An object oriented approach to data driven software development

David Benoit
CTO
Starscale Inc.

benoit@starscale.com

Example of a Data Driven Service: ENUM

- Take a phone number, find a server
  - The BEST server for that number
  - +1-212-xxx-xxxx goes to Verizon
  - +1-212-876-5309 goes to Jenny's personal server
- Many different implementations
  - Lots of regular expressions
  - A big tree with lots of leaves and not many internal data nodes
  - A big list of prefixes
- The question is simple, the implementation is tricky

What normally happens

```
string findServer(string num){
  while(true){
    query q = db.query("SELECT server
                        from the_big_list
                        where prefix = " + num);
    if(q.error()){
      throw exception("db error: " + q.message());
    }
    if(q.found()){
      return q.row[0].column[0];
    }
    num.removeLastChar();
  }
  throw exception("not found");
}
```

How we should do it

```
string findServer(string num){
  FindServer::result r = db.enum.findServer(num);
  if(r.error()){
    throw r.exception();
  }
  return r.server();
}
```

Encapsulation & Data Hiding: not a new idea

“The object-oriented approach encourages the programmer to place data where it is not directly accessible by the rest of the program. Instead, the data is accessed by calling specially written functions, commonly called methods, which are either bundled in with the data or inherited from "class objects." These act as the intermediaries for retrieving or modifying the data they control.”

– Wikipedia: Object Oriented Programming

Sample: A Simple Wiki

- Simple pages
  - Content
  - Creation info
  - Modification info
- Simple permissions
  - A simple list of who can modify a page
- Account Management functionality already exists
  - We can leverage that
  - Think of it as another object we can use

Schema: Pages

```
create table wiki."Pages" (
  -- The unique database id of this page
  "PageId"    bigserial not null,

  -- The content of the page, null if removed.
  "Content"   text null,

  -- The date this page was added to the system
  "CreatedOn" timestamptz not null,

  -- The id of administrator to create the page
  "CreatedBy" bigint not null,

  -- The date this page was last updated
  "UpdatedOn" timestamptz not null,

  -- The id of last administrator to set the page
  "UpdatedBy" bigint not null,

  -- The revision count for this page, starting at 1 per page
  "Revision"  bigint not null
);
```

Schema: Permissions

```
create table wiki."Permissions" (
  -- The member associated with these permissions
  "MemberId"      bigint not null,

  -- True if this member is an admin
  "Admin"         boolean not null,

  -- The pages to which this member has permission to write.
  -- Empty if none.  Does not apply for admins
  "Pages"         bigint[] not null,

  -- The date this permission was last updated
  "UpdatedOn"     timestamptz not null,

  -- The id of last administrator to set this permission
  "UpdatedBy"     bigint not null
);
```

Constraints

```
alter table wiki."Pages" add constraint "Pages_PK"
unique ("PageId");

alter table wiki."Pages" add constraint "Pages_FK1_Members"
foreign key ("CreatedBy") references am."Members" ("MemberId");

alter table wiki."Pages" add constraint "Pages_FK2_Members"
foreign key ("UpdatedBy") references am."Members" ("MemberId");

alter table wiki."Permissions" add constraint "Permissions_FK1_Members"
foreign key ("MemberId") references am."Members" ("MemberId");

alter table wiki."Permissions" add constraint "Permissions_FK2_Members"
foreign key ("UpdatedBy") references am."Members" ("MemberId");
```

If this was an OO language, that would be...

“private”

- These details are necessary, but they are details
- We don't need to know how it is implemented
- We don't care how it is implemented
  - Not all the time
- We don't want to be burdened if there are changes

- What would be “public”?

The Interface

- The stuff we want to do:
  - GetPage(num)
  - SetPage(num, content)
- Take some input, perform an action, return a result
  - Or many!
- No exposure of the internal data representation
- No ability to do anything else
- Related methods are in the same schema
  - Think of these like objects or classes

GetPage

GetPage::Status GetPage(integer id)

```
GetPage::Status {
        Success
        Failure
        PageNotFound
}
Results::Page(1) {
        Content
        Created { On, By }
        Updated { On, By }
        Revision
}
```

GetPage

SetPage::Status SetPage(integer id, text content,
integer baseRevision)

GetPage::Status {

Success

Failure

PageNotFound

PermissionDenied

MidAirCollission

}

No Results

You have to know too much

- Does the member have permission?
- Does the page exist?
- Was a conflicting change made?
- Is the schema the same?

- Things are never as simple as

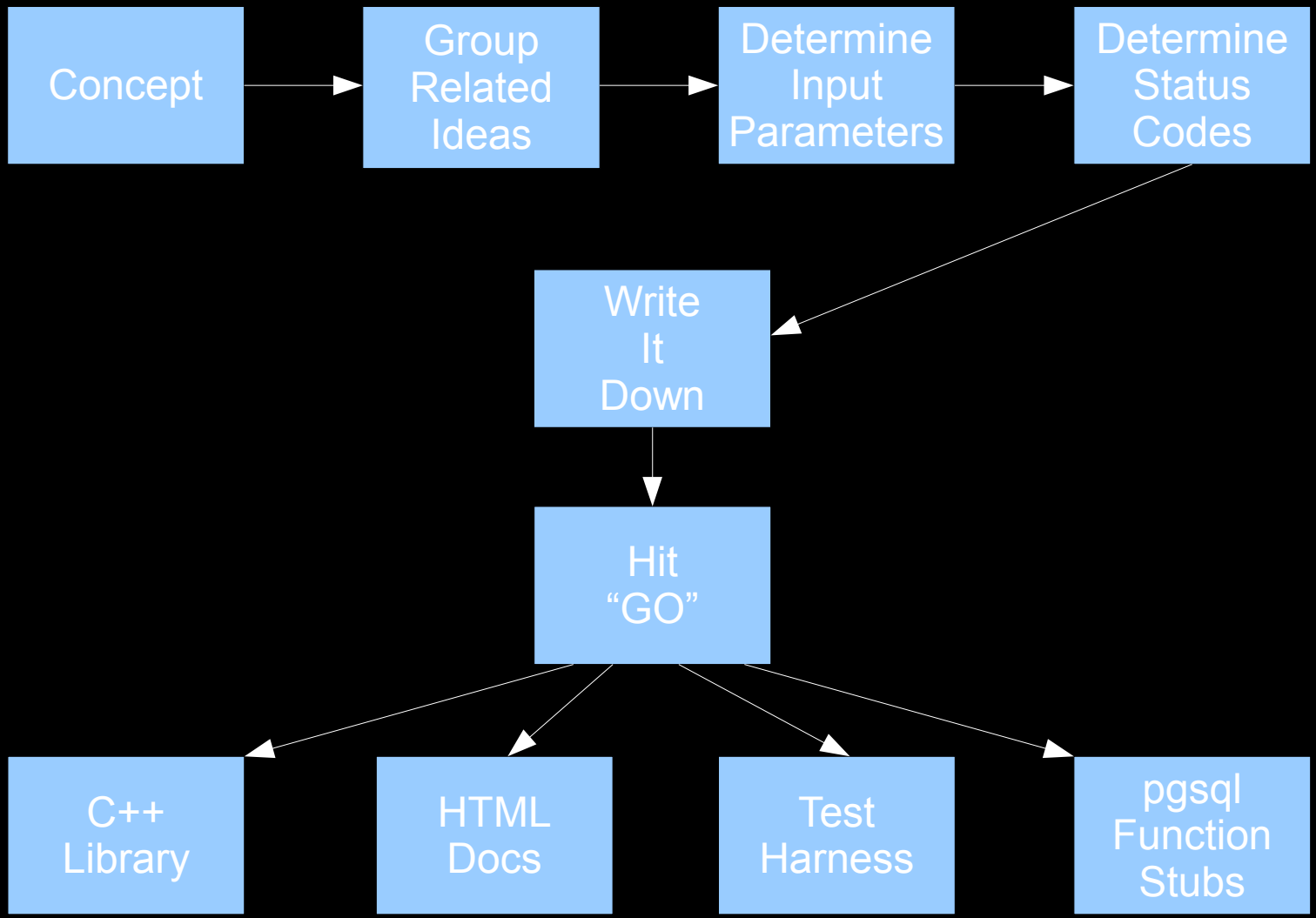
```
select content from pages where id = 4;
```

The Stub

```
create or replace function
   wiki."getPage" (_memberId bigint,
                   _pageId bigint)
   returns int
   security definer
   language plpgsql
   as $$
   declare
      _r1 refcursor := 'RS 1';
      _edit boolean;

   begin
...
      -- Return the info.
      open _r1 for
      select
...
      -- All ok
      return 0;
   end;
   $$;
```

Our Workflow



Division of Responsibility

- The application developer writes the application
- References the HTML documentation
- Can write fake procedures for testing

- The DBA writes the stored procedures
- References the HTML documentation
- Uses the testing harness to verify the implementation

More wins

- Precompilation
  - Most dataservers will pre-compile the code in a function, allowing for faster execution and checking ahead of time
  - Bad syntax, incorrect table/column names, etc. are all compile-time checks instead of run-time
  - Statement caching, optimization, etc.
- Security
  - Drop all permissions on all tables
  - Grant execute to the application user
  - Arguments passed as variables; aren't interpreted

Versioning

- Prepare for change; it is inevitable
- Two options
  - Change the API and update all applications
  - Allow for multiple versions of the same methods
- What you do depends on what you need
- We have done both
  - Legacy applications that we don't want to change, we leave alone and provide a legacy version of the function
  - New applications get all the new features

Testing

- Test the entire interface
  - Preconditions, postconditions, return values
  - etc.

```
const GetPage get(userId, p1);
get.success();
const GetPage::RowOfPage &row = get.getPage(0);
BOOST_REQUIRE_EQUAL(row.getContent(), "the first page");
BOOST_REQUIRE_EQUAL(row.getCreatedBy(), "User 001");
BOOST_REQUIRE_EQUAL(row.getUpdatedBy(), "User 002");
BOOST_REQUIRE_EQUAL(row.getRevision(), 3);
```

Is this enough?

- How can we make this better
- What other features should we explore
- Should this evolve to be the “standard” Application Interface?
- Can this generally act as a “high level” language for databases?

Questions?

Thank You