ODB, Advanced Weapons and Tactics

Boris Kolpackov

Code Synthesis

v1.0, Sep 2014

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_;
};
```

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);
    }
  });
```

Versioned Namespace?

```
namespace version2
{
  #pragma db object
  class user
  {
    std::string first_;
    std::string last_;
  };
}
```

Soft-Delete

```
#pragma db model version(1, 3)

#pragma db object
class user
{
  #pragma db deleted(3)
  std::string first_;

  #pragma db deleted(3)
  std::string last_;

  std::string name_;
};
```

Soft-Delete

```
#pragma db model version(1, 3)

#pragma db object
class user
{
  #pragma db deleted(3)
  std::string first_;

  #pragma db deleted(3)
  std::string last_;

  std::string name_;
};
```

Soft-Delete

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

  #pragma db value
  struct deleted_data
  {
    #pragma db deleted(3)
    std::string first_;

    #pragma db deleted(3)
    std::string last_;
  };

  #pragma db column("")
  std::unique_ptr<deleted_data> dd_;
};
```

Soft-Add

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

  #pragma db value
  struct deleted_data
  {
    #pragma db deleted(3)
    std::string first_;

    #pragma db deleted(3)
    std::string last_;
  };

  #pragma db column("")
  std::unique_ptr<deleted_data> dd_;
};
```

Soft-Add

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

Soft-Add

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

Soft-Add

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

Soft-Add

```
class user
{
  #pragma db added(3)
  std::string name_;

  #pragma db value
  struct deleted_data
  {
    #pragma db deleted(3)
    std::string first_;

    #pragma db deleted(3)
    std::string last_;
  };

  #pragma db column("")
  std::unique_ptr<deleted_data> dd_;
};
```

Containers

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

  #pragma db id auto
  unsigned long long id_;

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

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

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

Containers

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

  odb::vector<std::string> comments_;
};
```

Object Cache

```
#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_;
};
```

Object Cache

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

std::shared_ptr<user> u (db.load<user> (email));

t.commit ();
```

Object Cache

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

std::shared_ptr<user> u (db.load<user> (email));

t.commit ();
```

Object Cache

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

std::shared_ptr<user> u (db.load<user> (email));

t.commit ();
```

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.
```

Object Sections

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

  std::string description_;
  odb::vector<std::string> comments_;
};
```

Object Sections

- Load: `eager` or `lazy`
- Update: `always`, `change`, `manual`
- 1 byte overhead

Object Sections

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

  #pragma db load(lazy) update(change)
  odb::section details_;

  #pragma db section(details_)
  std::string description_;

  #pragma db section(details_)
  odb::vector<std::string> comments_;
};
```

Object Sections

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

  #pragma db load(lazy) update(change)
  odb::section details_;

  #pragma db section(details_)
  std::string description_;

  #pragma db section(details_)
  odb::vector<std::string> comments_;
};
```

Object Sections

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

  #pragma db load(lazy) update(change)
  odb::section details_;

  #pragma db section(details_)
  std::string description_;

  #pragma db section(details_)
  odb::vector<std::string> comments_;
};
```

Object Sections

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

  #pragma db load(lazy) update(change)
  odb::section details_;

  #pragma db section(details_)
  std::string description_;

  #pragma db section(details_)
  odb::vector<std::string> comments_;
};
```

Object Sections

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

for (bug& b: db.query<bug> (query::status == open))
{
  if (is_interesting (b))
  {
    db.load (b, b.details_);
    ...

    b.comments_.push_back ("I am working on a fix.");
    b.details_.change ();
  }
  ...

  db.update (b);
}

t.commit ();
```

Object Sections

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

for (bug& b: db.query<bug> (query::status == open))
{
  if (is_interesting (b))
  {
    db.load (b, b.details_);
    ...

    b.comments_.push_back ("I am working on a fix.");
    b.details_.change ();
  }
  ...

  db.update (b);
}

t.commit ();
```

Object Sections

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

for (bug& b: db.query<bug> (query::status == open))
{
  if (is_interesting (b))
  {
    db.load (b, b.details_);
    ...

    b.comments_.push_back ("I am working on a fix.");
    b.details_.change ();
  }
  ...

  db.update (b);
}

t.commit ();
```

Object Sections

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

for (bug& b: db.query<bug> (query::status == open))
{
  if (is_interesting (b))
  {
    db.load (b, b.details_);
    ...

    b.comments_.push_back ("I am working on a fix.");
    b.details_.change ();
  }
  ...

  db.update (b);
}

t.commit ();
```

Object Sections

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

for (bug& b: db.query<bug> (query::status == open))
{
  if (is_interesting (b))
  {
    db.load (b, b.details_);
    ...

    b.comments_.push_back ("I am working on a fix.");
    b.details_.change ();
  }
  ...

  db.update (b);
}

t.commit ();
```

Object Sections

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

  #pragma db load(lazy)
  odb::section details_;

  #pragma db section(details_)
  std::vector<odb::lazy_weak_ptr<bug>> reported_bugs_;
};
```

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 () << " "
       << r.last () << endl;
}

t.commit ();
```

Views

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

Declaring Views

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

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 << " "
       << b.last << 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 << " "
       << b.last << endl;
}
```

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

Aggregate View

```
#pragma db view object(bug)
struct bug_stats
{
  #pragma db column("COUNT(" + bug::id_ + ")")
  std::size_t count;
};
```

Aggregate View

```
#pragma db view object(bug)
struct bug_stats
{
  #pragma db column("COUNT(" + bug::id_ + ")")
  std::size_t count;
};
```

Aggregate View

```
#pragma db view object(bug)
struct bug_stats
{
  #pragma db column("COUNT(" + bug::id_ + ")")
  std::size_t count;
};


typedef odb::query<bug_stats> query;

bug_stats bs (
  *db.query<bug_stats> (
    query::status == closed).begin ());
```

Aggregate View

```
#pragma db view object(user) object(bug) \
  query ((?) + "GROUP BY" + user::email_)
struct user_stats
{
  std::string first;
  std::string last;

  #pragma db column("COUNT(" + bug::id_ + ")")
  std::size_t count;
};
```

Aggregate View

```
#pragma db view object(user) object(bug) \
  query ((?) + "GROUP BY" + user::email_)
struct user_stats
{
  std::string first;
  std::string last;

  #pragma db column("COUNT(" + bug::id_ + ")")
  std::size_t count;
};
```

Aggregate View

```
#pragma db view object(user) object(bug) \
  query ((?) + "GROUP BY" + user::email_)
struct user_stats
{
  std::string first;
  std::string last;

  #pragma db column("COUNT(" + bug::id_ + ")")
  std::size_t count;
};


for (const user_stats& us:
       db.query<user_stats> (
         query::user::last == "Doe" &&
         query::bug::status == open))
{
  ...
}
```

Stored Procedure Call

```
#pragma db view query("EXEC analyze_bugs (?)")
struct report
{
  unsigned long long id;
  std::string result;
};
```

Stored Procedure Call

```
#pragma db view query("EXEC analyze_bugs (?)")
struct report
{
  unsigned long long id;
  std::string result;
};
```

Stored Procedure Call

```
#pragma db view query("EXEC analyze_bugs (?)")
struct report
{
  unsigned long long id;
  std::string result;
};


typedef odb::query<report> query;

db.query<report> (query::_val ("abc") + "," +
                  query::_val (123));
```

Native Query

```
#pragma db view
struct sequence_value
{
  unsigned long long value;
};
```

Native Query

```
#pragma db view
struct sequence_value
{
  unsigned long long value;
};


sequence_value sv (
  *db.query<sequence_value> (
    "SELECT nextval('my_sequence')").begin ());
```

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

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

CREATE TABLE bug(
  id BIGINT NOT NULL PRIMARY KEY,
  platform TEXT NOT NULL,
  CONSTRAINT id_fk FOREIGN KEY(id) REFERENCES issue(id))

CREATE TABLE feature(
  id BIGINT NOT NULL PRIMARY KEY,
  votes INTEGER NOT NULL,
  CONSTRAINT id_fk FOREIGN KEY(id) REFERENCES issue(id))
```

Polymorphism

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

CREATE TABLE bug(
  id BIGINT NOT NULL PRIMARY KEY,
  platform TEXT NOT NULL,
  CONSTRAINT id_fk FOREIGN KEY(id) REFERENCES issue(id))

CREATE TABLE feature(
  id BIGINT NOT NULL PRIMARY KEY,
  votes INTEGER NOT NULL,
  CONSTRAINT id_fk FOREIGN KEY(id) REFERENCES issue(id))
```

Polymorphism

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

CREATE TABLE bug(
  id BIGINT NOT NULL PRIMARY KEY,
  platform TEXT NOT NULL,
  CONSTRAINT id_fk FOREIGN KEY(id) REFERENCES issue(id))

CREATE TABLE feature(
  id BIGINT NOT NULL PRIMARY KEY,
  votes INTEGER NOT NULL,
  CONSTRAINT id_fk FOREIGN KEY(id) REFERENCES issue(id))
```

Polymorphism

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

CREATE TABLE bug(
  id BIGINT NOT NULL PRIMARY KEY,
  platform TEXT NOT NULL,
  CONSTRAINT id_fk FOREIGN KEY(id) REFERENCES issue(id))

CREATE TABLE feature(
  id BIGINT NOT NULL PRIMARY KEY,
  votes INTEGER NOT NULL,
  CONSTRAINT id_fk FOREIGN KEY(id) REFERENCES issue(id))
```

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 ();
```

Bulk Operations

```
template <typename I>
void persist (I begin, I end);

template <typename I>
void update (I begin, I end);

template <typename I>
void erase (I begin, I end);
```

Bulk Operations

```
#pragma db object oracle:bulk(5000) mssql:bulk(7000)
class bug
{
  ...
};
```

Bulk Exceptions

```
catch (const multiple_exceptions& mex)
{
  for (const auto& e: mex)
  {
    cerr << "exception at " << e.position ();
    try
    {
      throw e.exception ();
    }
    catch (const odb::...)
    {
    }
    catch (const odb::...)
    {
    }
  }
}
```

Pimpl Idiom

```
#pragma db object
class bug
{




  ...

private:
  class impl;
  std::unique_ptr<impl> pimpl_;
};
```

Pimpl Idiom

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

  const std::string& summary () const;
  void summary (std::string);

  ...

private:
  class impl;
  std::unique_ptr<impl> pimpl_;
};
```

Virtual Data Members

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

  const std::string& summary () const;
  void summary (std::string);

private:
  class impl;

  #pragma db member(id) virtual(unsigned long long)
  #pragma db member(summary) virtual(std::string)

  #pragma db transient
  std::unique_ptr<impl> pimpl_;
};
```

Virtual Data Members

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

  const std::string& summary () const;
  void summary (std::string);

private:
  class impl;

  #pragma db member(id) virtual(unsigned long long)
  #pragma db member(summary) virtual(std::string)

  #pragma db transient
  std::unique_ptr<impl> pimpl_;
};
```

Virtual Data Members

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

  const std::string& summary () const;
  void summary (std::string);

private:
  class impl;

  #pragma db member(id) virtual(unsigned long long)
  #pragma db member(summary) virtual(std::string)

  #pragma db transient
  std::unique_ptr<impl> pimpl_;
};
```

Virtual Data Members

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

  const std::string& summary () const;
  void summary (std::string);

private:
  class impl;

  #pragma db member(id) virtual(unsigned long long)
  #pragma db member(summary) virtual(std::string)

  #pragma db transient
  std::unique_ptr<impl> pimpl_;
};
```

Virtual Data Members

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

  const std::string& summary () const;
  void summary (std::string);

private:
  class impl;

  #pragma db member(id) virtual(unsigned long long)
  #pragma db member(summary) virtual(std::string)

  #pragma db transient
  std::unique_ptr<impl> pimpl_;
};
```

Accessor/Modifier Expressions

```
#pragma db value
struct name
{
  name (std::string, std::string);
  std::string first, last;
};
#pragma db object
class user
{
  const std::string& first () const;
  const std::string& last () const;
  void first (std::string);
  void last (std::string);

private:


  name name_;
};
```

Accessor/Modifier Expressions

```
#pragma db value
struct name
{
  name (std::string, std::string);
  std::string first, last;
};
#pragma db object
class user
{
  const std::string& first () const;
  const std::string& last () const;
  void first (std::string);
  void last (std::string);

private:


  name name_;
};
```

Accessor/Modifier Expressions

```
#pragma db value
struct name
{
  name (std::string, std::string);
  std::string first, last;
};
#pragma db object
class user
{
  const std::string& first () const;
  const std::string& last () const;
  void first (std::string);
  void last (std::string);

private:


  name name_;
};
```

Accessor/Modifier Expressions

```
#pragma db value
struct name
{
  name (std::string, std::string);
  std::string first, last;
};
#pragma db object
class user
{
  const std::string& first () const;
  const std::string& last () const;
  void first (std::string);
  void last (std::string);

private:
  #pragma db get(name (this.first (), this.last ())) \
    set(this.first ((?).first); this.last ((?).last))
  name name_;
};
```

Accessor/Modifier Expressions

```
#pragma db value
struct name
{
  name (std::string, std::string);
  std::string first, last;
};
#pragma db object
class user
{
  const std::string& first () const;
  const std::string& last () const;
  void first (std::string);
  void last (std::string);

private:
  #pragma db get(name (this.first (), this.last ())) \
    set(this.first ((?).first); this.last ((?).last))
  name name_;
};
```

Accessor/Modifier Expressions

```
#pragma db value
struct name
{
  name (std::string, std::string);
  std::string first, last;
};
#pragma db object
class user
{
  const std::string& first () const;
  const std::string& last () const;
  void first (std::string);
  void last (std::string);

private:
  #pragma db get(name (this.first (), this.last ())) \
    set(this.first ((?).first); this.last ((?).last))
  name name_;
};
```

Index Definition

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

  #pragma db id
  std::string email_;

  #pragma db index
  std::string first_;

  std::string last_;

  #pragma db index("name_i") \
    unique                   \
    method("BTREE")          \
    members(first_, last_)
};
```

Index Definition

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

  #pragma db id
  std::string email_;

  #pragma db index
  std::string first_;

  std::string last_;

  #pragma db index("name_i") \
    unique                   \
    method("BTREE")          \
    members(first_, last_)
};
```

Index Definition

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

  #pragma db id
  std::string email_;

  #pragma db index
  std::string first_;

  std::string last_;

  #pragma db index("name_i") \
    unique                   \
    method("BTREE")          \
    members(first_, last_)
};
```

Prepared and Cached 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 and Cached 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 and Cached 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 and Cached 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 ();
```

Other Features

- Extended database types
- Objects as class template instantiations
- `on_delete`
- Connection management
- Database operation callbacks
- Transaction callbacks
- Recoverable exceptions (`deadlock`, `timeout`)

Customizations

- Custom value types
- Custom containers
- Custom smart pointers
- Custom `NULL` wrappers
- Custom session
- Custom profiles

Future

- SQL to C++ compiler
- Containers in queries
- Mass `UPDATE`

Maybe Future

- Persistence to XML, JSON
- Document databases (MongoDB, RethinkDB)
- Sharding

Resources

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