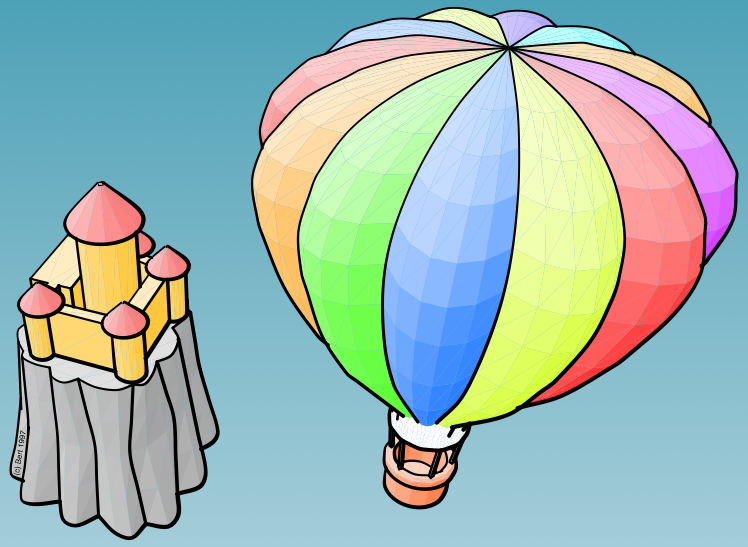


Smalltalk and Databases

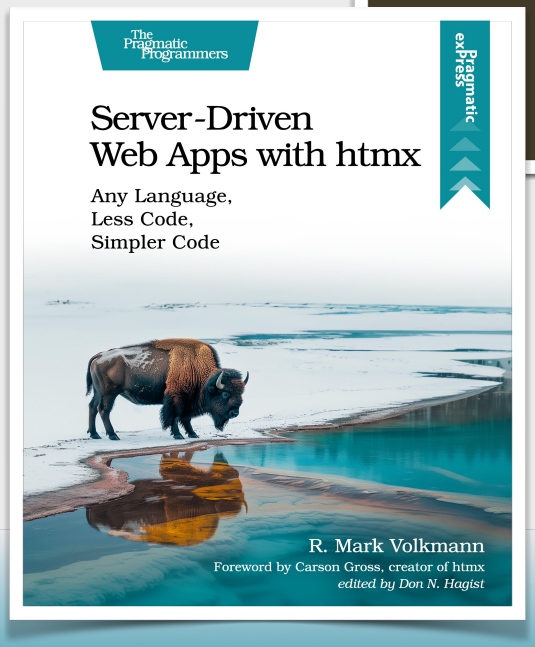
R. Mark Volkmann
Object Computing, Inc.
https://objectcomputing.com
mark@objectcomputing.com
@mark_volkmann

OBJECT COMPUTING
YOUR OUTCOMES ENGINEERED

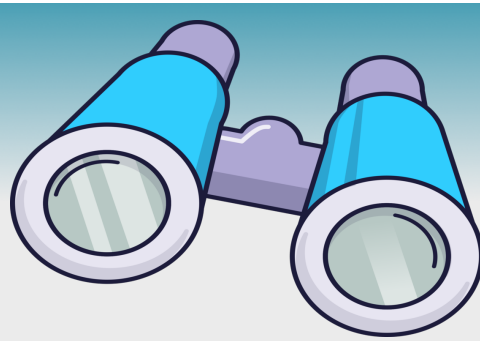
Slides at **https://github.com/mvolkmann/talks/**

About Me

- Partner and Distinguished Software Engineer at Object Computing, Inc. in St. Louis, Missouri USA
- 44 years of professional software development experience
- Writer and speaker
- Blog at https://mvolkmann.github.io/blog/
- Author of Manning book “Svelte ... in Action”
- Author of Pragmatic Bookshelf book “Server-Driven Web Apps with htmx”



ODBC Overview



- Acronym for Open DataBase Connectivity
- Defines an API for interacting with databases
  - primarily used with relational databases
- Originally created by Microsoft and Simba Technologies
  - version 1.0 was released in 1992
  - version 4.0 was released in 2016 (latest)
- Supported by a large number of databases
- Somewhat slower than directly accessing a specific kind of database

Installing ODBC

- In macOS
  - install Homebrew
  - open a Terminal
  - enter **`brew install unixodbc`**
- This installs
  - **`libodbc`** library
  - **`isql`** command
  - **`odbcinst`** command

Apologies for Mac-only instructions!

I need to get access to
Linux and Windows computers
to verify their installation steps.

Installing DB-specific Drivers

- **For Postgres** in macOS
  - in Terminal, enter **`brew install psqlodbc`**
  - creates files **`psqlodbcw.so`** and **`psqlodbca.so`** in **`/opt/homebrew/lib`**
  - "w" stands for "wide", meaning it supports Unicode characters
  - "a" stands for "ASCII", meaning it only supports 8-bit characters
- **For MySQL** in macOS
  - in Terminal, enter **`brew install mariadb-connector-odbc`**
  - creates file **`/opt/homebrew/lib/mariadb/libmaodbc.dylib`**
- **For SQLite** in macOS
  - in Terminal, enter **`brew install sqliteodbc`**
  - creates file **`/opt/homebrew/lib/libsqlite3odbc.so`**

compatible with MySQL

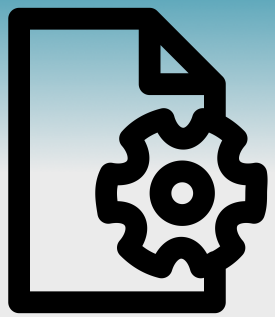
I was unable to get the official MySQL ODBC driver from https://dev.mysql.com/downloads/connector/odbc/ to work.

WARNING: Driver libraries may be installed in **`/usr/local/lib`** instead of **`/opt/homebrew`**.

Defining DSNs

- Acronym for Data Source Name
- DSNs and the drivers they use are configured in text files
- To determine files that must be created
  - in terminal, enter **`odbcinst -j`** `-j` prints config info
  - see “DRIVERS” in output
    - mine is **`/opt/homebrew/etc/odbcinst.ini`**
  - see “USER DATA SOURCES” in output
    - mine is **`.odbc.ini`** in my home directory
- Create those files with content similar to next slide

Configuration Files



/opt/homebrew/etc/odbcinst.ini — maps driver names to library files

```
[MySQL]
Description = MySQL ODBC Driver
Driver = /opt/homebrew/lib/mariadb/libmaodbc.dylib

[PostgreSQL]
Description = PostgreSQL ODBC Driver
Driver = /opt/homebrew/lib/psqlodbcw.so

[SQLite]
Description = SQLite ODBC Driver
Driver = /opt/homebrew/lib/libsqlite3odbc.so
```

WARNING: Driver libraries may be installed in **/usr/local/lib** instead of **/opt/homebrew**

~/.odbc.ini — maps data source names to drivers and databases

```
[MySQLTodosDSN]
Description = MySQL database for a Todo app
Driver = MySQL
Database = todos
User = root

[PostgresTodosDSN]
Description = PostgreSQL database for a Todo app
Driver = PostgreSQL
Database = todos

[SQLiteTodosDSN]
Description = SQLite database for a Todo app
Driver = SQLite
Database = /Users/volkmannm/todos.db
```

alternatively, can set to driver path

modify for location of your database

Listing Data Sources

- To list all configured DSNs, enter **`odbcinst -q -s`**
- To list all configured ODBC drivers, enter **`odbcinst -q -d`**
- To view details about a specific data source, enter **`odbcinst -q -s -n {dsn}`**

`-q` for query
`-s` for source (DSN)
`-d` for driver
`-n` for name

Creating Postgres Database

- Install Postgres: **`brew install postgresql@14`**
- Create required files: **`initdb`**

  directory defaults to **`/opt/homebrew/var/postgres`**
- Start database server: **`brew services start postgresql@14`**
- Create database: **`createdb todos`**
- Enter interactive mode: **`psql -d todos`**
- Create todo table
  - ```
    create table todos (
      id serial primary key,
      description text unique,
      completed boolean
    );
    ```
- Exit by entering **`\q`**
    ```

Creating MySQL Database

- Install MySQL commands: **`brew install mysql`**
- Start database server: **`brew services start mysql`**
- Enter interactive mode: **`/opt/homebrew/bin/mysql -u root`**

may be installed in **`/usr/local/bin`**

- Create todo table
  - ```
    create table todos (
      id int auto_increment primary key,
      description varchar(100) not null,
      completed boolean,
      unique (description)
    );
    ```

must use **`varchar`** instead of **`text`** to add unique constraint

- Exit by entering **`exit`**
    ```

Creating SQLite Database

- Install **`sqlite3`** command: **`brew install sqlite`**
- cd to directory where database will be created
- Create database and start interactive mode: **`sqlite3 todos.db`**
- Create todo table
  - ```
    create table todos (
      id integer primary key autoincrement,
      description string unique,
      completed numeric
    );
    ```
- Exit by pressing ctrl-d
    ```

Interactive Mode

- To interactively access a data source, enter **`isql {dsn}`**
  - to list existing tables, enter **`help`**
  - to see the schema for a specific table (column names and their types), enter **`help table-name`**
  - enter SQL statements like **`select * from todos;`**
  - to exit, enter **`quit`** or press ctrl-d

ODBC from Smalltalk

- In a Workspace, evaluate **`Feature require: 'ODBC'`**
- Evaluate code like the following

Only tested in Cuis Smalltalk ...
Will similar code work
in Squeak and Pharo?

```
| connection resultSet sqlString |

connection := ODBCConnection dsn: #SQLiteTodosDSN user: '' password: ''.

sqlString :=
    'insert into todos (description, completed) values (''buy milk'', 0)'.
connection execute: sqlString.

resultSet := connection execute: 'select * from todos'.
resultSet asTable do: [:row |
    row at: #description :: print
].

connection close.
```

`resultSet` is an instance of **`ODBCResultSet`**

`resultSet asTable` returns an instance of **`ODBCResultTable`** which is a subclass of **`OrderedCollection`**.

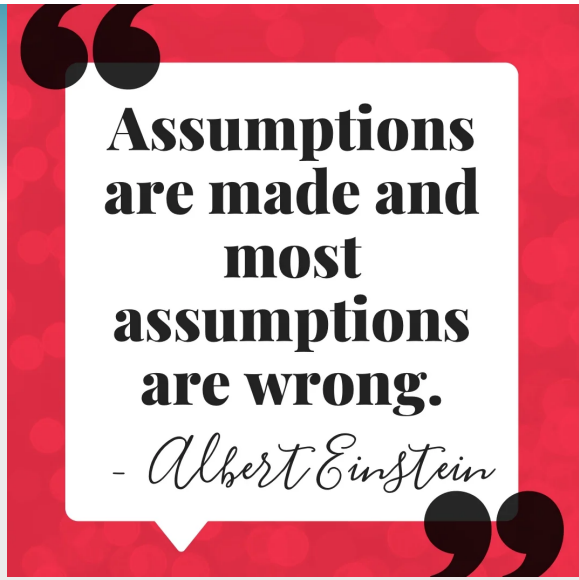
Active Record

- Design pattern described by **Martin Fowler**
  in book “Patterns of Enterprise Application Architecture”
  - “An object that wraps a row in a database table or view,
    encapsulates the database access,
    and adds domain logic on that data.”
  - “The classes match very closely the
    record structure of an underlying database.”

- Famously implemented as part of **Ruby on Rails** framework
- **Automatically discovers table column names and types**
- Ruby implementation uses `method_missing`
- My Smalltalk implementation uses `doesNotUnderstand:`

Assumptions

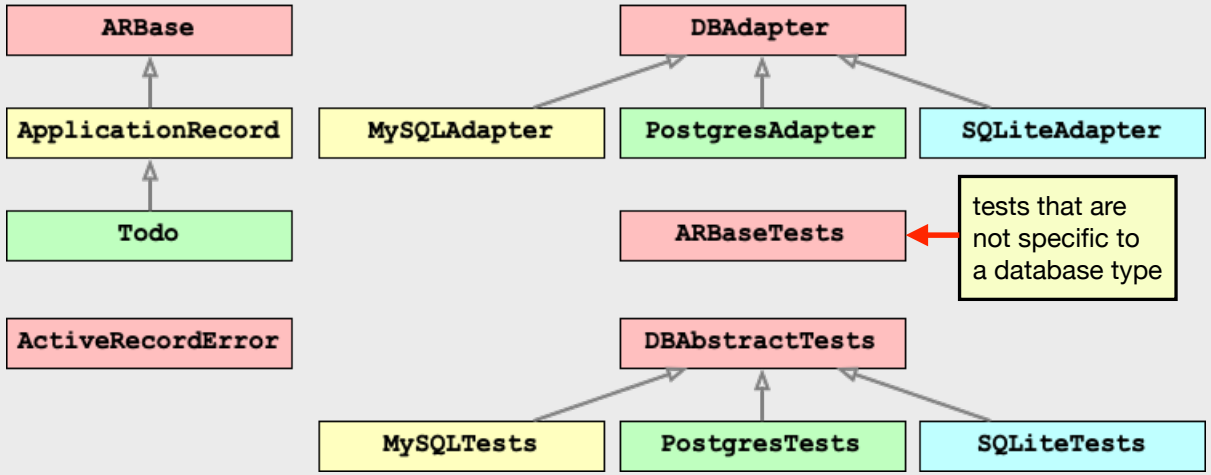
- All tables have an auto-increment `id` column
- All table names are the snake_case, plural version of the corresponding class name
  - class name -> table name
  - `Todo` -> `todos`
  - `GearShiftKnob` -> `gear_shift_knobs`
  - `Person` -> `people`



My Package

- Currently supports MySQL, Postgres, and SQLite
  - to support a new database, implement new subclass of **`DBAdapter`**
- Currently only tested in Cuis Smalltalk
  - hopefully only minimal changes will be needed for Squeak and Pharo
  - relies on having an ODBC package
- To install in a Cuis image
  - clone repository at https://github.com/mvolkmann/**Cuis-Smalltalk-ActiveRecord**
  - open a Workspace and evaluate
    ```
    Feature require: 'ActiveRecord'
    ```

Classes


ARBase
ApplicationRecord
Todo
ActiveRecordError
DBAdapter
MySQLAdapter
PostgresAdapter
SQLiteAdapter
ARBaseTests
tests that are not specific to a database type
DBAbstractTests
MySQLTests
PostgresTests
SQLiteTests

Example Usage ...

- Mimics Ruby on Rails implementation as closely as possible
- **`Todo`** class
  - maps to **`todos`** table
  - only needs to be a subclass of **`ApplicationRecord`**
  - no instance variables or methods are required

```
ApplicationRecord subclass: #Todo
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Demo'
```

... Example Usage

```
| id todo todos |

ARBase logSQL: true. "logs all SQL statements to Transcript"

Todo establishConnection: #SQLiteTodosDSN dbType: #SQLite.
id := Todo create: { #description->'eat tacos'. #completed->0 }.
id := Todo create: { #description->'cut grass'. #completed->1 }.
id := Todo create: { #description->'buy milk'. #completed->0 }.
id := Todo create: { #description->'study Smalltalk'. #completed->1 }.

todos := Todo all. "gets all rows"
todos do: [:todo | todo inspect print].

todos := Todo findWhere: {'completed = 0'}. "gets all matching rows"

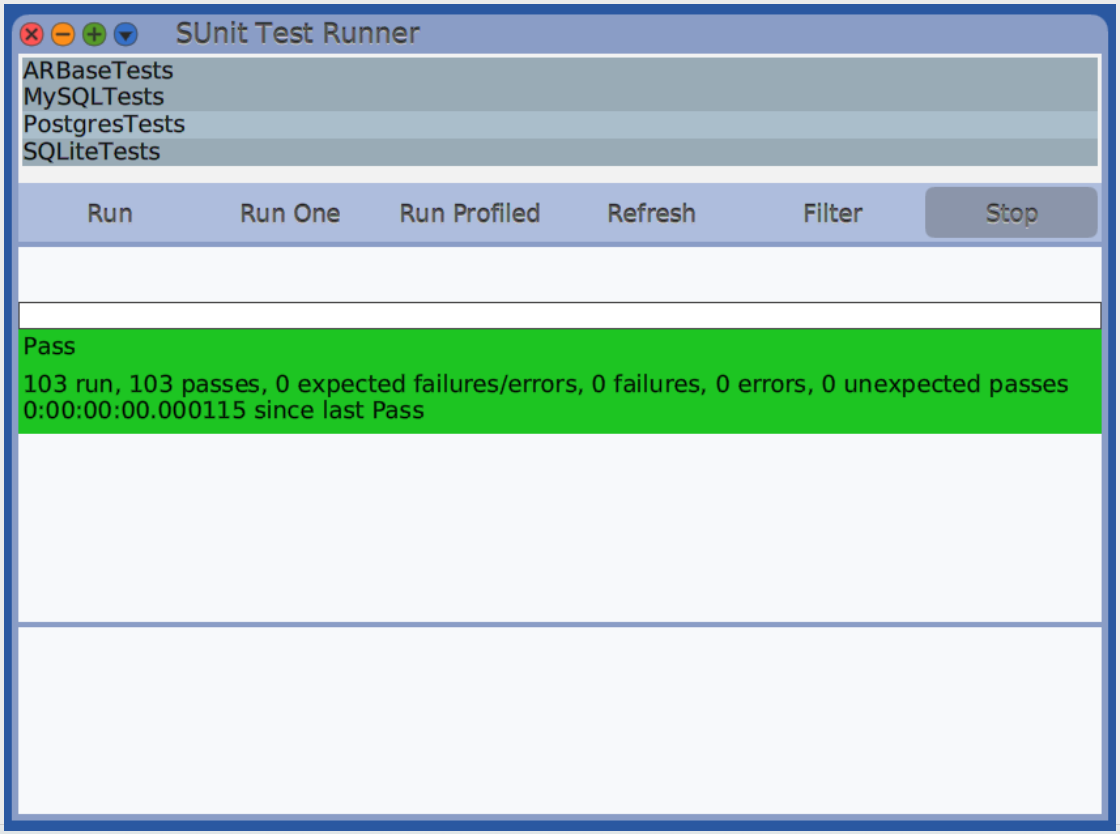
todo := Todo find: 19. "finds one row by its id"
todo description: 'go for a run'. "sets description column"
todo description. "gets description column"
todo destroy. "deletes row"
Todo destroyBy: #description->'buy milk'. "deletes all matching rows"
```

could this be determined from DSN?

outputs lines like
`Todo completed: 1.0, description: 'buy milk', id: 19579`

`inspect` is an ActiveRecord method that returns a string description of a record.

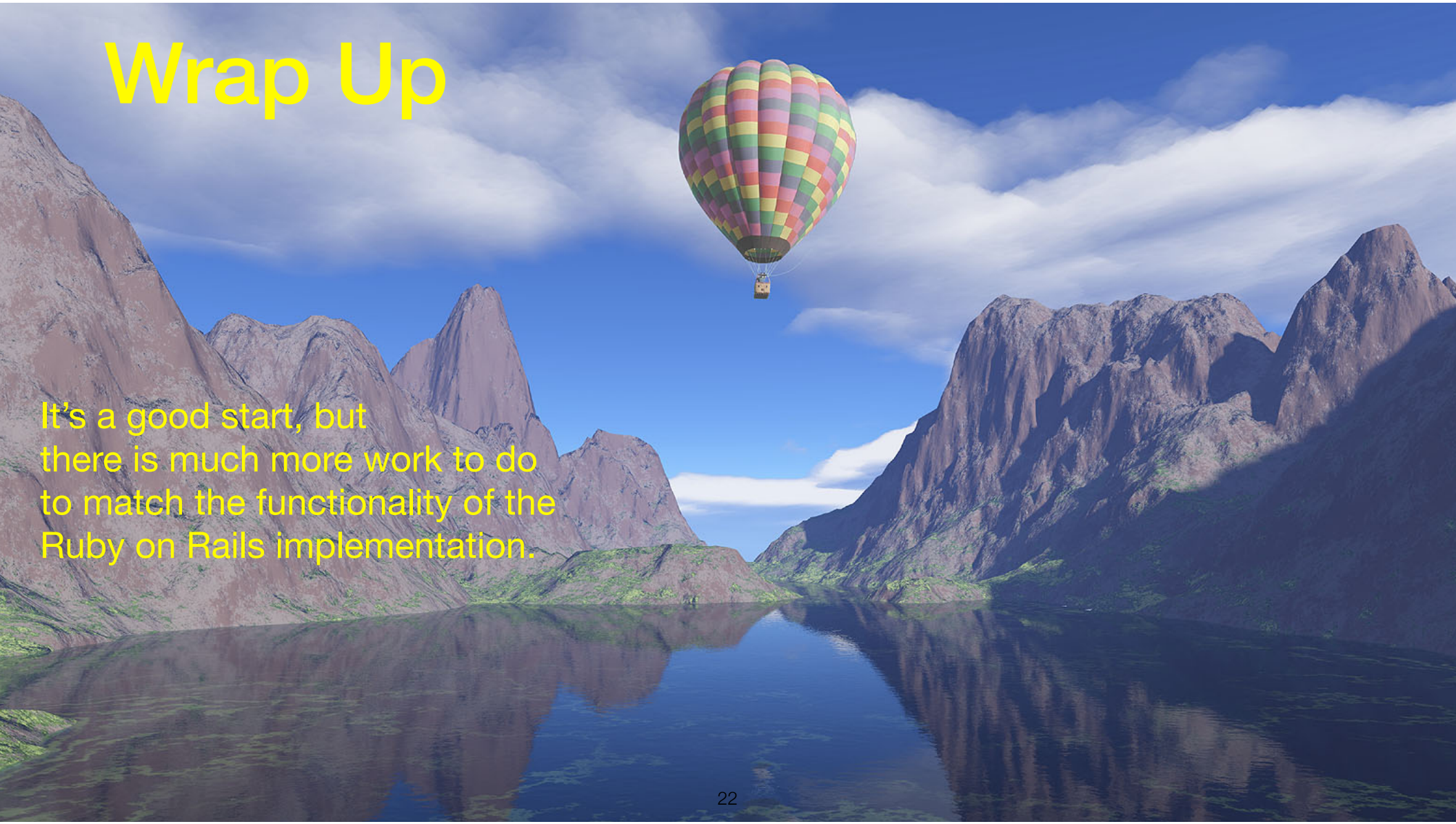
Comprehensive Test Suite



Rails Features Not Yet Implemented

- Describing table relationships
  - to support joins
- Transactions
  - currently all changes auto-commit
- Overriding assumptions
  - `id` column
  - mapping from classes to table names
- Table creation
- Migrations

Wrap Up



It's a good start, but
there is much more work to do
to match the functionality of the
Ruby on Rails implementation.