Gem #129: Базы данных API, безопасные по отношению к типам - Часть 1.

Автор: Emmanuel Briot — AdaCore

Краткое содержание: Для создания запросов к системам управления базами данных (СУБД) традиционно используется язык SQL. Этот язык в основном стандартизирован, несмотря на то, что каждый поставщик предлагает свои расширения и ограничения. Довольно удобно организовывать данные в виде таблицы и полей по реляционной модели. В последнее время выросла популярность так называемых баз данных «noSQL», использующих совершенно другую парадигму для лучшей производительности за счет большего объема ограничений. В данном Gem такие базы данных обсуждаться не будут.

Давайте начнём…

SQL-это традиционный язык, используемый для запросов к системе управления базами данных (СУБД). Этот язык в основном стандартизирован, хотя каждый поставщик предоставляет свои собственные расширения и ограничения. Удобно, когда данные могут быть организованы в таблицы и поля в соответствии с реляционной моделью. Совсем недавно мы стали свидетелями появления так называемых баз данных "noSQL", которые используют совершенно другую парадигму для повышения производительности в обмен на дополнительные ограничения. Мы не будем обсуждать базы данных noSQL в этом Gem.

Несмотря на все свои преимущества, SQL имеет множество ограничений, так как он традиционно используется.

**** Запрос записан как последовательность (в виде, например строки).***

С последовательностью не возможно проверить во время компиляции, корректен ли синтаксис. Также не возможно проверить, допустимы ли таблицы и поля, на которые ссылаются. Кроме того, не возможно иметь проверку типа (гарантирующий сравнение целых чисел с целыми числами, и т.д.).

**** Отправка запроса в СУБД зависит от поставщика.***

Каждый поставщик предоставляет API C, но все они отличаются друг от друга. Концепции в целом схожи: подготовка запроса; отправив его на сервер; передача параметров; получение результатов; и так далее. Существует стандартный API (`ODBC`), который реализуется большинством, но не всеми, поставщиками. Тем не менее, есть снижение производительности по сравнению с использованием нативного API в целом. Настройка также более сложная.

Некоторые поставщики предоставляют решения на основе препроцессора: вы пишете SQL-запрос непосредственно в исходном коде, который затем предварительно обрабатывается перед передачей компилятору. Однако это решение означает, что компилятор не видит именно тот код, который вы написали, что, например, может привести к сообщениям об ошибках, указывающим на неправильную строку в файле.

Коллекция компонентов GNAT (GNAT Components Collection) предлагает два пакета, `GNATCOLL.SQL` и `GNATCOLL.SQL.Exec`, которые обеспечивают решение проблем, выделенных выше. Одним из требований является то, что эти пакеты должны использоваться с существующим кодом, чтобы обеспечить более легкий переход.

Мы начнем с рассмотрения первого ограничения (SQL в виде строк). Пакет `GNATCOLL.SQL` предоставляет Ada API для написания SQL-запросов. Однако, чтобы обеспечить безопасность типов, нам сначала нужно описать схему нашей базы данных. Для этого `GNATColl` использует инструмент

командной строки под названием gnatcoll_db2ada. Если у вас уже есть существующая и работающая база данных, вы можете просто выполнить следующую команду:

```
gnatcoll_db2ada -dbname <name> -dbhost <host> -dbuser <user>
```

Эта команда сгенерирует два пакета Ada (по умолчанию они называются Database и Database_Names). Только первый предназначен для использования непосредственно в ваших приложениях. Он содержит пакеты Ada, соответствующие схеме базы данных, например:

```
package Database is
   type T_Table1 (...) is record
      Field1 : SQL_Field_Integer (...);
      Field2 : SQL_Field_Text (...);
   end record;

   type T_Table2 (...) is record
      Field3 : SQL_Field_Integer (...);
   end T_Table2;

   Table1 : T_Table1 (null);
   Table2 : T_Table2 (null);

   ...
end Database;
```

Фактические дискриминанты не имеют отношения к нашему обсуждению основ GNATCOLL.SQL. Они используются, когда вам нужно иметь несколько ссылок на одну и ту же таблицу (через разные псевдонимы) в одном запросе.

Имена T_Table1, Field1, Field2 и т. д. Являются фактическими именами таблиц и полей в схеме, поэтому должны быть знакомы разработчикам.

Если у вас нет существующей базы данных, можно описать схему в текстовом файле. Этот файл может быть создан изначально из существующей базы данных. Вот пример такого файла. Полная документация для формата доступна в документации GNATColl.

```
| TABLE  | Table1         |          | |
| Field1 | AUTOINCREMENT | PK       | | Documentation for this field
| Field2 | Text          | NOT NULL | | Documentation for field2

| TABLE  | Table2     | | |
| Field3 | FK(Table1) | | | This is a foreign key
```

После того как пакеты Ada сгенерированы, пользователь может писать SQL-запросы с помощью API Ada, предоставленного в GNATCOLL.SQL. Вот небольшой пример:

```
with GNATCOLL.SQL;   use GNATCOLL.SQL;
with Database;       use Database;   --  the generated package

...

declare
   Q : SQL_Query;
begin
   Q := SQL_Select
```

```
      (Fields => Table1.Field2 & Table2.Field3,   --  line 8
       From   => Table1 & Table2,                 --  line 9
       Where  => Table1.Field1 = Table2.Field3);  --  line 10

   --  Converting Q to a string via To_String will now display:
   --     "SELECT table1.field2, table2.field3 FROM table1, table2
   --      WHERE table1.field1 = table2.field3);"
end;
```

Хотя это может показаться более длинным, чем при использовании строки, этот код уже имеет несколько преимуществ:

**** Это чистая Ада.***

Нет необходимости в предварительной обработке, поэтому любое сообщение об ошибке, которое сообщает компилятор, будет указывать на правильную строку кода.

**** Это гарантирует, что SQL синтаксически правильный.***

По своей конструкции API не позволяет вводить синтаксически некорректный запрос (например, опечатка типа «SELCET», пропущенные пробелы или любые другие виды ошибок). API гарантирует, что вы можете указать только таблицу или список таблиц в предложении FROM, и есть аналогичное ограничение для предложения FIELDS.

**** Это гарантирует, что ссылки имеются только на допустимые таблицы и поля.***

Поскольку запрос написан с использованием сгенерированного API, это гарантирует, что если вы измените схему вашей базы данных, сгенерированный пакет базы данных также изменится, и компилятор будет отмечать запросы, которые ссылаются на поля, которые больше не существуют. Когда запрос записывается в виде строки, изменение схемы требует поиска в источнике возможных воздействий, а не использования компилятора для выполнения этой работы от нашего имени.* Проверяет типы во время компиляции.

Если мы изменим строку 10 для сравнения Table1.Field2 и Table2.Field3, компилятор пожалуется, что мы сравниваем целое число и текстовое поле.

Поскольку запросы строятся в виде дерева, которое затем объединяется в строку, GNATCOLL.SQL также может обеспечить автоматическое завершение запросов. Например, мы могли бы опустить строку 9 выше и добавить:

```
Auto_Complete (Q);
```

В этом случае нет никакой выгоды, и мы фактически теряем в удобочитаемости и эффективности. Но эта возможность автозаполнения особенно полезна для выполнения предложений GROUP BY. Например, когда одно из полей, возвращаемых запросом, является результатом функции агрегирования, и вы также возвращаете десять других полей, автозаполнение устраняет необходимость вручную поддерживать список из девяти полей в GROUP BY.

Конечно, существует снижение производительности, подразумеваемое созданием запроса в памяти, а затем его сериализацией в виде строки. Чтобы облегчить это, GNATCOLL.SQL обеспечивает поддержку подготовленных запросов (на стороне клиента или сервера), а также параметризованных запросов. Второй Gem в этой серии более подробно расскажет об этом, а также описывает пакет GNATCOLL.SQL.Exec, который абстрагирует взаимодействие с ядром СУБД в API, не зависящий от поставщика.

Продолжение следует в части 2 ...

Связанный со статьёй текст программы

Файлы примеров Ada Gems распространяются AdaCore и могут быть использованы или изменены для любых целей без ограничений.

Об авторе

Автор: Emmanuel Briot

AdaCore

Эммануэль Брио работает в AdaCore с 1998 года. Он принимал участие в различных проектах, в частности, ориентированных на графические пользовательские интерфейсы, включая GtkAda, GPS, XML / Ada, GnatTracker и нашу внутреннюю CRM. Он получил степень инженера в Национальной школе телекоммуникаций - Брест, Франция (Ecole Nationale des Telecommunications - Brest, France).

Last Updated: 10/13/2017
Posted on: 7/9/2012

Обсуждение…