Firebird Conference 2005

JayBird: JCA/JDBC driver

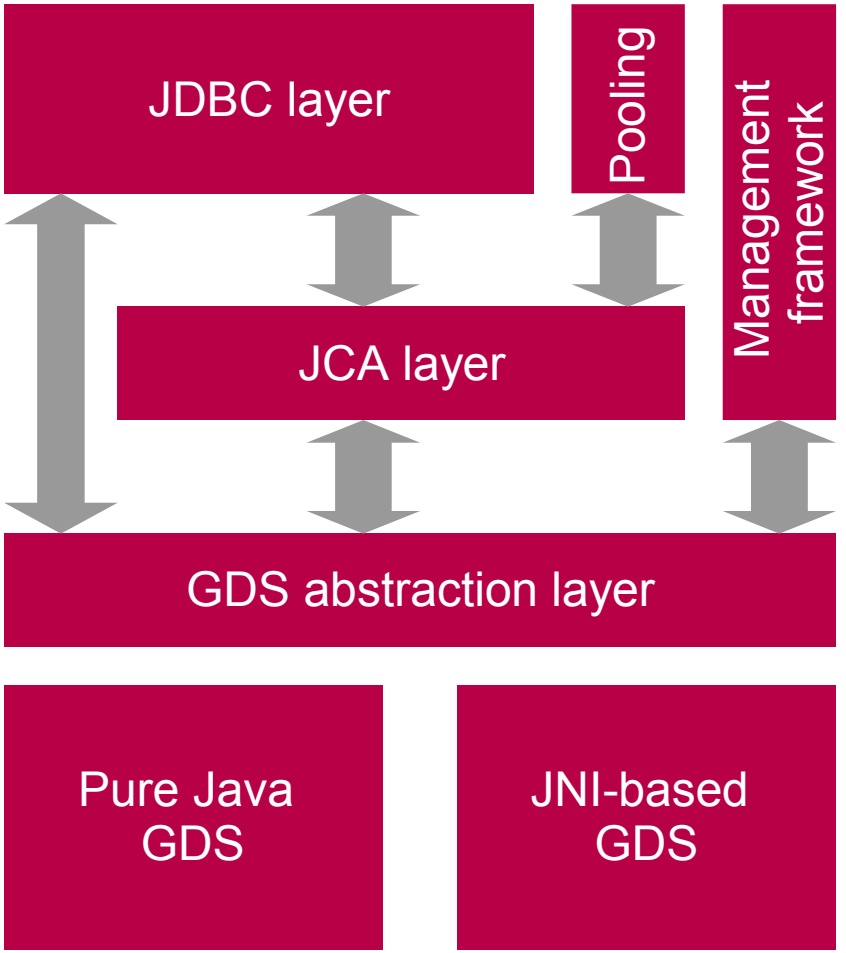
Roman Rokytskyy
Firebird Foundation

Agenda

- JayBird: introduction
  - History
  - Architecture
  - Connections
  - Statements
  - Result sets
  - Pooling
  - Embedded mode
  - Logging
  - Error handling
- JayBird: advanced
  - Transactions
  - Services API
  - Character sets
  - XA Transactions
  - Performance tracking

JayBird history and architecture

- History
  - Started by David Jencks in 2001
  - 1.0 Release on 29th April, 2003
  - 1.5 Release on 29th August 2004
  - 2.0 Release on 12th November 2005
- Architecture
  - GDS abstraction layer
  - JCA layer
  - JDBC layer
  - Pure Java GDS implementation
  - JNI-based GDS implementation
  - Pooling framework
  - Management framework
- Compliance
  - JDK 1.3.x, 1.4.x, 1.5.x
  - JDBC 3.0, JCA 1.0, JTA 1.0.1



Connections

- Driver manager
  - Register driver
    - org.firebirdsql.jdbc.FBDriver
  - JDBC URL
    - jdbc:firebirdsql:
    - jdbc:firebirdsql:wire:
    - jdbc:firebirdsql:local:
    - jdbc:firebirdsql:embedded:

```
import java.sql.*;

Class.forName("org.firebirdsql.jdbc.FBDriver");

Connection connection = DriverManager.getConnection(
"jdbc:firebirdsql:localhost/3050:c:/database/example.fdb
"
"SYSDBA", "masterkey");
```

```
jdbc:firebirdsql:[wire:]localhost/3050:c:/database/example.fdb
```

JDBC protocol

JDBC subprotocol, identifies driver to use

Driver type

RDBMS specific part, identifies the database to which driver must connect, in our case that is <host>/<port>:<path to database>

- Data Source
  - JNDI-based scheme
  - Instantiate in application
  - ConnectionPoolDataSource
  - XADataSource

```
import java.sql.*;
import org.firebirdsql.pool.*;

FBWrappingDataSource ds = new FBWrappingDataSource();

ds.setType("PURE_JAVA");

ds.setDatabase("localhost/3050:c:/database/example.fdb")
;
ds.setUserName("SYSDBA");
ds.setPassword("masterkey");

Connection connection = ds.getConnection();
```

Connection properties

- main
  - database
  - type
  - encoding and charSet
  - roleName
  - userName
  - password
- additional
  - sqlDialect
    - Default – 3
  - buffersNumber
  - nonStandardProperty
- Driver-specific
  - useStreamBlobs
    - Default – “false”
  - useStandardUdf
    - Default – “false”
  - socketBufferSize
    - Default – OS internal
  - defaultIsolation
    - Default – READ COMMITTED
  - tpbMapping
    - READ COMMITTED
      - isc_tpb_read_committed, isc_tpb_rec_version, isc_tpb_write, isc_tpb_wait
    - REPEATABLE READ
      - isc_tpb_concurrency, isc_tpb_write, isc_tpb_wait
    - SERIALIZABLE
      - isc_tpb_consistency, isc_tpb_write, isc_tpb_wait

Statements

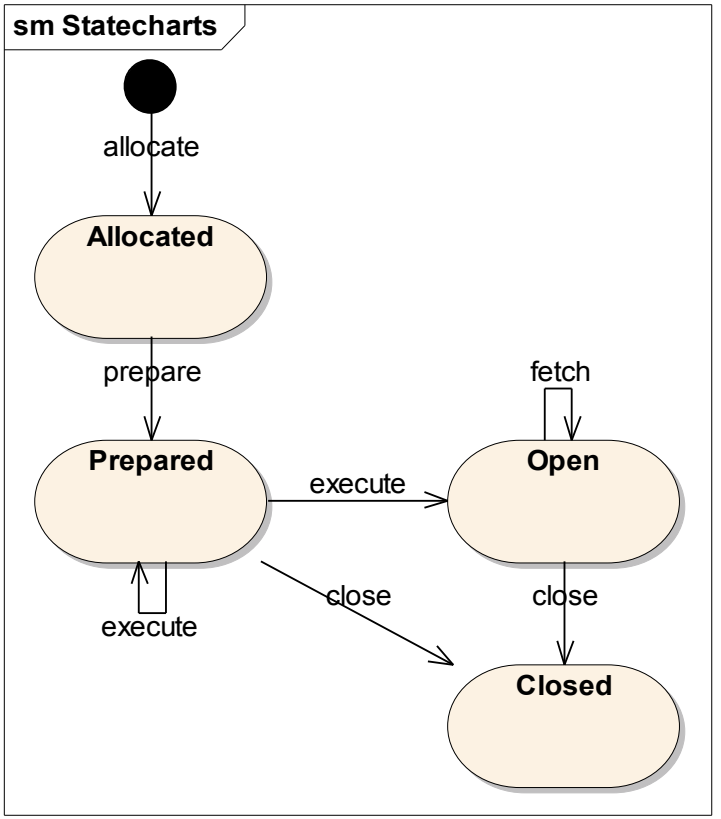
- java.sql.Statement
  - execute(String):boolean
  - executeQuery(String):ResultSet
  - executeUpdate(String):int
- java.sql.PreparedStatement
  - setXXX(...)
  - execute(), executeQuery() and executeUpdate()
- java.sql.CallableStatement
  - setXXX(...)
  - registerOutParameter(...)
  - execute()
  - getXXX(...)

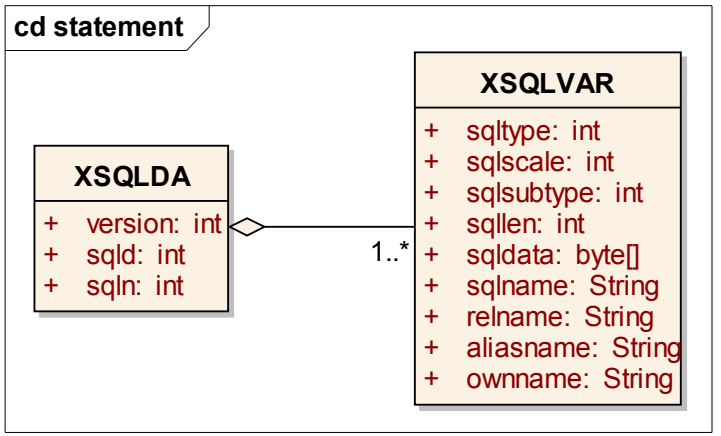
```
Statement stmt =
    connection.createStatement();
try {
    ResultSet rs = stmt.executeQuery(
        "SELECT firstName, lastName " +
        " FROM users" +
        " WHERE userId = 5");
    rs.next();
    String firstName = rs.getString(1);
    String lastName = rs.getString(2);
} finally {
    stmt.close();
}
```

```
procedure call ::= {[?=] call <params>}
params         ::= <param> [, <param> ...]
-------------------------------------------
CallableStatement stmt =
    connection.prepareCall(
        "{call factorial(?,?)}");
stmt.setInt(1, 2);
stmt.registerOutParameter(2, Types.INTEGER);
stmt.execute();
int result = stmt.getInt(2);
```

Statements on GDS level

- Firebird API calls
  - GDS.createIscStmtHandle
  - GDS.iscDsqlXXX
- Statement handle life-cycle
  - Allocate
  - Prepare
  - Describe
  - Execute
  - Fetch
  - Free
- Structures
  - XSQLDA structure
  - XSQLVAR structure





Statements in auto-commit mode

- Auto-commit mode
  - Commit transaction when the statement is ***completed***
- Life-cycle in auto-commit mode
  - Insert, Update, Delete or DDL statement is completed as soon as it finished executing
  - Select statement is completed as soon as its result set is closed
  - Callable statement is completed as soon as all associated result sets are closed

ResultSet is closed, when

All of the rows have been fetched

```
Statement stmt =
    connection.createStatement();
try {
  ResultSet rs = stmt.executeQuery(
      "SELECT firstName, lastName " +
      " FROM users");
  while(rs.next()) {
    String firstName = rs.getString(1);
    String lastName = rs.getString(2);
  }
  // we commit here
} finally {
    stmt.close();
    updateStmt.close();
}
}
    updateStmt.close();
}
  }
} finally {
    stmt.close();
    updateStmt.close();
}
```

The Statement is re-executed
Another Statement is executed on the same connection

Statement Extensions

- FirebirdStatement
  - getCurrentResultSet():ResultSet
  - hasOpenResultSet():boolean
  - getInsertedRowsCount():int
  - getUpdatedRowsCount():int
  - getDeletedRowsCount():int
  - getLastExecutionPlan():String
- FirebirdPreparedStatement
  - getExecutionPlan():String
  - getStatementType():int
  - TYPE_SELECT
  - TYPE_SELECT_FOR_UPDATE
  - TYPE_UPDATE
  - TYPE_INSERT
  - TYPE_DELETE
  - TYPE_DDL
  - TYPE_EXEC_PROCEDURE
  - TYPE_COMMIT
  - TYPE_ROLLBACK
  - TYPE_START_TRANS
  - TYPE_SET_GENERATOR

Statement Extensions (2)

- FirebirdCallableStatement
  - setSelectableProcedure(boolean)

```
procedure call ::= {[?=] call <params>}
params         ::= <param> [, <param> ...]
```

```
import java.sql.*;
import org.firebirdsql.jdbc.*;

...

CallableStatement stmt = connection.prepareCall(
    "{call factorial(?, ?, ?)}");

FirebirdCallableStatement fbStmt =
        (FirebirdCallableStatement)stmt;

fbStmt.setSelectableProcedure(true);

stmt.setInt(1, 5);
stmt.registerOutParameter(2, Types.INTEGER); // first OUT
stmt.registerOutParameter(3, Types.INTEGER); // second OUT

ResultSet rs = stmt.executeQuery();

while(rs.next()) {
    int firstCol = rs.getInt(1);              // first OUT
    int secondCol = rs.getInt(2);             // second OUT
    int anotherSecondCol = stmt.getInt(3);    // second OUT
}
```

Result sets

- Scrollable result sets
  - TYPE_FORWARD_ONLY
    - Only ResultSet.next()
  - TYPE_SCROLL_INSENSITIVE
    - Fully cached in memory
    - Absolute and relative positioning
    - Row count
  - TYPE_SCROLL_SENSITIVE
    - Not supported, downgraded to TYPE_SCROLL_INSENSITIVE
- Holdable result sets
  - HOLD_CURSORS_OVER_COMMIT
    - Only for scrollable result sets
  - CLOSE_CURSORS_AT_COMMIT
    - Default, all result sets
- Updatable result sets
  - Subset of single table
  - All columns from PK or RDB$DB_KEY
  - Not included columns allow NULL value
  - Not allowed
    - subqueries
    - DISTINCT predicate
    - HAVING clause
    - aggregate functions
    - joined tables
    - user defined functions
    - stored procedures
- FirebirdResultSet extension
  - getExecutionPlan():String

Pooling

- Connection pooling
  - Maintains pool of open connections
  - Supports connection “pinging” to detect broken connections and reconnect to the Firebird server behind the scenes
  - Closes idle connections after the specified timeout
  - Blocks for the specified time when no free connection can be found, clients are served on “first-in, first-out” basis
  - Provides runtime statistics about connection numbers
  - Remembers the place where Connection.close() was called and prints the stack trace if a “closed” connection is used
- Statement pooling
  - Saves time needed to prepare the statement
    - on AS3AP gives approx. 2x times boost
  - Maintains statement pool, however if application asks for more statements, request will always be satisfied
  - Remembers the place where PreparedStatement.close() was called and prints the stack trace if “closed” statement is used
  - Warning: Firebird supports approx. 20,000 open statements, keep the statement pool size low

JNI driver types

- LOCAL
  - Connects to Firebird server running on the localhost via IPC
  - On AS3AP tests gives approx. 30-40% performance boost
  - Requires synchronization on non-Windows platforms
  - JDBC URL
    - jdbc:firebirdsql:local:<path to database>
- NATIVE
  - Connects to Firebird over TCP/IP sockets
  - Is approx. 10% slower compared to pure Java mode
  - JDBC URL
    - jdbc:firebirdsql:native:<host>:<path to db>
- EMBEDDED
  - Accesses database file directly, no server required
  - Approx. 2x times faster than using server on the same host with pure Java connections
  - Exclusively locks database file on Windows
  - On Linux exclusive file lock does not exist, when multiple JVMs access same database, *it will be corrupted*
  - Requires synchronization on non-Windows platforms
- ORACLE
  - Same as NATIVE, but uses fyracle.dll to access Oracle-mode Firebird
  - Will be extracted from the main package into separate plugin

Logging

- How to switch it on
  - Log4J must be in CLASSPATH
  - -DFBLog4j=true
  - log4j.properties correctly configured

```
log4j.appender.stdout=\
    org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=\
    org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=\
    [%c{1},%p] %m%n

log4j.rootCategory=DEBUG, stdout

log4j.category.org.firebirdsql=DEBUG, stdout
```

- Log levels
  - DEBUG
    - Full wire protocol dump – file log grows very fast!
    - JCA processing
  - INFO
    - loading JNI libraries
    - when no connection cannot be obtained from the pool
  - WARN
    - serious issues happening in wire protocol handler,
    - unknown transactions on JCA level
    - pool experiences errors, etc.
  - ERROR
    - the specified host cannot be resolved when attaching to the database
    - driver cannot be registered
    - pooled statement is returned to pool, but the corresponding pool was not found

Error handling

- Error codes
- Typical usage