Persisting C++ Classes in Relational Databases with ODB

Boris Kolpackov

Code Synthesis

v1.0, Sep 2014

ODB, an ORM for C++

- Part I: Introduction and Basic Operations
- Part II: Advanced Technique and Mechanisms

What's an ORM, anyway?

Why ORM?

- Object-oriented vs relational mismatch
- Type and name safety
- Parameter binding and result set extraction
- Database schema evolution

Manual Schema Evolution

```
ALTER TABLE person
  ADD COLUMN age
    INTEGER UNSIGNED NOT NULL DEFAULT 0
```

Why not use an ORM?

- Hides too much
- Shoot yourself in the foot
- Framework
- Fun to roll your own

```
sword OCIBindDynamic ( OCIBind      *bindp,
                       OCIError     *errhp,
                       void         *ictxp,
                       OCICallbackInBind   (icbfp)(
                                void       *ictxp,
                                OCIBind    *bindp,
                                ub4        iter,
                                ub4        index,
                                void       **bufpp,
                                ub4        *alenp,
                                ub1        *piecep,
                                void       **indpp ),
                                void       *octxp,
                       OCICallbackOutBind  (ocbfp)(
                                void       *octxp,
                                OCIBind    *bindp,
```

Why Relational?

- Mature and reliable
- Tooling, support, and alternatives
- Flexible

What's ODB?

- Three levels
- Not a framework
- No magic
- One-to-one ORM-Database operation mapping

ODB, and ORM for C++

- Automatic generation of database code from C++ classes
- Target multiple databases
- Database schema evolution

C++ Standards

C++98 and C++11

- Rvalue references
- Range-based `for` loop
- `std::function` and lambdas
- C++11 Standard Library integration
- C++11 in examples

Databases

Cross-Database

- MySQL
- SQLite
- PostgreSQL
- Oracle
- Microsoft SQL Server

Platforms and Compilers

Cross-Platform

- Linux, Windows, Mac OS X, Solaris
- GCC, Visual C++, Clang, Sun Studio C++


Ultra II
SanDisk
2.0 GB

Mobile & Embedded

- ODB + SQLite
- “Hello, World” example is 500Kb
- Cross-compiler friendly
- Android, Raspberry Pi guides

Performance

High-Performance and Low Overhead

- Prepared statements, including custom queries
- Caching of connections, statements, and buffers
- Low-level native database C APIs
- Zero per-object memory overhead

Performance

High-Performance and Low Overhead

- Prepared statements, including custom queries
- Caching of connections, statements, and buffers
- Low-level native database C APIs
- Zero per-object memory overhead

Load performance

- SQLite — 60,000 object per second — 17 μs per object
- PostgreSQL — 15,000 objects per second — 65 μs per object

License

Dual-Licensed

- GPL + commercial license
- Can be used without restrictions within your organization
- License exceptions for open source projects

- ODB License
  - www.codesynthesis.com/products/odb/license.xhtml

C++ Support

ODB is implemented as a GCC plugin

C++ Support

ODB is implemented as a GCC plugin

- Mature, portable, and readily available
- One of the most complete C++11 implementations

C++ Support

C++ in, C++ out

C++ Support

C++ in, C++ out

Use any C++ compiler to build your application

C++ Support

C++ in, C++ out

Use any C++ compiler to build your application

Yes, even Sun Studio

Standard C++ In

Standard C++ In
Standard C++ Out

Persistent Class

```
enum class status {open, confirmed, closed};

class bug
{
public:
  ...

private:
  unsigned long long id_;

  status status_;
  std::string summary_;
  std::string description_;
};
```

Persistent Class

```
#include <odb/core.hxx>

#pragma db object
class bug
{
  ...
private:
  friend class odb::access;
  bug () {}

  #pragma db id auto
  unsigned long long id_;

  status status_;
  std::string summary_;
  std::string description_;
};
```

Persistent Class

```
#include <odb/core.hxx>

#pragma db object
class bug
{
  ...
private:
  friend class odb::access;
  bug () {}

  #pragma db id auto
  unsigned long long id_;

  status status_;
  std::string summary_;
  std::string description_;
};
```

Persistent Class

```
#include <odb/core.hxx>

#pragma db object
class bug
{
  ...
private:
  friend class odb::access;
  bug () {}

  #pragma db id auto
  unsigned long long id_;

  status status_;
  std::string summary_;
  std::string description_;
};
```

Persistent Class

```
#include <odb/core.hxx>

#pragma db object
class bug
{
  ...
private:
  friend class odb::access;
  bug () {}

  #pragma db id auto
  unsigned long long id_;

  status status_;
  std::string summary_;
  std::string description_;
};
```

Persistent Class

```
#include <odb/core.hxx>

#pragma db object
class bug
{
  ...
private:
  friend class odb::access;
  bug () {}

  #pragma db id auto
  unsigned long long id_;

  status status_;
  std::string summary_;
  std::string description_;
};
```

Persistent Class

```
#include <odb/core.hxx>

#pragma db object
class bug
{
  ...
private:
  friend class odb::access;
  bug () {}

  #pragma db id auto
  unsigned long long id_;

  status status_;
  std::string summary_;
  std::string description_;
};
```

Persistent Class

```
#pragma db object
class bug
{
public:
  unsigned long long id () const;
  void id (unsigned long long);

  status get_status () const;
  status& setStatus ();

  std::string& summary_please ();

  ...
private:

  #pragma db id auto
  unsigned long long id_;
  ...
```

Persistent Class

```
#pragma db object
class bug
{
public:
  unsigned long long id () const;
  void id (unsigned long long);

  status get_status () const;
  status& setStatus ();

  std::string& summary_please ();

  ...
private:

  #pragma db id auto
  unsigned long long id_;
  ...
```

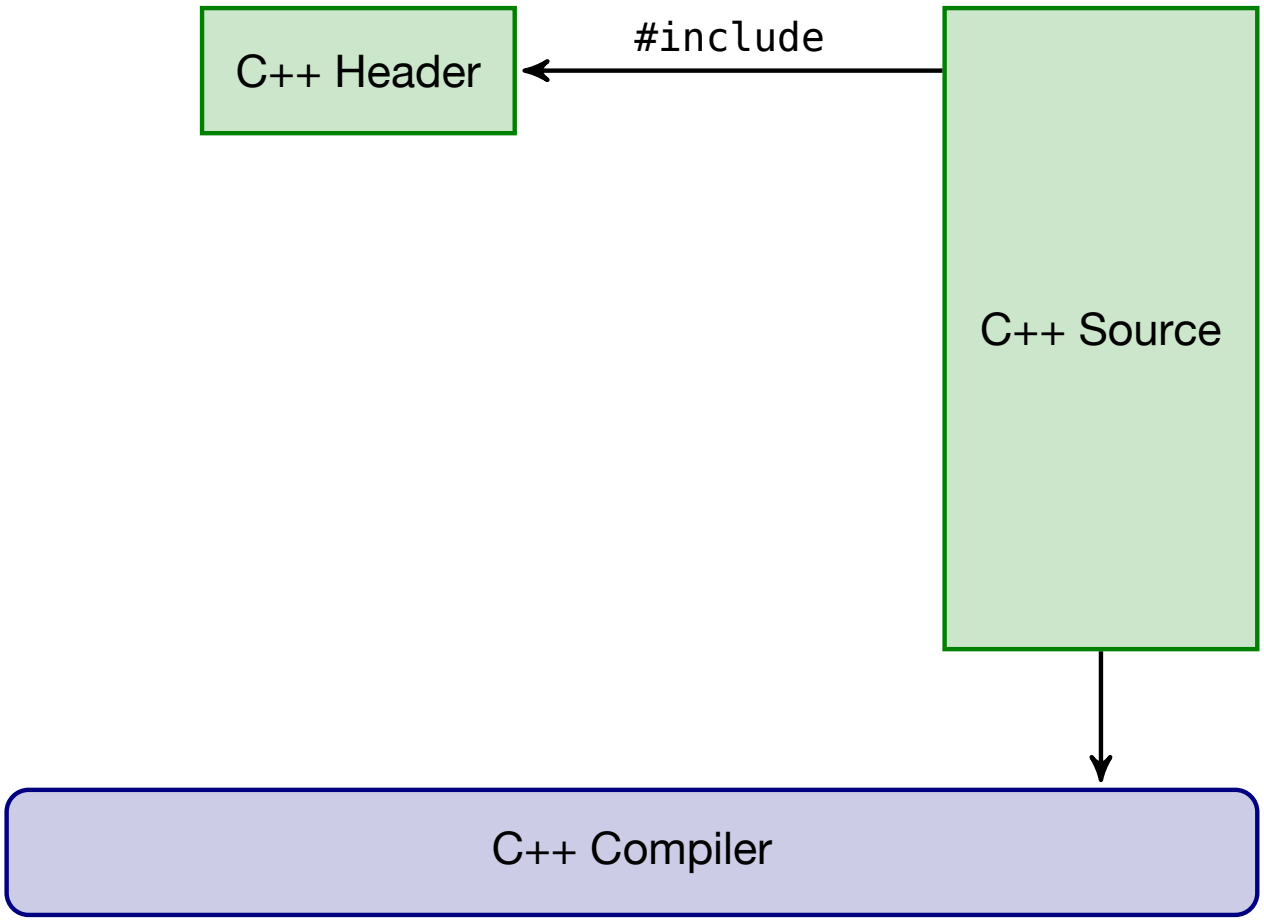
Persistent Class

```
class bug
{
  ...
private:
  unsigned long long id_;

  ...
};

#ifdef ODB_COMPILER
#  pragma db object(bug)
#  pragma db member(bug::id_) id auto
#endif
```

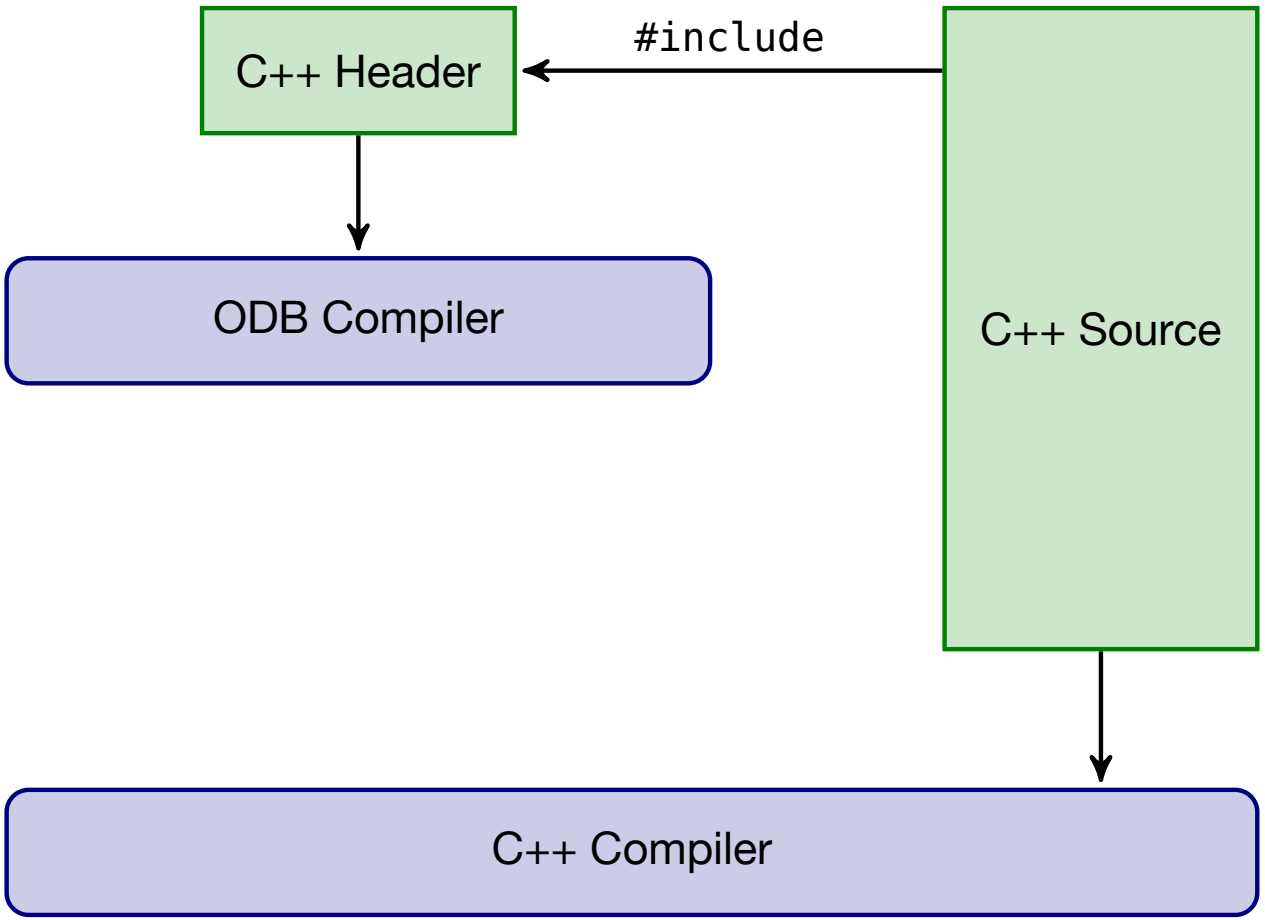
Persistent Class

```
// bug.hxx
class bug
{
  ...
private:
  unsigned long long id_;

  ...
};

// bug-mapping.hxx
#pragma db object(bug)
#pragma db member(bug::id_) id auto
```

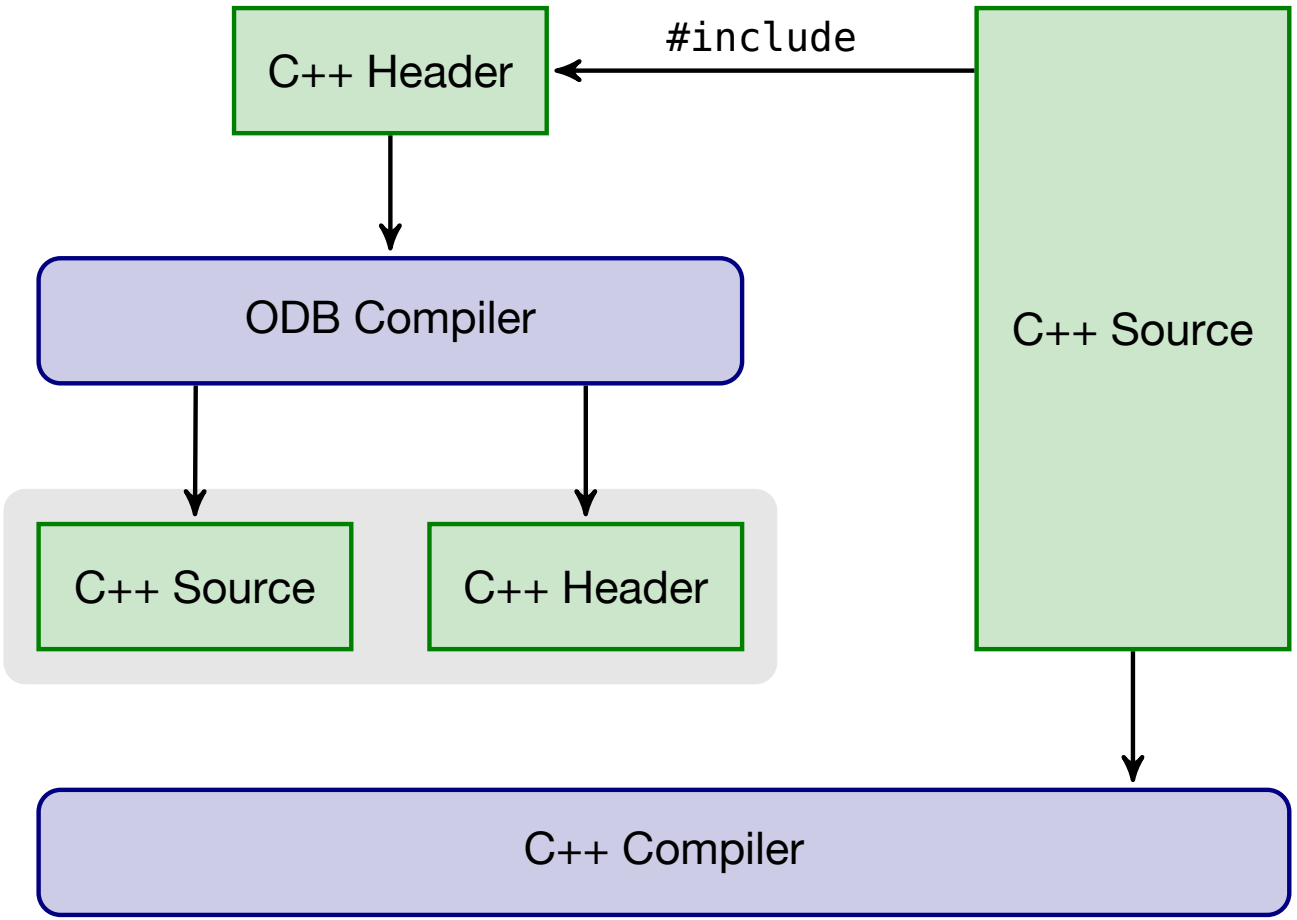
Workflow



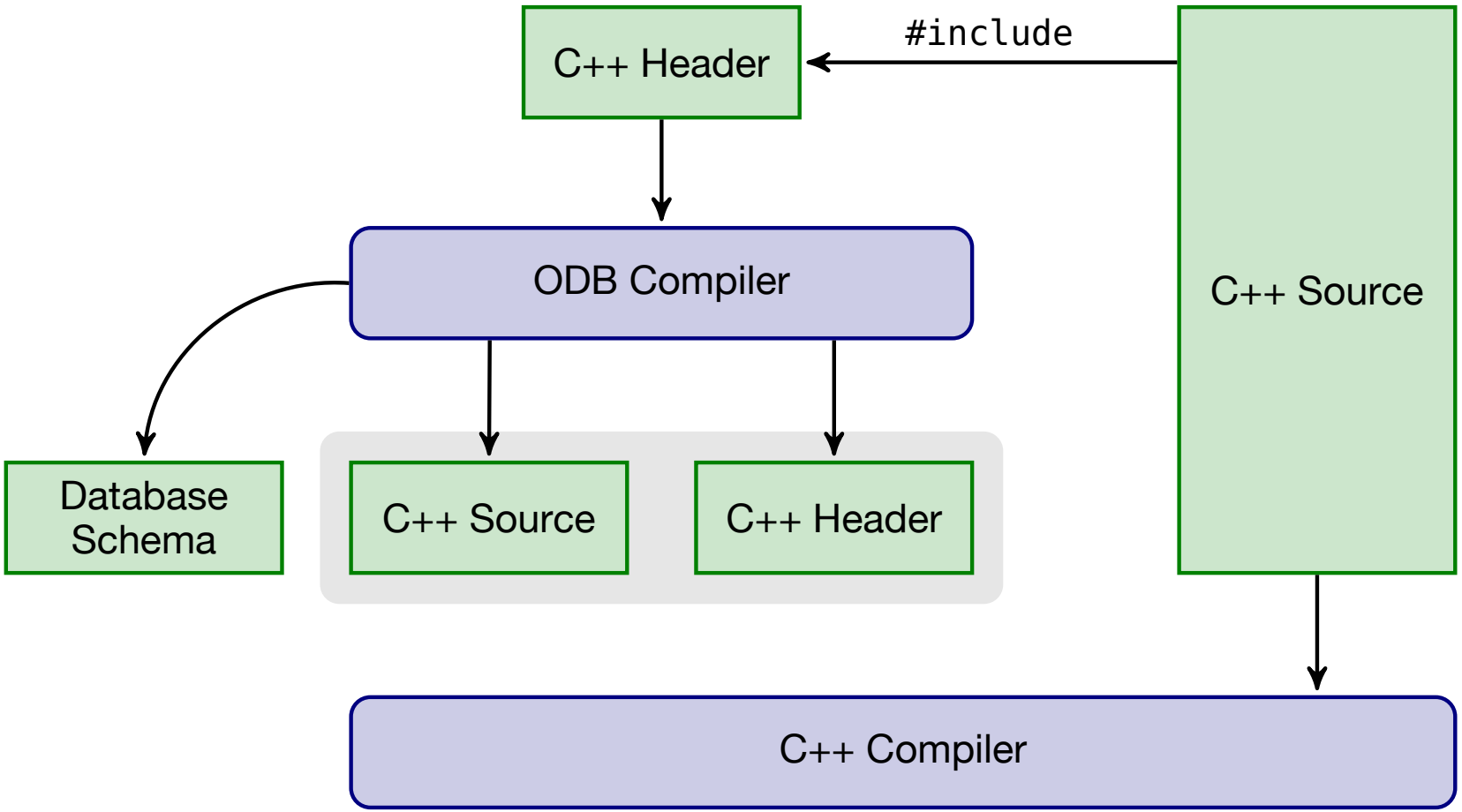
Workflow



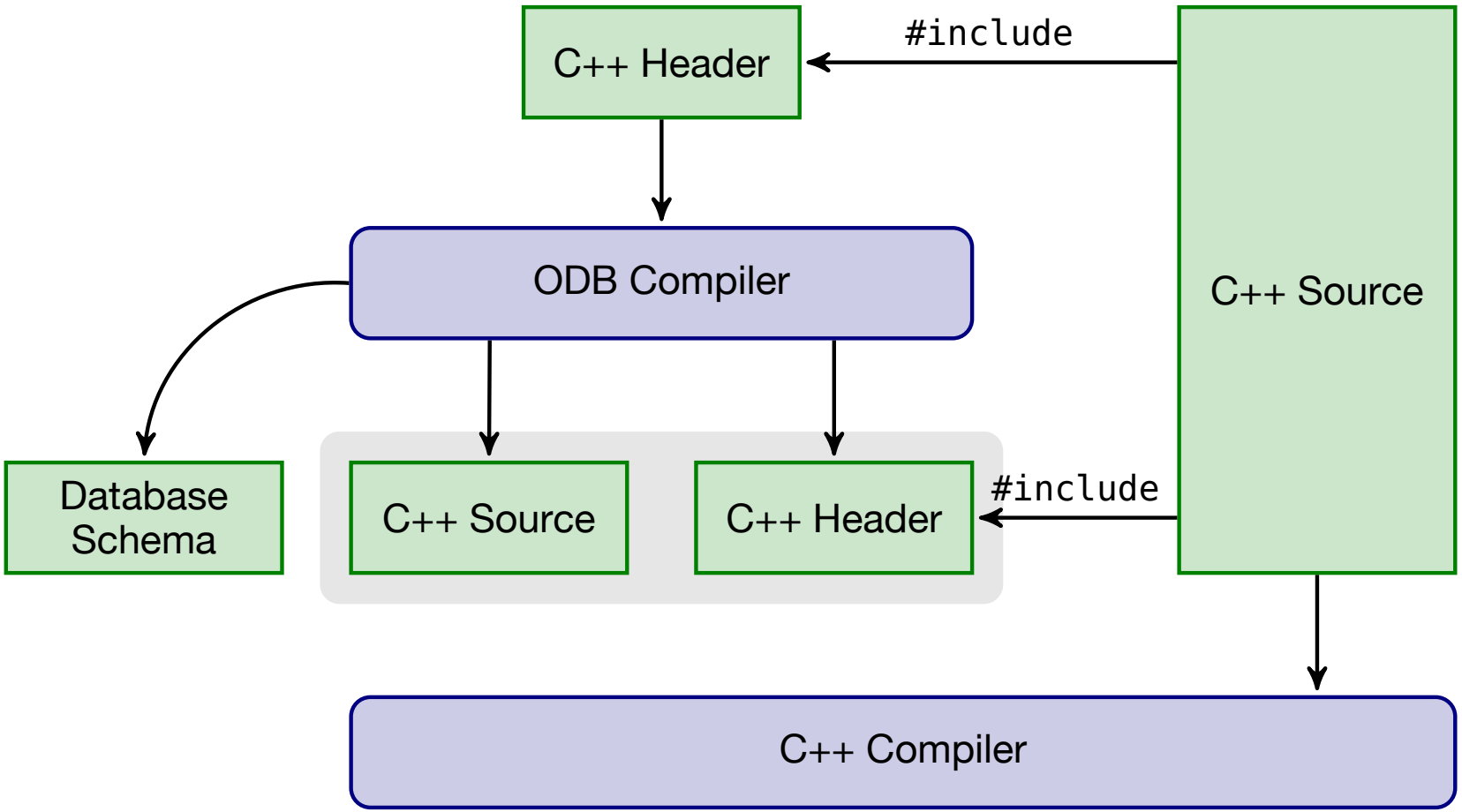
Workflow


C++ Header
#include
C++ Source
ODB Compiler
C++ Source
C++ Header
C++ Compiler

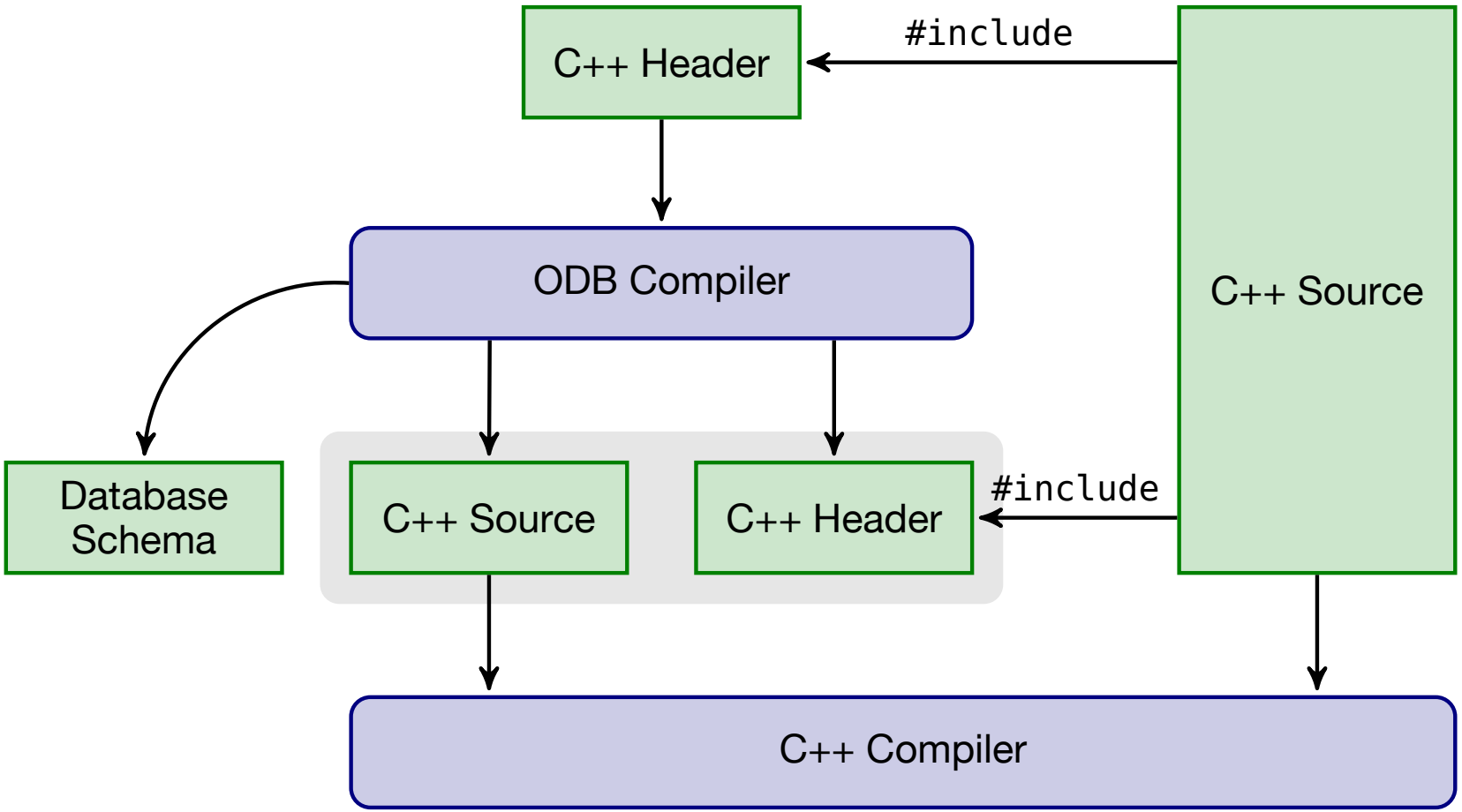
Workflow


C++ Header
#include
C++ Source
ODB Compiler
Database Schema
C++ Source
C++ Header
C++ Compiler

Workflow


C++ Header
#include
C++ Source
ODB Compiler
Database Schema
C++ Source
C++ Header
#include
C++ Compiler

Workflow


C++ Header
#include
C++ Source
ODB Compiler
Database Schema
C++ Source
C++ Header
#include
C++ Compiler

ODB Compiler

```
$ odb --database pgsql bug.hxx
```

ODB Compiler

```
$ odb --database pgsql bug.hxx

$ ls
bug.hxx
bug-odb.cxx
bug-odb.hxx
bug-odb.ixx
```

ODB Compiler

```
$ odb -I/opt/boost-latest -DENABLE_LASER_BEAMS ...
```

ODB Compiler

```
$ odb -I/opt/boost-latest -DENABLE_LASER_BEAMS ...

$ odb --std c++11 --default-pointer std::shared_ptr ...
```

ODB Compiler

```
$ odb --generate-schema -d mysql bug.hxx
```

ODB Compiler

```
$ odb --generate-schema -d mysql bug.hxx

$ ls
bug.hxx
bug-odb.cxx
bug-odb.hxx
bug-odb.ixx
bug.sql
```

ODB Compiler

```
$ odb --generate-schema -d mysql bug.hxx

$ ls
bug.hxx
bug-odb.cxx
bug-odb.hxx
bug-odb.ixx
bug.sql

$ cat bug.sql
```

ODB Compiler

```
$ odb --generate-schema -d mysql bug.hxx

$ ls
bug.hxx
bug-odb.cxx
bug-odb.hxx
bug-odb.ixx
bug.sql

$ cat bug.sql

CREATE TABLE bug (
  id BIGINT UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT,
  status ENUM('open', 'confirmed', 'closed') NOT NULL,
  summary TEXT NOT NULL,
  description TEXT NOT NULL)
```

Database

```
#include <odb/pgsql/database.hxx>

odb::pgsql::database db ("bugger",      // user
                         "secret",      // password
                         "bugs");       // database
```

Database

```
#include <odb/pgsql/database.hxx>

odb::pgsql::database db ("bugger",      // user
                         "secret",      // password
                         "bugs");       // database


#include <odb/sqlite/database.hxx>

odb::sqlite::database db ("bugs.db");  // database
```

Database

```
#include <odb/pgsql/database.hxx>

odb::pgsql::database db ("bugger",      // user
                         "secret",      // password
                         "bugs");       // database


#include <odb/sqlite/database.hxx>

odb::sqlite::database db ("bugs.db");  // database


#include <odb/database.hxx>

void do_it (odb::database& db);
```

Database Schema

- Automatically generated
- Map to a custom schema

Generated Schema

- Standalone SQL file
- Embedded into generated C++

Generated Schema

- Standalone SQL file
- Embedded into generated C++

```
#include <odb/schema-catalog.hxx>

transaction t (db.begin ());
schema_catalog::create_schema (db);
t.commit ();
```

Custom Schema

- Map classes to tables
- Map data members to columns
- Map C++ types to database types

Custom Schema

- Map classes to tables
- Map data members to columns
- Map C++ types to database types

```
#pragma db object table("bugs")
class bug
{
  #pragma db id auto column("bug_id")
  unsigned long long id_;

  #pragma db column("bug_status") type("SMALLINT")
  status status_;

  ...
};
```

Making Objects Persistent

```
bug b (open,
       ”Support for DB2”,
       ”ODB does not yet support IBM DB2.”);

transaction t (db.begin ());

db.persist (b);

t.commit ();
```

Making Objects Persistent

```
bug b (open,
       ”Support for DB2”,
       ”ODB does not yet support IBM DB2.”);

transaction t (db.begin ());



t.commit ();
```

Transactions

```
try
{
  transaction t (db.begin ());

  db.persist (b1);
  db.persist (b2);

  t.commit ();
}
catch (const odb::connection_lost&)
{
  // Try again.
  ...
}
```

Transactions

```
try
{
  transaction t (db.begin ());

  db.persist (b1);
  db.persist (b2);

  t.commit ();
}
catch (const odb::connection_lost&)
{
  // Try again.
  ...
}
```

Transactions

```
try
{
  transaction t (db.begin ());

  db.persist (b1);
  db.persist (b2);

  t.commit ();
}
catch (const odb::connection_lost&)
{
  // Try again.
  ...
}
```

Making Objects Persistent

```
bug b (open,
       "Support for DB2",
       "ODB does not yet support IBM DB2.");

transaction t (db.begin ());


unsigned long long id = db.persist (b);

t.commit ();
```

Making Objects Persistent

```
bug b (open,
       "Support for DB2",
       "ODB does not yet support IBM DB2.");

transaction t (db.begin ());
t.tracer (odb::stderr_tracer);

unsigned long long id = db.persist (b);

t.commit ();
```

Making Objects Persistent

```
bug b (open,
       "Support for DB2",
       "ODB does not yet support IBM DB2.");

transaction t (db.begin ());
t.tracer (odb::stderr_tracer);

unsigned long long id = db.persist (b);

t.commit ();
```

```
=> INSERT INTO bug (
     id,
     status,
     summary,
     description)
   VALUES (DEFAULT, $1, $2, $3)
   RETURNING id
```

Loading Persistent Objects

```
transaction t (db.begin ());

std::shared_ptr<bug> b (db.load<bug> (id));



t.commit ();
```

Loading Persistent Objects

```
transaction t (db.begin ());



bug b;
db.load (id, b);

t.commit ();
```

Loading Persistent Objects

```
transaction t (db.begin ());

std::shared_ptr<bug> b (db.load<bug> (id));

bug b;
db.load (id, b);

t.commit ();
```

```
=> SELECT
     status,
     summary,
     description
   FROM bug WHERE id = $1
```

Updating Persistent Objects

```
transaction t (db.begin ());

std::shared_ptr<bug> b (db.load<bug> (id));
b->status (confirmed);
db.update (b);

t.commit ();
```

Updating Persistent Objects

```
transaction t (db.begin ());

std::shared_ptr<bug> b (db.load<bug> (id));
b->status (confirmed);
db.update (b);

t.commit ();
```

```
=> UPDATE bug SET
     status = $1,
     summary = $2,
     description = $3
   WHERE id = $4
```

Querying the Database

```
typedef odb::query<bug> query;
typedef odb::result<bug> result;
```

Querying the Database

```
typedef odb::query<bug> query;
typedef odb::result<bug> result;

result r = ...

for (result::iterator i (r.begin()); i != r.end(); ++i)
  ...
```

Querying the Database

```
typedef odb::query<bug> query;
typedef odb::result<bug> result;

result r = ...
```

```
for (bug& b: r)
  ...
```

Querying the Database

```
typedef odb::query<bug> query;
typedef odb::result<bug> result;

transaction t (db.begin ());

result r (db.query<bug> (query::status == open));




t.commit ();
```

Querying the Database

```
typedef odb::query<bug> query;
typedef odb::result<bug> result;

transaction t (db.begin ());

result r (db.query<bug> (query::status == open));

for (const bug& b: r)
  cout << b.id () << " " << b.summary () << endl;

t.commit ();
```

Querying the Database

```
typedef odb::query<bug> query;

transaction t (db.begin ());

for (auto& b: db.query<bug> (query::status == open))
  ...

t.commit ();
```

Querying the Database

```
typedef odb::query<bug> query;

transaction t (db.begin ());

for (auto& b: db.query<bug> (query::status == open))
  ...

t.commit ();
```

```
=> SELECT
     id
     status,
     summary,
     description
   FROM bug WHERE status = $1
```

Querying the Database

```
db.query<bug> (query::status == open ||
               query::status == confirmed);

status s;
query q (query::status == query::_ref (s));

s = open;
db.query<bug> (q); // status == open

s = closed;
db.query<bug> (q); // status == closed

db.query<bug> ("status = " + query::_val (open));

db.query<bug> ("stats = " + query::_val (123));
```

Querying the Database

```
db.query<bug> (query::status == open ||
               query::status == confirmed);

status s;
query q (query::status == query::_ref (s));

s = open;
db.query<bug> (q); // status == open

s = closed;
db.query<bug> (q); // status == closed

db.query<bug> ("status = " + query::_val (open));

db.query<bug> ("stats = " + query::_val (123));
```

Querying the Database

```
db.query<bug> (query::status == open ||
               query::status == confirmed);
```

```
status s;
query q (query::status == query::_ref (s));

s = open;
db.query<bug> (q); // status == open

s = closed;
db.query<bug> (q); // status == closed
```

```
db.query<bug> ("status = " + query::_val (open));

db.query<bug> ("stats = " + query::_val (123));
```

Querying the Database

```
db.query<bug> (query::status == open ||
               query::status == confirmed);

status s;
query q (query::status == query::_ref (s));

s = open;
db.query<bug> (q); // status == open

s = closed;
db.query<bug> (q); // status == closed

db.query<bug> ("status = " + query::_val (open));

db.query<bug> ("stats = " + query::_val (123));
```

Querying the Database

```
db.query<bug> (query::status == open ||
               query::status == confirmed);

status s;
query q (query::status == query::_ref (s));

s = open;
db.query<bug> (q); // status == open

s = closed;
db.query<bug> (q); // status == closed

db.query<bug> ("status = " + query::_val (open));

db.query<bug> ("stats = " + query::_val (123));
```

Deleting Persistent Objects

```
transaction t (db.begin ());

db.erase<bug> (id);




t.commit ();
```

Deleting Persistent Objects

```
transaction t (db.begin ());


bug b = ...;
db.erase (b);


t.commit ();
```

Deleting Persistent Objects

```
transaction t (db.begin ());
```

```
db.erase_query<bug> (query::status == closed);
```

```
t.commit ();
```

Deleting Persistent Objects

```
transaction t (db.begin ());

db.erase<bug> (id);

bug b = ...;
db.erase (b);

db.erase_query<bug> (query::status == closed);

t.commit ();
```

```
=> DELETE FROM bug WHERE id = $1
```

Adding Timestamps

```
#pragma db object
class bug
{
  ...

  #pragma db id auto
  unsigned long long id_;

  status status_;
  std::string summary_;
  std::string description_;



};
```

Adding Timestamps

```
#pragma db object
class bug
{
  ...

  #pragma db id auto
  unsigned long long id_;

  status status_;
  std::string summary_;
  std::string description_;

  boost::posix_time::ptime created_;
  boost::posix_time::ptime updated_;
};
```

Profiles

- Generic integration mechanism
- Covers smart pointers, containers, and value types
- ODB includes profiles for Boost and Qt
- You can add your own profiles

```
odb -d pgsql -p boost bug.hxx

odb -d pgsql -p qt bug.hxx
```

Boost Profile

- `uuid`
- `date_time`
- `optional`

NULL Semantics

```
#pragma db object
class bug
{
  ...

  boost::optional<std::string> description_;
};


CREATE TABLE bug (
  ...
  description TEXT NULL)
```

Qt Profile

- Basic types: `QString`, `QUuid`, `QByteArray`
- Date-time types: `QDate`, `QTime`, `QDateTime`

Adding Creation and Modification Dates (Qt)

```
#pragma db object
class Bug
{
  ...

  #pragma db id auto
  unsigned long long id_;

  Status status_;
  QString summary_;
  QString description_;

  QDateTime created_;
  QDateTime updated_;
};
```

Containers

- Standard: `vector`, `list`, `set`, `map`, etc
- C++11: `array`, unordered (hashtable), etc
- Boost: unordered, `multi_index`
- Qt: `QList`, `QVector`, `QMap`, `QSet`, `QHash`, etc
- Easy to support custom containers

Adding Comments and Tags

```
#pragma db object
class bug
{
  ...

  #pragma db id auto
  unsigned long long id_;

  status status_;
  std::string summary_;
  std::string description_;

  boost::posix_time::ptime created_;
  boost::posix_time::ptime updated_;

  std::vector<std::string> comments_;
  std::unordered_set<std::string> tags_;
};
```

Adding Comments and Tags (Qt)

```
#pragma db object
class Bug
{
  ...

  #pragma db id auto
  unsigned long long id_;

  Status status_;
  QString summary_;
  QString description_;

  QDateTime created_;
  QDateTime updated_;

  QList<QString> comments_;
  QHash<QString> tags_;
};
```

Composite Value Types

- Class or struct type
- Mapped to more than one database column
- Contains composite values, containers, pointers to objects
- Can be used as an object id

Extending Comments

```
#pragma db value
class comment
{
  ...

  std::string text_;
  boost::posix_time::ptime created_;
};

#pragma db object
class bug
{
  ...

  std::vector<comment> comments_;
};
```

Relationships

- Relationships are represented as pointers to objects
- Standard: raw, `auto_ptr`, `tr1::shared_ptr`
- C++11: `std::shared_ptr`, `std::unique_ptr`
- Boost: `boost::shared_ptr`
- Qt: `QSharedPointer`
- Easy to support custom smart pointers

Adding User Object

```
#pragma db object
class user
{
  ...

  #pragma db id
  std::string email_;

  std::string first_;
  std::string last_;
};
```

Adding Bug Reporter

```
#pragma db object
class bug
{
  ...

  std::shared_ptr<user> reporter_;
};
```

Adding Bug Reporter

```
#pragma db object
class bug
{
  ...

  std::shared_ptr<user> reporter_;
};
```

unidirectional to-one relationship

Adding Bug List

```
#pragma db object
class user
{
  ...

  #pragma db id
  std::string email_;

  std::string first_name_;
  std::string last_name_;

  std::vector<std::shared_ptr<bug>> reported_bugs_;
};
```

Adding Bug List

```
#pragma db object
class user
{
  ...

  #pragma db id
  std::string email_;

  std::string first_name_;
  std::string last_name_;

  std::vector<std::shared_ptr<bug>> reported_bugs_;
};
```

bidirectional many-to-one relationship

We Have a Problem

```
#pragma db object
class user
{
  ...

  std::vector<std::shared_ptr<bug>> reported_bugs_;
};

#pragma db object
class bug
{
  ...

  std::shared_ptr<user> reporter_;
};
```

We Have a Problem

```
#pragma db object
class user
{
  ...

  std::vector<std::weak_ptr<bug>> reported_bugs_;
};

#pragma db object
class bug
{
  ...

  std::shared_ptr<user> reporter_;
};
```

Another Problem

```
CREATE TABLE bug (
  ...
  reporter TEXT NULL,
  CONSTRAINT reporter_fk
    FOREIGN KEY (reporter)
    REFERENCES user (email));


CREATE TABLE user_reported_bugs (
  ...
  bug_id BIGINT NULL,
  CONSTRAINT bug_id_fk
    FOREIGN KEY (bug_id)
    REFERENCES bug (id));
```

Another Problem

```
CREATE TABLE bug (
  ...
  reporter TEXT NULL,
  CONSTRAINT reporter_fk
    FOREIGN KEY (reporter)
    REFERENCES user (email));


CREATE TABLE user_reported_bugs (
  ...
  bug_id BIGINT NULL,
  CONSTRAINT bug_id_fk
    FOREIGN KEY (bug_id)
    REFERENCES bug (id));
```

Another Problem

```
#pragma db object
class user
{
  ...

  #pragma db inverse(reporter_)
  std::vector<std::weak_ptr<bug>> reported_bugs_;
};

#pragma db object
class bug
{
  ...

  std::shared_ptr<user> reporter_;
};
```

Adding Bug Reporter and Bug List (Qt)

```
#pragma db object
class User
{
  ...

  #pragma db inverse(reporter_)
  QList<QWeakPointer<Bug>> reportedBugs_;
};

#pragma db object
class Bug
{
  ...

  QSharedPointer<User> reporter_;
};
```

Relationships in Queries

```
typedef odb::query<bug> query;

db.query<bug> (query::reporter->last == "Doe");
```

Multi-Database Support

- Static
- Dynamic

Multi-Database Support

- Static
- Dynamic
- Mixed

Multi-Database Support

```
$ odb -m static -d common -d sqlite -d pgsql bug.hxx
```

Multi-Database Support

```
$ odb -m static -d common -d sqlite -d pgsql bug.hxx

$ ls
bug.hxx
bug-odb.cxx  bug-odb-sqlite.cxx bug-odb-pgsql.cxx
bug-odb.hxx  bug-odb-sqlite.hxx bug-odb-pgsql.cxx
bug-odb.ixx  bug-odb-sqlite.ixx bug-odb-pgsql.cxx
```

Static Multi-Database Support

```
#include ”bug-odb-pgsql.hxx”
#include ”bug-odb-sqlite.hxx”

odb::pgsql::database store (...);
odb::sqlite::database cache (...);

std::shared_ptr<bug> b;
{
  odb::transaction t (cache.begin ());
  b = cache.find<bug> (id);
  t.commit ();
}

if (b == nullptr)
{
  odb::transaction t (store.begin ());
  b = store.load<bug> (id);
  t.commit ();
}
```

Static Multi-Database Support

```
#include ”bug-odb-pgsql.hxx”
#include ”bug-odb-sqlite.hxx”

odb::pgsql::database store (...);
odb::sqlite::database cache (...);

std::shared_ptr<bug> b;
{
  odb::transaction t (cache.begin ());
  b = cache.find<bug> (id);
  t.commit ();
}

if (b == nullptr)
{
  odb::transaction t (store.begin ());
  b = store.load<bug> (id);
  t.commit ();
}
```

Static Multi-Database Support

```
#include ”bug-odb-pgsql.hxx”
#include ”bug-odb-sqlite.hxx”

odb::pgsql::database store (...);
odb::sqlite::database cache (...);

std::shared_ptr<bug> b;
{
  odb::transaction t (cache.begin ());
  b = cache.find<bug> (id);
  t.commit ();
}

if (b == nullptr)
{
  odb::transaction t (store.begin ());
  b = store.load<bug> (id);
  t.commit ();
}
```

Static Multi-Database Support

```
#include ”bug-odb-pgsql.hxx”
#include ”bug-odb-sqlite.hxx”

odb::pgsql::database store (...);
odb::sqlite::database cache (...);

std::shared_ptr<bug> b;
{
  odb::transaction t (cache.begin ());
  b = cache.find<bug> (id);
  t.commit ();
}

if (b == nullptr)
{
  odb::transaction t (store.begin ());
  b = store.load<bug> (id);
  t.commit ();
}
```

Static Multi-Database Support

```
#include ”bug-odb-pgsql.hxx”
#include ”bug-odb-sqlite.hxx”

odb::pgsql::database store (...);
odb::sqlite::database cache (...);

std::shared_ptr<bug> b;
{
  odb::transaction t (cache.begin ());
  b = cache.find<bug> (id);
  t.commit ();
}

if (b == nullptr)
{
  odb::transaction t (store.begin ());
  b = store.load<bug> (id);
  t.commit ();
}
```

Dynamic Multi-Database Support

```
#include "bug-odb.hxx"

std::shared_ptr<bug>
find_bug (odb::database& db, unsigned long long id)
{
  odb::transaction t (db.begin ());
  std::shared_ptr<bug> r (db.find<bug> (id));
  t.commit ();
  return r;
}

odb::pgsql::database store (...);
odb::sqlite::database cache (...);

std::shared_ptr<bug> b (find_bug (cache, id));

if (b == nullptr)
  b = find_bug (store, id);
```

Dynamic Multi-Database Support

```
#include "bug-odb.hxx"

std::shared_ptr<bug>
find_bug (odb::database& db, unsigned long long id)
{
  odb::transaction t (db.begin ());
  std::shared_ptr<bug> r (db.find<bug> (id));
  t.commit ();
  return r;
}

odb::pgsql::database store (...);
odb::sqlite::database cache (...);

std::shared_ptr<bug> b (find_bug (cache, id));

if (b == nullptr)
  b = find_bug (store, id);
```

Dynamic Multi-Database Support

```
#include "bug-odb.hxx"

std::shared_ptr<bug>
find_bug (odb::database& db, unsigned long long id)
{
  odb::transaction t (db.begin ());
  std::shared_ptr<bug> r (db.find<bug> (id));
  t.commit ();
  return r;
}

odb::pgsql::database store (...);
odb::sqlite::database cache (...);

std::shared_ptr<bug> b (find_bug (cache, id));

if (b == nullptr)
  b = find_bug (store, id);
```

Dynamic Multi-Database Support

```
#include "bug-odb.hxx"

std::shared_ptr<bug>
find_bug (odb::database& db, unsigned long long id)
{
  odb::transaction t (db.begin ());
  std::shared_ptr<bug> r (db.find<bug> (id));
  t.commit ();
  return r;
}

odb::pgsql::database store (...);
odb::sqlite::database cache (...);

std::shared_ptr<bug> b (find_bug (cache, id));

if (b == nullptr)
  b = find_bug (store, id);
```

Dynamic Loading

```
void
load_db (const std::string& db_name)
{
#ifdef _WIN32
  string dll ("bug-" + db_name + ".dll");
  HMODULE h (LoadLibraryA (dll.c_str ()));
#else
  string so ("libbug-" + db_name + ".so");
  void* h (dlopen (so.c_str (), RTLD_NOW));
#endif

  if (h == 0)
  {
    // Handle error.
  }
}
```

Database Schema Evolution

- No magic
- Simple, easy to understand building blocks
- Schema migration
- Data migration

Object Model Version

```
#pragma db model version(1, 1)

#pragma db object
class bug
{
  ...
};
```

Object Model Version

```
#pragma db model version(1, 1)

#pragma db object
class bug
{
  ...
};
```

```
#pragma db model version(1, 2)

#pragma db object
class bug
{
  ...

  std::string platform_;
};
```

Changelog

- XML file (human reviewable)
- Base model + changeset for each version
- Stored in source code repository

Changelog

- XML file (human reviewable)
- Base model + changeset for each version
- Stored in source code repository

```
<changeset version="2">
  <alter-table name="bug">
    <add-column name="platform" type="TEXT" null="false"/>
  </alter-table>
</changeset>

<model version="1">
  ...
</model>
```

Schema Migration

- SQL files or embedded into C++ code
- Pre and Post (`bug-002-pre.sql` and `bug-002-post.sql`)
- Pre-migration *relaxes* the schema
- Post-migration *tightens* it back

Schema Migration

- SQL files or embedded into C++ code
- Pre and Post (`bug-002-pre.sql` and `bug-002-post.sql`)
- Pre-migration *relaxes* the schema
- Post-migration *tightens* it back
- Data migration fits between the two

Schema Migration

```
/* bug-002-pre.sql */

ALTER TABLE bug
  ADD COLUMN platform TEXT NULL;
```

Schema Migration

```sql
/* bug-002-pre.sql */

ALTER TABLE bug
  ADD COLUMN platform TEXT NULL;
```

```sql
/* bug-002-post.sql */

ALTER TABLE bug
  ALTER COLUMN platform SET NOT NULL;
```

Data Migration

```
transaction t (db.begin ());

schema_catalog::migrate_schema_pre (db, 2);

for (bug& b: db.query<bug> ())
{
  b.platform (”Unknown”);
  db.update (b);
}

schema_catalog::migrate_schema_post (db, 2);

t.commit ();
```

Data Migration

```
transaction t (db.begin ());

schema_catalog::migrate_schema_pre (db, 2);

for (bug& b: db.query<bug> ())
{
  b.platform (”Unknown”);
  db.update (b);
}

schema_catalog::migrate_schema_post (db, 2);

t.commit ();
```

Data Migration

```
transaction t (db.begin ());

schema_catalog::migrate_schema_pre (db, 2);

for (bug& b: db.query<bug> ())
{
  b.platform (”Unknown”);
  db.update (b);
}

schema_catalog::migrate_schema_post (db, 2);

t.commit ();
```

Data Migration

```
schema_catalog::data_migration_function (
  2,
  [] (database& db)
  {
    for (bug& b: db.query<bug> ())
    {
      b.platform (”Unknown”);
      db.update (b);
    }
  });


transaction t (db.begin ());
schema_catalog::migrate (db);
t.commit ();
```

Schema Evolution

```
#pragma db model version(1, 2)

#pragma db object
class user
{
  std::string first_;
  std::string last_;
};
```

Schema Evolution

```
#pragma db model version(1, 2)

#pragma db object
class user
{
  std::string first_;
  std::string last_;
};


#pragma db model version(1, 3)

#pragma db object
class user
{
  std::string name_;
};
```

Changelog Diff

```
+ <changeset version="3">
+   <alter-table name="user">
+     <add-column name="name" type="TEXT" null="false"/>
+     <drop-column name="first"/>
+     <drop-column name="last"/>
+   </alter-table>
+ </changeset>
```

Data Migration

```
schema_catalog::data_migration_function (
  3,
  [] (database& db)
  {
    for (bug& b: db.query<bug> ())
    {
      b.name (b.first () + " " + b.last ());
      db.update (b);
    }
  });
```

Data Migration

```
schema_catalog::data_migration_function (
  3,
  [] (database& db)
  {
    for (bug& b: db.query<bug> ())
    {
      b.name (b.first () + " " + b.last ());
      db.update (b);
    }
  });
```

Resources

- ODB Page
  - www.codesynthesis.com/products/odb/
- ODB Manual
  - www.codesynthesis.com/products/odb/doc/manual.xhtml
- Blog
  - www.codesynthesis.com/~boris/blog/