DRUID TUTORIALS

Andrea Carboni, Mirko Caserta

Rev 2, 25-sep-2003

Contents

Chapter 1

Basic tutorial

Author Mirko Caserta

1.1 Introduction

Learn how to use the basic Druid's concepts. After this tutorial you will be able to:

- Add a new database
- Add tables and fields
- Add references to other tables
- Generate the db documentation
- Generate the sql-script to create the database
- Establish a JDBC connection to a database
- Create a simple E/R diagram

1.2 Druid at a glance

Don't let the first screen make a fool of you (figure 1.1). It looks like there's nothing to do; well, that is what it looks like, not what Druid can really do for you! For now, close the "Did you know" window.

Let's right-click the split-pane on the left and select Add Database (figure 1.2). Selecting this option we're effectively adding a new database to the project. We'll learn later that you can add more than just one db to the project.

At this point you should get to the Database Wizard dialog (figure 1.3) which allows you to choose between the three options:

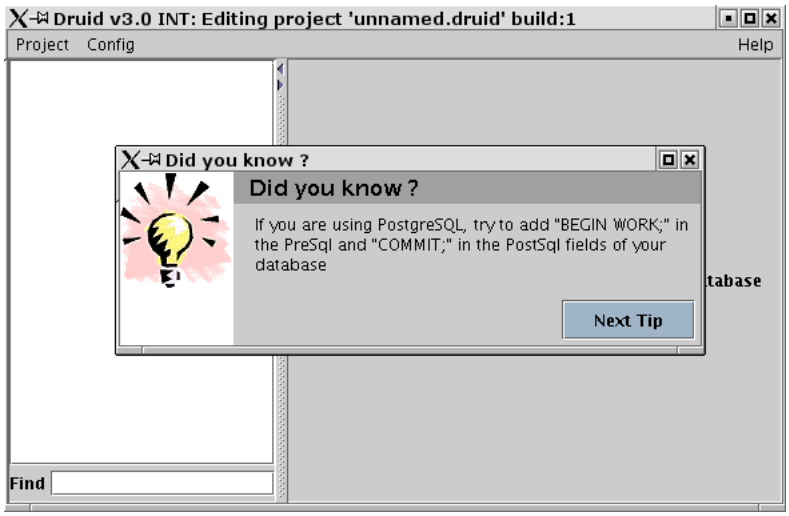

Figure 1.1: Druid at startup

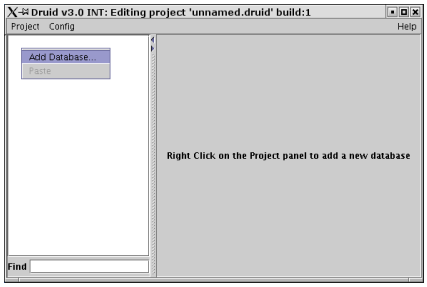

Figure 1.2: Adding a new database

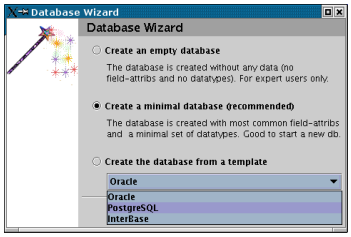

Figure 1.3: The database wizard

- Create an empty database
- Create a minimal database
- Create a database from a template

The first option should be considered for expert users only since the created db will really be empty, without data types and other nifty things that are no-doubt useful at the beginning stage.

The second option can be chosen with no worries and it’ll put at our disposal some basic data types defined by the SQL-92 standard.

The third option is definitely useful if we’re working with one of the RDBMS included in the list (InterBase, Oracle and PostgreSQL; hopefully more will be added soon) since it’ll automatically put at our disposal the appropriate data types supported by these applications. For our example we’re going to use the third option with the PostgreSQL variant since under Linux it’s a more often adopted solution.

At this point we should have in the left side split-pane a root node which corresponds to the database we’re operating on and in the split-pane on the right a handful of tabs that we’re going to examine with calm while we’ll get into the related issues. The data types we got, having chosen the PostgreSQL template from the wizard, are showed in figure 1.4.

1.3 An hypothetical database: the ”Fate Bene Fratelli” Hospital db

For our examples we’re going to use an hypothetical db which will hold some basic info about my favourite hospital (which really exists in Rome, Italy). Let’s double-click the db root node in the left side split-pane and rename the db as ”fatebenefratelli”. We’re going to use three tables, defined as follows:

```
CREATE TABLE doctors
(
   id        int4,
```

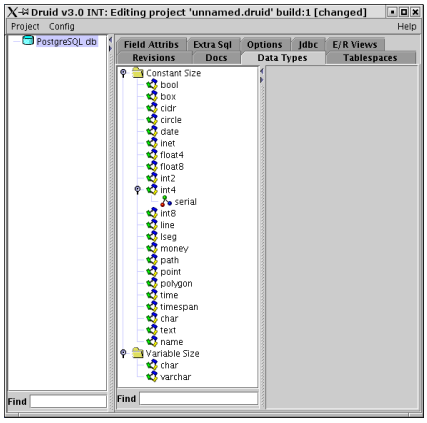

Figure 1.4: PostgreSQL datatypes

```
   name      varchar(32)   not null,
   surname   varchar(32)   not null,

   primary key(id),
   unique(name,surname)
);

CREATE INDEX doctorsNDX1 ON doctors(surname);

CREATE TABLE patients
(
   id         int4,
   name       varchar(32)     not null,
   surname    varchar(32)     not null,

   primary key(id),       unique(name,surname)
);

CREATE INDEX patientsNDX1 ON patients(surname);

CREATE TABLE prescriptions
(
   doctorid     int4            not null,
   patientid    int4            not null,
   medicine     varchar(64)     not null,
   quantity     varchar(32)     not null,
   when         timestamp       not null,

   primary key(doctorid,patientid,when),
   foreign key(doctorid) references doctors(id) on delete CASCADE,
```

```
    foreign key(patientid) references patients(id) on delete CASCADE
);
CREATE INDEX prescriptionsNDX1 ON prescriptions(patientid);
```

Those who are used to the SQL lingo are not going to have troubles understanding that the doctors and patients tables are identical and contain basic information about those persons. The prescriptions table is the *glue* between the other two tables and is supposed to keep the patients alive (though I would never enter an hospital which uses such a stupid and badly structured db!).

1.4 Defining available data types

Before we start structuring the database, we'll probably need to add/modify the available datatypes. This is possible by selecting the db root node and, in the split-pane on the right side, selecting Data Types. Basically, we can add data types of constant or variable dimensions.

A constant data type might be INTEGER or INT4 since the RDBMS is supposed to always use 4 bytes in order to store fields declared using this type. A variable type is instead a type which lets us specify the bytes length storage space. For instance, the CHAR data type, even being constant by its nature, allows us to specify the number of chars (and that really means bytes) that the field will be able to hold. The same applies to the VARCHAR datatype which, instead, tries to optimize the storage space used by a string; for instance, if we define a field of type VARCHAR(16), the RDBMS is probably going to store the string "foo" using only 4 bytes of which: 3 are used to store the string and 1 is used to store its length in bytes. By the way, all this talk about constant and variable types has been introduced only in order to let you understand what Druid intends for constant and variable data types.

Adding a constant data type is as simple as right-clicking the Constant Size node and selecting Add basic datatype. We should see a parent node appearing which we can rename by double-clicking on it.

Adding a variable data type is the same as adding a constant type. You should only (if you wish to specify the byte length) right-click on the newly added node and select Add alias. If we want, for instance, create a VARCHAR(128) data type, we should add an alias to the VARCHAR data type, rename it to, say, varchar128 and specify in the right side panel (it's called Var Alias) the value we wish to declare into the size text-box (in this case: 128).

1.5 How to create the structure of a db with Druid

Let's start adding tables. Right-click the db root node and select Add table. We'll see a child node appearing which we'll be going to rename with the table's right name by double-clicking it. At this point, in the right side split-pane, select the fields tab (it should be selected by default) and let's click the new button for each field we have to add. Then let's rename the fields by clicking on them in the name column. Now, let's specify the data type for each field by clicking

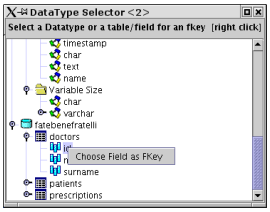

Figure 1.5: Choosing a foreign key for a field

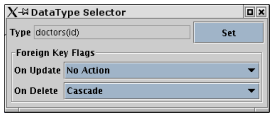

Figure 1.6: Specifing the foreign keys actions

into the appropriate grid cell in the type column. You should now see a pop-up window with all the available data types. In order to choose one, right-click it and select choose this datatype.

If the field in question is a FK to an id which is found in another table, you should scroll down the pop-up window until the node which represents the database, then open the node corresponding to the table which holds the target id, then Choose field as Fkey by right-clicking on the id in question (figure 1.5). This way we can tell Druid that the field in question is really a foreign key.

At this point you'll get a further pop-up window where you might specify the action to take for instance ON UPDATE and ON DELETE (figure 1.6). The most common option is to specify CASCADE for ON DELETE because, for instance, if we delete a patient, we might want not to keep track of his prescriptions (in a real hospital application this shouldn't happen though, since they'll want to keep track of the whole history a patient has had). In order to confirm the choice, you just need to close the pop-up window.

Now, let's get back to the tables definition: the meaning of the other columns of the grid is as follows:

PrKey the field is a primary key; you can even check this box on more than a field in the case of a multiple fields primary key, as in our prescriptions table example where we do have three PK fields: doctorid, patientid and when

Unique the field is of type UNIQUE, ie: there cannot be two tuples/rows in this table holding the same values in this field.

NotN NOT NULL, that is: you should necessarily specify a value for this field at the moment you're adding data into this table

MUnq Multiple Unique; allows you to group fields which should hold unique values inside the table

Def allows you to specify a default value which gets into the field in the case the INSERT omits an explicit input

Idx1 Creates an index for this field

You're allowed to modify the number and meaning of the fields' attributes by selecting the root db node and going to the Field Attribs tab. The use of PrKey, Unique, etc, is completely parametrized and, so, easily adaptable to the syntax of your RDBMS of choice. This allows the use of Druid even with RDBMS which are not fully SQL-92 compiant or have proprietary extensions and functionalities. Furthermore, these attributes can refer to the field or the table. Well, you can do almost everything!

1.6 Handling the documentation, ie: the lifetime of an application is directly proportional to the quantity and quality of the produced documentation

One of the most valuable things in a heavy weight project is the availability of its documentation. Rise your hands those of you who have never worked with a database whose structure was (to be fair and honest) cryptic? Uhmmmm, it looks like no hands are up. I can tell you that, in my personal experience in the IT world, one of the most important things I've learnt the hard way is that a correct and up-to-date documentation is an essential requisite for the life and the development of an application. So: do document, do document and then do document all over again. The time you'll spend during the documentation process will result invaluable once you have to put your hands on the project months or even weeks later. So: do document! Have I already said that? I just wanted to make sure the idea was clear enough ;-).

Druid allows you to handle the project documentation in an excellent way. For each node in the left side split-pane, including the single table fields (you can expand the node of a table by clicking on it) you're going to have a tab named Docs in the right side split-pane where you can add comments essential for the understanding of the structure.

Once you're done with the documentation work, you can automatically generate an XML or HTML output. The generated HTML files are in the classic javadoc style and extremely easy to navigate. Let's see how to setup HTML output generation.

- Select the database root node
- Into the right side split-pane select the Generation tab
- Now select the Modules tab
- Right-click the Docs folder
- Select Add module, then Browsable XHTML
- In the Output directory panel, specify the directory where the documentation will end up

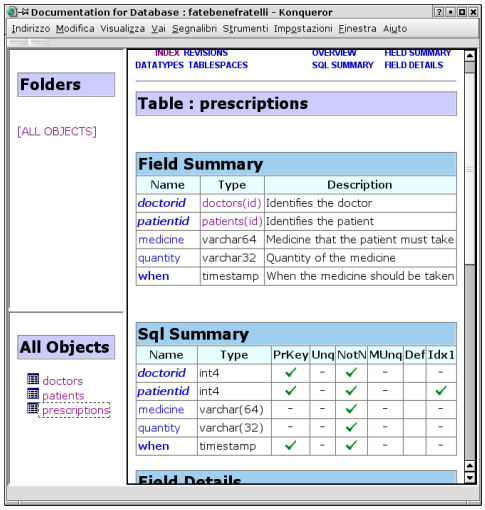

Figure 1.7: The generated docs

- Right-click the Browsable XHTML node in the Docs folder
- Select Generate

Now, point your favourite web browser on the index.html file which can be found in the directory we have previously specified and here is the index of the documentation (figure 1.7)!

The XHTML documentation generated by Druid reminds much of the *javadoc* style and is much precise, filled with details and well-structured. Just a side note before we go on: for particularly complex projects, Druid allows you to subdivide the tables inside folders. You can (well, you should!) add folders by right clicking the root db node and selecting Add folder. You can even drag'n'drop tables from one folder to the other in a transparent fashion.

1.7 Generating the SQL script for the db creation

One of the main features of Druid is its completely automated handling of links resolution and conflicts. If, for instance, something is not clear in the db structure, Druid is going to negate us the possibility to generate documentation or SQL scripts pointing us to the problem which has to be solved. Particularly funny (and maybe even useful) is the Statistics feature, which you can access with the usual right-click on the root db node.

To generate an SQL script for use by the RDBMS in order to create the database, the procedure to follow is similar to the one we've seen for generating the documentation:

- Select the root db node
- In the right side split-pane select the Generation tab
- Select the Modules tab now

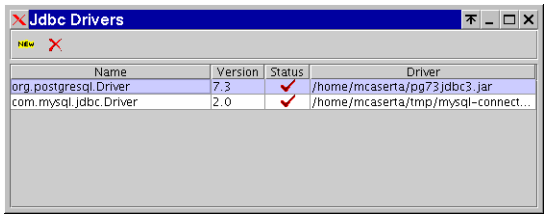

Figure 1.8: The installed JDBC drivers

- Right-click the SQL script folder
- Select Add module, then PostgreSQL
- Select the PostgreSQL node added above
- In the Output file panel, specify the file where the script will be output
- In the General tab you can choose the output order of the objects: sequential or optimized, moreover you can specify if you wish to include the comments (can be added using the Extra ▷ General ▷ Sql comment panel)
- Right-click the PostgreSQL node, then select Generate

Let's have a look at our file: we'll find our nicely formatted script.

1.8 Using the JDBC drivers to access the database

Through Druid you can directly access the databases you're working on using one or more JDBC drivers. In order to add a driver, let's go to the Config menu and select JDBC drivers. Let's click the new button and browse to the jar file which contains the driver we need. Here we can see how Druid is able to work at the same time with different JDBC drivers; in this case I've installed the PostgreSQL and MySQL drivers (figure 1.8).

Once you've completed this very simple operation it is possible to connect to a database by selecting the root db node, then the JDBC tab. In order to make the connection it is sufficient to specify the typical JDBC URL and a username plus password. Figure 1.9 ahows how Druid presents itself after having established a JDBC connection.

Once you've established the connection, you can handle the creation and modification of the tables directly from Druid. In order to create/recreate the structure of the database, right-click the root db node, then select Rebuild in jdbc DB. The same applies to the single tables. Be aware anyway that each rebuild via JDBC deletes all data in the database or table you're reconstructing.

Also in the JDBC tab, we have useful functionalities for importing existing entities from the db (if for instance we are working on a db we want to import the structure of), directly manipulate the tables' data just like in MS Access and to execute arbitrary SQL queries.

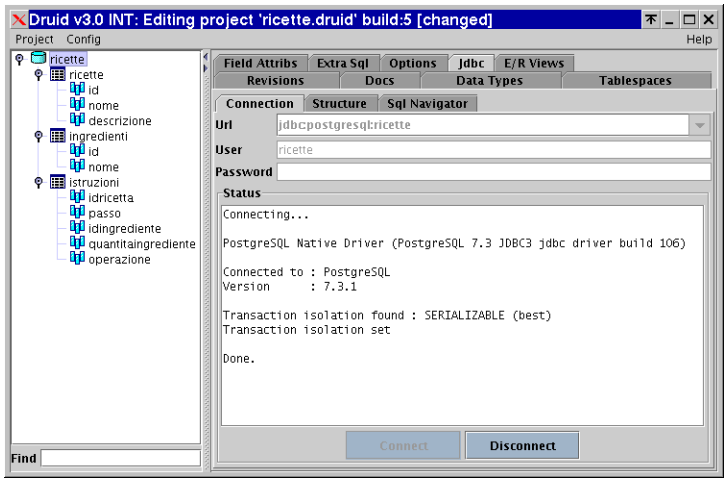

Figure 1.9: A JDBC session

1.9 Creating E/R diagrams

E/R stands for ”Entity/Relationship”. It is one of those things who can make people real happy, those just like me who do really love documentation. Creating one of these diagrams with Druid is extremely simple:

- Select the root db node
- Select the E/R Views tab
- Right-click into the left side split-pane (inside the E/R Views tab)
- Select Add E/R View
- Double-click to rename the diagram
- In the right side split-pane right-click and select Add entity
- Check one table, then click the OK button
- Repeat the Add entity for each table you wish to add

Druid will take care of automatically creating the links for dependencies and foreign keys. Once you’re done with the diagram, you can save it as a PNG image or, more simply, print it (the related options are accessible by right-clicking the name of the diagram). Figure 1.10 shows how an E/R diagram created with Druid presents itself.

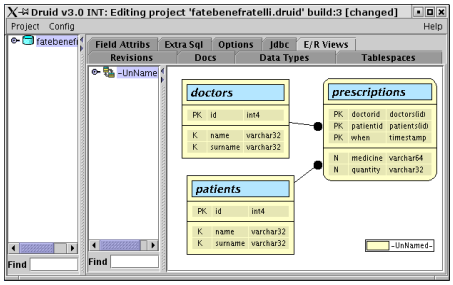

Figure 1.10: An E/R view of the database