ReStore: Relational Database Persistency for Smalltalk Objects

John Aspinall

December 2, 2021

Layout and typography based on the sbabook LaTeX class by Damien Pollet.

Contents

Illustrations

CHAPTER 1

Introduction

ReStore is a framework for Dolphin Smalltalk and Pharo which enables objects to be stored in and read from relational databases (SQLite, PostgreSQL, MySQL etc.).

ReStore aims to make relational persistency as simple as possible, creating and maintaining the database structure itself and providing access to stored objects via familiar Smalltalk messages. This allows you to take advantage of the power and flexibility of relational storage with no specialist knowledge beyond the ability to install and configure your chosen database.

CHAPTER 2

Getting Started

To install ReStore in your image follow the instructions on the GitHub project page for your Smalltalk dialect:

- Dolphin Smalltalk - https://github.com/rko281/ReStore
- Pharo - https://github.com/rko281/ReStoreForPharo

The class `SSWReStore` represents a ReStore session/connection; following installation a default singleton instance of `SSWReStore` is created and assigned to the global variable `ReStore`. We will use this global default throughout most of this document (see chapter 8 for information on working with multiple ReStore instances).

2.1 Choosing a Database

ReStore supports several different databases via the SSWSQLDialect class hierarchy. Currently defined SQL Dialects are:

- SQLite
- MySQL / MariaDB
- PostgreSQL
- SQL Server
- Access

Each subclass defines the different behavior, data types, functions etc. supported by a particular database. ReStore automatically selects the appropriate subclass after connecting to your chosen database; this ensures your application code is independent of database choice, enabling you to switch

databases easily if required. For example, for simplicity and speed you may use SQLite during development, then deploy to PostgreSQL for better scalability.

2.2 Configuring ReStore

After choosing and installing your database you must tell ReStore how to connect to it; the method for doing this varies by Smalltalk dialect:

Dolphin Smalltalk

ReStore for Dolphin accesses databases via ODBC. You must first create a Data Source Name (DSN) using the driver for your chosen database via the ODBC control panel. Since Dolphin is a 32-bit application ensure that you use the 32-bit ODBC control panel – you can open this from your Dolphin image by evaluating

```
ReStore openODBC
```

Once the DSN is created you can configure ReStore to use it as follows:

```
ReStore dsn: ‘’MyDataSourceName
```

Pharo

Pharo currently supports SQLite, MySQL and PostgreSQL. You must create the appropriate connection object then assign this to ReStore as follows:

```
"SQLite – see https://github.com/pharo-rdbms/Pharo-SQLite3 for more
    information"
ReStore connection: (SSWSQLite3Connection on: (Smalltalk
    imageDirectory / 'test.db') fullName)
```

```
"PostgreSQL – see https://github.com/svenvc/P3 for more information"
ReStore connection: (SSWP3Connection new url:
    'psql://user:pwd@192.168.1.234:5432/database')
```

```
"MySQL – see https://github.com/pharo-rdbms/Pharo-MySQL for more
    information"
ReStore connection:
(SSWMySQLConnection new
  connectionSpec:
    (MySQLDriverSpec new
      db: 'database'; host: '192.168.1.234'; port: 3306;
      user: 'user'; password: 'pwd';
    yourself);
yourself)
```

2.3 Connecting and Disconnecting

Once you have configured ReStore for your chosen database you may connect to and disconnect from the database as follows:

```
ReStore connect.
ReStore disconnect.
```

CHAPTER 3

Defining the Object Model

The first step in creating a ReStore application is to define your data model – the structure of your model classes. This allows ReStore to automatically create database rows from your objects, and also to create the actual database table in which those rows will exist.

Defining the structure of a class is done with the class method `reStoreDefinition`. This method should list the name of each persistent instance variable in the class, and define the type of object held in that instance variable. The `'type'` of object will be a class, a parameterized class, or a collection. These different types are highlighted in the following example for a hypothetical CustomerOrder class:

```
reStoreDefinition

  ^super reStoreDefinition
    define: #orderDate as: Date;  "Class"
    define: #customer as: Customer; "Class"
    define: #items as: (OrderedCollection of: CustomerOrderItem);
    "Collection"
    define: #totalPrice as: (ScaledDecimal withPrecision: 8 scale:
    2);  "Parameterized Class"
    yourself
```

3.1 Simple Classes

In the simplest case, just the class of object held is needed. Supported classes are:

- Integer
- Float
- Boolean
- String
- Date
- Time
- DateAndTime

Example for a hypothetical `Person` class:

```
define: #surname as: String;
define: #dateOfBirth as: Date;
define: #salary as: Float;
define: #isMarried as: Boolean;
```

Additionally, any other class defining a `reStoreDefinition` method may be used. This allows your classes to reference each other or even themselves:

```
define: #gender as: Gender;
define: #address as: Address;
define: #spouse as: Person;
```

3.2 Parameterized Classes

A parameterized class defines not only the class of an object but also additional information that may be required in relation to the class.

String

In Smalltalk an instance of String can be any size, with (to all intents) no upper bound. Within relational databases, however, there are usually three different types of Strings:

1. Fixed sized - usually referred to as a CHAR
2. Variable sized with some upper limit on the number of characters - this is usually referred to as a VARCHAR
3. An unbound, variable sized String - names vary; LONGTEXT, TEXT, MEMO etc.

For these reasons ReStore allows you to parameterize a String definition to enable the best choice of database type to be made. For Strings of a known, fixed number of characters (e.g. a postal/zip code, or a product code), you can specify a CHAR-type String using the String class method fixedSize:

```
define: #productCode as: (String fixedSize: 8);
```

For a variable sized String with a known maximum number of characters (VARCHAR) the method maxSize: is used:

```
define: #surname as: (String maxSize: 100);
```

Finally, if you just specify String (i.e. unparameterized) then a default value will be used as the maximum size of that String. This value will vary from database to database, but the net effect is usually to cause a LONGTEXT-type String to be used, although some databases may use an intermediate type with a large upper limit (e.g. MEDIUMTEXT). Example:

```
define: #notes as: String;
```

ByteArray

ReStore offers support for storing ByteArrays in a BLOB-type database column:

```
define: #imageData as: ByteArray;
```

Similar to with Strings, you may optionally specify a maximum size for the ByteArray – this will help ReStore choose the most appropriate BLOB type where the database offers multiple types with different (or no) maximum size:

```
define: #thumbnailImageData as: (ByteArray maxSize: 8192);
```

ScaledDecimal

Within Smalltalk an instance of ScaledDecimal has a scale - this defines the number of digits after the decimal point. In ReStore, when defining an instance variable as a ScaledDecimal as a minimum you must give the scale of that ScaledDecimal:

```
define: #totalPrice as: (ScaledDecimal withScale: 2);
```

Most relational databases support a type similar to ScaledDecimal (NUMERIC, DECIMAL etc.) but in addition to scale there is usually also precision - the total number of digits that may be held, including the scale. If you specify just a scale (as in the above example) a default precision of 15 will be used. Alternatively, you may specify the precision yourself:

```
define: #totalPrice as: (ScaledDecimal withPrecision: 8 scale: 2);
```

3.3 Unique IDs

Within ReStore every persistent object is automatically allocated an auto-incremented integer ID, unique to itself within its class. This happens completely transparently – you do not need to define this or store it within an

instance variable of your class. However there may be times where you wish to access this unique ID from your application code, for example to use as a customer or order reference.

Where this is the case you can declare the corresponding instance variable in your class as follows:

```
defineAsID: #customerNo;
```

This defines the instance variable customerNo as an Integer that holds the unique ID of the object. You do not need to instantiate this value yourself – ReStore will automatically allocate the next available ID when the object is persisted. Should you wish to allocate the unique ID yourself however, you can simply assign it prior to storing the object and ReStore will use the assigned value instead. In this latter case it is up to your application code to ensure the ID remains unique.

CHAPTER 4

Storing Objects – Transactions

CHAPTER 5

Storing Objects Manually

CHAPTER 6

Query Introduction

Bibliography