SQL Developer, Low Code and APEX Applications

Martin B. Nielsen

MBNDATA

Denmark

Keywords:

APEX, SQL Developer, Low Code, Data model, Process Model

Introduction

This session shows how to make the most of the SQL Developer when creating data models and database object, by automating part of the DDL creation (Low code). It also dives into the SQL Developer data modeler "reporting scheme" which enables us to re-use the artefacts from the model in our APEX applications.

SQL Developer model information can be exported into a reporting database schema. This information can then be providing input to the wizard generation of APEX pages, it can control navigation, provide help texts, domains and much more.

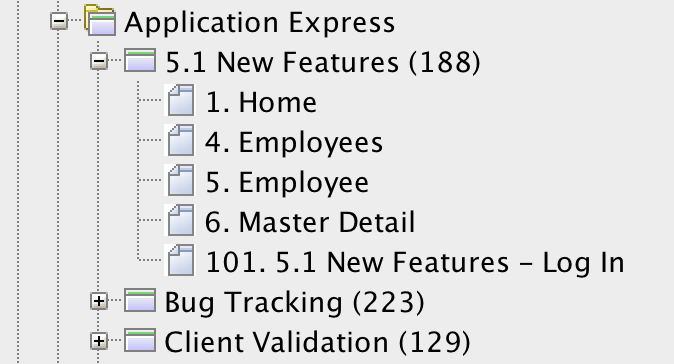
The session demos several examples of how this technique was used in a large Danish APEX project. It will also demo how easy it is to create data and process models in SQL Developer and how JavaScript (transformations) can be used to easily extend our models, make faster data models and automatically generate scripts for triggers and other database objects.

SQL Developer and the Datamodeller

The Oracle SQL developer is Oracle’s tools for maintaining databases and developing SQL and PL/SQL code. The tool is actually quite extensive and contains many integrated “modules” for handling various task that we as developers need when developing APEX applications.

Apart from the standard DBA, SQL and PL/SQL development capabilities, the SQL Developer contains a browser for our APEX applications, REST integration and many other useful tools.

The Browser gives us a tree structure overview of our APEX applications and have some interactions, which will actually change the application. The capabilities can be used to get an overview of the application and useful for DBA’s to enable / disable access to the application without actually logging into APEX:

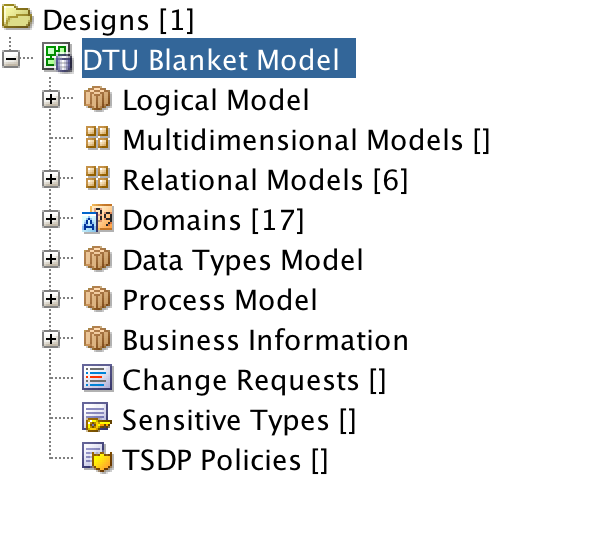


For more details about the APEX applications, I recommend looking into the APEX\_xxx views which are part of the APEX data dictionary.

Historically the Data Modeller was a separate product, which could be purchased from Oracle. Over time, this has been absorbed into the SQL Developer, thereby allowing free use of the product. Many people who has “converted” to SQL Developer, from TOAD and other products are often not aware that their development tool actually contains a complete designer tool (I will not say CASE – which is not a popular word any more).

The Datamodeller is well hidden among the many menus, but can by activated and used as a visual design tool for creating process and data models. The datamodel can already be useful to generate DDL for creating our database tables, indexes and constraints.

In this session I will go into details about how the Data Modeller artifacts (our visual design models) can be put to even further use, hence provide a “Low code” development environment for our applications.

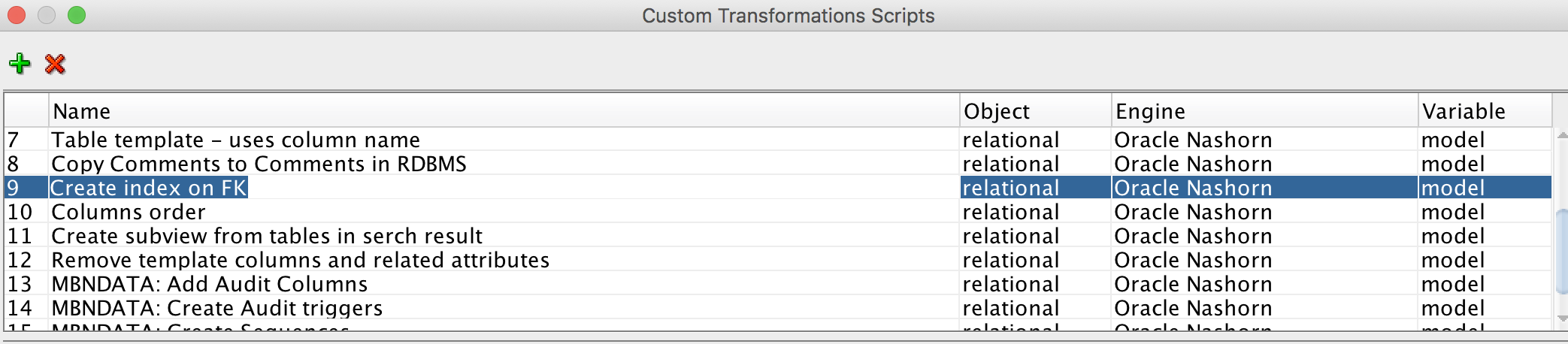


Low Code – generating code from SQL developer Datamodeller

As developers we tend to repeat out trusted development methods that we use form project to project. We can also be asked to implement and honor certain development standards during the project development. This could be anything from naming standard to index best practices (like always indexing foreign keys), having certain triggers to supply auditing information etc.

The best way to solve these tasks are by automation. The less we code (or copy/paste), the fewer errors we introduce. The Datamodeller itself has great tools for enforcing naming standard, prefixes etc. so we can focus on adding that extra functionality that our current project needs. The main tool for that, in the Data Modeller, is “Transformations”.

Transformation are snippets of JavaScript code that we write once, and save inside the Data Modeller.



Transformations can change the objects in out models, write to files and much more. There are a number of pre-supplied transformation from Oracle. Ex. “Create index on FK”, which will walk through the current datamodel and add indexes on all un-indexed foreign keys.

function getIndex(tab,cols){

keys = tab.getKeys();

for (var i = 0; i < keys.length; i++) {

index = keys[i];

if(!(index.isPK() || index.isUnique()) && !index.isFK() && index.isIndexForColumns(cols)){

return index

}

}

return null;

}

tables = model.getTableSet().toArray();

for (var t = 0; t<tables.length;t++){

table = tables[t];

indexes = table.getKeys();

for (var i = 0; i < indexes.length; i++) {

index = indexes[i];

if(index.isFK()){

columns = index.getColumns();

if(columns.length>0){

newIndex = getIndex(table,columns);

if(newIndex==null){

newIndex = table.createIndex()

table.setDirty(true);

for (var k = 0; k < columns.length; k++){

newIndex.add(columns[k]);

}

}

}

}

}

}

This snippet is making it quite fast for us to make sure we minimize locking issues and improve performance on our tables. The indexes are only implemented in the visual model, not the database (for that we need to generate DDL from the model).

The customer Transformations can also be used for generating other objects such as sequences, triggers, PL/SQL packages etc. This can be achieved by traversing the model and creating new files based on the tables in the model.

Here is an example of a custom transformation for creating sequences for all the tables in the model:

outFile = new java.io.FileWriter("/Users/martin/SVN/DTU/XXSUB/ddl/create\_sequences\_per\_diem.sql");

out = new java.io.PrintWriter(outFile);

out.print("-- -------------------------------------------------");

out.println();

out.print("-- Title : Sequences");

out.println();

out.print("-- Created: " + new Date() );

out.println();

out.print("-- -------------------------------------------------");

out.println();

out.println();

tables = model.getTableSet().toArray();

for (var t = 0; t<tables.length;t++)

{

table = tables[t];

tableName = table.getName();

out.println("CREATE SEQUENCE " + tableName + "\_S START WITH 1 INCREMENT BY 1 NOCACHE;");

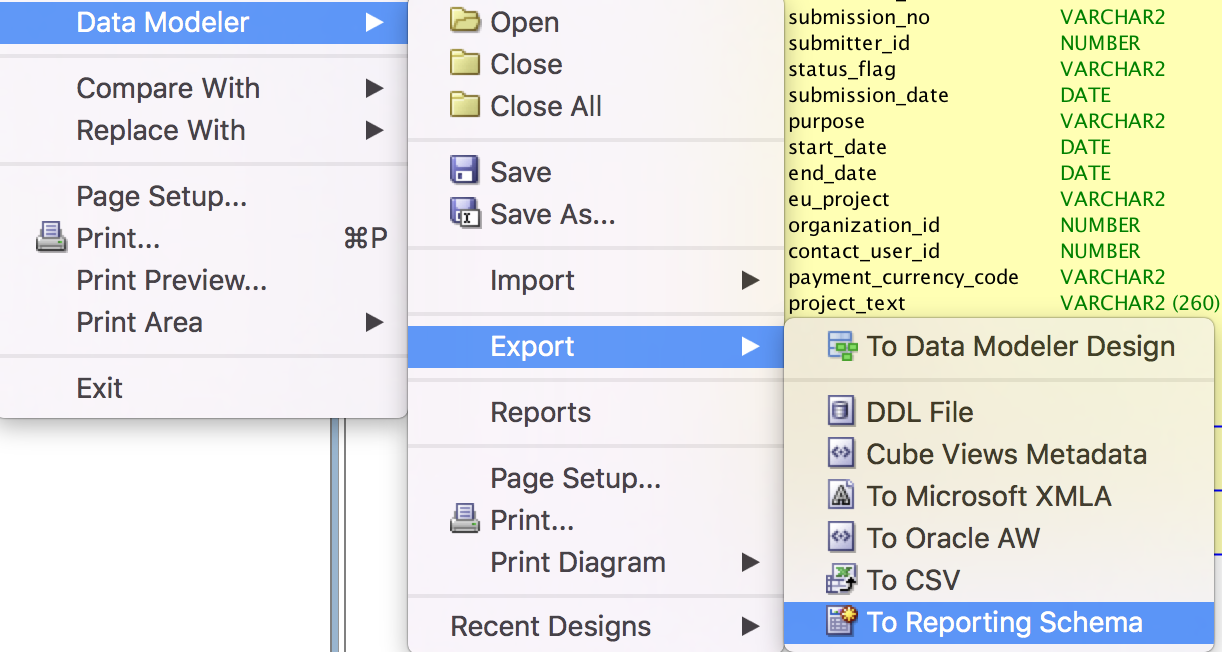
}

out.close();

The SQL Developer reporting schema

The great strength of the Data Modeller is that any artifact we create in the model can be exported to a so called “Reposting scheme”. Collections of models (our projects) are saved as a “Designs” which are a file based storage (consisting of lots of files in a directory, many of them XML files).

This is not so useful for us as database developers, so in order to use the information to something exiting, we need to export this to a datamodel (the reposting scheme):



Once our designs are in the reporting scheme we can use them for our APEX applications.

The reposting schema consists of a number of tables, prefixed with DMRS\_ and views prefixed by DBRV\_x: Ex.:

DMRS\_VIEW\_ORDER\_GROUPBY

DMRS\_VIEW\_CONTAINERS

DMRS\_VIEW\_COLUMNS

DMRS\_VALUE\_RANGES

DMRS\_TRANSFORMATION\_TASK\_INFOS

DMRS\_TRANSFORMATION\_TASKS

DMRS\_TRANSFORMATION\_PACKAGES

DMRS\_TRANSFORMATION\_FLOWS

DMRS\_TRANSFORMATIONS

DMRS\_TELEPHONES

DMRS\_TASK\_PARAMS\_ITEMS

DMRS\_TASK\_PARAMS

DMRS\_TABLE\_INCLUDE\_SCRIPTS

DMRS\_TABLE\_CONSTRAINTS

DMRS\_TABLEVIEWS

DMRS\_TABLES

DMRS\_STRUCT\_TYPE\_METHOD\_PARS

DMRS\_STRUCT\_TYPE\_METHODS

DMRS\_STRUCT\_TYPE\_ATTRS

DMRS\_STRUCTURED\_TYPES

DMRS\_SPATIAL\_DIMENSIONS

DMRS\_SPATIAL\_COLUMN\_DEFINITION

And

*DMRV\_DOMAINS*

*DMRV\_RAGGED\_HIER\_LINK\_ATTRS*

*DMRV\_COLLECTION\_TYPES*

*DMRV\_MODEL\_DISPLAYS*

*DMRV\_STRUCTURED\_TYPES*

*DMRV\_STRUCT\_TYPE\_METHOD\_PARS*

*DMRV\_DOMAIN\_AVT*

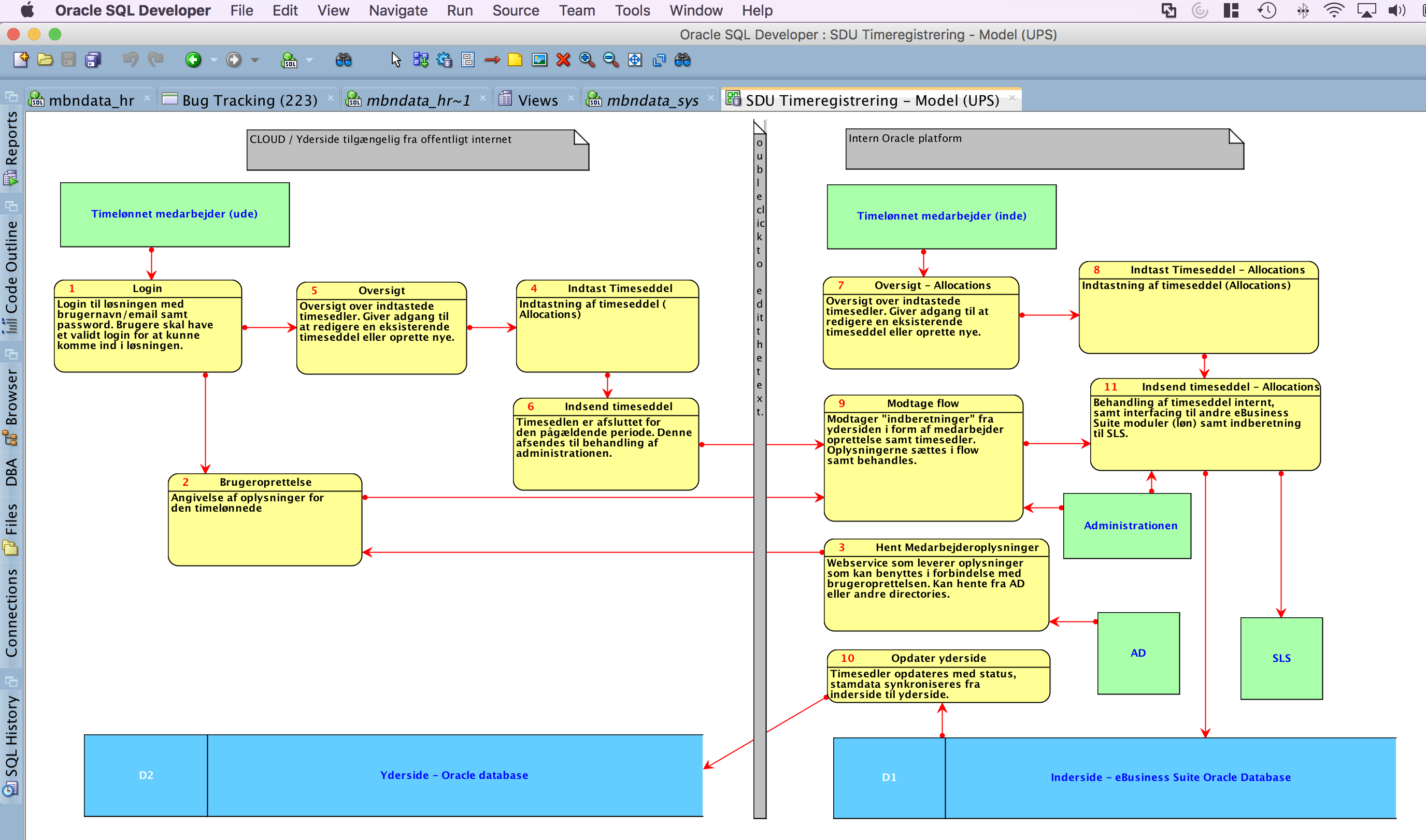
*DMRV\_HIERARCHY\_ROLLUP\_LINKS*

*DMRV\_RECORD\_STRUCT\_EXT\_DATAS*

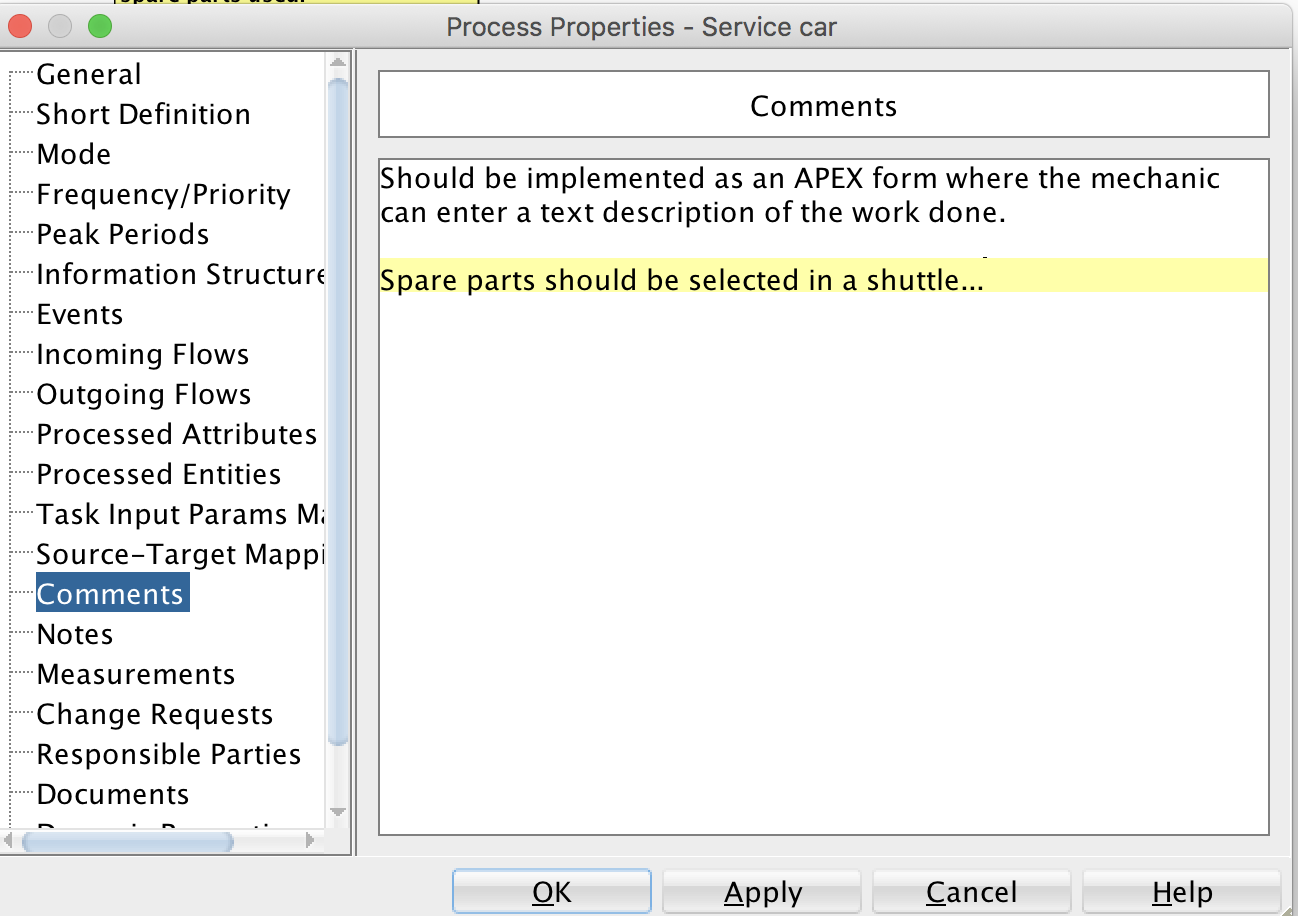
*DMRV\_RES\_PARTY\_ELEMENTS*

Integrating the process model with APEX

Now that our models are stored in database tables, we can start using the information in APEX applications. The process model contains our dataflow in the application. Each process can be a composite, consisting of many underlying processes.



Processes can be annotated by designers with all relevant information the developer needs when the process should be coded (ex. As an APEX page).



Information from the process model can be used for various purposes in our application. I will demonstrate each of these usages with a live demo and code examples:

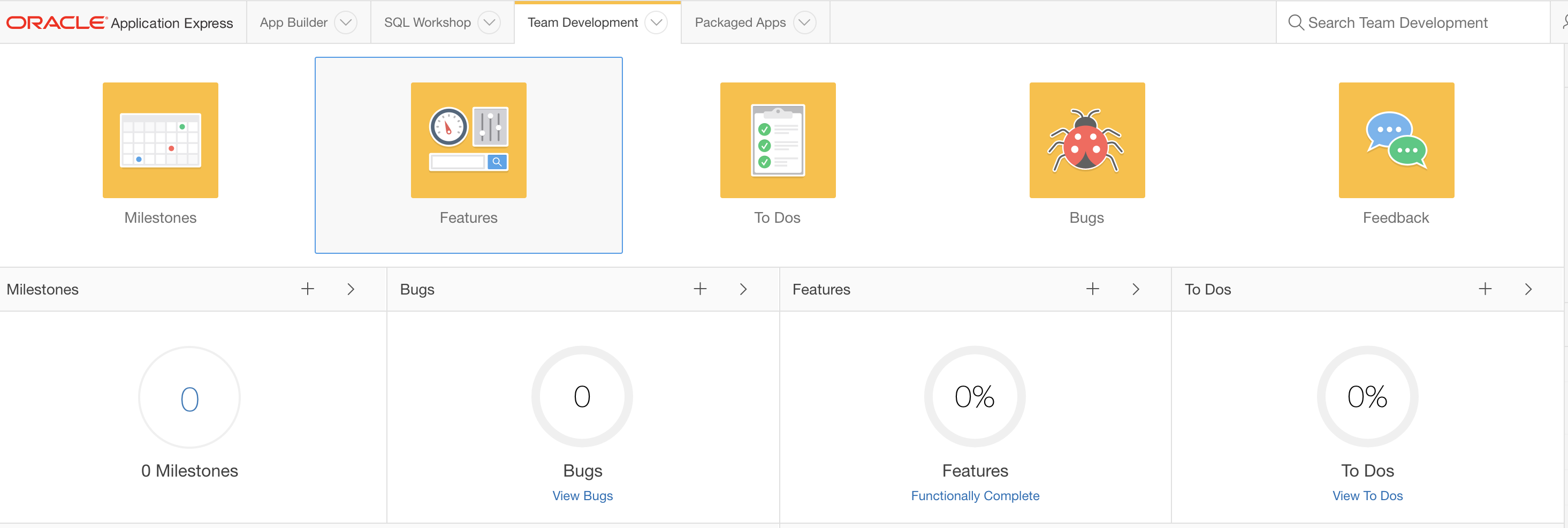
**Custom APEX Menu**

The hierarchical process model defined can be used to control the menu structure of our APEX applications. This can be achieved by creating a dynamic menu using a SQL statement, which looks in to the reporting Scheme.

**Integrating with APEX Team development**

The APEX team development module can be used to track the features which should be implemented in the new APEX application.

Processes can often be equal to “Features”, so why not create the features automatically based on the processes from out process models.



**Developer assistance**

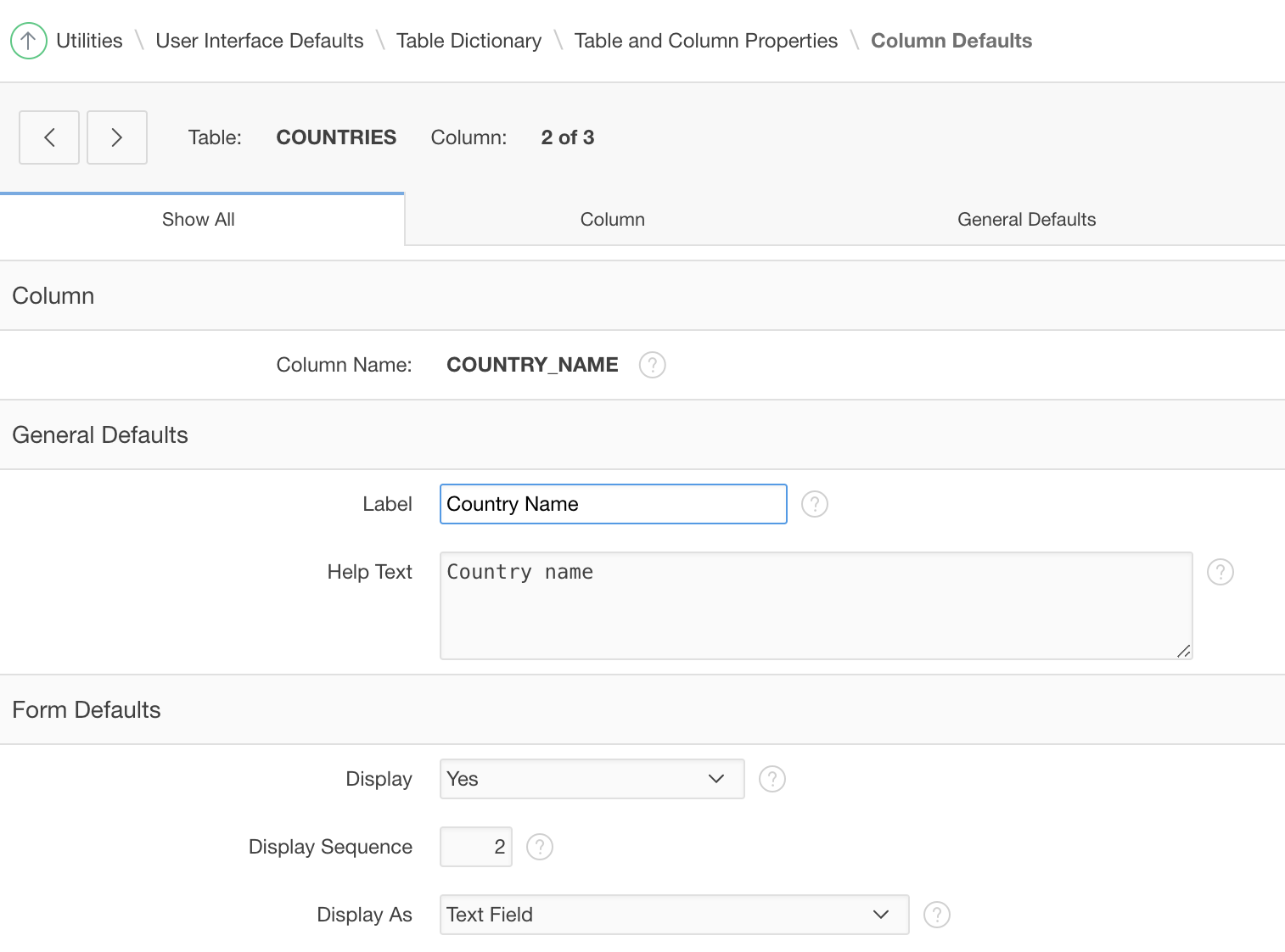
When the developer starts working on the new feature, we can supply all the information the developer needs from the model. The information can be available in regions that are only displayed during the development. Ex. Using “Build Options”

Integrating the data model with APEX

The Data Models contains a wealth of information, which can likewise be integrated into our APEX applications. The use cases are ex.:

**User Interface Defaults**

The data model is full of good information about our columns and tables. If this information can be placed in the APEX Users Interface defaults, our applications can use this when new pages are generated (using the wizards).



**Help text**

Information about tables and columns (entered by the designer), can be very useful for the end users. So they can be used to generate help information. Ex. As popup windows displaying information about the tables and fields which are part of the current screen.

**Contact information**

The Models also contain information about the contact persons for the project. This is useful information which can be displayed in the APEX application. Ex. In the support page.

**Best of all: by integrating the design model actively in the Application, we ensure that the model is kept up to date and will not get out of sync with the application over time. Thereby providing a valid system documentation for future supporters of the systems we develop.**

**Contact address:**

**Name**

Martin B. Nielsen

MBNDATA

Amtmandsvej 2

4300 Holbæk

Denmark

Phone: +45 40989637

Fax:

Email martin@mbndata.dk

Internet: mbndata.dk