CORaL

C-like Object Relational Language

Shane Chin - System Tester
Molly Karcher - Language Guru
Luis Peña - System Architect
Miguel Yanez - System Integrator
Brian Wagner - Project Manager
The Purpose of CORaL

Have you ever found yourself struggling with SQL?

Databases are used for everything, but they're hard to use.

Some languages have nice database support, they're scripting languages whose features are known by a lot of people who only know one or two languages.
What is CORaL?

- intuitive
- object-relational
- versatile
- portable
- easy-peasy
- vendor-neutral
- productive
- familiar
- universal
How CORaL helps

We provide a familiar environment for programmers with experience in C.

No need to learn a scripting language to interact with your database.

No need to use SQL.
Writing in CORaL

Demo!
Connecting to a Database

```bash
#cordbconn
server = "localhost";
user = "user";
password = "pass";
port = "8888";
DBName = "People";
type = "mysql";
#enddbconn
```
Adding to your Schema

#cordb
Table Person {
    firstName : string;
    lastName : string;
    age : int;
    primary_key(firstName);
};
#enddb
int main()
{
    user_t Person samplePerson;
    connectDB;

    samplePerson = Person(firstName = "John",
                           lastName = "Example",
                           age = 25);
    samplePerson.add();
    closeDB;
    return 0;
}
Demo!
The CORaL Compiler
Architecture

program.cl → Lexer → Parser → Syntax Analysis

Lexical tokens

AST

Semantic

Type Checking

.clx Python Script → Generator → Python Code
Intermediate Representations

type formal =
    Format of dtype + string

type stmt =
    Block of stmt list
    | Expr of expr
    | Return of expr
    | If of expr & stmt & stmt
    | For of expr & expr & expr & stmt
    | While of expr & stmt
    | ClosedDB
    | ConnectDB
    | Nostmt

type func_def = {
    return_type : dtype;
    name : string;
    formals : formal list;
    locals : var_decl list;
    body : stmt list;
}

type table_body =
    TableBody of attribute list & keydecls list & func_def list

type table = {
    tname : table_label;
    tbody : table_body;
}

type table_block =
    TableBlock of table list
    | NoTableBlock

type program = {
    conn : conn_block;
    tables : table_block;
    globals : var_decl list;
    funcs : func_def list;
}
The Executable

```python
#!/usr/bin/env python
from __future__ import print_function
import coral_backend
from coral_backend import *
from coral_backend.controller import *

setServer("server")
setUser("server")
setPass("server")
setPort("server")
setDBName("server")
setConnType("sqlite")
conn_block = True

def test_func():
    print("testing function\n", end='')
    return 0

def main():
    a = 1
    test_func()

if __name__ == '__main__':
    if (conn_block):
        connectDB()
    main()
```
The Development Process

The Makefile

- Top level Makefile to execute all other modules
- Different build stages for testing, development and installation
- Once installed, you can run make to see if it is already in your PATH

Development Tools

- git

Management Process

- GitHub
- GitLab
- JIRA

Prezi
Development Tools

- Git
- OCaml
- Vim
- SQLAlcheMy
- SQLite
- Sublime Text
The Makefile

GNU Make

Top-level Makefile to execute all other makefiles.

Different build stages for testing, development, and installation

Once installed you can run coralc as it is already in your PATH.
Testing CORaL

Testing Plan
- Test before code
- Regression Test on commit
- el ➔ elerr ➔ exp
- el ➔ out

Testing Methodology
- Regression Tests
- Installation Testing
- Test Coverage
Testing Plan

Test before code

.cl → .clx.err → exp

 Regression Test on commit
Testing Methodology

Regression Tests

Installation Testing

Test Coverage
Conclusions

Lessons Learned

- The power of functional languages
- Integration is easy in-person
- The team matters more than the roles

Thanks!
Questions?
Lessons Learned

The power of functional languages
Integration is easy in-person
The team matters more than the roles
Thanks!

Questions?