OTUS

ES.40: Avoid complicated expressions

Дмитрий Шебордаев, Ведущий специалист

OTUS

ОНЛАЙН-ОБРАЗОВАНИЕ

Дмитрий Шебордаев, Ведущий специалист

Хранилища

ORM

Дмитрий Шебордаев

Ведущий специалист

Сегодня

- RDBMS, DOD, KV
- SQL
- ORM
- DAO
- Repository

RDBMS, DOD, KV

- Реляционная арифметика
- Обстрактное представление данных
- Извлечение произвольных данных

SQL / MySQL Connector/C++

```
sql::mysql::MySQL_Driver *driver;
sql::Connection *con;
sql::Statement *stmt;

driver = sql::mysql::get_mysql_driver_instance();
con = driver->connect(
  "tcp://127.0.0.1:3306", "user", "password");

stmt = con->createStatement();
stmt->execute("USE " EXAMPLE_DB);
stmt->execute("DROP TABLE IF EXISTS test");
stmt->execute("CREATE TABLE test(id INT, label CHAR(1))");
stmt->execute("INSERT INTO test(id, label) VALUES (1, 'a')");

delete stmt;
delete con;
```


SQL / MySQL Connector/C++

```
stmt = con->createStatement();

res = stmt->executeQuery(
  "SELECT id, label FROM test ORDER BY id ASC");
while (res->next()) {
  cout << "id = " << res->getInt(1) << endl;
  cout << "label = " << res->getString("label") << endl;
}

delete res;
delete stmt;
```


ORM / ODB

```
#pragma db object
class person
{
  friend class odb::access;
  #pragma db id auto
  unsigned long id_;
public:
  const std::string& first () const {return first_; }
  const std::string& last () const {return last_; }
  unsigned short age () const {return age_; }
  void age (unsigned short age) {age_ = age; }
};
```


ORM / ODB

```
#pragma db object
class person
{
  friend class odb::access;
  #pragma db id auto
  unsigned long id_;
public:
  const std::string& first () const {return first_; }
  const std::string& last () const {return last_; }
  unsigned short age () const {return age_; }
  void age (unsigned short age) {age_ = age; }
};

odb –d mysql ––generate–query  ––generate–schema person.hxx
```


ORM / ODB

```
/* This file was generated by ODB, object-relational mapping (O
 * compiler for C++.
 */

DROP TABLE IF EXISTS `person`;

CREATE TABLE `person` (
  `id` BIGINT UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT,
  `first` TEXT NOT NULL,
  `last` TEXT NOT NULL,
  `age` SMALLINT UNSIGNED NOT NULL)
 ENGINE=InnoDB;
```


ORM / ODB

```
person joe (”Joe”, ”Dirt”, 30);
john_id = db—>persist (john);
```


ORM / ODB

```
person joe ("Joe", "Dirt", 30);
john_id = db->persist (john);

const char access::object_traits_impl< ::person, id_mysql >::p
"INSERT INTO `person` "
"(`id`, "
"`first`, "
"`last`, "
"`age`) "
"VALUES "
"(?, ?, ?, ?)";
```


ORM / ODB

```
result r (db->query<person> (query::age > 30));
for (result::iterator i (r.begin ()); i != r.end (); ++i) {
    cout << "Hello, " << i->first () << "!" << endl;
}
```


ORM / ODB

```
result r (db->query<person> (query::age > 30));
for (result::iterator i (r.begin ()); i != r.end (); ++i) {
    cout << "Hello, " << i->first () << "!" << endl;
}

const char access::object_traits_impl< ::person, id_mysql >::q
"SELECT "
"`person`.`id`, "
"`person`.`first`, "
"`person`.`last`, "
"`person`.`age` "
"FROM `person`";
```


ORM / ODB

```
auto_ptr<person> joe (db->load<person> (joe_id));
joe->age (joe->age () + 1);
db->update (*joe);
```


ORM / ODB

```
auto_ptr<person> joe (db->load<person> (joe_id));
joe->age (joe->age () + 1);
db->update (*joe);

const char access::object_traits_impl< ::person, id_mysql >::u
  "UPDATE `person` "
  "SET "
  "`first`=?, "
  "`last`=?, "
  "`age`=? "
  "WHERE `id`=?";
```


ORM / ODB

```
#pragma db view object(person)
struct person_stat
{
  #pragma db column("count(" + person::id_ + ")")
  std::size_t count;
  #pragma db column("min(" + person::age_ + ")")
  unsigned short min_age;
  #pragma db column("max(" + person::age_ + ")")
  unsigned short max_age;
};
```


Object-Relational Mapping (ORM) / ODB

```
query_statement (const query_base_type& q) {
  query_base_type r (
    "SELECT␣"
    "count('person'.'id'),␣"
    "min('person'.'age'),␣"
    "max('person'.'age')␣");
  r += "FROM␣'person'";
  if (!q.empty ()) {
    r += "␣";
    r += q.clause_prefix ();
    r += q;
  }
  return r;
}
```


OTUS

Data Access Object (DAO)

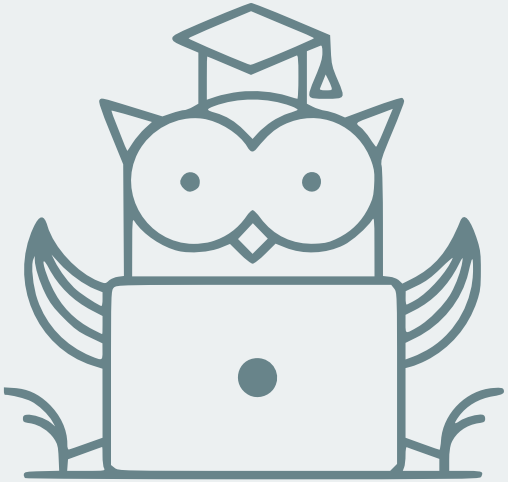
```
struct CTest {
  int id;
  std::string label;
}

class CTestDAO {
  void create(const CTest &test) {
    stmt->execute("INSERT INTO test (label) VALUES (?)", test.
  }
  void update(const CTest &test);
  CTest get(int id) {
    res = stmt->executeQuery(
      "SELECT id, label FROM test WHERE id=?", id);
    // ...
  }
  void remove(int id);
}
```


Repository / DDD

```
struct CTest {
  int id;
  std::string label;
  std::vector<CSlave> slaves;
}
class CTestRepository {
  void create(const CTest &test) {
    stmt->execute("INSERT INTO test (label) VALUES (?)", test.
    stmt->execute("INSERT INTO slave (test, value) VALUES (?,
  }
  void update(const CTest &test);
  CTest get(const CTest &test) {
    res = stmt->executeQuery(
      "SELECT id, label FROM test WHERE id=?", test.id);
    // ...
  }
  void remove(const CTest &test);
}
```


Почитать

- https://codesynthesis.com/products/odb/
- Предметно-ориентированное проектирование. Структуризация сложных программных систем (Эрик Эванс)



Спасибо за внимание!