1 Stack overflow when building the relocatable libraries

Once built and installed the sql subproject (with relocatable, static-pic and static libraries), it is the turn of postgres and sqlite. But in these cases the binder raises the following message:

```
Build Libraries
   [gprlib]       gnatcoll_postgres.lexch
   [bind SAL]     gnatcoll_postgres

raised STORAGE_ERROR : stack overflow or erroneous memory access
gprlib: invocation of /opt/GNAT/2021/bin/gnatbind failed
gprbuild: could not build library for project gnatcoll_postgres
make: *** [Makefile:123: build-relocatable] Error 4
```

Modifying to an exaggerated degree the process stack size produces the same result.
The only way to proceed is to forget relocatable/static-pic libraries and have only the static one.
So change the makefile.setup to ENABLE_SHARED=no.

The environment was Linux Xubuntu 22.04, GNAT Community 2021 and Postgresql 13.6.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

2 Calling gnatcoll_all2ada with no “-omit-schema <the-sql-schema>”

Suppose that the database has a sql schema. If you try gnatcoll_all2ada with no “-omit-schema” then 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.

However, if gnatcoll_all2ada is called with such option, all emitted SQL statements in your program will be rejected because the database sql schema is not in the search path of the database search path.

The solution is to insert in your program a SQL statement indicating the search path:

```
Execute (Connection => DB (Session),
         Query => "set schema '<the sql-schema>';");
```

However that should be solved in the generated code, not in your program.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

3 Loosing information when mapping SQL varchar (n) to Ada unbounded string

Even if this is a sensible design solution, it should be noted that the length information is lost.
Eventually runtime errors may arise.

Gnatcoll-db has the length information of the field. Any way to check at compile time?

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

4 gnatcoll_all2ada has enough information to generate better readable schema.

For example:
gnatcoll_all2ada should have some parameters to generate:

- if there is a table or view description that starts by (<some-name>), the 3rd column for tables should be automatically generated as <some-name>
- if the name of the table or view ends in “s”, the 3rd column should be automatically generated using the name but removing the “s”. However, in some cases the result may not be corrected and should be modified by hand. Think in “currencies”.
- To control producing comments or not.
- To align columns. In addition, column alignment should be global to all tables.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

5 gnatcoll-db should include support for more “financial” types

Money is numeric(16,2) which is normally insufficient for financial applications (Money is intended for such applications, is it no?). Types T_Numeric_24_8 and T_Numeric_8_4 should be supported. For example, Forex (foreign currency market) is expressed with at least 6 decimals, many investment funds are also expressed with 6 decimals, etc.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

6 Missing function “-” for Money

The function "-" (T : T_Money) return T_Money renames SQL_Impl."-"; was missing.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

7 Error in translating Smallint

Smallint is mapped to Integer, it should be mapped to Short_Integer instead.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

8 Type Real is unknown to gnatcoll-db

For example, Postgresql Real type is IEEE 754 Binary32 with a precision of at least 6 decimals, which is equivalent to Ada Float (also IEEE 754 Binary32).

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

9 Double Precision is not supported

It should be mapped to Ada Long_Float (IEEE 754 Binary64).

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

10 Postgresql Interval type has larger range than Ada Duration

Really gnatcoll_all2ada produces a unknown type “interval”, supposedly ISO8601, whose range is larger than Ada Duration. Runtime errors may arise.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

11 Generating the schema: numeric qualifiers not honored

Default negative numbers such as -5.0 are returned by Postgresql as strings instead of numbers. These strings are qualified by tags such as “::numeric” or “::integer”.

This cause an error in gnatcoll_all2ada.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

12 Generating the schema: remove type cast for default values

gnatcoll_all2ada generates type cast for default values, which is redundant.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

13 Generating the schema: constraint error for Money with default values

The constraint error is raised in gnatcoll-sql-inspect.ads. The solution is to move Type_To-SQL to the body.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

14 Generating the schema: wrong mapping of “timestamp without time zone”

As it is mapped to timestamp with time zone.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

15 Generating the schema: wrong mapping of “timestamp”

SQL timestamp is really timestamp without time zone. Instead, it is mapped to timestamp with time zone.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

16 Generating the schema: unknown field “time without time zone”

In this case gnatcoll db2ada reports unknown field type time without time zone.

For more explanations, you can read the report

https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

17 Generating the schema: unknown field “time with time zone”

In this case gnatcoll db2ada reports unknown field type time without time zone.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

18 Generating the schema: unknown field “time”

In this case gnatcoll db2ada reports again unknown field type time without time zone.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

19 Generating the schema: error when mapping “Timestamp”, “Time” and “Interval” with a precision (see Postgresql documentation)

In these cases gnatcoll db2ada reports again unknown field types.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

20 Generating the schema: index information not extracted from the database

Index information is not extracted from the database, therefore no index lines are produced in the schema.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

21 Generating the schema: Database field order is not honored in the generated schema

Given that fields are alphabetically classified by gnatcoll_all2ada.

We must assume that the order of fields in the Database is defined by the Database Designer and such order must not be changed.

It is in gnatcoll-sql-postgres-builder.adb procedure For_Each_Field having the clause “ ORDER BY pg_attribute.attname”. Please comment it out.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

22 Generating the schema: inconsistency in field names (upper-lower case)

Some fields (Integer, Float, Money, Text, Character, perhaps others) are mapped to entities starting with lower case initials (smallint, bigint, double precision, real, numeric, timestamp with time zone, data, interval, boolean).

For more explanations, you can read the report

https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

23 Generating Ada for compound foreign keys

Compound foreign keys are not fully processed by File_IO.Read_Schema in gnatcoll-sql-inspect.adb, emitting multiple messages such as

```
Invalid foreign key: some_table.some_field references an invalid table or field
```

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

24 Generating Ada for “timestamp with/without time zone” and for “timestamp”

They are mapped to Ada Calendar Time, obtained in GNAT from the Operating System, which already included the time zone. The generate code does not deal with time zones. Also note that SQL ranges are larger than GNAT Time range (reported in a previous issue).

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

25 Generating Ada: wrong numeric initialization in dborm.py

All numeric types are initialized to -1, really they should be initialized to ‘First.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

26 Generating Ada: constraint error for values of type Money

These values are returned by Postgresql as “35,6 €” or as “$35.6”, for example, which causes a constraint error in the generated ORM. The currency symbol should be cleaned when converting to T_Money (are least for the most used currencies).

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

27 Generating Ada: Views with primary keys and with foreign keys

Views cannot have primary keys, an error should be reported.

Views with foreign keys must be reported also as errors.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

28 Generating Ada: updating all fields of a table

The generated code contains procedures to update fields of a table, one by one. It is convenient to have a procedure to update all fields at once.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

29 Generating Ada: Missing procedure to define a primary key

Note that not all primary keys are autoincrement, so for tables with primary keys that are not autoincrement, there is not way to define a new row, as it is nor possible to assign a value to the primary key field.

The solution is that the functions New_<object> include parameters for the primary key fields. That implies modifications to dborm.py, which is in python, a surprise as this is Ada generating Ada. In the indicated github path an full Ada solution is included, replacing the python module.

The documentation is also misleading. It is indicated that it is highly recommended to set a primary key on all tables, also recommending using integers for such primary keys for efficiency reasons (i. e. autoincrement). This is not an accurate description: it must say that tables having non-autoincrement primary keys cannot be expanded (using gnatcoll-db) with new rows as the generated Ada code do not include routines to define new rows in these cases.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

30 Generating Ada: updating views

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 must be defined for gnatcoll_all2ada: -updatable-views, for example, default false. In this case, no routines to update views shall be generated in the ORM code.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

31 Autoincrement Fields for Postgresql

Autoincrement fields (in Postgresql terminology *smallserial, serial or bigserial*) are mapped 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.

This situation must be corrected, for example by including a new field property (named Serial) and then set it depending upon the information returned by Postgresql. Obviously the function "*is_autoincrement*" has to be modified. The python module dborm.py has to be modified too, admitting now in the third column the key "PK,SERIAL" for *smallint, integer and bigint.*

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

32 Generating Ada: Autoincrement Fields for sqlite

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 mapped in the schema to 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, this has to be changed having its Field_Mapping changed to SQL_Field_Bigint.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

33 Generating Ada: routine "from_cache"

dborm.py generates a routine "*from_cache*" only for tables with just one PK and integer. This situation must be modified to contemplate the case of just one PK but now smallint, integer, bigint.

Note that gnatcoll-sql-sessions is also affected, in particular the cache part. Even sqlite autoincrement are PKs but with 64 bits, so the actual implementation has this error: change Integer to Long_Long_Integer.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

34 Generating Ada: supporting multiple Foreign Keys (1)

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.

gnatcoll-sql-inspect.adb reads from the SQL database and correctly builds the internal schema. But the final step is to check orphan Fks. This last test fails because the function Get_PK for a table with multiple Pks returns No_Fields.

The solution is to purge such orphan FKDs.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

35 Generating Ada: supporting multiple Foreign Keys (2)

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.

The reverse relation is not translated from the individual Fks to the multiple "FK:"

For more explanations, you can read the report

https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

36 Generating Ada: supporting multiple Foreign Keys (3)

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.

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 should be to postfix these routines with the individual FK name.

This solution has been implemented in the proposed replacement of dborm.py in the new Ada package.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

37 Generating Ada: supporting multiple Foreign Keys (4)

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.

In some obscure cases it seems that dborm.py assumes tables with just one PK. A comprehensive review is recommended. Or better, replace dborm.py by the proposed Ada module.

For example, dborm.py still complains on multiple Fks, cannot see the effect. 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.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

38 Generating Ada: supporting multiple Foreign Keys (5)

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.

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.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

39 Proposed Improvements to gnat-sql-inspect

1. There should exist a function returning if a field is Autoincrement or not.
2. There should be a function returning the reverse relation (if applicable, otherwise return the empty string).
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).
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.
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.
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.
8. Gnatcoll-sql-inspect always initializes table.row_name to table.name, even if table.row_name was given in the generated schema.
9. There should be routines to insert synthetic Tables and Fields, marked adequately.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt

40 Proposed Improvements to the documentation

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, they 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.
11. The documentation should indicate that the generated schema must not be modified by hand at least in the following cases:
    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.
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.

For more explanations, you can read the report
https://github.com/fdesp87/gnatcoll-db/examples/mytests/Report.odt