Unit Testing a C++ Database Application with Mock Objects

Ray Lischner

PNSQC
October 11, 2011

Problem

- Write a library to support simple applications that need to access a SQL database
- Must support unit-testing of applications
- Must have a C++ API
- Must be easy to write, support

Typical Usage (Simplified)

```
struct Name { std::string first, last };

typedef std::vector<Name> Names;

class DbApp
    {
    public:
       DbApp(db::sql* sql);
       void get_names(std::string last_name, Names& name);

private:
      static std::string query_;
      std::auto_ptr<db::sql> sql_;
      std::auto_ptr<db::statement> stmt_;
    }
```

Typical Usage (Simplified)

```
std::string DbApp::query_(
      "SELECT * FROM names WHERE last LIKE ?");

DbApp::DbApp(db::sql* sql): sql_(sql),
      stmt_(sql->prepare(query_))
   {}

void DbApp::get_names(std::string last_name, Names& names) {
      stmt_->bind_param(last_name);
      std::string first, last;
      stmt_->bind_result(first, last);

   stmt_->execute();
      while (stmt_->fetch())
         names.push_back(Name(first, last));
   }
```

Typical Unit Test

```
db::mock::sql sql;
    s.store_result(DbApp::query_, "Ray", "Lischner");
    s.store_result(DbApp::query_, "Cheryl", "Klipp");

DbApp dbapp(&sql);
    Names names;

dbapp.get_names("name", names);

BOOST_CHECK_EQUAL(std::size_t(2), names);
    BOOST_CHECK_EQUAL(std::string("Ray"), names.at(0).first);
    BOOST_CHECK_EQUAL(std::string("Cheryl"),
    names.at(1).first);
    BOOST_CHECK_EQUAL(std::string("Lischner"),
    names.at(0).last);
    BOOST_CHECK_EQUAL(std::string("Klipp"), names.at(1).last);
```

Unit Test Error Cases

```
db::mock::sql sql;
    sql.store_result(DbApp::query_, db::mock::error);
    DbApp dbapp(&sql);
    Names names;

BOOST_CHECK_THROW(dbapp.get_names("name", names),
        db::exception);
```

API

- sql
  - void connect(connection parameters)
  - void execute(query)
  - statement* prepare(query)
- statement
  - void bind_param(...)
  - void bind_result(...)
  - void execute()
  - bool fetch()

Abstract Interfaces

```
class sql {
    public:
        virtual statement* prepare(std::string const& query) = 0;
        virtual void execute(std::string const& query) = 0;
    };

class statement {
    public:
        virtual void execute() = 0;
        virtual bool fetch() = 0;

    bind?

};
```

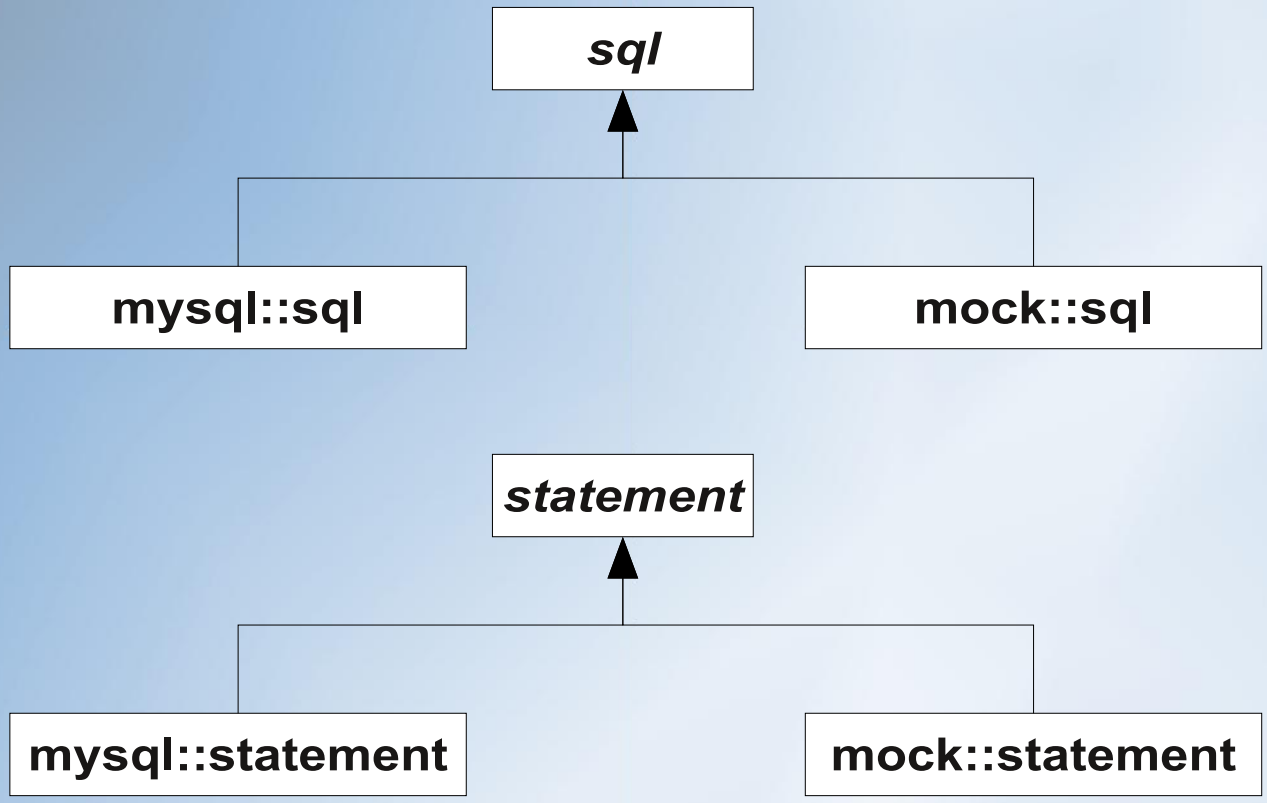
Impure Abstract Interface

```
class statement {
    public:
       template<class T1, class T2>
       void bind_param(T1 const& arg1, T2 const& arg2) {
          params p;
          p.push_back(param(arg1));
          p.push_back(param(arg2));
          bind_params(p);
       }

private:
       void bind_params(params const& p) = 0;

};
```

Real and Mock Classes



Characteristics of a Unit Test

- **F**ast
- **I**ndependent
- **R**epeatable
- **S**elf-validating (automated)
- **T**imely

- Bob Martin. *Clean Code*. Prentice-Hall, 2008.

libdb

- Designed for unit testing
- Mock and real libraries designed together

Questions