Relational/JDBC database support

JdbcDataContext

The Apache MetaModel JDBC module is built such that we use as much of the powerful SQL capabilities of JDBC databases as possible (see Query execution strategies). So as such the module does little, except translate JDBC into the MetaModel API, but on the other hand it provides a single interface which translates into any of the many SQL dialects that are out there.

Features of the connector includes:

- Full implementation of DataContext and UpdateableDataContext.
- Mapping of MetaModel's data types to the particular set of data types available for the database. For instance MetaModel's STRING will be automatically converted to 'VARCHAR', 'TEXT' etc. depending on the database.
- Automatic formatting and narrowing of values when interacting with the database.
- Contains specialization and optimization for many databases, including:
    - PostgreSQL
    - MySQL
    - Microsoft SQL Server
    - Oracle
    - IBM DB2
    - Apache Hive
    - H2
    - SQLite
    - Hsqldb
    - Apache Derby
- Support for JDBC batch operations when you use BatchUpdateScript.
- Support for reusable prepared statements (as Compiled queries in MetaModel).
- Works with single `Connections` or connection pools (`DataSource`)

Creating from plain old java code - JdbcDataContext

This is really simple - if you have a JDBC based `Connection` or a `DataSource`, simply pass it to new `JdbcDataContext(...)`.

Creating from properties - JdbcDataContextFactory

The JDBC connector can also be driven by properties via MetaModel's DataContextFactory SPI:

```
final DataContextPropertiesImpl properties = new DataContextPropertiesImpl();
properties.put("type", "jdbc");
properties.put("url", "jdbc:postgresql://localhost:5432/mydb");
DataContext dataContext = DataContextFactoryRegistryImpl.getDefaultInstance().createDataContext
(properties);
```

The following table outlines all the relevant properties for this style of instantiation:

Property	Example value	Required	Description
`type`	`jdbc`	✅	Must be set to 'jdbc' or else another type of DataContext will be constructed.
`url`	`jdbc:mysql://localhost/sakila`	✅	The JDBC URL to use.
`driver-class`	`com.mysql.jdbc.Driver`		The JDBC driver class name.
`username`	`johndoe`		The username to use when connecting.
`password`	`qwerty`		The password to use when connecting.
`table-types`	`TABLE,VIEW`		A comma-separated list of table types to include in JDBC metadata discovery.
`catalog`	`cat01`		The JDBC 'catalog' value

Additional behaviour configuration

The module contains a few features that are specific to the JDBC interactions. These are all configured as Java **system properties**.

System property	Default value	Description
`metamodel.jdbc.convert.lobs`	`"false"`	Whether or not to automatically convert CLOBs into `String` and BLOBs into `byte[]`.

`metamodel.jdbc.batch.updates`	`DatabaseMetaData.supportsBatchUpdates()`	Whether or not using the JDBC batch API should be allowed. Set to "true" or "false" to override the default dynamic behavior.
`metamodel.jdbc.compiledquery.pool.max.size`	`-1`	The maximum number of open prepared statements to keep in a pool for the compiled queries.
`metamodel.jdbc.compiledquery.pool.idle.timeout`	`500`	The minimum idle timeout before a prepared statement is marked as evictable.
`metamodel.jdbc.compiledquery.pool.eviction.period.millis`	`1000`	The time between eviction runs in the compiled queries pool.