JDBC 4.0 – Overview

• Themes
  – Ease of Development
  – Extend JDBC 3.0 API
  – Resolve JDBC 3.0 Specification Issues

• Target Delivery platforms
  – J2SE 6 and J2EE 6

• And a warning...
IMPORTANT DISCLOSURE

The following is a proposal only and may change
JDBC 4.0 Focus

- Ease of Development
- Expand and Develop the API
Ease of Development (EoD) Goals/Non-Goals

- Intended for people who want quick access to some rows
- Not intended to support complex object abstractions
  - Technologies like JDO and EJB 3.0 are better for this
  - Provide a solid, complete object persistence framework
Ease of Development (EoD)

• Common database operations are complex
  – Executing queries, processing results and mapping to objects
  – Updating and modifying data

• Use J2SE 5 EoD Features to make this easier
  – Annotations
  – Generics
Example: Querying the Old Way

```java
PreparedStatement select = con.prepareStatement("SELECT id, fname, lname, description FROM people");
HashMap people = new HashMap();
ResultSet result = select.executeQuery();

while(result.next()) {
    Person person = new Person();
    person.id = result.getInt(1);
    person.fname = result.getString(2);
    person.lname = result.getString(3);
    person.description = result.getString(4);
    people.add(person.id, person);
}
```
Example: Querying with JDBC 4

```java
interface MyQueries extends BaseQuery {
    @Query(SQL="SELECT id, fname, lname, description FROM people")
    DataSet<Person> getAllPeople();
}
...
QueryFactory queryFactory = QueryFactory.newInstance();
MyQueries myQueries =
    queryFactory.create(MyQueries.class);
myQueries.setConnection(con);

DataSet<Person> rows = myQueries.getAllPeople();
```
EoD Classes/Interfaces

- “Data Class”
- DataSet
- BaseQuery
- QueryFactory
Data Class

- A class that maps to a row in a result set
- Think of it as a “struct” for holding a row

```java
public class Person {
    public int id;
    public String fName;
    public String lName;
    public String description;
}
```

- Can also be defined using getters and setters
DataSet

- Generic class (e.g. DataSet<T>) where <T> is a data class
- A Collection of data objects
- Represents a result set
- Can map to a single table or multiple tables
Query interface

- Interface whose methods define a set of database statements
- Extends BaseQuery interface
- Each statement defined as a method on the interface
- Annotations on the method describe the statement to be executed and how results are mapped to the DataSet
QueryFactory

- Used to creates instances of a given Query interface
- QueryFactory can be provided by JDBC driver vendor or a generic implementation provided by the JDK can be used.
- Uses annotations defined on Query methods to correctly execute statements and map results
@Query Annotation

- Annotates a method on a Query interface
- Defines a statement that returns results

interface MyQueries extends BaseQuery {
    @Query(SQL="SELECT id, fname, lname, description FROM people")
    public DataSet<Person> getAllPeople();
}
Parameterized Queries

- @Query annotation can be parameterized
- Parameter names in annotation map directly to parameter names in method

interface MyQueries extends BaseQuery {
    @Query (SQL="SELECT fname, lname, description FROM people 
             WHERE fname={fname} AND lname={lname}")
    public DataSet<Person> getPerson(String fname, String lname);
}
interface MyQueries extends BaseQuery {
    ...
    @Update (SQL="UPDATE PEOPLE SET description=
    {description} WHERE lname={lname}")
    int setDescription(String lname, String description);
}
...
int rowsUpdated;
rowsUpdated = myQueries.setDescription("Cleese", 
    "A very silly man");
@Table annotation

```java
@Table(name="people")
class Person {
    public int id;
    public String fname;
    public String lname;
    public String description;
}
```

- Maps a data class to a specific table
- Allows for easier insert, update and delete of rows in the table
- Supported only for single-table query methods
interface myQueries extends BaseQuery {
    public DataSet<Person> create();
    ...
}

DataSet<Person> newrow =
    myQueries.create();

newrow.id = getPersonId();
newrow.lname = "Cleese";
newrow.fname = "John";
newrow.description = null;
newrow.insert();
Inserting with an Existing DataSet

```java
DataSet<Person> rows = myQueries.getPeople();

...;

Person person = new Person();
person.id = getPersonId();
person.lname = "Idle";
person.fname = "Eric";
person.description = "funny in an odd sort of way";
rows.insert(person);
```
In-Place Update

```java
DataSet<Person> people = myQueries.getPeople();

for ( Person person : people ) {
    if ( person.description==null ) {
        person.description="<none>";
        people.modify();
    }
}
```
Deleting

```java
DataSet<Person> people;
people = myQueries.getAllPeople();
for ( Person person : people ) {
    if ( person.lname.startsWith("?") ) {
        people.delete();
    }
}
```
Disconnected Operation

- Use “connected=false” parameter to the @Query annotation
- Acts as a CachedRowSet
- Modifications are synced with database when you call DataSet.sync()
Disconnected Example

@Query(SQL="SELECT id, fname, lname, description FROM people" connected=false)
public DataSet<Person> getAllPeople();
...
DataSet<Person> people = myQueries.getAllPeople();
for ( Person person : people ) {
    if ( person.lname.equals("Palin") ) {
        person.description="Very funny with Cleese";
        people.modify();
    }
    if ( person.lname.startsWith("?" ) ) {
        people.delete();
    }
}
people.sync();
JDBC EoD compared to EJB3 POJO Persistence

- Definite similarities
  - Both use annotations to map SQL data to objects
  - Both reduce amount of JDBC code to write

- JDBC EoD focused purely on making it easier to do JDBC programming
  - Many customers not ready for/interested in full POJO Persistence
  - “I just want to get some data out of the database”
ROWID Type

• Standardize SQL ROWID type in JDBC API
  – Permits applications to obtain a specific and unique address of a row in a DBMS
• Potential for significant performance boosts
  – Direct access to underlying record
  – High performance update, deletes within a transaction
SQL Exception Changes

• Adding support for J2SE 1.4 chained exceptions for both SQLException and SQLWarning

• Useful SQLException subclasses
  – provide a standard mapping to common SQLState class values and to other common error states
  – two main subclasses: SQLTransientException and SQLNonTransientException
  – transient exceptions mean you can retry statement (e.g. deadlocks, lost connection). Needed for HA databases.
Proposed New Exceptions

- **Non Transient (retry won't help)**
  - SQLNonTransientConnectionException
  - SQLDataException
  - SQLIntegrityConstraintViolationException
  - SQLInvalidAuthorizationException
  - SQLSyntaxErrorException

- **Transient (retry will probably help)**
  - SQLTransientConnectionException
  - SQLTransactionRolledBackException
  - SQLTimeoutException
BLOB Management

- Additional interfaces and standard implementations to assist in creating BLOBs
  - Created using streams
  - Created using byte/char arrays
- Add additional control to close BLOB instances for improved cleanup
SQL 2003 and the XML Datatype

- Introduce standard description of XML data that is stored in a DBMS into API
- Working in conjunction to other JSRs to ensure consistency across standards
  - JSR-225 – XQJ
  - XQuery API for the Java Platform
  - SQL:2003
  - SQL/XML
JDBC 4 - Other Features

- APIs for connection pooling
- `Connection.cancel()`
  - Explicit cancel of currently executing statement
- Wrapper Pattern
  - Pools often implement proxies, and sometimes you need to get to the “real” connection to invoke vendor-specific methods
  - Allows you to “unwrap” a connection
Conclusion

- JDBC 4.0 brings
  - very useful Ease-of-Development features
  - API updates meet the needs of advanced users.
- More details at JavaOne 2005!
  - http://java.sun.com/javaone
JDBC – Contacting us

• Email
  – jsr-221-feedback@jcp.org
  – jdbc-interest@java.sun.com

• Web
  – Java Developer Connection
  – Java.Net
  – http://java.sun.com/jdbc
Questions & Answers

David Van Couvering
david.vancouvering@sun.com

java.sun.com/jdbc