SqueakSave
An Automatic Object-Relational Mapping Framework

Thomas Kowark
Robert Hirschfeld
Michael Haupt

Software Architecture Group
Hasso-Plattner-Institut Potsdam
www.hpi.uni-potsdam.de/swa
Outline

• motivation
• basic usage
• framework architecture
• performance
• summary & outlook
Available Persistence Approaches

- image storing
- object databases
- (object-)relational persistence
SqueakSave – Project Goals

- automatic mapping deduction
- simplistic API
- seamless integration into existing applications
Guiding Example

User
- rname : string
- sword : string

Admin

Author

Blog
- title : string
- text : string

BlogPost
- title : string
- text : string

Comment
- author : string
- title : string
- text : string

+blogPosts 0..* +administeredBlogs 0..*
API – Configuration

• configuration based on naming conventions

SqsConfig subclass: #BlogExampleSqsConfig
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: ''
  category: 'BlogExample'

BlogExampleSqsConfig
class>>#connectionSpecification
  ^ SqsMySQLConnectionSpecification
     user: 'admin'
     password: 'password'
     database: 'blog_example_db'
API – Basic Operations

```smalltalk
author := Author new
    password: 'password';
    username: 'testuser';
    email: 'user@example.org'.

author blog: (Blog new title: 'My Blog').

author save.
...

author destroy.
```
API – Queries

(SqsSearch for: User) detect: [:aUser | aUser username = 'testuser']

(SqsSearch for: Author) select: [:anAuthor | anAuthor blog blogPosts size > 10 ]

(SqsSearch for: Blog) anySatisfy: [:aBlog | aBlog blogPosts noneSatisfy: [:aBlogPost | aBlogPost comments isEmpty ] ]

(SqsSearch for: Blog) findByTitle: 'testblog'

(SqsSearch for: Comment) findByAuthor: 'author' andTitle: 'comment'.
SqueakSave – Architecture

The mapping of the data types is implemented within class side methods that are named `sqsType`. For all classes that are trivially mappable, this method has been implemented and returns a SqueakSave internal string representation of the according SQL type. If the columns have to be created, those internal representations are translated by the `SqsDatabaseAdapter` classes into the specific values that are required by the current database servers' SQL implementation. Types with variable lengths, like strings, are additionally enriched with the information about the current length of the respective object. Hence, a string of length 0 will not only be mapped to TEXT or VARCHAR, but VARCHAR(0).

Non-trivial attribute types are mapped by a foreign key reference to the corresponding entry in the table that represents the class of the respective object. The reference will always point to the table of the base class, i.e., the first class in the inheritance chain below Object or a class that is marked like depicted in section ... which is especially important in class table inheritance structures. They are created in such a way that a separate table for each subclass is created and only contains the attributes that are defined within this class. Therefore, a foreign key constraint pointing to only such a sub table would prevent the possibility to reference objects of super or subclasses.
The mapping of the data types is implemented within class side methods that are named `sqsType`. For all classes that are trivially mappable, this method has been implemented and returns a SqueakSave internal string representation of the according SQL type. If the columns have to be created, those internal representations are translated by the `SqsDatabaseAdapter` classes into the specific values that are required by the current database servers' SQL implementation.

Types with variable lengths, like strings, are additionally enriched with the information about the current length of the respective object. Hence, a string of length 0 will not only be mapped to TEXT or VARCHAR, but VARCHAR("0"), etc.

Non-trivial attribute types are mapped by a foreign key reference to the corresponding entry in the table that represents the class of the respective object. The reference will always point to the table of the base class, i.e., the first class in the inheritance chain below Object or a class that is marked like depicted in section ..., which is especially important in class table inheritance structures. They are created in such a way that a separate table for each subclass is created and only contains the attributes that are defined within this class. Therefore, a foreign key constraint pointing to only such a sub table would prevent the possibility to reference objects of super or subclasses.
SqueakSave – Architecture

Creation or fetching of unique SqsStorage wrapper instance
Creation or update of mapping descriptions
SqueakSave – Architecture

Calculation of changes to the relational database schema
SqueakSave – Architecture

The mapping of the data types is implemented within class side methods that are named `sqsType` for all classes that are trivially mappable, this method has been implemented and returns a SqueakSave internal string representation of the according SQL type. If the columns have to be created, those internal representations are translated by the SqsDatabaseAdapter classes into the specific values that are required by the current database servers' SQL implementation. Types with variable lengths, like strings, are additionally enriched with the information about the current length of the respective object. Hence, a string of length 0 will not only be mapped to TEXT or VARCHAR, but VARCHAR_0.

Non-trivial attribute types are mapped by a foreign key reference to the corresponding entry in the table that represents the class of the respective object. The reference will always point to the table of the base class, i.e., the first class in the inheritance chain below Object or a class that is marked like depicted in section , which is especially important in class table inheritance structures. They are created in such a way, that a separate table for each subclass is created and only contains the attributes that are defined within this class. Therefore, a foreign key constraint pointing to only such a sub table would prevent the possibility to reference objects of super or subclasses.

Connection pooling
SqueakSave: An Automatic Object-Relational Mapping Framework

SqueakSave – Architecture

The mapping of the data types is implemented within class side methods that are named sqsType. For all classes that are trivially mappable, this method has been implemented and returns a SqueakSave internal string representation of the according SQL type. If the columns have to be created, those internal representations are translated by the SqsDatabaseAdapter classes into the specific values that are required by the current database servers' SQL implementation. Types with variable lengths, like strings, are additionally enriched with the information about the current length of the respective object. Hence, a string of length 0 will not only be mapped to TEXT or VARCHAR, but VARCHAR-0.

Non-trivial attribute types are mapped by a foreign key reference to the corresponding entry in the table that represents the class of the respective object. The reference will always point to the table of the base class, i.e., the first class in the inheritance chain below Object or a class that is marked like depicted in section, which is especially important in class table inheritance structures. They are created in such a way, that a separate table for each subclass is created and only contains the attributes that are defined within this class. Therefore, a foreign key constraint pointing to only such a sub table would prevent the possibility to reference objects of super or subclasses.

Schema update and object insertion or update
Query Analysis

- SQL statement generation through block execution with placeholder objects
- One placeholder class per ‘simple type’, SqsQueryObject and SqsQueryCollection for complex cases

(SqsSearch for: User) detect: [:aUser | aUser username = 'testuser']
Query Analysis

- SQL statement generation through block execution with placeholder objects
- one placeholder class per ‘simple type’, SqsQueryObject and SqsQueryCollection for complex cases

```
(SqsSearch for: User) detect: [:aUser | aUser username = 'testuser']
```

```
queryObject := SqsQueryObject new
depictedClass: User.
result := aBlock value: queryObject.
```
Query Analysis

- SQL statement generation through block execution with placeholder objects
- One placeholder class per ‘simple type’, SqsQueryObject and SqsQueryCollection for complex cases

(SqsSearch for: User) detect: [:aUser | aUser username = 'testuser']

The query object does not know what #username does, but generates the SQL to scope to the respective column.
Query Analysis

• SQL statement generation through block execution with placeholder objects
• one placeholder class per ‘simple type’, SqsQueryObject and SqsQueryCollection for complex cases

(SqsSearch for: User) detect: [:aUser | aUser username = 'testuser']

WHERE users.username
Query Analysis

- SQL statement generation through block execution with placeholder objects
- one placeholder class per ‘simple type’, SqsQueryObject and SqsQueryCollection for complex cases

```scheme
(SqsSearch for: User) detect: [:aUser | aUser username = 'testuser']
```

The result of the first call is an SqsQueryString. It knows how to map the #= to SQL properly.
Query Analysis

• SQL statement generation through block execution with placeholder objects
• one placeholder class per ‘simple type’, SqsQueryObject and SqsQueryCollection for complex cases

(SqsSearch for: User) detect: [:aUser | aUser username = 'testuser']

WHERE users.username =
Query Analysis

- SQL statement generation through block execution with placeholder objects
- one placeholder class per ‘simple type’, $\text{SqsQueryObject}$ and $\text{SqsQueryCollection}$ for complex cases

$\text{(SqsSearch for: User) detect: [:aUser | aUser username = 'testuser']}$

$\text{WHERE users.username = 'testuser'}$
Evaluation

- evaluation based on OO7 benchmark
  - CAD application data structure
  - complex object model with many cyclic dependencies
- set of queries with increasing complexity
- number of traversals of an object graph
- comparison with GLORP
Evaluation – Query Performance

• approx. 20% slower than GLORP
• two exceptions
  – caching mechanism (10x slower)

```small
(SqsSearch for: SqsAtomicPart) detect: [:ap | ap oid = id].
```

– query creation with joins (1/3x faster)

```small
(SqsSearch for: SqsBaseAssembly) select: [:ba | ba unsharedParts anySatisfy: [:part | part document = id ]].
```
Evaluation – Traversal Performance

- missing eager loading (n+1 queries problem)
- minimal intrusion into object models (only collection proxies)
Summary and Outlook

• simple usage & setup
  – integration into existing applications almost seamless
• automatic deduction of database structures
Summary and Outlook

• simple usage & setup
  – integration into existing applications almost seamless
• automatic deduction of database structures

• possible extensions
  – SqueakDBX usage
  – eager loading
  – performance optimizations