ODB, an ORM for C++(11)

Boris Kolpackov

Code Synthesis

v1.1, May 2013

No, thanks

- No manual parameter binding
- No manual result set extraction
- No hand-written mapping or registration code
- No magic
- Not a framework

ORM for C++

Yes, please

- Automatic generation of database code from C++ classes
- Handle any standard C++
- Object-oriented database API
- Statically-typed, C++-integrated query language
- Database portability
- Database schema evolution support
- Flexible and customizable

Databases

Cross-database

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

Databases

Cross-database

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

- *Multi-database support*

C++ Standards

C++98, TR1, and C++11

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

C++ Support

ODB is implemented as a GCC plugin

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

C++ Support

ODB is implemented as a GCC plugin

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

- C++ in, C++ out
- Use any C++ compiler to build your application

Platforms & Compilers

ODB works with any modern C++ compiler

- GNU/Linux with GCC 4.2 and up, Clang 3.x
- Mac OS X with GCC 4.2 and up, Clang/XCode
- Solaris (x86 and SPARC) with Sun Studio C++ 12.2 and up
- Windows with GCC (MinGW) or VC++ 2008, 2010, 2012


Ultra II
SanDisk
2.0 GB

Mobile & Embedded

- ODB + SQLite
- “Hello, World” example is 533Kb
- Cross-compiler friendly
- Raspberry Pi-based guide

Performance

High-performance and low overhead

- Prepared statements, including custom queries
- Caching of connections, statements, and buffers
- Low-level native database 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 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

Declaring Persistent Classes

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

class bug
{
public:
  ...

private:
  unsigned long long id_;

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

Declaring Persistent Classes

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

#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_;
};
```

Declaring Persistent Classes

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

#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_;
};
```

Declaring Persistent Classes

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

#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_;
};
```

Declaring Persistent Classes

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

#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_;
};
```

Declaring Persistent Classes

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

#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_;
};
```

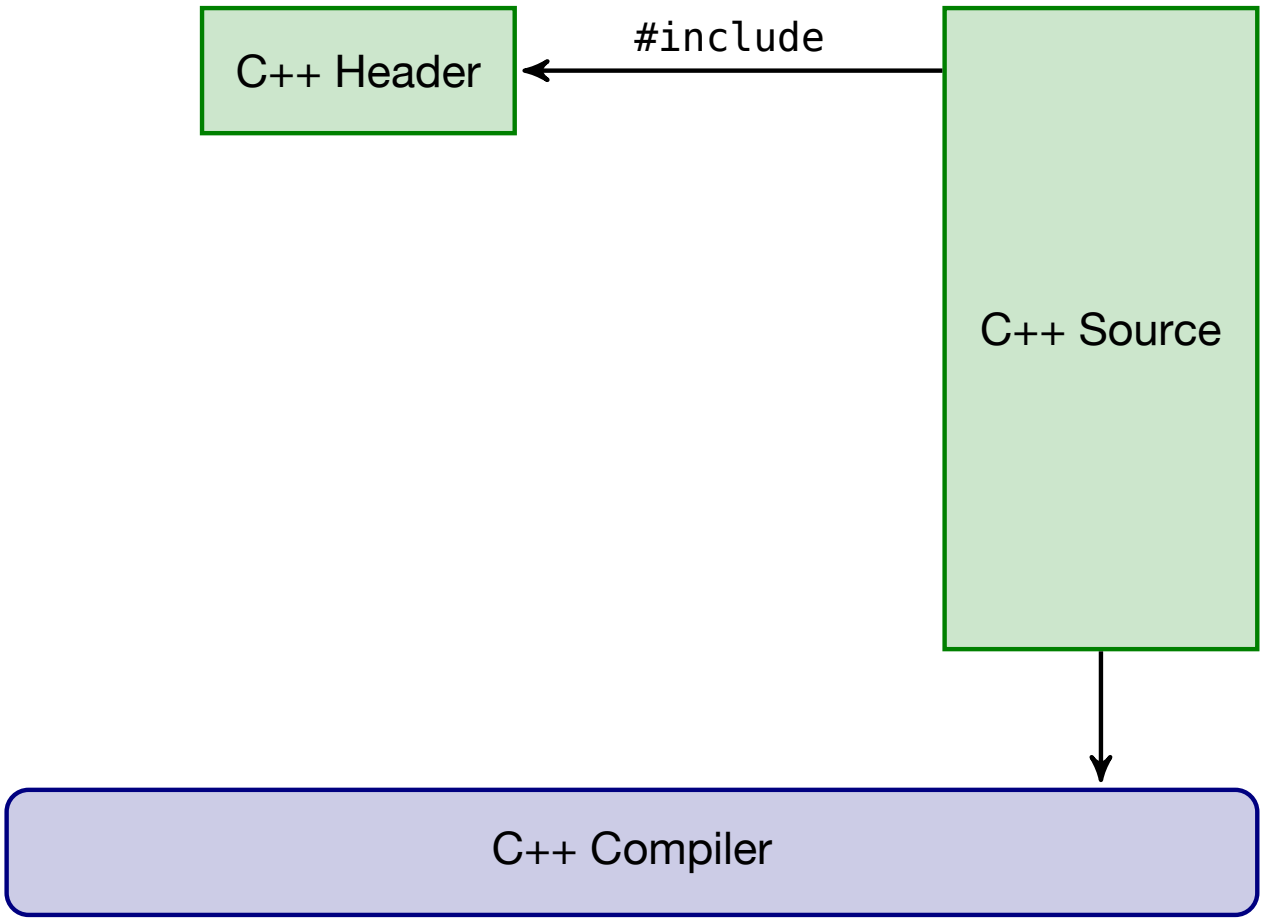
Declaring Persistent Classes

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

#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_;
};
```

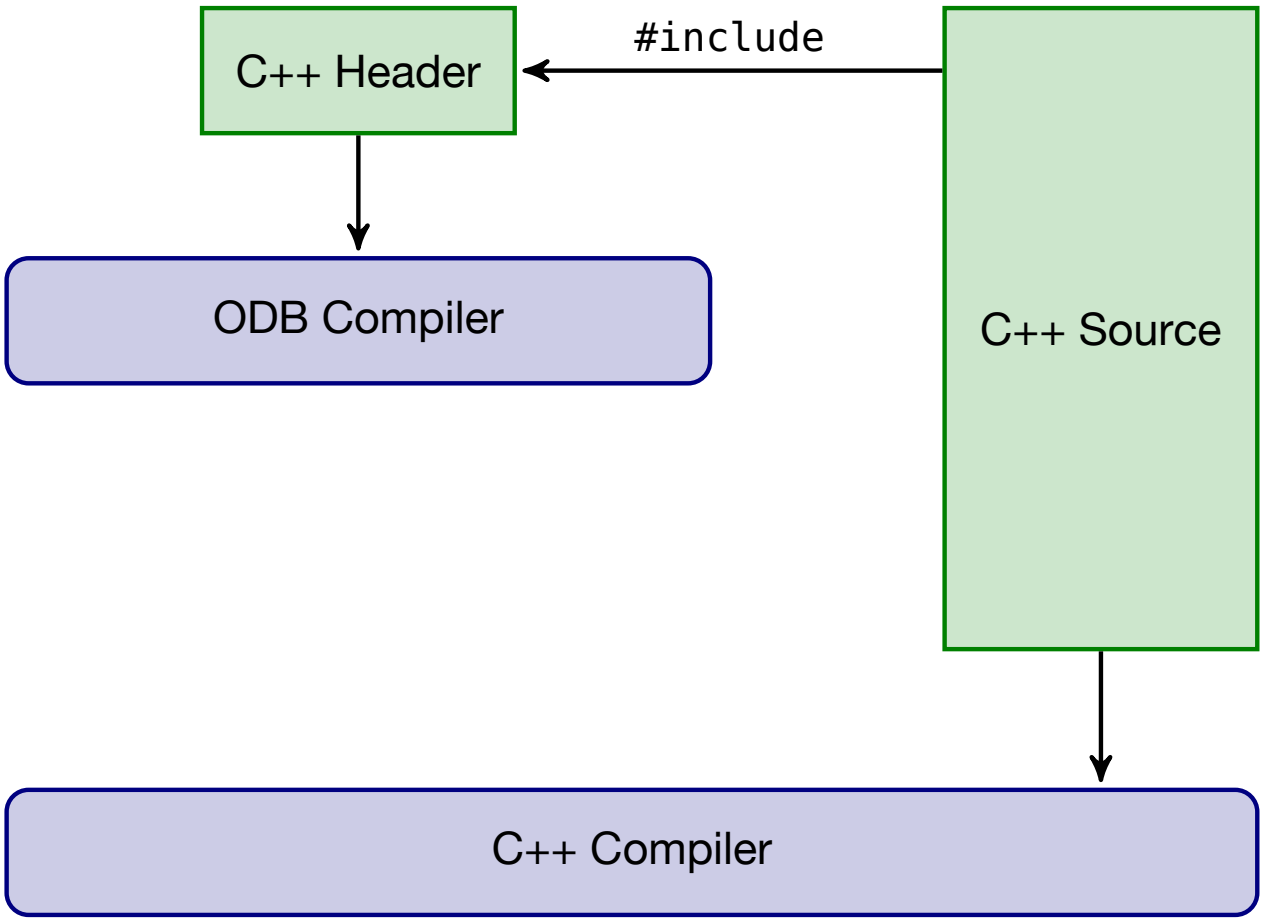
Declaring Persistent Classes

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

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

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

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

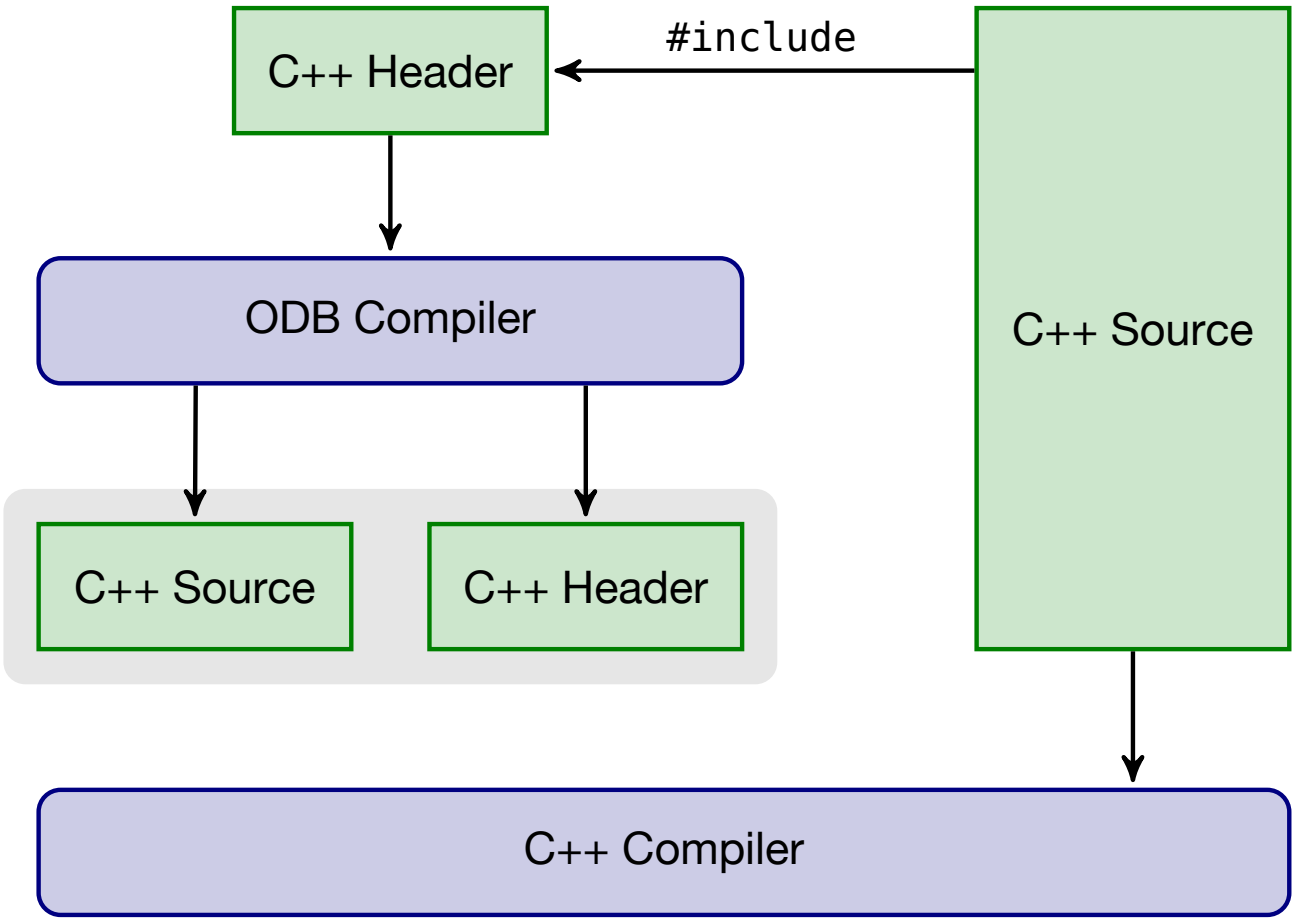
Workflow



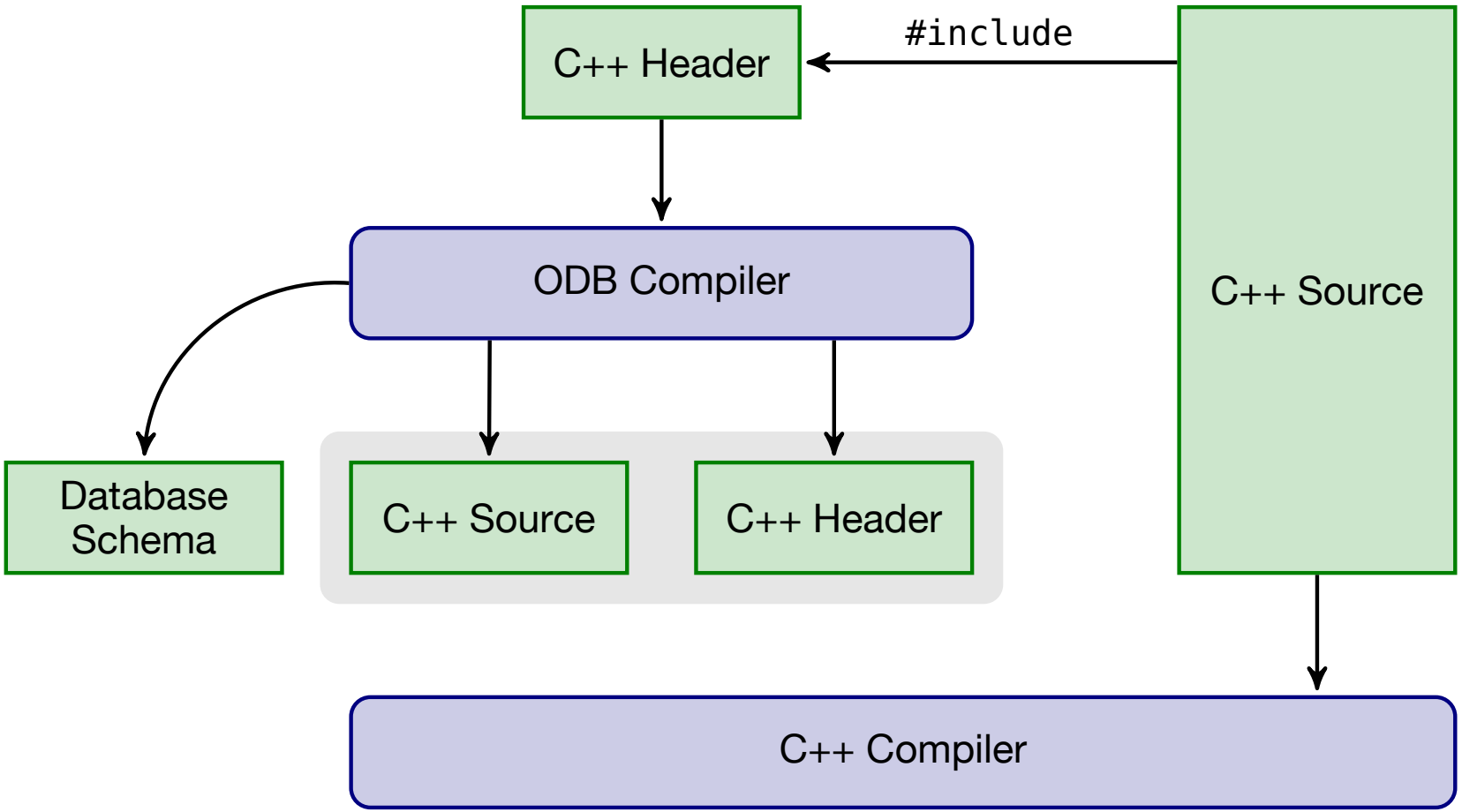
Workflow



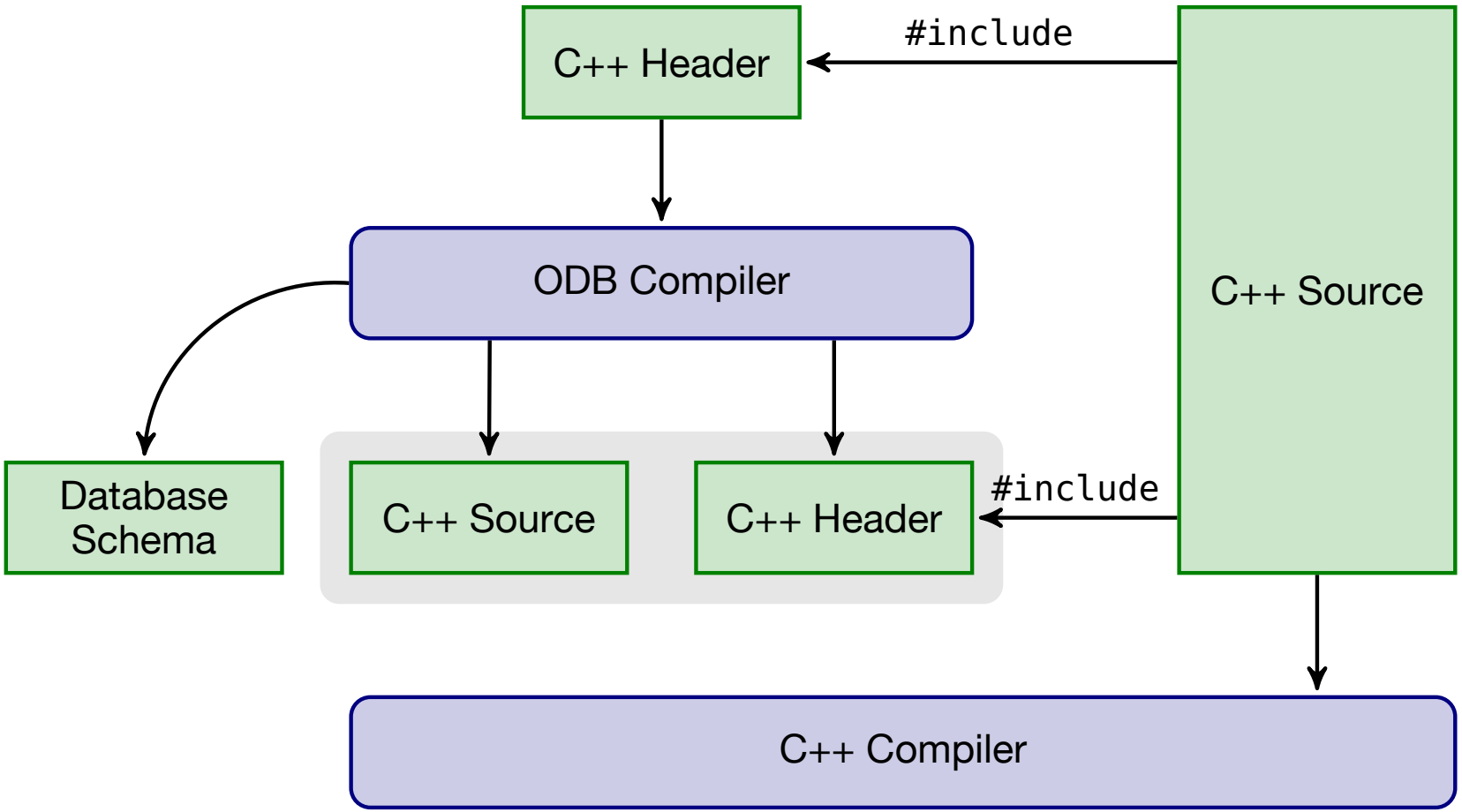
Workflow


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

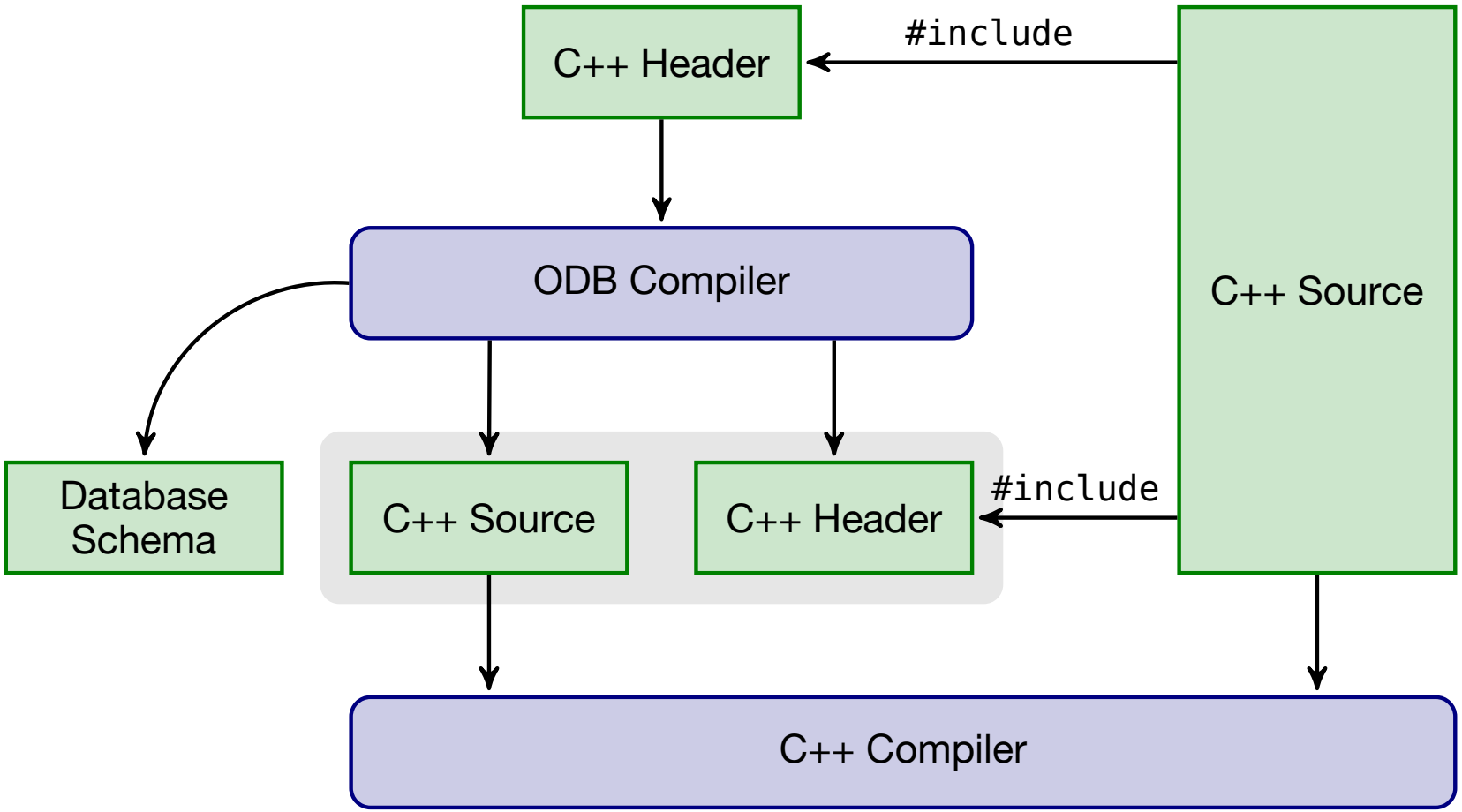
Workflow


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

odb -I/opt/boost-latest -d sqlite bug.hxx
```

ODB Compiler

```
odb --database pgsql bug.hxx

odb -I/opt/boost-latest -d sqlite bug.hxx

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

ODB Compiler

```
odb --database pgsql bug.hxx

odb -I/opt/boost-latest -d sqlite bug.hxx

odb --std=c++11 --default-pointer std::shared_ptr ...

odb -d pgsql --generate-schema bug.hxx
```

ODB Compiler

```
odb --database pgsql bug.hxx

odb -I/opt/boost-latest -d sqlite bug.hxx

odb --std=c++11 --default-pointer std::shared_ptr ...

odb -d pgsql --generate-schema bug.hxx
```

```
CREATE TABLE bug (
  id BIGSERIAL NOT NULL PRIMARY KEY,
  status INTEGER NOT NULL,
  summary TEXT NOT NULL,
  description TEXT NOT NULL);
```

Database

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

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

Database

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

odb::pgsql::database db ("bugtracker", // 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 ("bugtracker", // 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);
```

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;

transaction t (db.begin ());

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

for (result::iterator i (r.begin ()); i != r.end (); ++i)
  cout << i->id () << " " << i->summary () << endl;

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 (result::iterator i (r.begin ()); i != r.end (); ++i)
  cout << i->id () << " " << i->summary () << endl;

t.commit ();
```

Querying the Database

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

transaction t (db.begin ());

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

t.commit ();
```

Querying the Database

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

transaction t (db.begin ());

for (bug& 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));
```

Prepared Queries

```
typedef odb::query<bug> query;
typedef odb::prepared_query<person> prep_query;

transaction t (db.begin ());

status s;
query q (query::status == query::_ref (s));
prep_query pq (db.prepare_query<bug> ("bug-query", q));

s = open;
pq.execute ();

s = confirmed;
pq.execute ();

...

t.commit ();
```

Prepared Queries

```
typedef odb::query<bug> query;
typedef odb::prepared_query<person> prep_query;

transaction t (db.begin ());

status s;
query q (query::status == query::_ref (s));
prep_query pq (db.prepare_query<bug> ("bug-query", q));

s = open;
pq.execute ();

s = confirmed;
pq.execute ();

...

t.commit ();
```

Prepared Queries

```
typedef odb::query<bug> query;
typedef odb::prepared_query<person> prep_query;

transaction t (db.begin ());

status s;
query q (query::status == query::_ref (s));
prep_query pq (db.prepare_query<bug> ("bug-query", q));

s = open;
pq.execute ();

s = confirmed;
pq.execute ();

...

t.commit ();
```

Prepared Queries

```
typedef odb::query<bug> query;
typedef odb::prepared_query<person> prep_query;

transaction t (db.begin ());

status s;
query q (query::status == query::_ref (s));
prep_query pq (db.prepare_query<bug> ("bug-query", q));

s = open;
pq.execute ();

s = confirmed;
pq.execute ();

...

t.commit ();
```

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 Creation and Modification Dates

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

  #pragma db id auto
  unsigned long long id_;

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



};
```

Adding Creation and Modification Dates

```
#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 write your own

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

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

Boost Profile

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

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_;
};
```

Change-Tracking Containers

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

std::shared_ptr<bug> b (db.load<bug> (id));
b->add_comment ("I also have this problem! Help me!");
db.update (b);

t.commit ();
```

Change-Tracking Containers

- Drop-in replacements for ordinary containers
- `odb::vector` equivalent for `std::vector`
- `QOdbList` equivalent for `QList`
- 2-bit per element overhead

Composite Value Types

- Class or struct type
- Mapped to more than one database column
- Can contain composite values, containers, and pointers to objects
- Can be used as 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_;
};
```

Example of a *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_;
};
```

Example of a *bidirectional many-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_;

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

Example of a *bidirectional many-to-one* relationship.

We Have a Problem

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

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

#pragma db object
class bug
{
  ...

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

We Have a Problem

Actually, we have two:

1. Ownership cycle between `user` and `bug`
2. Eager loading of the `bug` list in `user`

We Have a Problem

Actually, we have two:

1. Ownership cycle between `user` and `bug`
2. Eager loading of the `bug` list in `user`

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

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

Lazy Pointers

- Finer-grained control over relationship loading
- Every supported pointer has a corresponding lazy version

Lazy Pointers

- Finer-grained control over relationship loading
- Every supported pointer has a corresponding lazy version

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

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

odb::lazy_weak_ptr<bug> lb = ...
std::shared_ptr<bug> b (lb.load ()); // Load and lock.
```

Adding Bug Reporter and Bug List (Qt)

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

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

#pragma db object
class Bug
{
  ...

  QSharedPointer<User> reporter_;
};
```

Views

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

transaction t (db.begin ());

for (const bug& b: db.query<bug> (query::status == open))
{
  const user& r (b.reporter ());

  cout << b.id () << " "
       << b.summary () << " "
       << r.first_name () << " "
       << r.last_name () << endl;
}

t.commit ();
```

Views

- Load a subset of data members from objects/tables
- Join multiple objects/tables
- Handle results of arbitrary SQL queries (aggregate, etc)

Declaring Views

```
#pragma db view object(bug) object(user)
struct bug_summary
{
  unsigned long long id;
  std::string summary;
  std::string first_name;
  std::string last_name;
};
```

Using Views

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

for (const bug_summary& b:
       db.query<bug_summary> (query::bug::status == open))
{
  cout << b.id << " "
       << b.summary << " "
       << b.first_name << " "
       << b.last_name << endl;
}
```

Using Views

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

for (const bug_summary& b:
       db.query<bug_summary> (query::bug::status == open))
{
  cout << b.id << " "
       << b.summary << " "
       << b.first_name << " "
       << b.last_name << endl;
}
```

```
=> SELECT bug.id, bug.summary,
          user.first_name, user.last_name
   FROM bug LEFT JOIN user ON bug.reporter = user.email
   WHERE bug.status = $1
```

Optimistic Concurrency

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

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

cout << "current status: " << b->status () << endl
     << "enter new status: ";

status s;
cin >> s;

b->status (s);
db.update (b);

t.commit ();
```

Optimistic Concurrency

```
std::shared_ptr<bug> b;
{
  transaction t (db.begin ());
  b = db.load<bug> (id);
  t.commit ();
}

cout << "current status: " << b->status () << endl
     << "enter new status: ";

status s;
cin >> s;
b->status (s);

{
  transaction t (db.begin ());
  db.update (b);
  t.commit ();
}
```

Optimistic Concurrency

- ”Hope for the best, prepare for the worst”
- ODB uses object versioning
- Works best for low to medium contention levels

Declaring Optimistic Classes

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

  #pragma db id auto
  unsigned long long id_;

  #pragma db version
  unsigned long long version_;

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

Declaring Optimistic Classes

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

  #pragma db id auto
  unsigned long long id_;

  #pragma db version
  unsigned long long version_;

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

Using Optimistic Classes

```
...

for (bool done (false); !done;)
{
  cout << "current status: " << b->status () << endl
       << "enter new status: ";

  cin >> s;
  b->status (s);

  transaction t (db.begin ());

  try{ db.update (b); done = true; }
  catch (const odb::object_changed&) { db.reload (b); }

  t.commit ();
}
```

Polymorphism

```
class issue
{
  unsigned long long id_;

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

class bug: public issue
{
  std::string platform_;
};

class feature: public issue
{
  unsigned int votes_;
};
```

Polymorphism

- Table-per-difference mapping
- Can use static or dynamic types in database operations

Declaring Polymorphic Classes

```
#pragma db object polymorphic
class issue
{
  ...

  virtual ~issue () = 0;

  #pragma db id auto
  unsigned long long id_;

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

Declaring Polymorphic Classes

```
#pragma db object polymorphic
class issue
{
  ...

  virtual ~issue () = 0;

  #pragma db id auto
  unsigned long long id_;

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

Declaring Polymorphic Classes

```
#pragma db object
class bug: public issue
{
  ...

  std::string platform_;
};

#pragma db object
class feature: public issue
{
  ...

  unsigned int votes_;
};
```

Using Polymorphic Classes

```
std::shared_ptr<issue> i (new bug (...));

transaction t (db.begin ());

db.persist (i); // Persist bug.

i->status (confirmed);
db.update (i);  // Update bug.

db.reload (i);  // Reload bug.

t.commit ();
```

Using Polymorphic Classes

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

transaction t (db.begin ());

// Load bug or feature.
std::shared_ptr<issue> i (db.load<issue> (id));

for (const issue& i:
       db.query<issue> (query::status == open))
  // i is either bug or feature.

db.query<issue> (query::status == open)   // Both.
db.query<bug> (query::status == open)     // Bugs.
db.query<feature> (query::status == open) // Features.

t.commit ();
```

Using Polymorphic Classes

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

transaction t (db.begin ());

// Load bug or feature.
std::shared_ptr<issue> i (db.load<issue> (id));

for (const issue& i:
       db.query<issue> (query::status == open))
  // i is either bug or feature.

db.query<issue> (query::status == open)   // Both.
db.query<bug> (query::status == open)     // Bugs.
db.query<feature> (query::status == open) // Features.

t.commit ();
```

Using Polymorphic Classes

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

transaction t (db.begin ());

// Load bug or feature.
std::shared_ptr<issue> i (db.load<issue> (id));

for (const issue& i:
       db.query<issue> (query::status == open))
  // i is either bug or feature.

db.query<issue> (query::status == open)   // Both.
db.query<bug> (query::status == open)     // Bugs.
db.query<feature> (query::status == open) // Features.

t.commit ();
```

Using Polymorphic Classes

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

transaction t (db.begin ());

// Load bug or feature.
std::shared_ptr<issue> i (db.load<issue> (id));

for (const issue& i:
       db.query<issue> (query::status == open))
  // i is either bug or feature.

db.query<issue> (query::status == open)   // Both.
db.query<bug> (query::status == open)     // Bugs.
db.query<feature> (query::status == open) // Features.

t.commit ();
```

Using Polymorphic Classes

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

transaction t (db.begin ());

// Load bug or feature.
std::shared_ptr<issue> i (db.load<issue> (id));

for (const issue& i:
       db.query<issue> (query::status == open))
  // i is either bug or feature.

db.query<issue> (query::status == open)   // Both.
db.query<bug> (query::status == open)     // Bugs.
db.query<feature> (query::status == open) // Features.

t.commit ();
```

Database Schema

- Database schema can be automatically generated
- Or we can map persistent classes to a custom schema

Generated Schema

- Standalone SQL file
- Embedded into the generated C++ code

Generated Schema

- Standalone SQL file
- Embedded into the generated C++ code

```
#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_;
};
```

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
- 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)
  {
    typedef odb::query<bug> query;

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

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

Other Features

- `NULL` mapping to pointers, `odb::nullable`, or `boost::optional`
- Session (object cache)
- Database indexes
- Virtual data members and `pimpl` idiom

Customizations

- Custom value types
- Custom containers
- Custom smart pointers
- Custom `NULL` wrappers
- Custom session
- Custom profiles
- Extended database type mapping
- Per-class database operations callback
- Connection management (connection pool by default)

Future

- SQL to C++ compiler
- Generalized lazy loading, including containers
- Bulk operations
- Additional schema migration operations

Maybe Future

- Support more databases (DB2, Firebird)
- Support NoSQL databases, MongoDB
- Persistence to XML, JSON
- Sharding

Resources

- ODB home page
  - www.codesynthesis.com/products/odb/
- ODB manual
  - www.codesynthesis.com/products/odb/doc/manual.xhtml
- My Blog
  - www.codesynthesis.com/~boris/blog/