**Ներածություն**

JDBC – java database connectivity:

JDBC – ն հանդիսանում է JavaSE (standart edition) – ի մաս և իրենից ներկայացնում է java api, որի միջոցով կարող ենք աշխատել տվյալների բազաների հետ։

**JDBC Drivers**

Db – ին միանալու համար jdbc – ն օգտագործում է դրայվերներ։ Driver – ը դա հենց այն ծրագրային կոմպոնենտն է որը թույլ է տալիս Java application – ին աշխատել տվյալների բազայի հետ։ Driver – ը JDBC call – երը կոնվերտացնում է տվյալների բազայի սպեցիֆիկ call – երի: Յուրաքանչյուր db – ի համար պետք է դրայվեր։

Տվյալների բազաներին միանալու համար JDBC – ն օգտագործում է դրայվերներ, որոնք լինում են 4 տեսակի։

* JDBC-ODBC Bridge Driver (open database connectivity)

JDBC call - երի կոնվերտացնում է ODBC call - երի, որից հետո կանչվում է ODBC – ի դրայվերը, որի call – երը vendor database library – ի համար nativ են և արդեն կոնվերտացնում է տվյալների բազայի սպեցիֆիկ call – երի։ ODBC – ի կոնվերտացվում է տարբեր տիպի db – ների համար որոշակի ունիվերսալություն ապահովելու համար։ Ամենադանդաղն է քանի որ կազմված է մի քանի մակարդակներից։ Նաև պետք է JDBC – ից բացի ունենալ ODBC driver։ Java 8 – ից սկսած support չի արվում։

Native Driver

Շատ նման է 1 – ին տիպի դրայվերին այն տարբերությամբ որ արանքից դուրս է գալիս ODBC – ի մասը և call - երը միանգամից փոխարինվում է vendor database library – ի համար nativ call – երով։

Պետք է instal անել vendor – ը։ Որ բոլոր db – ի provider – ներն են որ տրամադրում են vendor api:

Network Protocol Driver

JDBC – ի կանչը ուղարկում է միջանկյալ սերվերի, որն էլ իր հերթին կոնվերտացնում է call - երը և միանում տվյալների բազային։ Միջանկյալի ռեսուրսի համար պետք է լրացուցիչ կոդ, նաև ազդում է արագության վրա։

-Thin Driver

Ամբողջությամբ Java դրայվեր է, որը անմիջականորեն միանում է տվյալների բազային և միաժամանակ կատարում է նաև vendor – ի գործառույթները այսինք JDBC call – երը միանգամից կոնվերտացնում է db սպեցիֆիկ call - երի։ Յուրաքանչյուր տիպի database ունի կոնկրետ իր համար ռեալիզացվածthin driver։ Դրայվերների տիպերից ամենաարդյունավետն է և չի պահանջում միջանկյալ api – ներ։

**Java database connectivity steps**

JDBC api – ի միջոցով տվյալների բազային միանալու համար անհրաժեշտ են հետևյալ գործողությունները։

* Հայտարարել driver կլասս
* Ստեղծել connection
* Ստեղծել statement
* Իրականացնել querie – ները
* Փակել connection – ը

JDBC 4.0 – ից սկսած դրայվեր կլասսի հայտարարում Class.forName() չի պահանջվում։ DriverManager – ի վրա կանչվում է registerDriver() ստատիկ մեթոդը և որպես պարրամետր փոխանցվում է տվյալների բազային համապատասխան դրայվերի կլասսի էկզեմպլյար։

Connection ինտերֆեյսի տիպի օբյեկտ ստեղծվում է DriverManager – ի getConnection() ստատիկ մեթոդով։ Մեթոդը որպես պարրամետր ընդունում է տվյալների բազայի url – ը նաև կարող է ընդունել user, password: Throw է անում SQLException:

Statement ինտերֆեյսի տիպի օբյեկտ ստեղծվում է Connection ինտերֆեյսի տիպի օբյեկտի վրա createStatement() մեթոդի միջոցով։ Statement – ը պատասխանատու է տվյալների բազայում querie – ների իրականացման համար: Statement ինտերֆեյսի մեթոդներից exequteQuery() – ը վերադարձնում է ResultSet տիպի օբյեկտ որի միջոցով հասանելիություն ենք ստանում querie – ի արդյունքներին։ Աշխատանքի ավարտից հետո connection – ն փակվում է connection տիպի օբյեկտի close() մեթոդով։

Հիմա ավելի մանրամասն խոսեմ JDBC api – ի հաճախ օգտագործվող ինտերֆեյսների և դրանց մեթոդների մասին։

**DriverManager class**

JDBC – ի կոմպոնենտ է և աշխատում է user – ի և դրայվերների միջև։ DM – ի միջոցով հայտարարվում է (ռեգիստր է լինում դրայվերը) դրայվերը, ինչպես նաև կապ է հաստատվում տվյալների բազայի հետ։ Եթե դրայվերը ռեգիստր չի լինում ապա կունենանք SQLException:

**Connection** **interface**

(DriverManager.getConnection()) Connection – ը դա սեանս է java application – ի և տվյալների բազայի միջև, որը օգնում է կապ հաստատել տվյալների բազայի հետ։ Connection – ը տրմադրում է Statement, PreparedStatemen, DatabaseMetadata և այլ մեթոդներ։

**Statement interface**

(connection.createStatement()) Statement – ը օգտագործվում է querie – ների իրականացման համար։

public ResultSet executeQuery(String sql), ruseltset – ը null չի կարող լինել այն կարող է լինել դատարկ։

public int executeUpdate(String sql) // վերադարձնում է փոփոխված տողերի քանակը

public boolean execute (String sql) // եթե querie – ն կարող է վերադարձնել 1 – ից ավելի արդյունք, օր փոփոխված տողերի, եթե արդյունքը իր մեջ ResultSet չի պարունակում կվերադարձնի falses

**ResultSet interface**

(statement.getResultSet()) ResultSet օբյեկտը տրամադրում է select querie – ի վերադարձրած արդյունքը։ Այն կարելի է ներկայացնել աղյուսակի տեսքով որը ձևավորվում է querie – ի վերադարձրած արդյունքի հիման վրա։ Յուրաքանչյուր Statement – ի համար կարող ենք ունենալ միայն մեկ ResultSet: ResultSet – ի օբյեկտը support է անում կուրսոր, որը ցույց է տալի ընթացիկ տողը։ Default – ով resultset – ի կուրսորը կարող է շարժվել միայն առաջ, բայց ResultSet.TYPE\_SCROLL\_INSENSITIVE ResultSet.CONCUR\_UPDATABLE Յուրաքանչյուր statement – ի համար կարող է լինել միայն 1 Resultset:

**PreparedStatement interface**

(connection.preparedStatement(query)) Օգտագործվում է պարամետրավորված հարցումների համար։ Պարամետրը փոխանցվում է ? – ի միջոցով, որի արժեքը սահմանվում է PreparedStatement - ի seter մեթոդների միջոցով։ Ավելի արդյունավետ է արագության տեսանկյունից և ապահով SQL injection – ների տեսանկյունից։ Եթե query – ն դինամիկ է ձևավորվում և իր մեջ օգտագործում է դրսից ստացվող data ապա sql injection – ի ժամանակ այդ data – ն կարող է գա sql սկրիպտի տեսքով ու որի հետևանքով db – ի հետ անցանկալի բաներ տեղի ունենան։

Վերոհիշյալի աշխատանքը ցույց տալ օրինակով։

(app – ում գրանցել user, ցույց տալ user ի պրոֆիլը առանց permission – ի, admin – ով user – ին permission տալ, ավելացնել նոր գիրք, հետո գիրքը ջնջել)։

**ResultSetMetaData interface**

Metadata նշանակում է տվյալներ տվյալների մասին։ Օգտագործվում է table – ի մասին լրացուցիչ տվյալներ ստանալու համար։ Օր՝ սյուների ընդհանուր քանակ, անուն, տիպ և այլն։

**DatabaseMetaData interface**

Metadata նշանակում է տվյալներ տվյալների մասին։ Օգտագործվում է database – ի մասին լրացուցիչ տվյալներ ստանալու համար։ Օր՝ driver - ի, անուն, վերսիա և այլն։

Binary store retrieve

File store retrieve

**CallableStatement interface**

Տալիս է մեթոդներ sql պրոցեդուրաների կանչելու համար։

**Transactions**

Տրանզակցիաները դա գործողությունների հավաքածու է որոնք պետք է իրականացվեն բոլորը միասին, եթե որևէ գործողություն չի իրականացվում ապա մյուսները չեղարկվում են (այդ թվում նաև իրականացվածները)։ Default – ով connection – ը գտնվում է querie - ների ավտոմատ ֆիքսացիայի (commit) ռեժիմում։

Տրանզակցիայի ավտոմատ commit – ը միացնելու / անջատելու համար setAutoCommit(true / false)

Գործողությունների իրականացման համար connection.commit()

Չեղարկման համար connection.rollback(), օգտագործվում է հիմնականում catche բլոկներում

**Batch Processing**

Եթե կան իրար հաջորդող querie – ներ ապա դրանք առանձին առանձինի փոխարեն կարող ենք exequte անել բոլորը միասին։ Առանձինի դեպքում ամեն անգամ connection է հաստատվում և օր շատ insert – ների դեպքում անարդյունավետ է։ Դրա համար կարող ենք statement տիպի օբյեկտի վրա կանչել setButch(query) և վերջում statement.executeButche()։ Վերադարձված մասիվի երկարությունը butch – երի քնակն է։

**RowSet interface**

ResultSet – ի հետ աշխատելու համար գոյություն ունի RowSet ինտերֆեյսը որի իմպլեմենտացիաները ունեն տվյալների պահման այլ մեխանիզմներ։ Ունի 5 իմպլեմենտացիա (ժառանգ ինտերֆեյսներ)։ Հիմնական առավելությունն այն է որ կարող է պահել պասիվ կապ Db – ի հետ քեշավորման շհնորիվ։

RowsetProvider.newFactory(). մեթոդով

JdbcRowSet – ը wrap է անում ResultSet – ը ու ավելացնում է RowSet – ի մեթոդները կատարելագործված տարբերակն է։ Default – ով scrollable ու updateable է։

ChachedRowSet – ը քեշավորում է տվյալները որի շնորհիվ db – ի հետ պահում է պասիվ կապ։

WebRowSet – ը ի լրումն ChachedRowSet – ի հատկությունների կարող է նաև տվյալները գրել xml ֆայլի մեջ, որը հնարավորություն է տալի տվյալները փոխանցել նաև ինտերնետ պրոտոկոլներով։

JoinRowSet – ով կարող ենք rowSet օբյեկտները իրար միավորել

FilteredRowSet – ի միջոցով կարող ենք ֆիլտրել db – ից ստացված արդյունքը

RowSet – ունի նաև setCommand մեթոդ որի միջոցով կարող ենք querie – ներ execute անել։

/jdbc/Demo.jsp?example=rsMetadata

/jdbc/Demo.jsp?example=dbMetadata

/jdbc/Demo.jsp?example=transaction&failed=yes

/jdbc/Demo.jsp?example=batch