`

Darwin Sabbatical  
User Guide

# Introduction

Darwin Sabbatical is a toolset that generates object types based on table definitions in the datadictionary of the Oracle Database.

Darwin Sabbatical as a toolset can be compared with Java persistence frameworks as Hibernate or Oracle TopLink. Not in the feature richness of those frameworks but in the Object-Relational mapping it provides.

## What are possible usages of Darwin Sabbatical?

Darwin Sabbatical is originated in two Oracle SOASuite projects. One was a project about schedules of tasks for the generation of different files that needed to be transferred to different destinations. With BPEL these tasks needed to be handled and then it is very convenient to query a complete object structure from all the inter-related tables the database using the primary key value of the base entity. Doing so just one database adapter configuration is needed, instead of one per detail table. In that project only the base objects were generated, no support for foreignkey based objects was added. Those were created by hand.

In another project, a few years later, cases that are transferred from another application system using Oracle Service Bus needed to be inserted in a target application system. The desire was that this was done in one transaction. In

## What does it provide?

The following kinds of object types are generated:

* Entity or base objecttypes: types that reflect the table definitons: for each column in the database an attribute is generated based on the column definition (datatype) of the column. Additional generated parts are:
  + Additional attributes (collections or base types) based on the configuration of the ForeignKey relations
  + Optional derivation columns allow you to have lookup attributes filled based on a selection method in the derivation attribute. Before an insert the select-method in the derivation attribute is called, after which the result attribute is copied to the corresponding column in the base type.
  + A parameterless constructor to allow instantiation of an empty object, to set attribute values in a later stage.
  + A Primary Key based constructor to allow instantiation based on a query on the table with the primary key columns as a where clause. Based on foreign key definitions the constructors of child/detail or lookup types are called as well.
  + An Insert method to do an insert including a call to the insert method of Lookup- and ChildDetail attributes.
  + An Update method for only the base type (no relational updates are done).
  + Delete method for only the base type
  + Delete Cascade method doing also delete-cascade for the lookup and collection attribute (complementary to the insert)
  + A To\_XML() method to create an xmltype with the attributes of the base type, including a call to the To\_XML methods of child/detail and Lookup Attributes. This allows the simple generation of an XML of the whole object tree, to be printed, interfaced with another application, or for example stored and/or investigated using an XMLEditor.
* Table of Object types that are a simple collection of the base/entity types.
* Collection Object Types. These objects are in fact a wrapper around the Table Of Object Types. They supply foreign key based constructors, insert, delete and to\_xml methods, to travers the table-of attribute to call the corresponding method of each table-member.

# Install/Setup

## Install Database objects

### Install tables:

Run the following scripts to create the necessary tables:

|  |  |  |
| --- | --- | --- |
| Script | Object | Purpose |
| xxx\_derived\_columns.sql | Table xxx\_derived\_columns | To add custom code for column derivations like foreign keys based technical id’s of lookup tables. |
| xxx\_fk\_definitions.sql | Table xxx\_fk\_definitions | To drive the generation of code based on foreign keys. |
| xxx\_log\_id\_seq1.sql | Sequence xxx\_log\_id\_seq1 | Sequence to generate id’s for the logging table. |
| xxx\_logs.sql | Table xxx\_logs | Helper package to do logging. |
| xxx\_objects.sql | Table xxx\_ objects | Table to store generated code-artefacts and scripts. |
| xxx\_xmldocuments.sql | Table xxx\_ xmldocuments | Table to store generated xml documents and XSL-templates. |

### Insert data

The tables *xxx\_derived\_columns* and *xxx\_fk\_definitions* drive the code generation. The contents is used in the queries on the datadictionary to complement the xml extracted from the queries.

It is recommended to edit the contents of those tables using the insert scripts. Run the corresponding scripts to fill/update the tables:

* Insert\_Derived\_columns.sql
* Insert\_fk\_definitions.sql

### Install packages

The following packages are needed for the toolset:

|  |  |  |
| --- | --- | --- |
| Script | Object | Purpose |
| xxx\_log.pls | xxx\_log | Logging |
| xxx\_table\_definitions.pls | xxx\_table\_definitions | Methods to query data from the datadictionary, ammended with extra info into XML. |
| xxx\_xml.pls | xxx\_xml | Helper package to do xml processing. |
| xxx\_xslt.pls | xxx\_xslt | Helper package to do xsl processing. |
| xxx\_xmldoc.pls | xxx\_xmldoc | Save xml documents in *xxx\_xml\_documents*. |
| xxx\_xml\_gen.pls | xxx\_xml\_gen | Helper package to parse xml documents and generate code using xsl-templates. |
| xxx\_obj.pls | xxx\_obj | Helper package to store generated objects into *xxx\_objects* table. |
| xxx\_gen\_objects.pls | xxx\_gen\_objects | Generate the objects for the datamodel of a project. |

## Add XSL-Object Templates Templates

# Configuration and Object Generation

## Edit XXX\_GEN\_OBJ Package

## Configure ForeignKey Relations

## Derived Columns

## Generate Objects

# Wishlist

* Create an extra table to add custom members to be generated along with the entity types and/or the collection types. More or less comparable to the derived-columns functionality.
* Create a table to drive the generation of objects for the tables. Now in the XXX\_GEN\_OBJECTS package specification 2 lists of tables are kept. One to drive the generation of creation of objects in the correct order of dependency and one reversed list for the generation of drop-type statements. A drive-table would make the generation of types more generic and helps to make the toolset customer/project independent.
* Having a table to drive the generation could also help in registering alternative names for the object names. Currently the names of the types are derived from the tablenames, that are normally plural. But object-names in Oracle can at most be 30 characters long. But to differentiate between entity, table and collection types a suffix is used. So table names currently must be shorter to cater for the extra suffix in the type-names.