REPORT ON USING GNATCOLL-DB WITH POSTGRESQL

Prepared by

Juan L. Freniche

fdesp87@gmail.com

Updated 2022-10-22

Table of Contents

1 Scope

1.1 Identification

This document describes the modifications to ACT's gnatcoll-db tool that were implemented to remove some limitations and expand some other aspects of the tool.

This document version 6 corresponds to ACT gnatcoll-db version 23.

1.2 Purpose

ACT's gnatcoll-db is a tool used to interface Ada to some databases (currently Postgresql and sqlite). It consist of several Ada packages distributed in the following parts:

- sql: this part is in charge of interfacing Ada with SQL.
- postgres: this part is in charge of interfacing Ada with Postgresql database management system.
- sqlite: this part in in charge of interfacing Ada with sqlite database management system.
- gnatcoll_db2ada: this part is in charge of generating Ada packages implementing the cited interfaces. If dealing with more than one database management system it is named as gnatcoll_all2ada.

Although very promising, the complete package exhibit some limitations in its intended use as well it can be extended to support more completely the facilities provided by the databases management system.

Therefore it has been modified in these lines, widening its benefits.

1.3 Introduction

Traditionally there has been a difficulty in using Ada for database applications. Gnatcoll-db came to resolve such difficulty, at two levels:

1. Interfacing Ada to the database management system through the use of SQL.
2. Interfacing Ada on top of the previous level using an ORM approach.

However when using seriously gnatcoll-db with medium to large databases, it arose severe limitations in the implemented part as well lack of features that render gnatcoll-db only valid for limited databases applications. The modifications here described are an attempt to expand the use of gnatcoll-db by solving these issues.

In spite of the required or proposed improvements, the facility gnatcoll-db is a great idea which allows to access RDBMS in the programming language Ada, using object oriented features or Object Relational Mapping (ORM). In definitive it is an excellent work.

This document is composed by the following sections:

Section 1 is the scope. Section 2 contains the necessary references. Section 3 describes the modifications. Section 4 describes the tests, both objectives, methods and results. Finally section 5 specifies the delivery method. The document ends with an appendix with used acronyms.

2 References

1. Ada 2012 Language Reference Manual, ISO/IEC 8652:2012(E). Available online at http://www.ada-auth.org
2. Structured Query Language (SQL), ISO/IEC 9075:2016
3. Postgresql documentation, available online at https://www.postgresql.org/
4. sqlite documentation, available online at https://www.sqlite.org/
5. gnatcoll-db documentation, available online at https://docs.adacore.com/live/wave/gnatcoll-db/html/gnatcoll-db_ug/index.html
6. R. Elmasri, S. B. Navathe: *Fundamentals of Database Systems*, 6th ed., 2011, Addison-Wesley. In particular Part 3, Conceptual Modeling and Database Design and Part 4, Object, Object-Relational and XML: Concepts, Models, Language and Standards.
7. IEEE Standard for Floating-Point Arithmetic, IEEE 754-2019, available at the IEEE.
8. Alire, the GNAT package management tool: https://alire.ada.dev/

3 Description of the Modifications

3.1 Environment

1. Linux Xubuntu 22.04 “jammy”
2. Meld (visual comparison tool) 3.20.4
3. Initially GNAT Community 2021 and then GNAT Compilation System from alire
4. Posgresql 13.6
5. pgAdmin4 6.8
6. Gnatcoll-db version 23 from *https://github.com/AdaCore/gnatcoll-db.git*

3.2 Building and installing Gnatcoll-db

1. sql: using the makefile.setup produced by make setup (i.e., ENABLE_SHARED=yes and BUILD=PROD, there is the following error:

   ```
   gprbuild -p -m --target=x86_64-linux  -j2 -XGNATCOLL_VERSION=0.0 -XBUILD=PROD
   -XLIBRARY_TYPE=relocatable -XXMLADA_BUILD=relocatable -XGPR_BUILD=relocatable
   \
   gnatcoll_sql.gpr
   gnatcoll.gpr:12:04: value "relocatable" is illegal for typed string "build"
   gprbuild: "gnatcoll_sql.gpr" processing failed
   ```

   This is traced to an installation of gnatcoll-core with no relocatable library. Just recompile and reinstall gnatcoll-core with relocatable libraries.

2. Postgres: when trying to build the relocatable library:
   …
   …
   ```
      [Ada]          gnatcoll-sql-postgres-gnade.adb
   Build Libraries
      [gprlib]       gnatcoll_postgres.lexch
      [bind SAL]     gnatcoll_postgres

   raised STORAGE_ERROR : stack overflow or erroneous memory access
   ```

   Modifying by an exaggerated degree the Unix process stack size produces the same result. It was not the objective to resolve this issue, so all makefile.setup were changed to ENABLE_SHARED=no. Then only the static library is obtained.

   However, moving the compilation system to the new alire compilation, the problem disappeared, we can have the relocatable libraries.

3. sqlite: same issue with relocatable. Move to the new alire compilation system.
4. gnatcoll_2db2ada: here DB_BACKEND=all to support Postgresql. Then the generator program is named gnatcoll_all2ada.

3.3 Building the Required Components from alire Compilation System

The process is not so easy as alire documentation presents. Follow these steps:

1.- Download the compiler and gprbuild and move (as downloaded) to /opt/GNAT/12.2.1

```
alr -v -get -o gnat_native
```
move it to /opt/GNAT/12.2.1/
alr -v -get -o gprbuild
move it to in /opt/GNAT/12.2.1/

2.- Download and Install xmlada

```
alr -v -get -o xmlada
./configure --prefix=/opt/GNAT/12.2.1/ --enable-shared
make all
make install
```

3.- Download and Install libgpr

```
make setup and change num_processors to 1
make libgpr.build
make libgpr.install
```

4.- Download and Install gnatcoll (core)

```
make setup
change num_processors to 1, version info 22.0.0
make
cd doc
make html
cd ..
make install
```

5.- Download and Install gnatcoll_icon. Really you will download the complete gnatcoll bindings (gmp, iconv, lzma, omp, python, python3, readline, syslog, zlib). Only icon is required.

```
alr -v get gnatcoll_iconv
change version to 22.0.0
cd iconv
./setup.py build
./setup.py install
```

6.- Now instead of using the gnatcoll-db package, use the one with all modifications described here. See Section 5 to see how to obtain it.

Follow this order to compile/link/install: sql, postgres, sqlite, xref, gnatinspect, gnatcoll_all2ada

In each subdirectory:

```
make setup
edit makefile.setup:
      change prefix to /opt/GNAT/12.2.1,
      processors 1,
      version 22.0.0
```

```
make
make install
```

3.4 Using gnatcoll_all2ada and Some Terminology

RDBMS is the acronym for **R**elational **D**ata **B**ase **M**anagement **S**ystem, referring to the software that manipulates the data. In our case, it is Postgresql, whilst sqlite will be considered in a few cases in this report.

The SQL DDL (Data Definition Language) is the script used to generate the database, which normally can also be obtained later using the RDBMS. By convention, this file is ended in ".sql".

gnatcoll_all2ada can connect to the RDBMS (Postgresql or sqlite) and build an internal Ada structure called the DB_Schema. To do that, use gnatcoll_all2ada with the option -text and all necessary RDBMS parameters (database name, host, port, user, password).

In addition, using such option -text also produces a file containing a reduced database schema. We will call it the "generated schema". By convention , these files will end in ".orm".

Using again gnatcoll_all2ada with -api -orm, this program will generate the ORM Ada code using the previously generated schema. This last part is obtained by internally calling a python module called dborm.py (which will be replaced by an Ada module, see section 3.12).

3.5 Recommendations:

1. If the Postgresql database has a sql schema (see SQL documentation for this concept):
   1. When calling gnatcoll_all2ada, the option -omit-schema <the_sql_schema> shall be included, otherwise lots of
      ```
      tmp_orm:5047:14: error: child unit allowed only at library level
      tmp_orm:5440:16: error: missing "is"
      ```
      are issued by gnatchop.
   2. On the other hand, if such option is included when calling gnatcoll_all2ada, all emitted sql statements are rejected because the sql schema is not in the search path of the sql statements. Solution:
      Insert in your program a SQL statement=> set schema '<the sql-xchema>;' such as this one:
      ```
      Execute (Connection => DB (Session),
               Query => "set schema '<the sql-schema>';");
      ```
      However that should be solved in the generated code.
2. gnatcoll-db maps Varchar(n) to text (unbounded string) loosing the limit. This is a sensible design decision but the length of the string is lost. Longer strings will be reported by the RDBMS as runtime errors.

3.6 Pretty-Print the generated schema

1. A new option was included in gnatcoll_all2ada: "-noautodata". Default false. If not given (i.e., noautodata is false so you are requesting automatic data generation), then in the generated schema:
   1. if in the sql schema the table or view description starts by "(<some_name>) etc.", the 3rd column for tables is automatically generated as <some_name>. Note that the ")" shall be followed by a space. Same for views. **Implemented.**

2. In other case, if the table name ends in "s", the 3rd column for tables is automatically generated by removing the "s". Same for views in case the view name ends in "s_view". **Implemented.**
   Warning: note that this approach may produce incorrect words (for example, “currencies” is translated to “currencie”). Also Name conflicts may arise if the generate table’s row_name is identical to some field in some table. Therefore manual intervention may be required.
2. See below for automatic reverse relations (section 3.8 point 14). **Implemented.**
3. Two new options have been placed under user control (they were already implemented): “-noshowcomments” and “-noaligncolumns”. Default false. **Implemented.**
4. Aligning columns is now global for all tables. **Implemented.**

3.7 New Supported Types

1. The function "-" (T : T_Money) return T_Money renames SQL_Impl."-"; was missing. **Implemented.**
2. *Money* is numeric(16,2). This is normally insufficient, so *numeric(24,8)* and *numeric(8,4)* were included similarly to *Money*. **Implemented.**
3. Note that in gnatcoll-sql-inspect.ads, the overriding functions Type_To_SQL for these new types are in the body. **Implemented.**
4. *Smallint*: gnatcoll db2ada translates it to Integer instead of Short_Integer. **Corrected.**
5. *Real* is unknown to gnatcoll_all2ada. Postgresql *Real* type is IEEE 754 Binary32 with a precision of at least 6 decimal, which is equivalent to Ada Float (also IEEE 754 Binary32). **Implemented.**
6. Unsupported SQL type: *double precision*. It should be mapped it to Long_Float (IEEE 754 Binary64). **Implemented.**
7. *Interval*: In Postgresql, interval is a duration but with larger range than Ada Duration. gnatcoll_all2ada produces unknown field type *interval*. A new routine to parse Postgresql Duration (ISO8601, Sql_Standard, normal Progresql stype and Postgresql verbose) has been included in gnatcoll-sql-exec_private.adb. **Implemented.**

3.8 Observations when generating the schema

1. Default negative numbers such as -5.0 are returned by Postgresql as strings instead of numbers. However gnatcoll_all2ada do not honor in all cases the "*::numeric*" qualifier nor the “*::integer*”, etc. **Corrected**.
2. Remove the writing of type cast for default values in the generated schema. **Implemented.**
3. *Money with default value* produces a constraint error in gnatcoll-sql-inspect.ads:199:15. The solution is to move Type_To_SQL to the body. **Corrected.**
4. *Timestamp without time zone* is translated by gnatcoll_all2ada to *timestamp with time zone*.
5. *Timestamp* alone should be translated to *timestamp without time zone*, required by SQL, but it is translated by gnatcoll db2ada to *timestamp with time zone*.
6. *Time without time zone*: gnatcoll db2ada produces unknown field type *time without time zone*.
7. *Time with time zone*: gnatcoll db2ada produces unknown field type *time without time zone*.
8. *Time*: gnatcoll db2ada produces unknown field type *time without time zone*.
9. *Timestamp*, *Time* and *Interval* with a precision: gnatcoll db2ada reports unknown field type.
10. Note that index information, even if it is in the Postgresql database, is no included in the generated schema because it is not extracted from the RDBMS.

11. Fields in the generated schema are alphabetically classified, not respecting the order of the fields in the database (assuming this order was defined by the database designer by some reason". It is in gnatcoll-sql-postgres-builder.adb procedure For_Each_Field having the clause " ORDER BY pg_attribute.attname". **Corrected.**
12. Trivial: some fields are generated with upper case initial (Integer, Float, Money, Text, Character), others are generated with lower case initial (smallint, bigint, double precision, real, numeric, timestamp with time zone, date, interval, boolean).
13. If the SQL script contains a foreign relation name, such name is taken as the reverse relation name when generating the schema. **Implemented.**
14. Automatic Reverse Relation when generating the schema from a database: if -noautodata is false and the comment in a FK field in the database is "(some_name) etc. etc." then some_name is automatically taken as reverse relation name. This takes preference over previous point 13. **Implemented.**

3.9 Observations when generating Ada or in the generated Ada code

1. Having compound foreign key is not fully processed by File_IO.Read_Schema in gnatcoll-sql-inspect.adb, producing multiple messages such as
   ```
   Invalid foreign key: some_table.some_field references an invalid table or field
   ```
   Corrected. See also section 3.11.
2. *Timestamp without time zone, Timestamp with time zone* and *Timestamp* are mapped to Ada Time which in GNAT is obtained from the Operating System that already includes the time zone. The generated code does not deal with the time zones. Also note that the SQL ranges are larger than GNAT Time range.
3. In dborm.py, all numeric types (except autoincrement) are now initialized to 'First instead to -1. **Corrected.**
4. In ORM: values of type *Money* are returned by Postgresql as "35,6 €" or "$35,6" for example. That produces a constraint error in programs using the generated ORM. The solution is to clean such value converting it to T_Money. Corrected for $ and € in gnatcoll-sql-exec_private.adb routine Money_Value. **Corrected (only for $ and €).**
5. Views cannot have primary keys, an error should be reported. **Implemented.**
6. In the generated ORM: to get a new object, say for example Ship, the function New_Ship is generated, without parameters, returning a new Ship with all fields initialized to default values . Then every field can be updated using Ship.Set_Some_Field (some value), field by field. It would be convenient a procedure to update all fields in one Ada statement.
7. As in the previous example: There is no routine to set the value of a field that is also a primary key. The function New_Ship may obtain automatically a new primary key only for the case of using autoincrement primary keys, not this case where the primary key is the IMO (International Maritime Organization) ship identification.
   This is an important point. The function New_"object" must include the primary key fields as parameters. dborm.py must be modified (by the way, in python. Why not in Ada?). Until a solution is available, **the generated ORM Ada code can be modified by hand** (all New_"object" have to include as parameters the primary key fields and in the body, they must be set as modified). **Implemented** in a new module in gnatcoll_all2ada but the definitive solution shall go in dborm.py or better, get rid of dborm.py and recode it in Ada (see section 3.12).

8. As in the previous example: gnatcoll-db documentation indicates that it is highly recommended to set a primary key on all tables. Then, it is recommended using integers for the primary keys for efficiency reasons (i.e., autoincrement). This is not an accurate description: it must say that tables with non-autoincrement primary keys cannot be expanded with new rows as the generated SQL insert statements do not include the primary key field. Until a solution is available, **the generated ORM Ada code can be modified by hand** (all procedures Insert_Or_Update have to included a section for handling the primary key and the check for insert or update must be "if Missing_PK OR Mask (number of the primary key)". **Also an important point.** The solution is to partially replace dborm.py (the part that generates ORM) by an Ada package generated the same files but with solving the previous issues. **Implemented.**
9. Views: although views with Pks are marked correctly as an error, views with FKs are accepted in error. **Corrected.**
10. Originally, SQL views were intended to be read-only combination of tables, however some DMBSs support updating views under limiting conditions (by automatically defining triggers to update the involved tables). That is beyond gnatcoll_all2ada, therefore a new option has been defined for gnatcoll_all2ada: -updatable-views, default false. In this case, no routines to updates are generated for views in the ORM code. **Implemented** in the new replacement for dborm.py (see section 3.12). Specifically:
    1. *insert_or_update*: in the spec it will declared with null body.
    2. *internal_delete*: already covered by a test on having PKs.
    3. *key*: already covered by a test on having PKs.
    4. *on_persist*: omitted in spec and body.
    5. *set*: omitted in spec and body.

3.10 Special Section on AutoIncrement Fields

1. *Autoincrement* fields (in Postgresql terminology *smallserial, serial or bigserial*) are generated in the schema correctly to *smallint, integer and bigint*. If some of these fields is primary key, it is also correctly marked. But when extracting the information from the Postgresql database, the autoincrement property is not marked at all in the generated schema. The correction has been to include a new property of field (named Serial) and then set it depending upon the information returned by Postgresql. Obviously the function *is_autoincrement* has been also modified. The python module dborm.py has been modified too, admitting now in the third column the key "PK,SERIAL" for *smallint, integer and bigint.* Note that the key must be exactly "PK,SERIAL", no blanks, etc. **Implemented.**
2. Case of sqlite: on the other hand, *Autoincrement* fields in sqlite are implemented as 64-bit signed integer (see *https://www.sqlite.org/autoinc.html*) and automatically are primary keys (different from Postgresql where autoincrement fields are not automatically primary key). They are generated in the schema as Integer Autoincrement Primary Key (see gnatcoll-sql-sqlite-builder.adb around line 1155), which is correct. However they are mapped later to Ada Integer instead of Ada Long_Long_Integer. Also in gantcoll-sql-inspec.ads it has been changed its Field_Mapping to SQL_Field_Bigint. **Both Corrected.**
3. Case of Postgresql: if the generated schema is modified by hand and a field is set as AUTOINCREMENT, it is always mapped to Ada Integer. But the Postgresql database may have this field as *smallserial, serial* or *bigserial,* so the mapping to Ada should be Short_Integer, Integer and Long_Long_Integer. So don't do that.

4. Case of Postgresql: if the generated schema is modified by hand and a field is set as AUTOINCREMENT, it is considered a Primary Key. **But in Postgresql, Primary Keys and AutoIncrement are ortogonal properties**, there may exist autoincrement fields that are no primary keys. So don't do that.
5. Dborm.py generates a routine "*from_cache*" only for tables with just one PK and integer. This has been modified to contemplate the case of just one PK but now smallint, integer, bigint. Other similar tests in dborm.py have been modified too. **Implemented.**
6. The above modification also impacts in gnatcoll-sql-sessions, in particular the cache part (lines 407 and so on in the spec). As mentioned, even sqlite autoincrement are Pks with 64 bits, so this is an error. This part of code is based on having just one PK and integer. The solution has been to change in gnatcoll-sql-sessions and others the supported key from *Integer* to *Long_Long_Integer*. **Implemented.**
   The impact is:
   a. dborm.pry, routine *"from_cache_hash"*, to generate *Long_Long_Integer* for the parameter. That forces that the generated routine "*From_Cache*" include this conversion (from *Short_Integer*, *Integer* and *Long_Long_Integer* to *Long_Long_Integer*; this last case could be saved).
   b. Gnatcoll-sql-exec_privade.ads: repeat function *Last_Id* now with parameters SQL_Field_Smallint and SQL_Field_Bigint.
   c. Gnatcoll-sql-exec-tasking: same.
   d. Gnatcoll-sql-exec: repeat function *Last_Id* now with parameters SQL_Field_Smallint and SQL_Field_Bigint.
   e. Gnatcoll-sql-sessions: change *Element_Key* to *Long_Long_Integer*, change body of function *Image (Element_Key)* to use a new copy of the function *Image* of gnatcoll-utils valid for *Long_Long_Integer* and in the body of function *Hash* introduce a type cast to Key.Table.
   f. Gnatcoll-sql-exec, function Last_ID, repeat these functions for smallint and bigint.
   g. Gnatcoll-sql-postgres-builder and gnatcoll-sql-builder: there is an overriding function Last_Id in the generic package Postgresql_Cursors; insert then two new overriding functions for Last_Id with smallint and bigint. Also copy and modify the bodies.
7. Note that gnatcoll-db2ada can generate the SQL schema from a previously obtained schema described in the gnatcoll-db documentation. Even if we are not interested in this facility, only commenting: even if the generate schema is modified by hand inserting or modifying an autoincrement field, it is mapped in the generated SQL schema to SERIAL PRIMARY KEY, which may be incorrect as Postgresql supports smallserial, serial and bigserial.

3.11 Supporting Multiple Fks

Even if the documentation contains information on the "FK:" lines for multiple foreign keys (implying tables with multiple PKs), both gnatcoll-sql-inspect.adb and dborm.py fails in some way in these cases.

1. gnatcoll-sql-inspect.adb reads from the SQL database and correctly builds the internal schema. But the final step is to check orphan FKs, test that fails because the use of the function Get_PK for a table with multiple Pks returns No_Fields (see point 6 of section 3.13). The solution is to purge such orphan FKDs (partially done, left to ACT the final deletion). **Corrected.**
2. The reverse relation is not translated from the individual Fks to the multiple "FK:". **Corrected.**

3. dborm.py generates procedures for the reverse relation. In case of multiple Fks, there is a name conflict as the reverse relation name is the same for all Fks. The solution has been to postfix these routines with the individual FK name. **Implemented** in the new Ada package.
4. In some obscure cases it seems that dborm.py assumes tables with just one PK. A comprehensive review is recommended. Or better, replace it. For example, dborm.py still complains on multiple Fks, cannot see the effect. The message "`Error`…" has been changed to "`Warning:`…". In other cases there is a mismatch between parameters (see the generated code for functions of reverse relations in case of multiple Fks). In other cases, dborm.py changes second, etc. PK to the first PK (because it assumes it is the only one PK). It has been also detected that even it changes the data type of some FK.
5. dborm.py generates procedures named *Set_<Pointed_Table_Name>* to modify the FK(s) of a detached object using the values of a pointed detached object or table. Once the values are recorded, it generates also *Self.Set_Modified (n)* to know later (if the modification is committed) which field was modified, using n as the field order. But in the case of multiple Fks, only the last field number in the multiple FK is issued, missing the others. **Corrected.**

3.12 Replacing dborm.py by a new package in Ada

Given that dborm.py is a module written in python, and then maintainable only by python experts, a new Ada module with the same functions and increased with some points as indicated in this report has been included. For the moment on, it is running after dborm.py to facilitate testing and comparisons.

Features of this new Ada module are:

For the specs:

- Generation of *procedure Set_<primary_key>* for non-autoincrement or non-serial primary keys. By contrast, dborm.py emits such functions for serial fields, which is an error.
- Generation of *function New_<Table>(<primary_keys>)* for non-autoincrement or non-serial primary keys.

For the bodies:

- Generation of *procedure Set_<primary_key>* for non-autoincrement or non-serial primary keys. By contrast, dborm.py emits such functions for serial fields, which is an error.
- Generation of *function New_<Table>(<primary_keys>)* for non-autoincrement or non-serial primary keys. Their bodies have a call to the corresponding *Set_<primary_key>* with the value specified.
- The procedures I*nsert_Or_Update* now include testing the masks for all fields, including primary keys. Masks for primary keys are used too to decide between insert or update.
- Generation of bodies for functions *From_Cache*, *Key* and *Insert_or_Update* with the improvements of point 6 of section 3.10.

Differences pending of solution are:

1. The constant Upto_*<Table>*_*<num>* for the last multiple-FK (if there are more than one multiple FK).

2. The constant *Alias_<Table>* as the algorithm could not be understood.

The workarounds are to copy such constants from the file (ORM body) produced by dborm.py. Hope ACT can solve these differences.

Finally, the newly generated Ada files support all new field types and the corrections to other fields.

3.13 Improvements to gnatcoll-sql-inspect

1. There should exist a function returning if a field is Autoincrement or not. **Implemented.**
2. There should be a function returning the reverse relation (if applicable, otherwise return the empty string). **Implemented.**
3. There should exist a boolean function returning is a field is a foreign key (now there exists a function returning the referenced field or no_field). **Implemented.**
4. For_Each_Table has a parameter to select alphabetic or as the order reported by the corresponding schema. However this feature is not available for the procedure For_Each_Field.
5. There should exist a procedure similar to For_Each_FK but for primary keys of a table. **Implemented.**
6. Using <table>.Get_PK is misleading as there is not difference between no PK and several Pks and also if the table inherits of an abstract table and the PK belongs to it, No_Field is returned too in this case. See the alternative solution in the Ada package replacing dborm.py (section 3.12).
7. Using <field>.Id is also misleading. The comment indicates that it is a unique number within a table, however if the table inherits of an abstract table, numbers will repeat. See the alternative solution in the Ada package replacing dborm.py (section 3.12).
8. Gnatcoll-sql-inspect always initializes table.row_name to table.name, even if table.row_name was given the generated schema. Modified in Parse_Table around lines 1393. **Corrected.**
9. There should be routines to insert synthetic Tables and Fields, marked adequately.

3.14 Recommendations about the Documentation

No attempt has been made to modify the documentation. There follows some comments:

1. In general, the documentation should be reviewed by a person not involved in the development of gnatcoll-db, to identify points not totally clear.
2. It is indicated that the generated schema contains the information to generate a database in a RDBMS. Too strong statement as some information is not included (for example, how views are defined, length of varying string, possibly more).
3. The documentation shall indicate that Table definitions shall be separated by a blank line at least (if the schema is modified by hand).
4. The documentation shall indicate that FK statements in the generated schema shall not contain blanks between the foreign table and the optional reverse relation.
5. The documentation separates FKs by comma in the case of “FK:” clause, should be separated by spaces.
6. The database should be in 3NF. Otherwise, name conflicts may appear in the generated code.
7. When modifying the generated schema by hand, avoid to repeat names within a table (2nd and 3rd columns) with fields in the same table.
8. *Float* is seen by Postgresql as *double precision*, so it is necessary to use *Float(24)* in Postgresql to obtain the equivalent Ada Float. However *Float(24)* is unknown in the translation by

gnatcoll_all2ada. Instead of Float, you should use explicit *Double Precision* in Postgresql and use *Real* to obtain the Ada Float type. Indicate that in the documentation.

9. *Numeric* is translated to *Float* by gnatcoll_all2ada . Instead of *Numeric*, you should use some available fixed point type in the extracted schema. Indicate that in the documentation.
10. *Decimal* is translated to *Float* by gnatcoll_all2ada . Instead of *Decimal*, you should use some available fixed point type in the extracted schema. Indicate that in the documentation.

4 Testing

4.1 Objectives

Test objective shall be limited to test those parts that were modified or non-modified parts that can be affected by the modifications.

4.2 Test Methods

Two test methods are used:

1. Comparing the Ada files generated by dborm.py and by the new Ada module, using a visual difference tool (meld in this case).
2. Executing some test cases to check correct access to the database management system.

Tests files for the new Ada package are as follows:

1. One table with all supported types and no PK, accessing to a Postgresql database for reading.
2. A reduced example of a ship/country/skipper database, with PKs, FKs and Views, accessing to a Postgresql database for reading, updating, inserting and deleting.
3. The original example of gnatcoll-db but autoincrement changed to bigint PK,SERIAL
4. Same as test 1 but this time the table has PK integer.
5. Several tables, each one with one PK respectively: smallint, integer, bigint, text, increment, smallserial, serial and bigserial, float, numeric and time.
6. One table with one PK and several serial fields but no PK.
7. One table with two PKs
8. Three tables with PKs each and one FK from the first table to the second one and another FK from the first table to the third one.
9. One table with two Pks, another table with three Pks and one table with one PK and also with two Fks pointing to the first table (that is, a case of multiple Pks and multiple Fks) and three Fks pointing to the second table. The corresponding Postgresql database is also used.
10. Four tables with PK and the last table having Fks each one pointing to one of the previous tables.

4.3 Test Results

All tests generate the Ada files with dborm.py and also the new package generate the same two files (with names post fixed with "_new"). To facilitate comparisons, all Ada files are passed through the gnat pretty printer "`gnatpp --no-compact`" . The results are, apart of cosmetic differences (mainly because different order of code generation) were satisfactory.

Testing with Postgresql databases have been carried out for tests 1, 2 and 9, which include all new features**.** Test cases for listing, filtering, inserting, updating and deleting were included. The results were also satisfactory.

5 Delivery and Installation

The delivery will be upon request to the author of this document in the form of a zip file including the gnatcoll-db directory tree with all modifications, ready to be compiled.

The recommended installation is:

1. Make a local clone of https://github.com/AdaCore/gnatcoll-db.git
2. Unzip the distribution and copy it on top of the cloned directory
3. Review recompile.sh and the several makefile.setup
4. Build it
5. For the examples:
    1. Create the three Postgresql databases using the sql scripts
    2. use the generateXX.sh scripts to obtain the “*.orm” and all Ada generated modules
    3. Compile the examples and execute.

Alternatively the complete package can be obtained from https://github.com/fdesp87/gnatcoll-db which is a fork of the original ACT package.

6 Annexes

6.1 Used Acronyms

ACT	Ada Core Technologies
DDL	Data Definition Language
FK	Foreign Key
IEC	International Electro-technical Commission, Geneva, Switzerland
IEEE	Institute of Electrical and Electronics Engineers, NJ, USA
ISO	International Standards Organization, Geneva, Switzerland
ORM	Object Relational Mapping
PK	Primary Key
RDBMS	Relational Database Management System
SQL	Structured Query Language

- o – o – o -