READ_ONLY`
    kursor możne tylko czytać dane z tabeli;
  - `ResultSet.CONCUR_UPDATABLE`
    rekord, na który wskazuje kursor można dowolnie zmodyfikować i zmiany te będą zapamiętane w bazie danych.
- Przykład obiektu `Statement`, który może utworzyć `ResultSet`, w którym będzie można modyfikować dane:

```
Statement stm = conn.createStatement(
    ResultSet.TYPE_SCROLL_SENSITIVE,
    ResultSet.CONCUR_UPDATABLE);
```

Modyfikowanie rekordów

- Metody modyfikujące dane w rekordzie pod kursorem w obiekcie `ResultSet`:
  - `void updateXxx (int col, Xxx val)`
  - `void updateXxx (String col, Xxx val)`
  - `void updateNull (int col)`
  - `void updateNull (String col)`
  - `void updateRow ()`
  - `boolean rowUpdated ()`
  - `void cancelRowUpdates ()`
- Metody `updateXxx` modyfikują pole typu `Xxx`, na przykład `updateInt`, `updateDouble`, `updateString`.
- Pola w rekordzie są numerowane od 1.
- Metoda `updateRow` zapisuje zmienione wartości i musi być wywołana przed przejściem do następnego rekordu.

Usuwanie rekordów

- Metody usuwające rekord pod kursorem w obiekcie `ResultSet`:
  - `void deleteRow ()`
  - `boolean rowDeleted ()`
- Metody `deleteRow` nie wolno używać w stosunku do nowowstawionego rekordu.

Wstawianie rekordów

- Metody wstawiające nowy rekord do tabeli w obiekcie `ResultSet`:
  - `void moveToInsertRow ()`
  - `void insertRow ()`
  - `boolean rowInserted ()`
- Metoda `moveToInsertRow` przesuwa kursor do miejsca, w którym będzie można zapisać nowy rekord.
- Po wypełnieniu rekordu danymi za pomocą metod `updateXxx` wstawiamy nowy rekord do tabeli metodą `insertRow`.

Metainformacje o bazie danych

- Metainformacje o bazie danych uzyskujemy za pomocą metody `getMetaData`:
  ```
  DatabaseMetaData md =
  conn.getMetaData();
  ```
- W obiekcie `DatabaseMetaData` mamy cały zbiór metod do wyciągania informacji o bazie danych:
  - `getDatabaseProductName ()`
  - `getDatabaseProductName ()`
  - `getDriverName ()`
  - `getURL ()`
  - `getUserName ()`
  - `supportsTransactions ()`
  - i wiele innych

Metainformacje o tabeli wynikowej

- Metainformacje o tabeli wynikowej uzyskujemy za pomocą metody `getMetaData`:
  ```
  ResultSetMetaData rsmd =
  rs.getMetaData();
  ```
- W obiekcie `ResultSetMetaData` mamy zbiór metod do wyciągania informacji o tabeli wynikowej:
  - `getcolumnCount ()`
  - `getColumnName (int n)`
  - `getColumnDisplaySize (int n)`
  - `getColumnType (int n)`
  - `getColumnTypeName (int n)`
  - `getColumnClassName (int n)`
  - i wiele innych

Bezpieczeństwo transakcji

- Baza danych blokuje tabele i rekordy na czas wykonywania modyfikacji.
- Rozwiązanie problemu:
  `SELECT … FOR UPDATE;`

Zapytanie preinterpretowane

- W JDBC można tworzyć procedury parametryzowane, w których podaje się wzorzec ze znacznikami (pytajniki) zastępowanymi konkretnymi wartościami przed wywołaniem (obiekty typu `PreparedStatement`).
- Obiekt klasy `PreparedStatement` tworzymy podobnie jak `Statement`:
  ```
  PreparedStatement stmt =
  conn.prepareStatement("…");
  ```
- W parametrze mogą wystąpić znaki zapytania (numerowane od 1), które uzupełniamy odpowiednimi wartościami za pomocą metod `setXxx` (gdzie `Xxx` oznacza typ parametru, na przykład `stmt.setString(nr,arg)`, `stmt.setInt(nr,arg)`, itp); po wypełnieniu znaczników konkretnymi wartościami można wykonać polecenie (używając na przykład metody `execute()`).

Procedury przechowywane w bazie danych

- Większość komercyjnych baz danych posiada wewnętrzny język programowania (PL/SQL w przypadku Oracle'a), w którym można definiować procedury działające na bazach danych (obiekty typu `CallableStatement`).
- Obiekt klasy `CallableStatement` tworzymy podobnie jak `Statement`:
  `CallableStatement stmt = conn.prepareCall("…");`
- W parametrze mogą wystąpić znaki zapytania (numerowane od 1), które uzupełniamy odpowiednimi wartościami za pomocą metod `setXxx` (gdzie `Xxx` oznacza typ parametru, na przykład `stmt.setString(nr,arg)`, `stmt.setInt(nr,arg)`, itp); parametry wyjściowe rejestrujemy metodą `registerOutParameter`; po wypełnieniu znaczników konkretnymi wartościami można wykonać polecenie (używając na przykład metody `execute()`).

Procedury przechowywane w bazie danych

- Przykład procedury napisanej w PL/SQL:

```
CREATE OR REPLACEPROCEDURE daj_wiek (
  id IN INTEGER,
  w OUT INTEGER
) IS
BEGIN
  SELECT wiek INTO w FROM osoba
  WHERE osoba.id = id
EXCEPTION
  WHEN NO_DATA_FOUND
    THEN w := -1;
  WHEN OTHERS
    THEN NULL;
END;
```

- Przykład programu korzystającego z powyższej procedury:

```
String st = "{call daj_wiek(?,?)}";
CallableStatement cstm = conn.prepareCall(st);
cstm.registerOutParameter(2,java.sql.Types.ITEGER);
cstm.setInt(1,id);
cstm.execute();
```

Wsadowe zapytania modyfikujące

- Polecenia modyfikujące (`INSERT`, `DELETE` i `UPDATE`) można wysłać do bazy danych w postaci bloku poleceń, wykorzystując obiekt `Statement`.
- Metoda `addBatch("...")` dodaje polecenie do kolekcji poleceń w obiekcie `Statement`.
- Polecenia wsadowe są wykonywane za pomocą metody `executeBatch()` w takiej kolejności w jakiej były dodawane do `Statement`; metoda `executeBatch()` zwraca tablicę `int[]`, w której są umieszczane wyniki wykonania kolejnych poleceń.

Literatura (JDBC)

- M.Grochala: *Java – aplikacje bazodanowe Wydanie 2. Rozdział 5: JDBC, rozdział 6: URL w aplikacjach bazodanowych, rozdział 7: Aplikacje bazodanowe, rozdział 9: Zaawansowane techniki obsługi bazy danych.* Wydawnictwo HELION, Gliwice 2001.
- K.Barteczko: *Java – od podstaw do technologii. Tom 2, część C, rozdział 1: Java i bazy danych (JDBC).* Wydawnictwo MIKOM, Warszawa 2004.
- C.S.Horstmann, G.Cornell: *Core Java – techniki zaawansowane. Wydanie 8. Rozdział 4: Połączenia do baz danych (JDBC).* Wydawnictwo HELION, Gliwice 2009.
- JDBC(TM) Tutorial: http://download.oracle.com/javase/tutorial/jdbc/