Advanced O/R Mapping with Glorp

Alan Knight (knight@acm.org)
Cincom Systems of Canada

Metaphor/Motivating Example

- Ruby on Rails/ActiveRecord
  - Reading Database Schema
    - » Some interesting mapping issues
  - Creating Mappings Dynamically
    - » Conventions/Grammar
  - APIs
    - » Querying
    - » Globals, Singletons, Transactions
  - Schema Evolution
  - Web Integration

cincom smalltalk

Ruby on Rails

- “Opinionated software”
- Rails
  - Go really fast, but only in one direction
- Reaction against J2EE/.NET
- “Greenfield” projects
- Ruby-Based
  - Smalltalk-like, scripting language
  - Some very Smalltalkish “tricks” in rails
- ActiveRecord pattern for persistence

cincom smalltalk

Architecture: Glorp and ActiveRecord

- Metadata vs. convention-driven
- Glorp: Explicit metadata
  - Tables
  - Classes
  - Descriptors/Mappings
- ActiveRecord
  - Strict naming conventions
  - Aware of language forms
  - Hints at the class level
  - Code Generation (mostly for web)

cincom smalltalk

Brokers

- Glorp
  - Single broker (session)
    - » Responsible for object identity
    - » Manages automatic writes
  - Multiple clients use multiple sessions
  - Independent of other frameworks
- ActiveRecord
  - Classes as brokers
    - » No object identity
    - » Single global session
  - Explicit writes
  - Tightly integrated with web framework

cincom smalltalk

Domain Model

- Glorp
  - No metadata in domain classes
  - Premium on flexibility, ability to optimize
  - Expect existing classes, schema
- ActiveRecord
  - Premium on simplicity
  - Minimal metadata, but on domain classes
  - May not even be a domain model
    - » Use ActiveRecord directly
    - » Instance variables as properties

cincom smalltalk

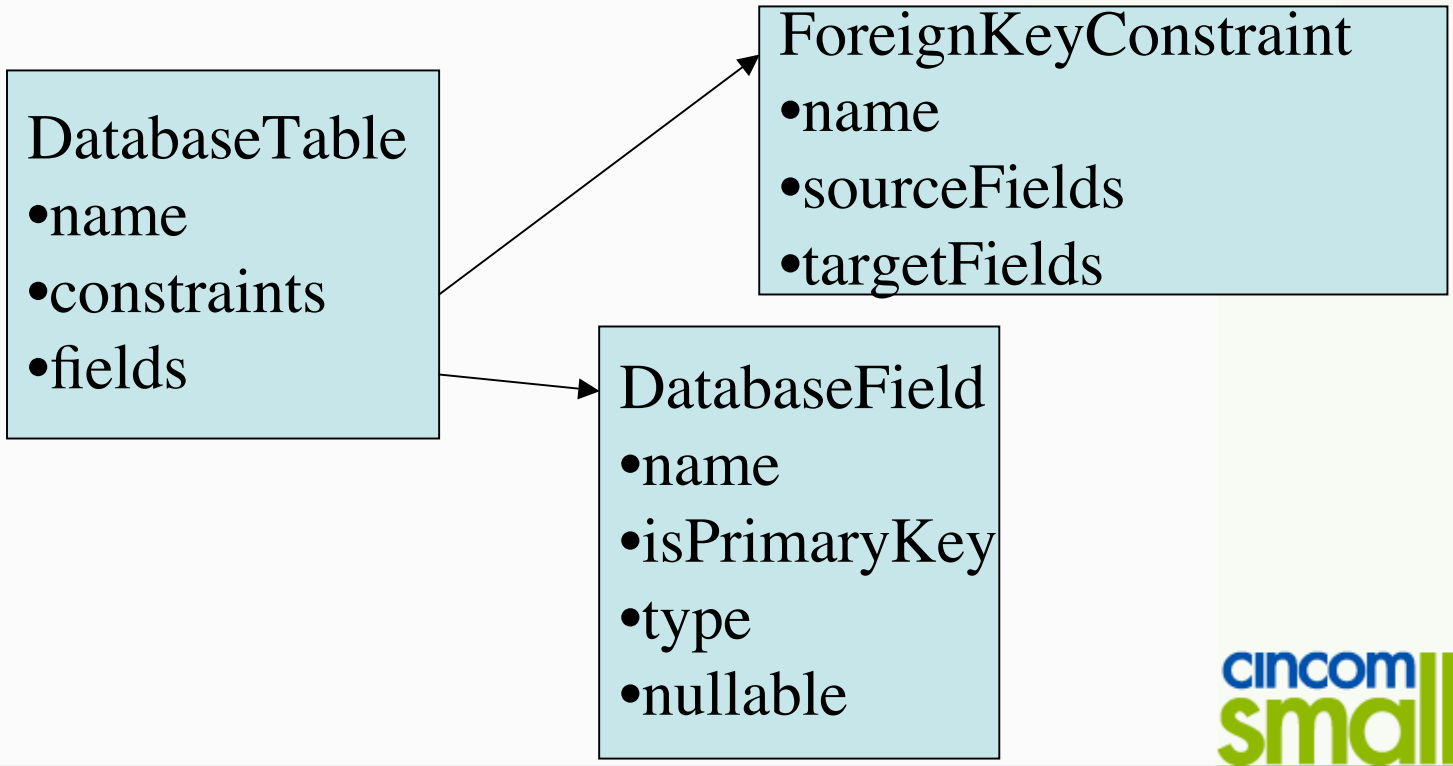
Goal

- Can we provide some of the benefits, but without losing our advantages
- “Hyperactive Records”
  - Automatic persistence
  - Convention-driven
  - But be less restrictive
  - Use a bit more information (constraints)
  - Allow a graceful transition

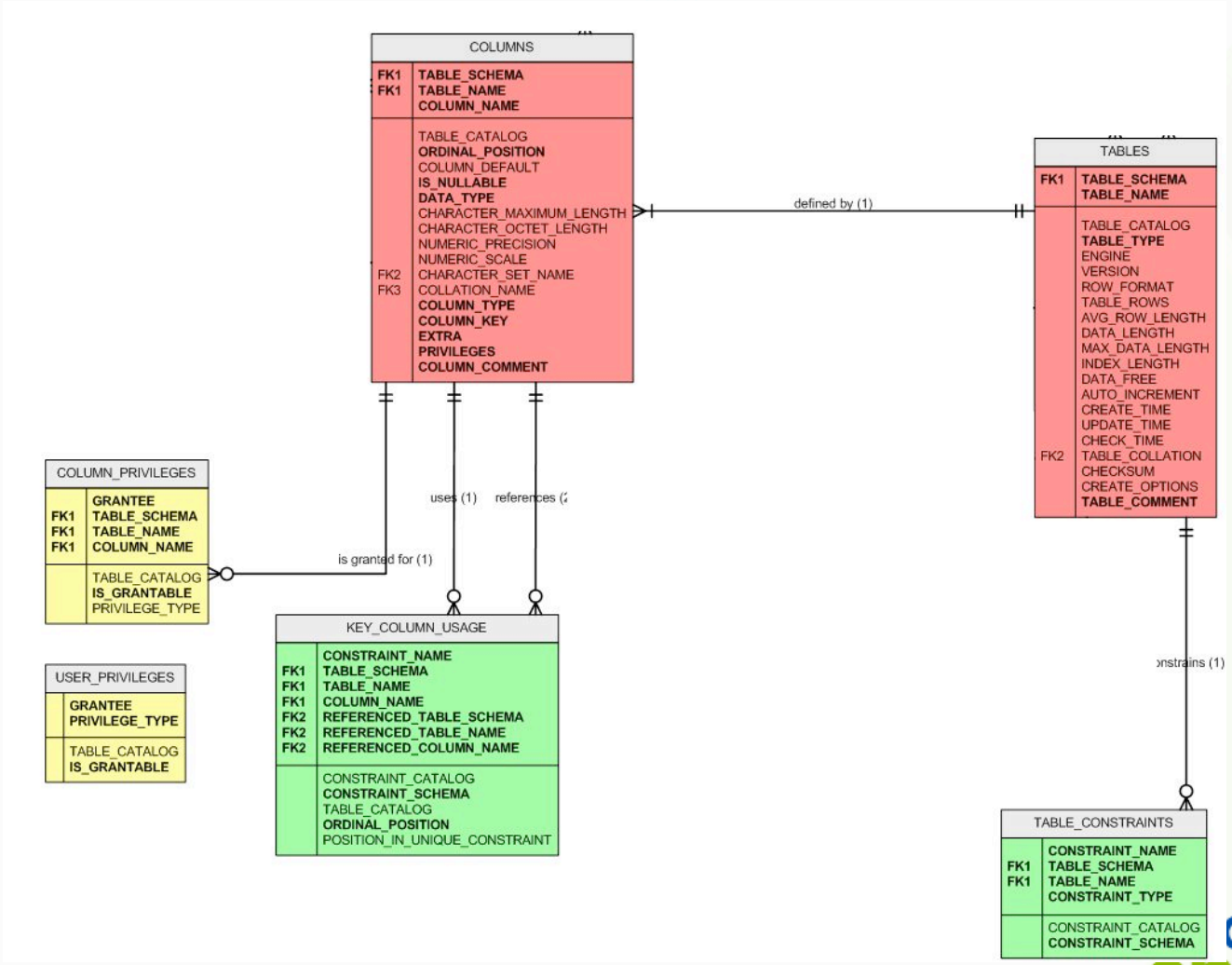
cincom smalltalk

Issue: Reading Schema

- Before we can automate, we need to read the database schema.
- A nicely recursive problem



INFORMATION_SCHEMA



Mapping DatabaseField

DatabaseField
- name
- *isPrimaryKey*
- type
- nullable

COLUMNS
- TABLE_NAME
- COLUMN_NAME
- DATA_TYPE
- IS_NULLABLE

```
table := self tableNamed: 'columns'.
(aDescriptor newMapping: DirectMapping)
    from: 'name'
    to: (table fieldNamed: 'column_name').
```

Mapping #isPrimaryKey

- ST: a boolean value
- DB: primary key constraints are entities
- Columns used in a constraint are listed in key_column_usage
- For a field, do any primary key constraints exist that make use of it
- Mapping a two-level join to a boolean

Mapping #isPrimaryKey

```
(aDescriptor newMapping: DirectMapping)
    from: #isPrimaryKey
    to: [:each |
        each primaryKeyConstraints notEmpty].
```

- Direct mapping, but to an expression
  - “each” is the field we’re referring to
  - primaryKeyConstraints is another relationship
  - notEmpty is a subselect operation

Subselects

- In queries, several “collection” operations that turn into different kinds of subselects
- isEmpty/notEmpty
- select:
- anySatisfy:/noneSatisfy:
- sqlCount (also aggregation)

```
read: Customer
  where: [:each |
    (each orders select: [:order |
      order amount > 1000])
        sqlCount > 5].
```

cincom smalltalk

Reading Schema Summary

- sourceFields and targetFields worse
- Information_schema variations, limits
- Works for Oracle, Postgresql, MySQL
- No changes at all to the domain model
  - But easier because read-only
  - Several pseudoVariables
- Good motivation for automatic mapping

Back to ActiveRecord

- Glorp metadata
  - defined in DescriptorSystem
  - Methods for tables, classes, mapping
  - E.g. #descriptorForCustomer:
  - Lists allTables, allClasses

cincom smalltalk

ActiveRecord DescriptorSystem

- All subclasses of ActiveRecord
- Read allTables from the database
  - For each class name, find table name
  - Find link tables from constraints or hints
- For each inst var/field name, figure out the mapping
  - Naming convention
  - Database constraints

cincom smalltalk

Aside: Inflector

- Ruby on Rails class
- Knows basic grammar forms (English)
- Knows class/inst var/field/table naming and capitalization
  - Person class -> PEOPLE table
  - OVERDUE_ORDER_ID -> overdueOrder
- Big ball of regular expressions

cincom smalltalk

Aside: Hints

- Ruby on Rails uses class data to tell it how to create relationships that are ambiguous
- hasMany, hasAndBelongsToMany
- tableNames (added)

cincom smalltalk

Aside: Class Generation

- Generate a package
- Class for each database table
  - Filtered
- Empty descriptor system with root class

cincom smalltalk

Incremental Setup

- We want to do as little work as necessary
- How to “tweak” an automatically generated mapping
- #mappingNamed:do:

```
self mappingNamed: #bankCode do:
  [:mapping | mapping type: Integer].
```

cincom smalltalk

Rails Migrations

- Define tables in Ruby code
  - Multiple versions, ordered by naming convention
  - First version full
  - Subsequent versions define how to upgrade and downgrade
- In the database, store a version number for the schema
- Run the instructions in sequence

cincom smalltalk

Smalltalk Migrations

- We have full metadata definition of tables
- Keep multiple classes
  - Subclasses?
- Modify the schema to conform to the newest
- Prototype level
  - No upgrading instructions, can lose data
  - Jumps directly from one to the other

cincom smalltalk

Web Integration

- Equivalent to automatic web forms
- MagritteGlorp (Ramon Leon)
  - Extend Magritte with additional information about relationship types
  - Generate Glorp descriptors based on Magritte metadata

Web Integration

- GlorpActiveRecordMagritteSupport
- Magritte metadata based on Glorp
  - Assume Magritte editor based on data type
- Glorp metadata based on database

cincom smalltalk

References

- GLORP
  - http://www.glorp.org
  - http://glorp.sourceforge.net
- Ruby on Rails
  - http://www.rubyonrails.org/
  - Lots of other links

cincom smalltalk

The End

cincom smalltalk

Subselects

- In SQL terms, a nested query
- Many different uses
  - tend to make the brain hurt
- Glorp provides various shortcuts for specific Smalltalk semantics, plus a general mechanism
  - sometimes also make the brain hurt
  - still settling on semantics, naming

cincom smalltalk

Subselect Example

- e.g.

```
… where: [:each | each members
    anySatisfy: [:eachMember | eachMember
    name like: 'Alan%']].
```

```
SELECT <project fields>
  FROM PROJECT t1
  WHERE EXISTS (
    SELECT <whatever> FROM MEMBER s1t1 WHERE
      s1t1.proj_id = t1.id)
```

Aggregating

- Two forms of aggregating
- At the query level
  - aQuery retrieve: [:each | each value sum]
  - Puts an aggregate into the fields of the SQL
  - `SELECT ... SUM(t1.value)`
- Within a where clause
  - where: [:each | (each value sqlSum) > 10]
  - Creates a subselect of that aggregate
  - `SELECT ... WHERE (SELECT SUM(s1t1.value) FROM ... WHERE ...) > 10`
- min, max, average, count, etc.

Still More Aggregating

- Also within a where clause

```
expression count: [:x | x attribute]
```

- or more generally

```
expression
  count: [:x | x attribute]
  where: [:x | x something = 5].
```

- More awkward than

```
expression sqlCount
```

- Not really more powerful

General Aggregations

- General facility

```
read: GlorpCustomer
  where: [:each | each
   (each
      aggregate: each accounts
      as: #countStar
      where: [:acct | acct price > 1000]])
    = 1].
```

- Really awkward
- More general
  - Only requires the underlying function to exist

Select:

- count:where: suggests a more Smalltalk-like form

```
where: [:each |
   (each users select: [:eachUser |
      eachUser name like: 'A%])
         sqlCount > 3].
```

- Or we could apply other operations e.g. anySatisfy: to the filtered collection.

cincom smalltalk

Fully General Subselects

- A subselect is represented by a query.

```
aCustomer accounts
   anySatisfyExists: [:eachAccount |
      eachAccount in:
         (Query
            read: GlorpBankAccount
            where: [:acct |
               acct balance < 100])]].
```

- Very general, but awkward
- Often putting queries into block temps, setting retrieve: clauses, etc.

Correlated Subselects

- Are the internal selects effectively constants, or do they refer back to things in outer queries
- Slower in database, but more powerful

```
read: StorePackage where: [:each |
   | q |
   q := Query read: StorePackage
      where: [:eachPkg |
         eachPkg name = each name].
   q retrieve: [:x | x primaryKey max].
   each username = 'aknight'  & (each primaryKey
   = q)].
```

OK, No More Subselects

- Yes, these are complicated
- Sometimes you need them
- The tests are a good resource for code fragments to copy from
- Or just experiment until you get SQL (and results) you want

cincom smalltalk