Database Design Using JDeveloper

Susan Duncan
Oracle Corporation
United Kingdom

Keywords:

JDeveloper, Database, Versioning, SQL, Modeling

Introduction
JDeveloper provides extensive functionality for the database developer. You can create logical models using the UML class modeler and transform these models to offline physical database designs for a variety of databases. The offline physical database objects are stored as XML metadata that can be reported on using a SQL-like interface. You can apply standards in the form of templates to those database objects and share these between different database models. You can version the database objects and compare, merge and resolve conflicts between different versions of objects using a declarative UI. You can reverse engineer and forward engineer your database objects and compare differences between offline and live databases to produce CREATE, ALTER and REPLACE scripts that can be versioned or run directly against a live database. In this paper understand how to achieve all of this and more

Logical Modeling
JDeveloper is shipped with a UML class modeler that can be used to do logical database modeling. As a default each class is displayed with both attributes and operations shown in the model, but by using the class model diagram preferences (Tools -> Preferences -> Diagrams -> Class) you can edit the display of objects on the diagram to more closely represent a more traditional entity relationship diagram

- Display tab: only check Show Attributes
- Attributes tab: uncheck Sort Alphabetically and Show Visibility

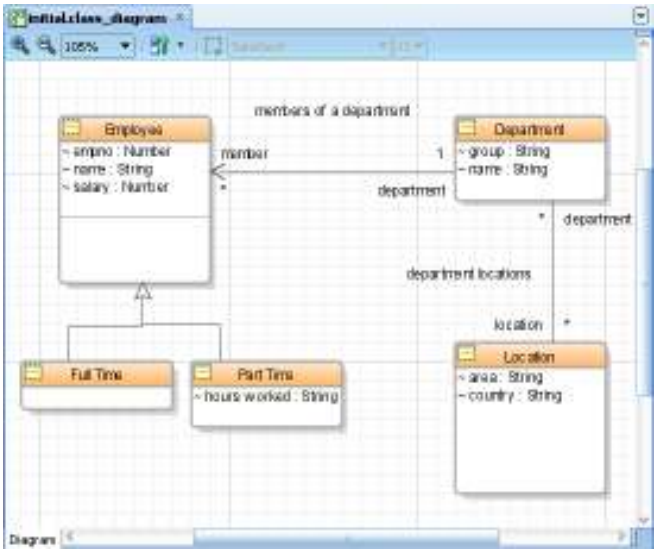

Illustration. 1: UML Logical Database Model

Illustration 1 shows a portion of a UML Class model being used for logical modeling. Note the conventions the developer has chosen to use in this example

- UML notation to name the association ends
  - Read 'Employee is a member of 1 Department'
- Named the association (relationship)
  - 'members of a department'
- Created a m:m association between Department and Location
- Specified the UML type of each attribute
  - empno:Number
  - name:String
- Has not specified generic 'key' attribute(for example 'ID')
- Used UML Generalisations to specify two sub-types of Employee: Full Time, Part Time
  See illustration 2 for an alternative and perhaps more database developer aware way of displaying sub-types

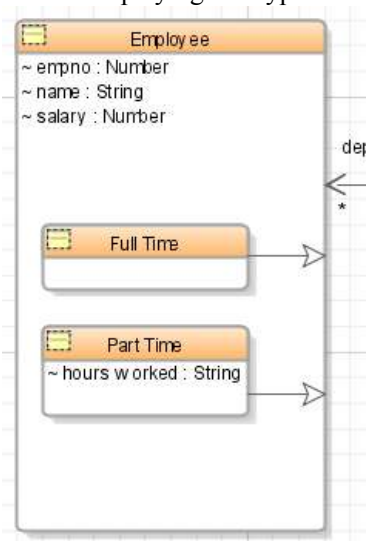

Illustration. 2: Sub-types in UML Logical Model

Transform UML Logical Model to Physical Database Model

Having created and versioned (perhaps many times) the logical model it is now ready to be transformed to a physical database model. This is done by selecting Context menu -> Transform -> UML to Database. This opens the transform wizard to declaratively guide you through the transform options:

- Choose the project and offline database and schema* the logical model
- Set naming options such as
  - Quote the UML name
  - Capitalize the name and insert underscores between words (part_time)
  - Attempt to pluralize names (EMPLOYEE to EMPLOYEES)

- Invert UML association role names
  for instance if rather than using the UML notation in the example above the user could have used a more ERD-like approach
  Read 'Employee from 1 dept' (illustration 3)

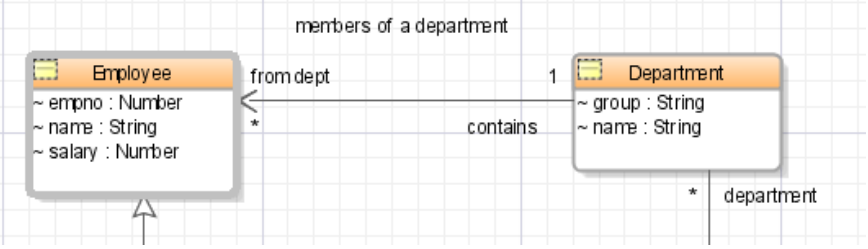

Illustration. 3: ERD approach

- Set options for transforming classes
  - How to deal with sub-types (discriminator column, separate tables etc.)
  - Create intersection table for m:m associations

Before running the transformer the developer can preview the expected model and make changes in the wizard

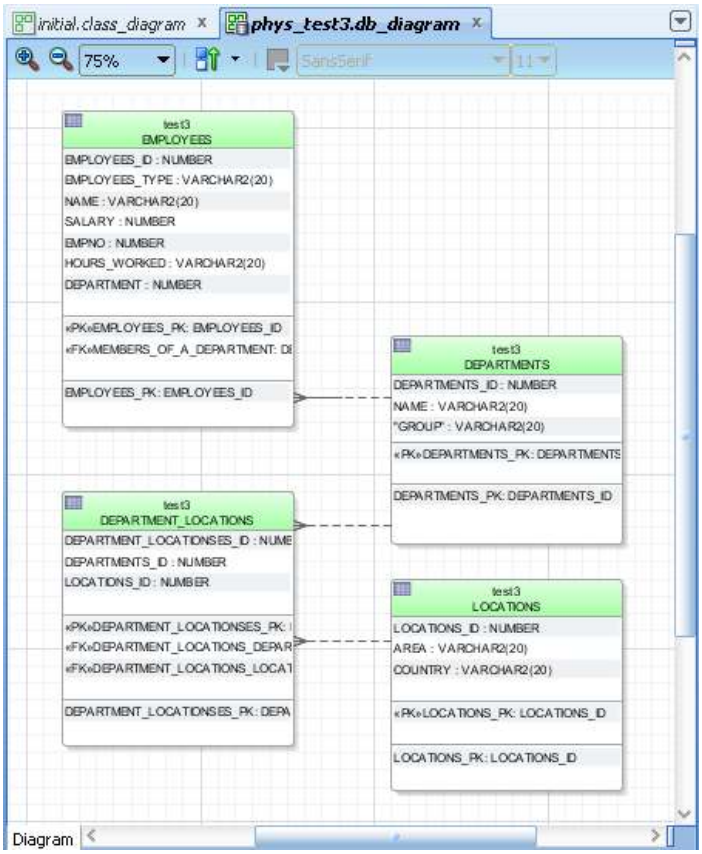

Illustration. 4: Physical database model

Illustration 4 shows the first cut default physical data model for this example. Notice that the crows feet notation is used for the foreign key relationships, a more UML like notation is also possible using a Property Inspector setting for the diagram. Like the class diagram the database diagram can also be customized through simple preference setting. Illustration 5 shows a variation of the display of the same model. A Primary key column <table>_ID:Number has been created for each table.

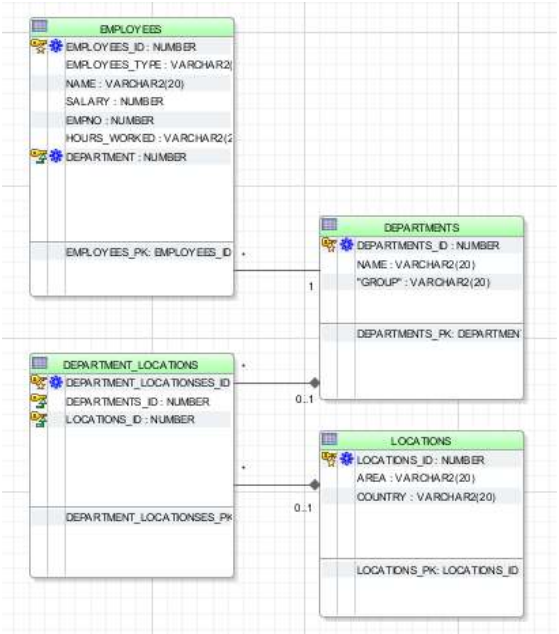

Illustration. 5: Enhanced physical model

Templates

To provide more standardization and consistency across database objects JDeveloper provides database object templates. These allow you to create tables, triggers, sequences etc with standard functionality either applicable to a specific database, schema, project, application or across applications.

There are various ways to create templates. One is to utilize a set of default templates as you create an offline database. In the offline database creation dialog you specify the name of the DB, the default schema, the database type to emulate (for instance: Oracle, Sybase, Times Ten, IBM, MySQL) and two checkboxes:

- Initialize Default Templates
  - This gives you a set of default objects (for instance, each table created will automatically have one column named <table name>_ID that will be marked as the primary key
- Edit Default Templates
  - This takes you to the default template objects so you can review and edit them as you create the specific database.

Another way is to create your own template objects from scratch in a specific offline database (perhaps in a separate project) and apply this set of templates to another offline database. For instance, you could create a table template with

- a column 'ID', a number that is set as the primary key
- audit columns

- a sequence and trigger to populate the ID column

Illustration 6 shows the table dialog where MY_TEMPLATE_TABLE is being defined. It shows that a sequence called MY_TEMPLATE_TABLE_SEQ populated by MY_TEMPLATE_TABLE_TRG will be applied to a column called ID. Note that the table is also called MY_TEMPLATE_TABLE. When this template table object is applied to a new table, the user's name will replace the template table name

To share a template between different offline databases you must create a dependency between the databases and choose the template object that is to be applied. First, go to the properties of the offline database and add a dependency. Then, again through the properties dialog:Default Templates select the offline database schema that holds your templates (in illustration 7: TEMPLATE_SCHEMA), select the object in your database that you want to associate with a template object and select the template. This example shows that for Tables the MY_TEMPLATE_TABLE template should be used.

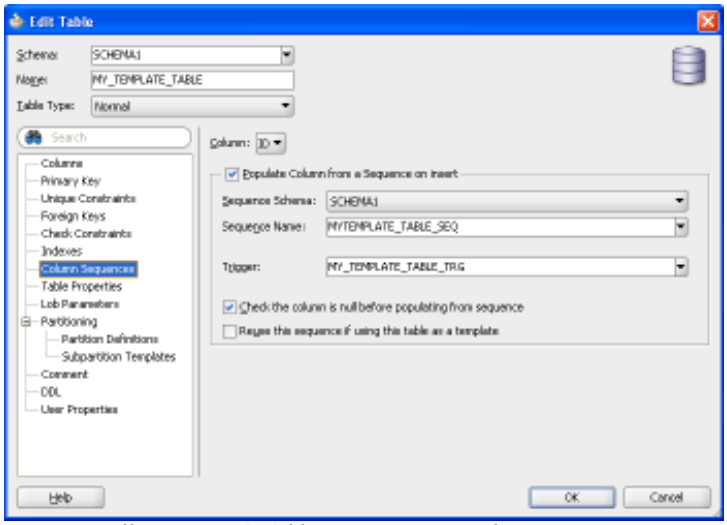

Illustration. 6: Adding a sequence and trigger

Templates can also be used in conjunction with the UML to Database Transformer. Not only does this add to consistency across your database objects but also means that common attributes (such as audit columns) do not have to be modeled at the logical level nor repeatedly added in the physical layer.

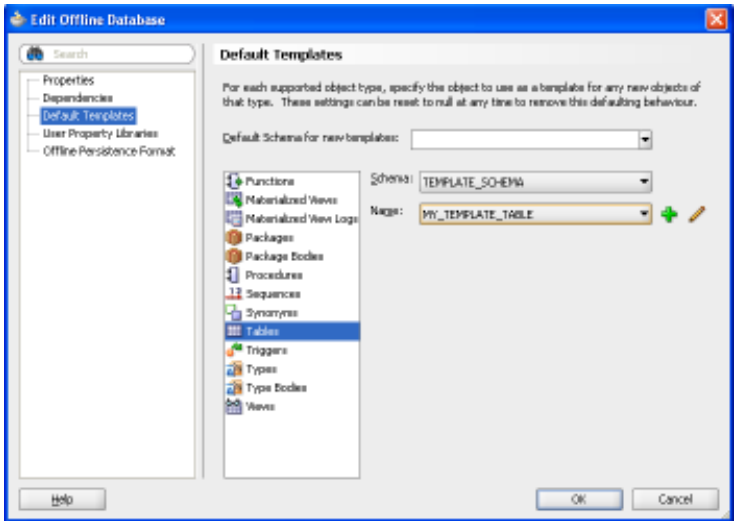

Illustration. 7: Using Template objects

Versioning

JDeveloper is integrated with a number of SCM systems including Subversion (SVN), Perforce, Dimensions and ClearCase. This paper will use SVN to demonstrate some of the versioning functionality that is available. Note that not all systems provide the same level of integration.

The UML class objects and the physical database objects so far described in this paper are all stored as individual objects by JDeveloper. This means that they can be versioned and compared as such. Illustration 8 shows part of the Application Navigator. Note that each UML object (class diagram, Location class, 'members of a department' association etc), the 'phys_test3' database diagram and each of the database tables (DEPARTMENT_LOCATIONS, DEPARTMENTS, EMPLOYEES) are separately accessible. Taking the EMPLOYEES table – it is currently at version 1055

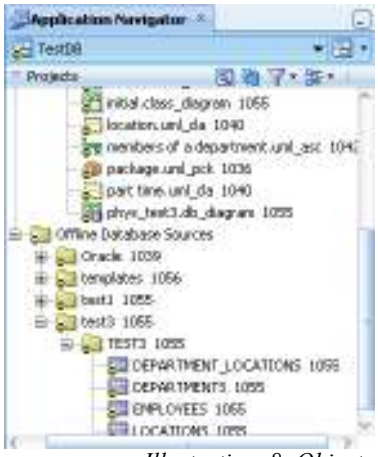

Illustration. 8: Objects in the navigator

Now look at the version history of the Employees table in illustration 9. The left hand pane shows version 1055, the right shows new version 1058. You can see that the size of column EMPLOYEES_TYPE has been changed to 2 and that a new constraint on the table has been defined and the indexes have changed. Note that those changes can be reverted using the central bar of the history dialog.
In addition you can view each versioned object's history in a graphical tree.

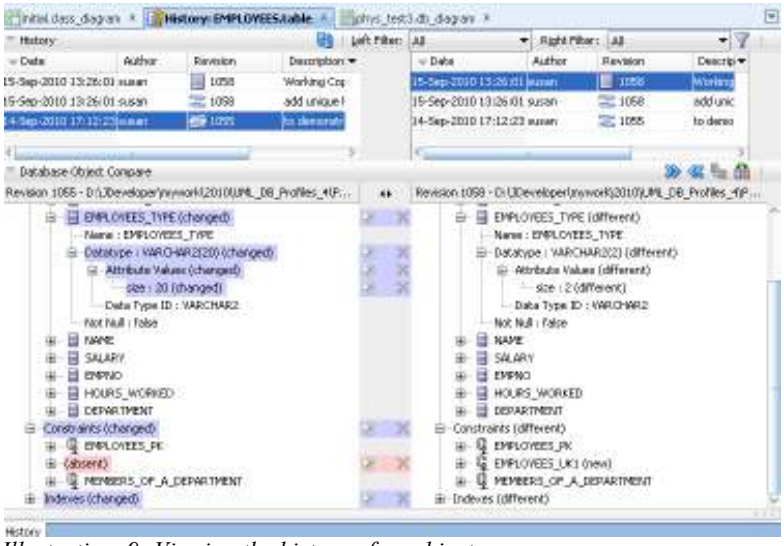

Illustration. 9: Viewing the history of an object

JDeveloper provides a Pending Changes window that is invaluable to both individual developers and to multi-development teams. The Candidates tab shows the developer which files have been added to his working copy since his last commit. He can use the tab to review the files and Add them as appropriate. If he wishes rather to have new files automatically added he can automate that using the Tools -> Preferences -> Versioning menu.
The Outgoing tab shows all the files that need commiting to the repository. From this tab he can select one or more specific files to Commit or he can choose either Commit or Commit Working Copy from the Versioning menu.
Commit (or Update) Working Copy will always commit all the files in the checked out working copy (most likely the whole application) regardless of what file has the focus in the application. Commit (or Update) will only commit those files that have been specifically selected.
The Incoming tab shows any files that have been committed to the repository by another user. Illustration 10 shows that the Employees table has been updated in the repository.

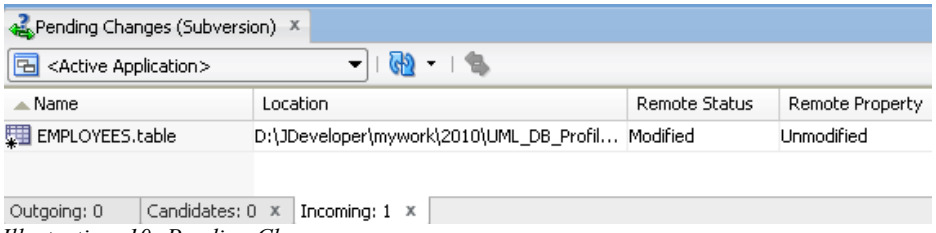

Illustration. 10: Pending Changes

Now the user can choose to either Update this one file or Update Working Copy to update the whole application. In many cases Subversion and the XML-aware parser can resolve any potential conflicts during the update. But in this case there is a direct conflict that only human intervention can resolve. JDeveloper provides a Merge Conflict dialog for the user to work from (Illustration 11). In this example it can be seen that the size of the EMPLOYEES_TYPE column of the developer's working copy (the left hand panel) is 3, whilst the repository (right hand panel) is 4. The centre panel that shows the common ancestor value is 2. The developer can now choose which value to shuffle to the middle panel to resolve and save the conflict and complete the update.
JDeveloper also supports Subversion Branching, tagging and SVN property setting.

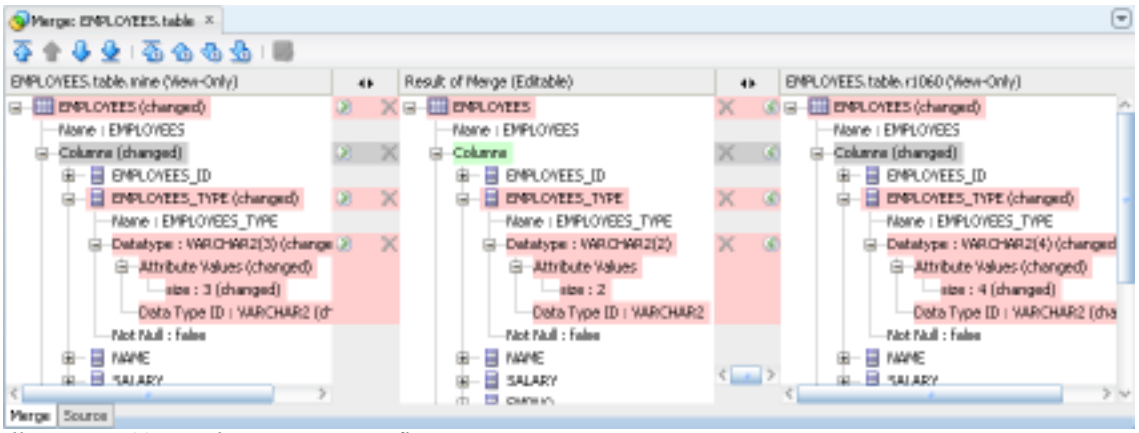

Illustration. 11: Resolving a merge conflict

Forward and Reverse Engineering

JDeveloper provides full forward and reverse engineering capabilities. The menu options Generate and Copy to Project are your entry point to both of these functionalities. You can reverse engineer from a live database and store the offline representation of that database in JDeveloper. Many teams use JDeveloper and an SVN repository as the source of truth of their database versions. Other customers generate CREATE, ALTER and REMOVE scripts and version those to use as their source of truth.

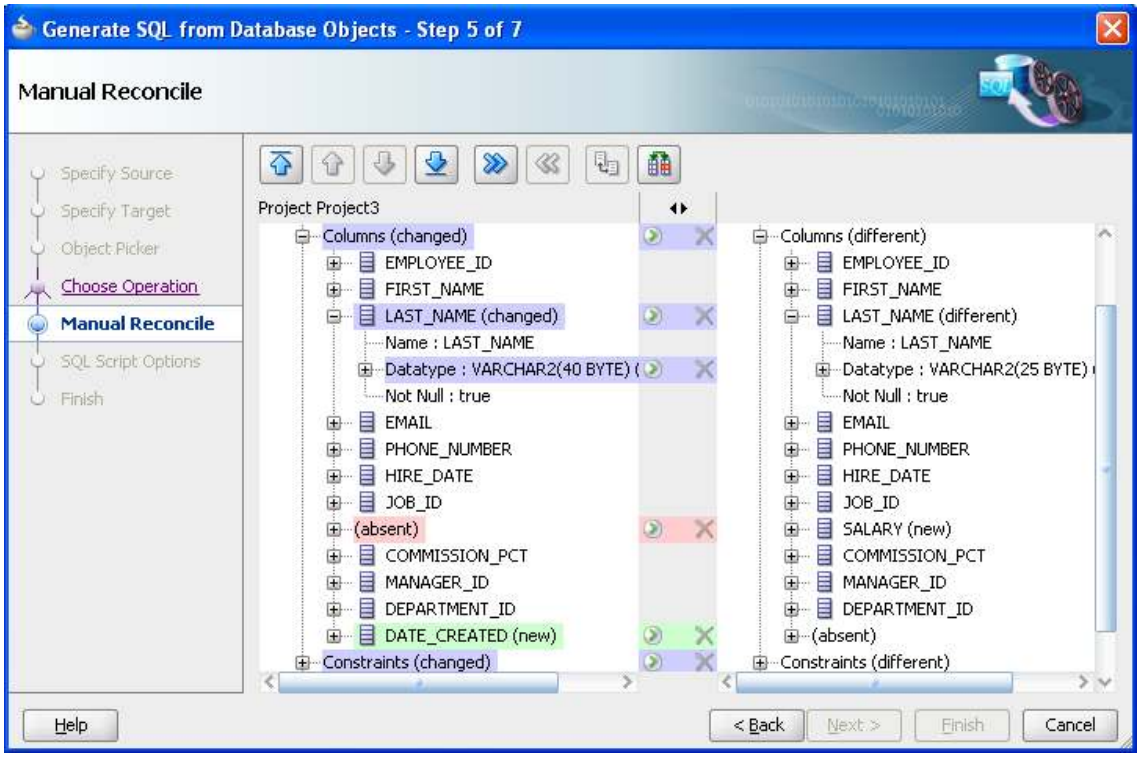

Illustration. 12: Manually reconciling two databases

You can also compare offline and online databases and manually select changes to be written to an ALTER script or directly update the compared database. Illustration 12 shows the Manual Reconcile pane when comparing an offline database with the live version. Notice that the length of LAST_NAME has changed and that SALARY is not present in the offline version whilst the offline version has an audit column that the live version does not. The user can shuffle each of these potential changes across to produce the required ALTER script.

Database Reporting

A number of pre-built reports are available to the database developer working with offline databases:

- Number of schema objects of each object type
- Schema objects in the offline database
- Tables with their column information
- Tables with their column count
- Tables with no primary key

In addition the developer can create custom reports using a SQL-like interface to interrogate an offline database. Illustration 13 shows the quick-pick interface for the Tables with no primary key pre-built report. The developer can use this report as a base for understanding and creating custom reports. Notice that from DB_PKCONSTRAINTS the foreign key relationships can be followed to DB_TABLES

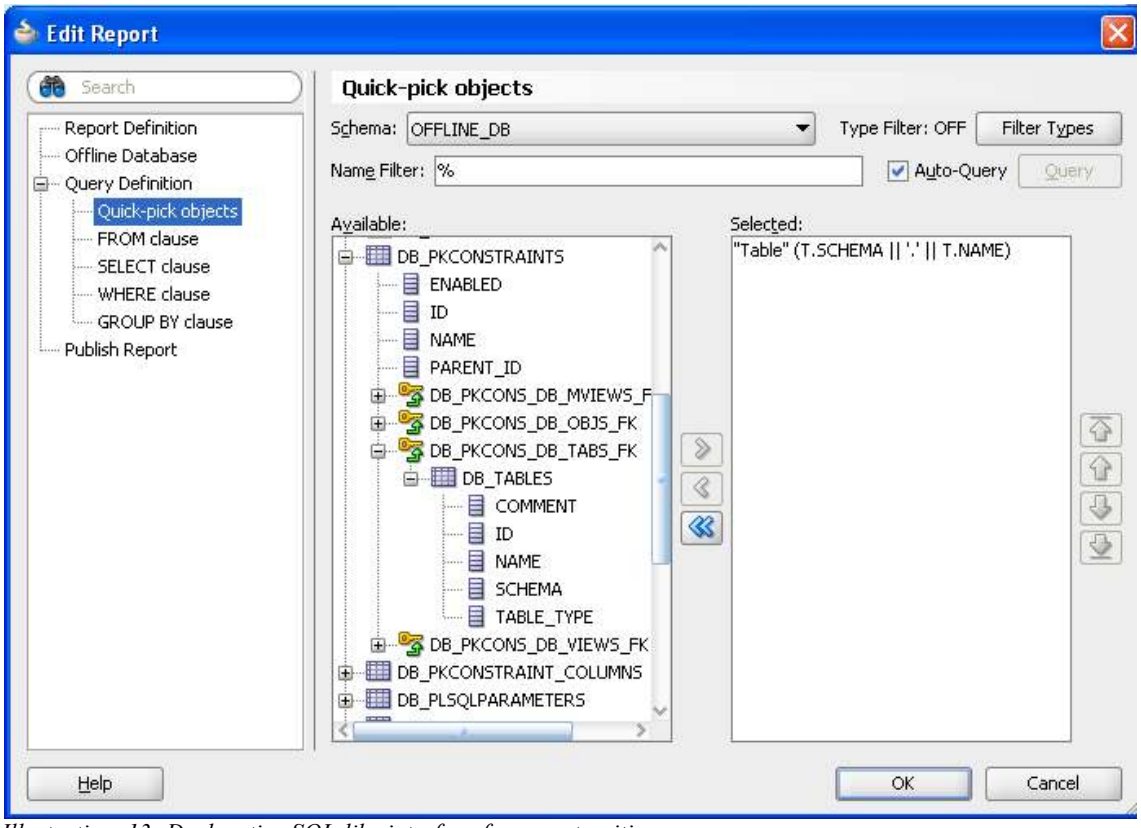

Illustration. 13: Declarative SQL-like interface for report writing

The developer can use this declarative approach to create reports or write the SQL directly as shown in illustration 14

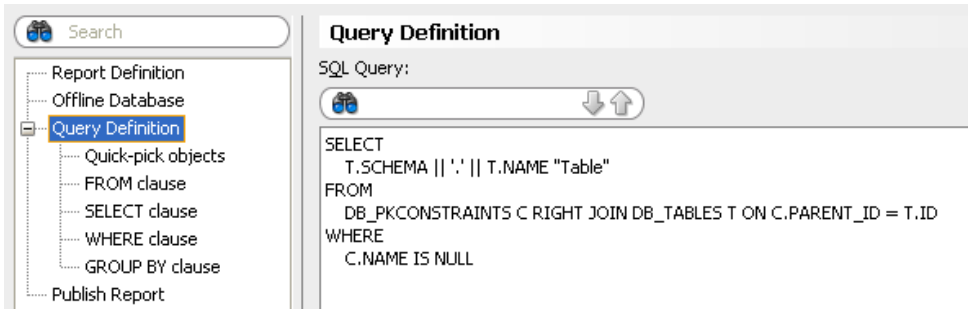

Illustration. 14: Writing SQL-like reports directly

The reports structures can be saved and versioned and the reports themselves can be run in and output to the log file or saved as HTML or CSV files for incorporating into documentation and reporting process

Conclusion

This paper has given an outline of how JDeveloper can be used by the database developer to design, model, refine, version and report on databases – both live and offline.

Contact address:

Susan Duncan
Oracle Corporation
Thames Valley Park
Reading, UK

Phone: +44(0) 118 924 5896
Email susan.duncan@oracle.com
Internet: http://www.susanduncan.blogspot.com