© Copyright 2008, Matthew Wakeling

InterMine

An open source data warehousing tool, ready for prime time

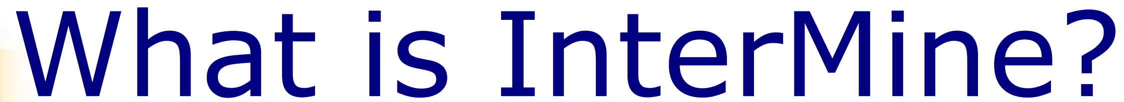
Matthew Wakeling

FlyMine Group, Department of Genetics

University of Cambridge

matthew@flymine.org

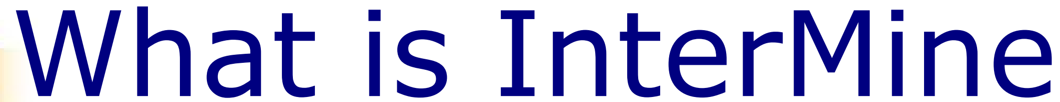
© Copyright 2008, Matthew Wakeling



What is InterMine?

- InterMine is a data warehousing system written for the FlyMine project.
  - Written over six years with five full-time developers in the University of Cambridge.
  - Open Source (LGPL).
  - Designed to import data from many sources for optimised read access.
  - Includes a full O/R mapping tool and integration system.
  - Web application is designed for non-programmers to access the database.

© Copyright 2008, Matthew Wakeling



What is InterMine

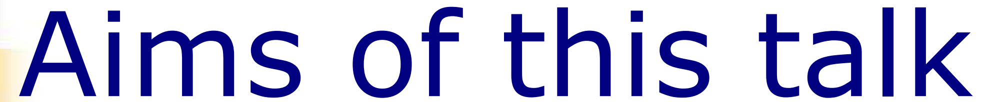
- Used by many projects:
  - FlyMine – www.flymine.org
    - 30+ data sources, 39 million objects
  - modENCODE – www.modencode.org
    - C. elegans/D. melanogaster high throughput, $57M.
  - BOKU & IMP – Vienna
  - MitoMiner – mitochondria
  - MilkMine – milk proteins
  - DeinoMine, Japan
  - Genes to Cognition, Sanger Centre
  - Yeast, Rat, Zebrafish model organism databases

© Copyright 2008, Matthew Wakeling

Overview

- InterMine's project architecture
- The "objectstore" project
  - Setting up a database
  - Reading and writing
- The "integrate" project
  - Importing data
- The web application
  - Creating a user interface to the database

© Copyright 2008, Matthew Wakeling



Aims of this talk

- Build a working InterMine database.
  - Define an object model.
  - Import and merge data from two sources.
  - Set up and demonstrate a web application.
  - Query the data and get results.
- Explain some of the inner workings and architecture of InterMine.
- Give a tour of FlyMine to show the features that are available.

© Copyright 2008, Matthew Wakeling

Aims of this talk

InterMine v 1.0
Object integration and warehousing software

Home | Templates | Lists | QueryBuilder | Data | MyMine | Log in

Search Identifiers for Go

InterMine > QueryBuilder

QueryBuilder

Advanced users can use a flexible query interface to construct their own data mining queries. The QueryBuilder lets you view the data model, apply constraints and select output. You can also export queries to share them with others.

Browse data model
Import query from XML
Login to view saved queries

Select a Data Type to Begin a Query

Click on a class name for a description or double click on a class name to create a new query starting at that class

----
Department
Employee
HasName
InterMineObject

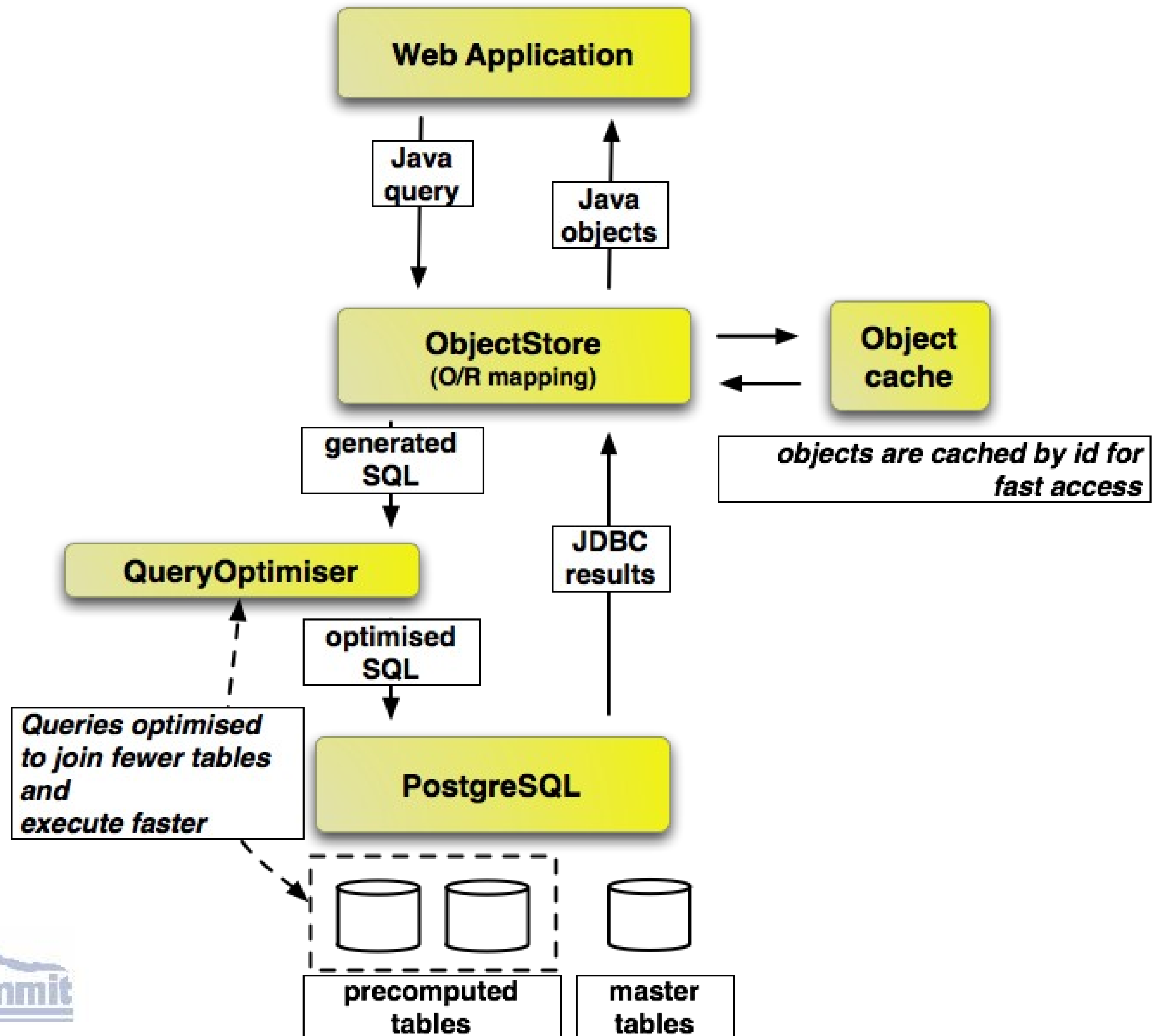
Select

Query History

Queries that you have run during this session. Log in to save your queries permanently.

© Copyright 2008, Matthew Wakeling

Overall architecture



© Copyright 2008, Matthew Wakeling

InterMine's project architecture

- InterMine uses an ant-driven system of separate projects covering different areas of functionality.
  - To create a new database, the easiest way is to create a new InterMine project and make use of the dependency system to pull in all the required code and ant rules for you.
  - This talk will demonstrate how to do this, and give a tour of the existing projects and their functionality.

© Copyright 2008, Matthew Wakeling

InterMine's project architecture

FlyMine

Your project

Web App

Path Query

Integration

ObjectStore

SQL

© Copyright 2008, Matthew Wakeling

The ObjectStore Project

- The InterMine ObjectStore is a general-purpose object-oriented database.
  - Backed by an SQL database (PostgreSQL).
  - Performance optimised for reads and writes.
  - Advanced query interface.
  - Differs from Hibernate:
    - Designed to take a object model and create an optimal SQL database, rather than mapping an object model onto a pre-set SQL database.
    - Designed for bulk data and complex queries rather than online transaction processing.

© Copyright 2008, Matthew Wakeling

Creating a project

- Make a directory for your project.
  - Now we create a sub-project for the db.
  - A project is an ant project, so we need a directory “dbmodel” with a build.xml.

```
<project name="tutorial" default="default" basedir=".">
  <description>For CSS2008</description>
  <import file=".../imbuild/objectstore.xml"/>
</project>
```

  - Also need a project.properties file.

```
compile.dependencies = intermine/integrate/main
objectstore.name = os.tutorial
model.name = tutorialmodel
intermine.properties.file = csstutorial.properties
default.intermine.properties.file =
    ../default.intermine.integrate.properties
```

© Copyright 2008, Matthew Wakeling

Defining a model

- An object model is defined in XML.

```
<model name="tutorialmodel" namespace="http://www.intermine.org/model/tutorialmodel#">
    <class name="tutorialmodel.HasName" is-interface="true">
       <attribute name="name" type="java.lang.String"/>
    </class>
    <class name="tutorialmodel.Employee" extends="tutorialmodel.HasName" is-interface="false">
       <attribute name="fullTime" type="boolean"/>
       <attribute name="age" type="int"/>
       <reference name="department" referenced-type="tutorialmodel.Department" reverse-reference="employees"/>
    </class>
    <class name="tutorialmodel.Department" extends="tutorialmodel.HasName" is-interface="false">
       <collection name="employees" referenced-type="tutorialmodel.Employee" reverse-reference="department"/>
       <reference name="company" referenced-type="tutorialmodel.Company" reverse-reference="departments"/>
    </class>
    <class name="tutorialmodel.Company" extends="tutorialmodel.HasName" is-interface="false">
       <collection name="departments" referenced-type="tutorialmodel.Department" reverse-reference="company"/>
    </class>
</model>
```

- Attributes can be of types:
  - boolean, short, int, long, float, double, Boolean, Short, Integer, Long, Float, Double, String, Date, BigDecimal.

© Copyright 2008, Matthew Wakeling

- A database is defined using properties.

```
db.tutorial.datasource.class=org.postgresql.jdbc3.Jdbc3PoolingDataSource
db.tutorial.datasource.dataSourceName=db.common-tgt-items
db.tutorial.datasource.maxConnections=10
db.tutorial.driver=org.postgresql.Driver
db.tutorial.platform=PostgreSQL
db.tutorial.datasource.serverName=server.flymine.org
db.tutorial.datasource.databaseName=tutorial
db.tutorial.datasource.user=fred
db.tutorial.datasource.password=whatever
```

- An ObjectStore is defined from the db.

```
os.tutorial.class=org.intermine.objectstore.intermine.ObjectStoreInterMineImpl
os.tutorial.db=db.tutorial
os.tutorial.noNotXml=false
os.tutorial.model=tutorialmodel
os.tutorial.minBagTableSize=10000
osw.tutorial.class=org.intermine.objectstore.intermine.ObjectStoreWriterInterMineImpl
osw.tutorial.os=os.tutorial
```

   - Put it in a file named ~/.intermine/csstutorial.properties

© Copyright 2008, Matthew Wakeling

Creating a project

- Also need to create some empty files and directories:

```
$ tree
.
|-- dbmodel
|    |-- build.xml
|    |-- lib
|    |-- project.properties
|    |-- resources
|    |    `-- tutorialmodel_keyDefs.properties
|    |-- src
|    `-- tutorialmodel_model.xml
`-- default.intermine.integrate.properties
```

- Create a Postgres database called “tutorial” and run “ant build-db”.

Colorado Software Summit

© Copyright 2008, Matthew Wakeling

Database Mapping

- Each class and interface in the model is mapped onto a table in the database.
  - The table contains a column for each field in the class, plus an “OBJECT” column containing the serialised object and a “class” column.
  - Each object is stored in all the tables associated with classes and interfaces that the object extends.
    - Multiple copies makes querying easier and faster, but storing slower.

© Copyright 2008, Matthew Wakeling

Database Mapping

- Several properties change the layout:
  - Truncated tables merges several tables into one – all classes extending a given class are combined in one table.
  - Missing tables are useful to reduce the storage overhead if it would not be queried.
  - Missing NotXML removes the OBJECT column from all tables except the root table. This means each query needs to go to the database twice – once to run the query, and again to fetch the objects.

© Copyright 2008, Matthew Wakeling

Database Mapping

```
test=# \d employee
             Table "public.employee"
           Column            |  Type   | Modifiers
-----------------------------+---------+-----------
 object                      | text    |
 fulltime                    | boolean | not null
 id                          | integer | not null
 name                        | text    |
 age                         | integer | not null
 departmentid                | integer |
 class                       | text    |
Indexes:
    "employee_pkey" UNIQUE, btree (id)
```

© Copyright 2008, Matthew Wakeling

Accessing the database

- Read access to the database is provided through the ObjectStore interface.

```
import org.intermine.objectstore.*;
ObjectStore os = ObjectStoreFactory.getObjectStore(“os.tutorial”);
```

  - “os.tutorial” was set up using properties earlier.

- Write access is provided by an ObjectStoreWriter – a sub-interface of ObjectStore.

```
ObjectStoreWriter osw =
    ObjectStoreWriterFactory.getObjectStoreWriter(“osw.tutorial”);
```

© Copyright 2008, Matthew Wakeling

Accessing the database

- To write an object to the database, use the ObjectStoreWriter.store() method:

```
import tutorialmodel.*;
ObjectStoreWriter osw =
    ObjectStoreWriterFactory.getObjectStoreWriter("osw.tutorial");
Employee e = new Employee();
e.setName("Fred");
e.setAge(5);
e.setFullTime(true);
System.out.println("ID before storing: " + e.getId());   ← null
osw.store(e);
System.out.println("ID after storing: " + e.getId());    ← 872341
```

- To delete an object:

```
osw.delete(e);
```

  - The object must be of the right class and have the ID filled in to delete properly.

© Copyright 2008, Matthew Wakeling

Accessing the database

- Many to many collections are stored using an indirection table.
  - When storing the object, many to many collections are updated.
  - You can independently update collections:

```
osw.addToCollection(e.getId(), Employee.class, "collection",
    other.getId());
```

- One to many collections are managed by the reverse reference.
  - To update them, store the object in the collection with a new reference back.

© Copyright 2008, Matthew Wakeling

Accessing the database

- Transactions are supported:

```
try {
     osw.beginTransaction();
     osw.store(e);
     System.out.println("In transaction: " + osw.isInTransaction());
     osw.commitTransaction();
} finally {
     osw.abortTransaction();
}
```

- There is also an asynchronous commit method:

```
osw.batchCommitTransaction();
```

  - Forces a transaction commit and re-begin, but does not wait for the database.

© Copyright 2008, Matthew Wakeling

Accessing the database

- Performance-wise, transactions are good.
  - Each write performed outside a transaction, and each transaction commit needs to wait for a round-trip to the database.
  - If you write lots of objects in one transaction, the software will group the writes together and send them to the database later, in the background.
    - For Postgres, we send the changes using a binary representation, to minimise conversion overheads.

© Copyright 2008, Matthew Wakeling

Accessing the database

- The ObjectStoreWriter uses a single connection to the database, in order to provide transaction support.
  - The ObjectStore uses a pool of connections, for parallel access.
  - Each ObjectStoreWriter is attached to an ObjectStore – accessed by getObjectStore().
- To fetch an object from the database:

```
Employee e = os.getObjectById(872341, Employee.class);
```

  - Results are cached.

© Copyright 2008, Matthew Wakeling

Data Storage

- InterMine is very flexible in storing data.
  - All stored objects* inherit InterMineObject, which has the ID field.
  - Will store any subclass of InterMineObject, whether it is in the model or not!
  - The model dictates what you can search for, not what you can store.

```
osw.store(new Employee() {
        private int extraField;
        public void setExtraField(int extraField) {
             this.extraField = extraField; };
        public int getExtraField() { return extraField; };
    });
```

* Except simple objects.

© Copyright 2008, Matthew Wakeling

Querying the database

- So far we have just accessed objects by ID. InterMine contains a powerful query interface.
  - Similar to SQL, but object-oriented.
  - You can select fields:

```
SELECT e.name FROM Employee AS e
```

  - You can select whole objects:

```
SELECT e FROM Employee AS e
```

  - You can add constraints:

```
SELECT e FROM Employee AS e WHERE e.name = 'Fred'
```

© Copyright 2008, Matthew Wakeling

Querying the database

- Queries can be written as text (IQL) or programmatically created in Java.

```
SELECT e.name FROM Employee AS e WHERE e.age > 25
```

- is the same as:

```
Query q = new Query();
QueryClass qc = new QueryClass(Employee.class);
q.addFrom(qc);
q.addToSelect(new QueryField(qc, “name”));
q.setConstraint(new SimpleConstraint(
        new QueryField(qc, “age”),
        ConstraintOp.GREATER_THAN, 25));
q.setDistinct(false);
```

- For conciseness I will use the text version.

© Copyright 2008, Matthew Wakeling

Querying the database

- The FROM list can contain multiple classes – even multiple copies of the same class.
  - The tables are joined together, just like SQL. The results are then restricted by the WHERE clause.
  - There is no SQL-like “JOIN” keyword – the classes are just listed separated by commas.
  - Each class can be aliased using the “AS” keyword, just like SQL.

- There are several types of constraint.
  - SimpleConstraint:

```
WHERE e.age > 25
```

  - ContainsConstraint:

```
WHERE e.department CONTAINS d
```

  - A question mark can represent a parameter:

```
WHERE e = ?                  (an InterMineObject)
WHERE e IN ?          (a collection of objects)
WHERE e.name IN ?            (a collection of Strings)
```

  - Constraints can be combined with AND, OR, NOT, and brackets (ConstraintSet).

© Copyright 2008, Matthew Wakeling

Querying the database

- The SELECT list is a list of the columns you want in your results.
  - May have more than one column, like SQL.
  - You can put fields, whole objects, mathematical expressions, and aggregate functions (COUNT, MIN, MAX, *etc.*) on the SELECT list.
  - If you use aggregate functions, you can set the GROUP BY list too, just like SQL.
  - There is no HAVING though – that is merged with the WHERE clause.

© Copyright 2008, Matthew Wakeling

Querying the database

- The ORDER BY clause behaves exactly like SQL.
  - Putting a class on the ORDER BY clause is like putting its ID on the ORDER BY.
- The DISTINCT keyword also behaves exactly like SQL.
- Subqueries can be put on the FROM list, and all the columns are available in the main query.

© Copyright 2008, Matthew Wakeling

Querying the database

- Path expressions are extra columns that are added to the SELECT list, but do not change the number of rows in the results.

```
SELECT d, d.employees FROM Department AS d
```

  - Returns results with two columns, of types Department and Collection of Employee.
  - The main query only involves Department.
  - `d.employees` is a path expression – extra data added to the results afterwards, so it cannot be accessed elsewhere in the query.

© Copyright 2008, Matthew Wakeling

Querying the database

- However, a path expression has query elements in its own right:

```
SELECT d, d.employees(WHERE default.age > 25) FROM
Department AS d
```

  - Returns a table of Department against a Collection of Employees older than 25.
  - Returns all Departments even if there are no Employees.
  - A bit like an SQL “outer join”.

© Copyright 2008, Matthew Wakeling

Executing queries

- The ObjectStore provides two main methods for executing queries:
  - ObjectStore.execute(Query q);
    - Returns a List of rows, each of which is a List of columns.
  - ObjectStore.executeSingleton(Query q);
    - Returns a List of elements, each of which is the value of the only column of the Query.
  - The List is a Results object – lazily-loaded from the database as it is accessed.
    - Uses a separate thread to prefetch data.

© Copyright 2008, Matthew Wakeling

Executing queries

- The Results object provides a few extra methods beyond just a lazy List:
  - Results.getInfo() returns an object containing estimates for how many rows there are, and how long the database will take to produce them all.
  - Results.setNoOptimise() switches off the use of the QueryOptimiser.
  - Results.setNoPrefetch() switches off the background prefetch thread.
  - Results.setBatchSize() sets the fetch size.

© Copyright 2008, Matthew Wakeling

Executing queries

- Because the results are lazily fetched, the data may change half-way through.
  - The Results object throws ConcurrentModificationException.
  - If you want to modify or delete objects as you read them, then execute your query on the ObjectStore, and write the changes to an ObjectStoreWriter in a single transaction.
    - The changes are only visible to the ObjectStore once you commit the transaction.

© Copyright 2008, Matthew Wakeling

Executing queries

- A helpful program is provided – IqlShell.
  - Provides an interactive console connected to a database.
  - You can type in queries and have them executed, optimised, or explained.
  - You can also manipulate the Query Optimiser from the program.
  - It is found in intermine/scripts/iql_shell
  - Uses Java-Readline.

© Copyright 2008, Matthew Wakeling

Query Optimiser

- InterMine provides a generic SQL query optimiser.
  - Precomputes the results of some common queries, and places the results into a table.
  - This table can then be substituted into other queries to speed them up.
  - Requires a suitable set of precomputed queries to make a difference.
  - Can improve performance significantly for very complex queries.

© Copyright 2008, Matthew Wakeling

QueryOptimiser

- Allows the database to separate object model design from performance optimisation.
  - Precomputed tables will act as the performance-optimised model.
  - The queries on the designed model will be rewritten to use the performance-optimised tables.

© Copyright 2008, Matthew Wakeling

Data Integration

- InterMine provides an integration project containing code for loading data into the database.
  - Will import data in several common formats.
  - Additional converters can be plugged in.
  - Will merge data from multiple sources.
    - Identifies duplicated objects.
    - Reconstructs object graphs.
    - Resolves field value conflicts.

© Copyright 2008, Matthew Wakeling

Data Integration

- Each data source can provide:
  - Extensions to the object model.
  - Code for converters.
  - Primary keys.
- The pattern is to build the entire database from scratch each time new versions of the source data are released.
  - It is an Integrated Data Warehouse.

© Copyright 2008, Matthew Wakeling

Data Integration

- Integration is controlled by project.xml.

```
<project type="tutorial">
  <property name="target.model" value="tutorialmodel"/>
  <property name="common.os.prefix" value="common"/>
  <property name="intermine.properties.file"
         value="csstutorial.properties"/>
  <property name="default.intermine.properties.file"
         location="../default.intermine.integrate.properties"/>
  <sources>
    <source name="source1" type="intermine-items-xml-file">
      <property name="src.data.file" location="data/source1.xml"/>
    </source>
    <source name=”source2” type=”intermine-items-xml-file”>
      <property name=”src.data.file” location=”data/source2.xml”/>
    </source>
  </sources>
</project>
```

- Here we are specifying two sources importing files in InterMine data XML format.

© Copyright 2008, Matthew Wakeling

Data Integration

- Then, create an “integrate” sub-project with a build.xml file:

```
<project name="tutorial-int" default="default" basedir=".">
  <description>perform integration</description>
  <import file=".../imbuild/integrate.xml"/>
</project>
```

  - Also need a project.properties file:

```
compile.dependencies = intermine/integrate/main,\
                .../tutorial/dbmodel
intermine.properties.file = csstutorial.properties
default.intermine.properties.file =
    ../default.intermine.integrate.properties
extra.project.dependencies = .../tutorial/dbmodel
```

© Copyright 2008, Matthew Wakeling

Data Integration

- Need some more properties in ~/csstutorial.properties:

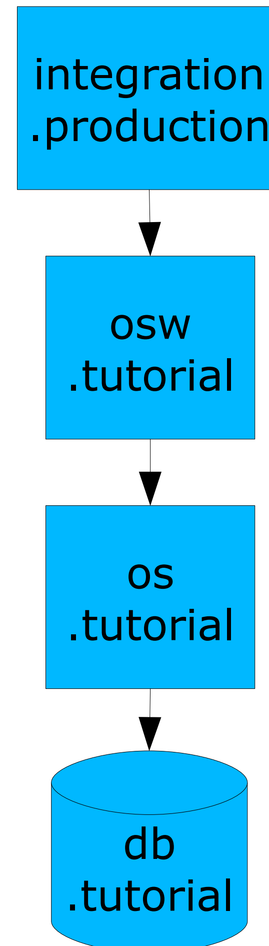
```
integration.production.class=org.intermine.dataloader.IntegrationWriterDataTrackingImpl
integration.production.osw=osw.tutorial
integration.production.datatrackerMaxSize=100000
integration.production.datatrackerCommitSize=50000
db.common-tgt-items.datasource.class=org.postgresql.jdbc3.Jdbc3PoolingDataSource
db.common-tgt-items.datasource.dataSourceName=db.common-tgt-items
db.common-tgt-items.datasource.maxConnections=10
db.common-tgt-items.driver=org.postgresql.Driver
db.common-tgt-items.platform=PostgreSQL
db.common-tgt-items.datasource.serverName=server.flymine.org
db.common-tgt-items.datasource.databaseName=common-tgt-items
db.common-tgt-items.datasource.user=fred
db.common-tgt-items.datasource.password=whatever
os.common-tgt-items-std.class=org.intermine.objectstore.intermine.ObjectStoreInterMineImpl
os.common-tgt-items-std.db=db.common-tgt-items
os.common-tgt-items-std.missingTables=InterMineObject
os.common-tgt-items-std.model=fulldata
os.common-tgt-items-std.minBagTableSize=10000
osw.common-tgt-items.class=org.intermine.objectstore.intermine.ObjectStoreWriterInterMineImpl
osw.common-tgt-items.os=os.common-tgt-items-std
os.common-tgt-items.class=org.intermine.objectstore.fastcollections.ObjectStoreFastCollectionsImpl
os.common-tgt-items.os=os.common-tgt-items-std
os.common-tgt-items.model=fulldata
os.common-translated-std.class=org.intermine.objectstore.translating.ObjectStoreTranslatingImpl
os.common-translated-std.model=tutorialmodel
os.common-translated-std.os=os.common-tgt-items
os.common-translated-std.translatorClass=org.intermine.dataconversion.ItemToObjectTranslator
os.common-translated.class=org.intermine.objectstore.fastcollections.ObjectStoreFastCollectionsForTranslatorImpl
os.common-translated.model=tutorialmodel
os.common-translated.os=os.common-translated-std
```

© Copyright 2008, Matthew Wakeling

Data Integration

```
integration.production.class=
   org.intermine.dataloader.IntegrationWriterDataTrackingImpl
integration.production.osw=osw.tutorial
integration.production.datatrackerMaxSize=100000
integration.production.datatrackerCommitSize=50000
```

- Creates an "Integration Writer".
  - Provides more complex store methods than the ObjectStoreWriter, which merges object graphs properly.
  - Includes a data tracker to track which data source data came from, so conflicts can be resolved.

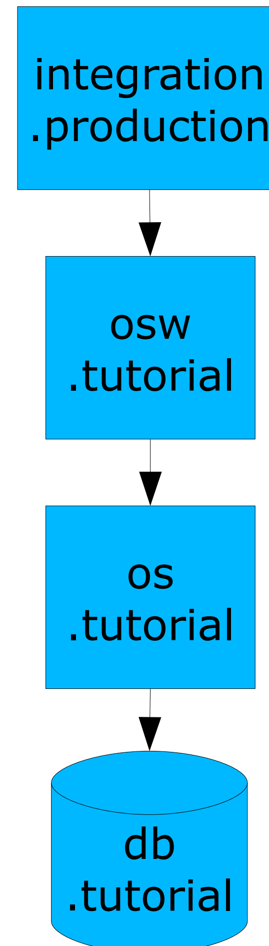


© Copyright 2008, Matthew Wakeling

Data Integration

```
db.common-tgt-items.datasource.serverName=server.flymine.org
db.common-tgt-items.datasource.databaseName=common-tgt-items
db.common-tgt-items.datasource.user=fred
db.common-tgt-items.datasource.password=whatever
```

- Defines an intermediate database.
  - InterMine uses an intermediate database to store converted data while importing.
  - Allows searches to be performed in order to reconstruct object graphs from flat data sources.
  - All taken care of by the system, easy to set up.

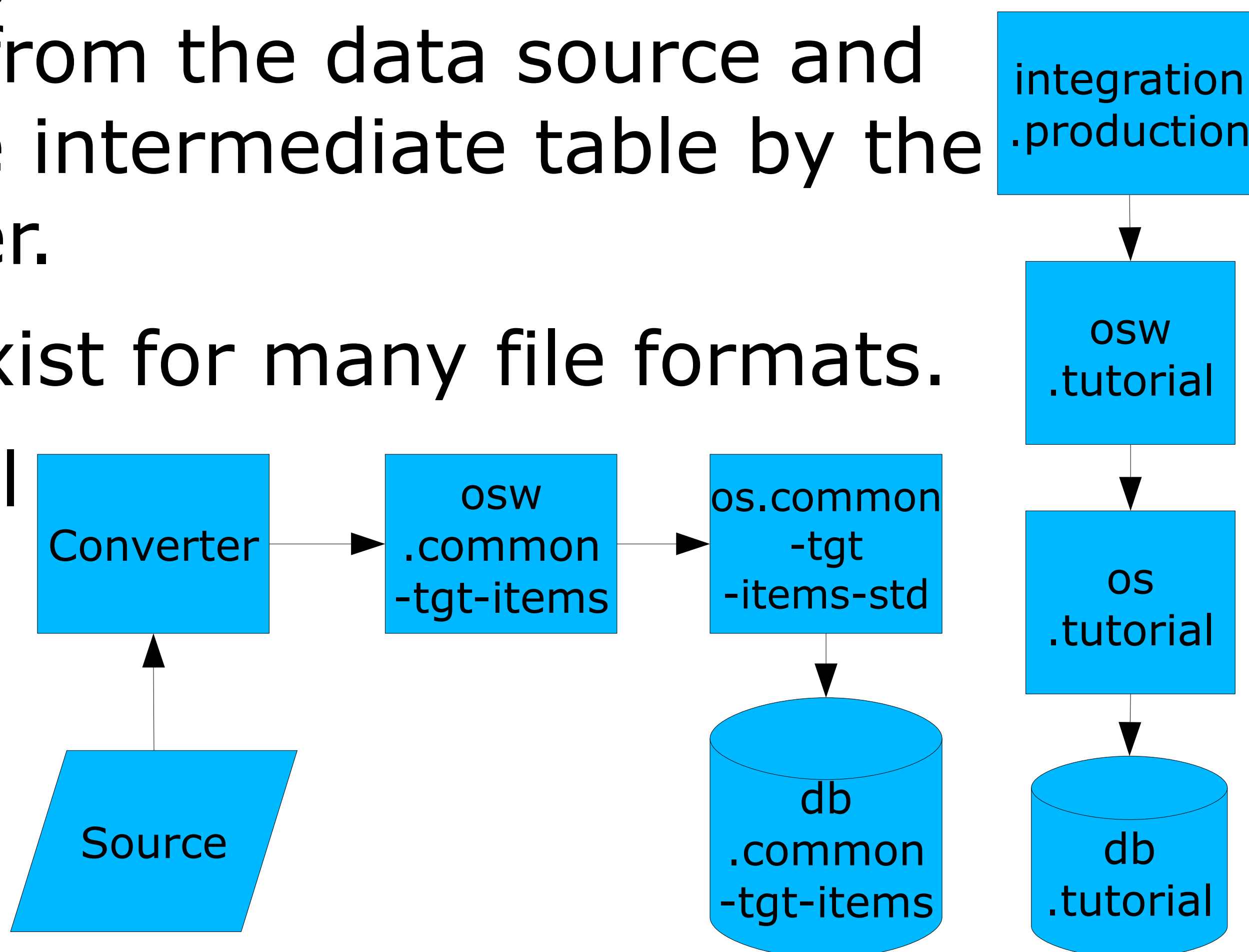


© Copyright 2008, Matthew Wakeling

Data Integration

```
os.common-tgt-items-std.class=org.intermine.objectstore.intermine.ObjectStoreInterMineImpl
os.common-tgt-items-std.db=db.common-tgt-items
os.common-tgt-items-std.missingTables=InterMineObject
os.common-tgt-items-std.model=fulldata
os.common-tgt-items-std.minBagTableSize=10000
osw.common-tgt-items.class=org.intermine.objectstore.intermine.ObjectStoreWriterInterMineImpl
osw.common-tgt-items.os=os.common-tgt-items-std
```

- Data is read from the data source and written to the intermediate table by the DataConverter.
- Converters exist for many file formats.
- Uses a special "flat mode" for performance.

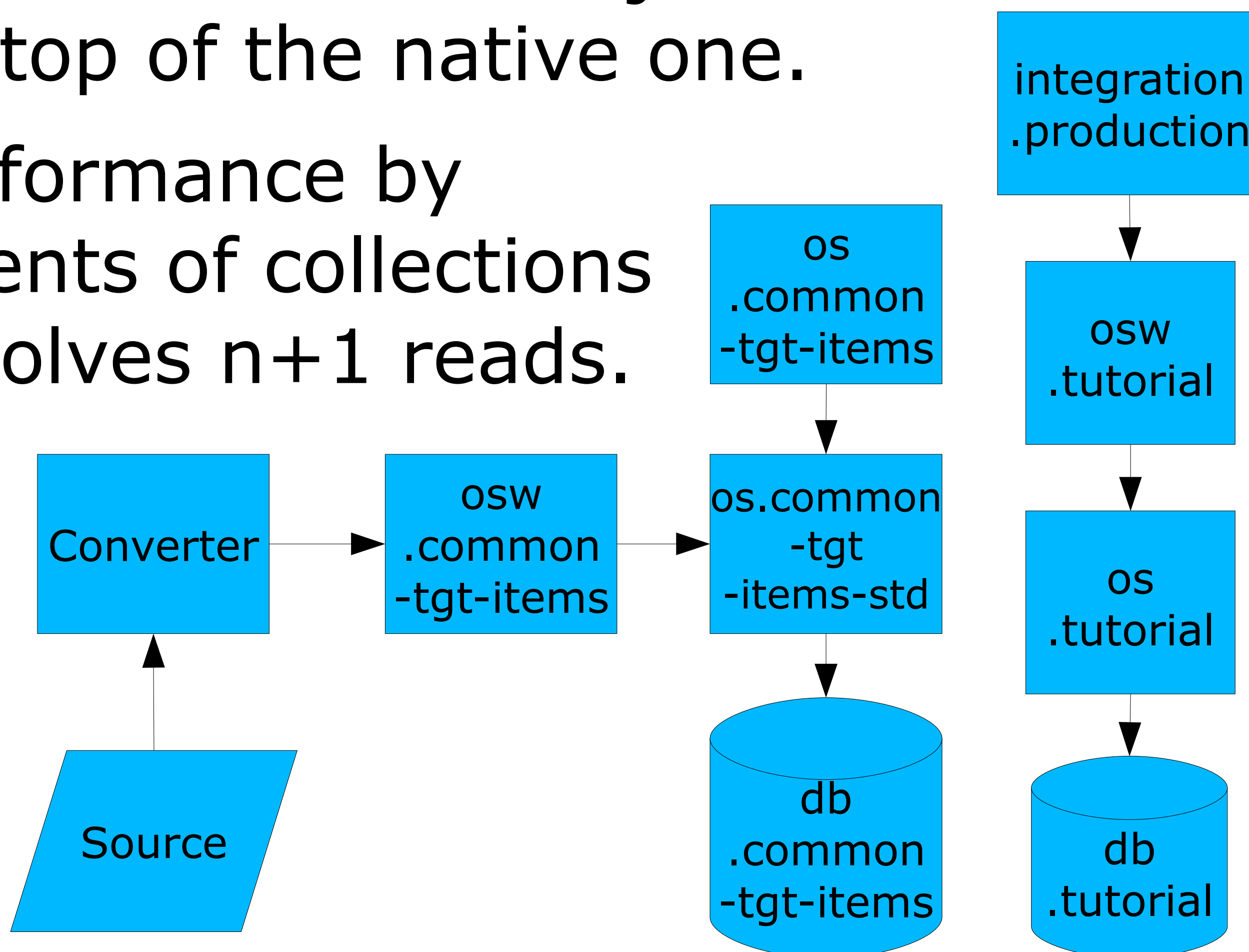


© Copyright 2008, Matthew Wakeling

Data Integration

```
os.common-tgt-
   items.class=org.intermine.objectstore.fastcollections.ObjectStoreFastCollectionsImpl
os.common-tgt-items.os=os.common-tgt-items-std
os.common-tgt-items.model=fulldata
```

- Another implementation of ObjectStore is layered on top of the native one.
- Improves performance by fetching contents of collections in advance. Solves n+1 reads.
- ObjectStores can be quite diverse.

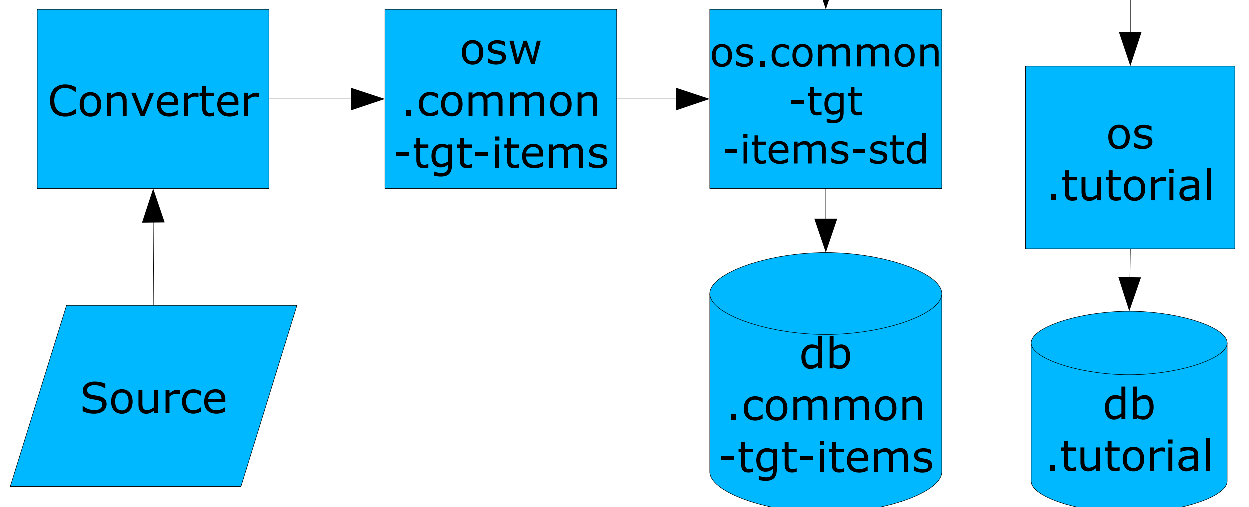


© Copyright 2008, Matthew Wakeling

Data Integration

```
os.common-translated-std.class
    =org.intermine.objectstore.translating.ObjectStoreTranslatingImpl
os.common-translated-std.model=tutorialmodel
os.common-translated-std.os=os.common-tgt-items
os.common-translated-std.translatorClass
    =org.intermine.dataconversion.ItemToObjectTranslator
os.common-translated.class=org.intermine.objectstore.fastcollections
    .ObjectStoreFastCollectionsForTranslatorImpl
os.common-translated.model=tutorialmodel
os.common-translated.os=os.common-translated-std
```

- Another layered ObjectStore translates database formats.
- Finally, another performance improvement layer.
- Layers add functionality.



© Copyright 2008, Matthew Wakeling

Data Integration

- The layers of ObjectStores and the IntegrationWriter take care of all translation and merging.
  - The DataLoader is a simple loop, reading objects from the translated ObjectStore and storing them in the IntegrationWriter.
  - For each object, the IntegrationWriter performs a query in the destination database to find an equivalent object, by primary key, and merges any existing objects with the object being stored.

© Copyright 2008, Matthew Wakeling

Data Integration

- Obviously running a query per object would be slow.
  - So the DataLoader actually takes a shortcut.
  - It keeps a summary of the data in the ObjectStore to avoid any obviously fruitless queries.
  - It uses another ObjectStore layer to “look ahead” at the objects that are coming, and runs queries for whole groups of objects.
  - Manages to load ~200,000 objects per minute.

© Copyright 2008, Matthew Wakeling

Data Integration

- We need to define the primary keys.
  - This is done in tutorial_keyDefs.properties.

```
Employee.key1=name
Department.key1=name
Company.key1=name
```

  - Multiple primary keys which can have multiple fields can be defined for each class.
- We also need to declare which primary keys are used by this data source.
  - This is done in source1_keys.properties

```
Department=key1
Employee=key1
```

© Copyright 2008, Matthew Wakeling

Data Integration

- Finally, we need some data:

```
<?xml version="1.0"?>
<items>
  <item id="1" class="Employee">
    <attribute name="name" value="Fred"/>
    <attribute name="age" value="27"/>
    <attribute name="fullTime" value="true"/>
    <reference name="department" ref_id="2"/>
  </item>
  <item id="2" class="Department">
    <attribute name="name" value="Marketing"/>
  </item>
</items>
```

```
<?xml version="1.0"?>
<items>
  <item id="1" class="Department">
    <attribute name="name" value="Marketing"/>
    <reference name="company" ref_id="2"/>
  </item>
  <item id="2" class="Company">
    <attribute name="name" value="ACME"/>
  </item>
</items>
```

- This is the "InterMine data XML" format.
- The item IDs are only used to refer inside the file – they are renumbered on loading.
- Also need a tutorialmodel_priorities.properties config file (can be empty).

© Copyright 2008, Matthew Wakeling

Data Integration

- Now run “ant -Dsource=all”.
  - This loads the data into the main database.
  - You should get a message in a file called “intermine.log” saying “Finished dataloading 2 objects” for each source.
  - Since the two Departments from the two sources match, the integration process will have merged them into one Department.
  - If any of the fields disagreed, the priority config would need to tell the integration which source to rely on for field values.

© Copyright 2008, Matthew Wakeling

Web Application

- InterMine provides a web application to provide access to the database.
  - It is designed as a data mining interface, allowing complex queries to be built and executed. We run it inside Apache Tomcat.
  - So create another sub-project called “webapp” with another build.xml:

```
<project name="tutorial-webapp" default="default" basedir=".">
  <description>Web application</description>
  <import file=".../imbuild/application.xml"/>
</project>
```

© Copyright 2008, Matthew Wakeling

Web Application

- It has another database for user accounts:

```
db.tutorial-userprofile.datasource.class=org.postgresql.jdbc3.Jdbc3PoolingDataSource
db.tutorial-userprofile.datasource.dataSourceName=db.tutorial-userprofile
db.tutorial-userprofile.datasource.maxConnections=30
db.tutorial-userprofile.driver=org.postgresql.Driver
db.tutorial-userprofile.platform=PostgreSQL
db.tutorial-userprofile.datasource.serverName=server.flymine.org
db.tutorial-userprofile.datasource.databaseName=tutorial-userprofile
db.tutorial-userprofile.datasource.user=fred
db.tutorial-userprofile.datasource.password=whatever
os.tutorial-userprofile.class=org.intermine.objectstore.intermine.ObjectStoreInterMineImpl
os.tutorial-userprofile.model=userprofile
os.tutorial-userprofile.db=db.tutorial-userprofile
os.tutorial-userprofile.noNotXml=true
os.tutorial-userprofile.minBagTableSize=100
os.tutorial-userprofile.missingTables=InterMineObject
osw.tutorial-userprofile.class
        =org.intermine.objectstore.intermine.ObjectStoreWriterInterMineImpl
osw.tutorial-userprofile.os=os.tutorial-userprofile
```

Web Application

- Also need a project.properties file:

```
compile.dependencies = intermine/webapp/main,\
                       .../tutorial/dbmodel,\
                       intermine/webtasks/main
deploy.dependencies = .../tutorial/dbmodel
objectstore.name = os.tutorial
userprofile.objectstorewriter.name = osw.tutorial-userprofile
userprofile.objectstore.name = os.tutorial-userprofile
userprofile.db.name = db.tutorial-userprofile
userprofile.model.name = userprofile
base.webapp.path = intermine/webapp/main/dist/intermine-webapp.war
intermine.properties.file = csstutorial.properties
default.intermine.properties.file = ../default.intermine.integrate.properties
```

- Defines which databases are to be used by the application.
- Also need classDescriptions.properties, class_keys.properties, webconfig-model.xml, and default-template-queries.xml

© Copyright 2008, Matthew Wakeling

Web Application

- The final config file determines where the application will be deployed:

```
www.serverlocation=fred@server:public_html/csstutorial
superuser.account=fred@flymine.org
project.sitePrefix=http://server/~fred/csstutorial
project.helpLocation=http://server/~fred/csstutorial/help
project.releaseVersion=1.0

# Web application
webapp.deploy.url=http://server:8080
webapp.baseurl=http://server:8080
webapp.path=csstutorial
webapp.manager=fred
webapp.password=whatever
webapp.viewByID.prefix=objectDetails.do?id=
webapp.logdir=/path/to/tomcat/logs
webapp.os.alias=os.tutorial
webapp.userprofile.os.alias=osw.tutorial-userprofile
project.standalone=true
```

© Copyright 2008, Matthew Wakeling

Web Application

- In the webapp directory, run “ant ; ant build-db-userprofile ; ant release-webapp”.
  - This compiles a WAR file and deploys it to Tomcat using the parameters in your properties file.
  - This will create a usable but simple web site that allows you to build queries on the database.
  - There is much more functionality that can be plugged in or otherwise configured.

© Copyright 2008, Matthew Wakeling

Web Application

- Webapp features:
  - Non-programmers can build complex queries for data mining.
  - Queries can be made into template queries for common query patterns.
  - Users can create "lists" of objects from results, and use them in new queries.
  - Quick search and autocompletion.
  - Also provides a RESTful web service.
  - Embedding of results in other web sites.

© Copyright 2008, Matthew Wakeling

Web Application

- The web site is mostly configurable on-line by a super-user.
  - Configuring does not require programming knowledge.
  - Create public template queries.
  - Create public lists.
  - Configure what information appears on report pages.
  - Configure what appears in each data category.

© Copyright 2008, Matthew Wakeling

FlyMine

- FlyMine is a fully-configured instance of InterMine.
  - It is beyond the scope of this presentation to show you how to add all these features, but I may as well show them off!
  - Widgets – a framework is provided to easily generate graphs and charts on report pages.
- Let's take a tour.

© Copyright 2008, Matthew Wakeling

Acknowledgments

The FlyMine team: Richard Smith, Matthew Wakeling, Xavier Watkins, Julie Sullivan, Jakub Kaluviak, Hilde Janssens, Rachel Lyne, Dan Tomlinson, and Gos Micklem

The InterMine system (www.intermine.org) is a generic object-oriented database and data integration system, open source and licensed under the LGPL.

FlyMine (www.flymine.org) is a biology-specific application of the InterMine system, also licensed under the LGPL.

FlyMine is funded by the Wellcome Trust.

FlyMine Group, Cambridge Systems Biology Centre, University of Cambridge, Tennis Court Road, Cambridge, CB2 1QR, UK
Tel: +44 1223 760262 Email: info@flymine.org

Colorado Software Summit

wellcome trust