Конфигурация БД MySQL

\*\*\*

# Основные понятия

**Корневой ссылочный тип (класс)** – это зарегистрированный в базе ссылочный тип данных, для которого соответствующий базовый тип не зарегистрирован.

Ссылочный тип данных называется **зарегистрированным в базе**, если для него в таблице type имеется соответствующая запись и существует соответствующая таблица в базе данных.

# Проектирование шаблона базы данных

## Идентификация объектов в базе данных

Для идентификации объектов в базе данных необходимо, чтобы у каждого зарегистрированного класса присутствовало поле Key типа DbKey. Сам тип DbKey не регистрируется в таблице ссылочных типов, т.к. является структурой и обязательно присутствует в каждой базе. Также DbKey не заносится в таблицу полей.

### Хранение ключей DbKey в базе

Помимо хранения собственных полей типа DbKey: SystemId и RawRevision, в базе хранится тип объекта, к которому относится ключ. Это необходимо для решения вопроса работы с базовыми ссылками. Таким образом, таблица для ключей объектов – db\_key, имеет следующие столбцы: sys\_id, raw\_revision и real\_type\_id. На связанные с db\_key поля таблиц ставится CONSTRAINT ON DELETE CASCADE.

## Представление информации о типах

Информация о ссылочных типах в базе представлена таблицами, которые содержат столбцы, соответствующие полям классов. Кроме того присутствует вспомогательная таблица type, хранящая список всех зарегистрированных в базе классов. Эта таблица содержит следующие столбцы:

* id - идентификатор типа в таблице;
* base\_type\_id - идентификатор базового типа:
  + NULL - базовый тип не зарегистрирован в базе – признак корневого класса;
* name - имя типа в системе;
* table\_name - имя соответствующей классу таблицы в БД.

Генерация имен таблиц происходит на стороне клиента по следующему алгоритму: в имени класса каждая буква верхнего регистра преобразуется к нижнему регистру и перед ней ставится символ "\_". Пример: User -> \_user, CoolCarForBirthday -> \_cool\_car\_for\_birthday. Ограничение на длину имени таблицы: 64 символа. [TODO]

### Хранение информации о полях классов

Описания полей классов содержатся в таблице field, содержащей следующие столбцы:

* id - идентификатор поля в таблице;
* owner\_type\_id - id типа, содержащего поле;
* name - имя поля в системе;
* column\_name - имя соответствующего столбца в таблице типа;
* type\_name – имя типа поля в системе (если ссылочный – можно не указывать);
* type\_id – id ссылочного типа (если тип‑значение – NULL);
* compare\_options – параметры для сравнения (применяется к строкам);
* back\_ref\_field\_id – id поля, являющегося обратной ссылкой (см. Организация обратных ссылок).
* is\_list – признак списка.

Имена колонок генерируются на стороне клиента по тому же алгоритму, что и имена таблиц, с тем же ограничением на длину.

Каждое поле соответствует одному классу, т.е. для производного класса записи полей базового класса в таблице полей не дублируются.

### Пример 1

Пусть в базе зарегистрировано 3 класса: DbObject, User и Admin (рисунок 1). Содержание таблиц type и field приведено в таблицах 1 и 2 соответственно.

Рисунок 1. Пример 1: иерархия классов

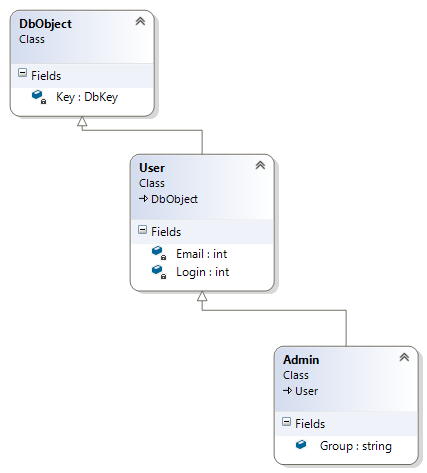


Таблица 1. Пример 1: содержание таблицы type

|  |  |  |  |
| --- | --- | --- | --- |
| id | name | table\_name | base\_type\_id |
| 0 | DbObject | db\_obj | NULL |
| 1 | User | user | 0 |
| 2 | Admin | admin | 1 |

Таблица 2. Пример 1: содержание таблицы field\*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| id | owner\_type\_id | name | column\_name | type\_name | type\_id | back\_ref\_field\_id | Is\_list |
| 0 | 1 | Login | login | string | NULL | NULL | FALSE |
| 1 | 1 | Email | email | string | NULL | NULL | FALSE |
| 2 | 1 | Rating | rating | int | NULL | NULL | FALSE |
| 3 | 2 | Group | grp | string | NULL | NULL | FALSE |

\* Здесь и в последующих примерах опущена колонка compare\_options.

### Варианты организации наследования на уровне таблиц классов

Независимо от способа организации иерархии, каждая таблица класса должан содержать столбец db\_key, по которому происходит идентификация объектов в базе.

#### Наследование без дублирования полей

При отображении иерархии классов без дублирования полей базовых классов в таблицах производных классов уменьшается избыточность базы данных на уровне структуры таблиц, что упрощает дальнейшие изменения схемы базы (добавление/удаление/изменение полей, добавление новых классов).

Однако возникает избыточность на уровне записей, т.к. один объект распределяется между всеми таблицами от терминального до корневого класса. Значительно увеличивается сложность чтения, изменения, удаления и вставки новых записей в таблицы, так как для получения полного списка полей объекта и их значений требуется обойти все таблицы от текущего класса до корневого. При чтении объекта это просто последовательность JOIN-ов по одинаковому ключу, но при добавлении, удалении или изменении записей становится очень трудно.

Учитывая специфику работы с базой, данный способ организации наследования нецелесообразен.

#### Наследование с дублированием полей

При дублировании полей базовых классов в таблицах производных классов объект хранится в виде одной записи, что значительно упрощает процедуру чтения, изменения, добавления и удаления записей из базы данных. В то же время становится более трудоемким процесс изменения схемы базы данных: при внесении изменений в базовый класс требуется пройтись по всем таблицам его производных классов.

Данный вариант приемлем, так как изменение структуры базы данных по определению дорогостоящая операция и ради ее упрощения нецелесообразно жертвовать скоростью работы с записями.

[TODO: ввести доп. таблицу для оптимизации выборки полного списка полей класса - не ходить до корня иерархии каждый раз для получения имен; формировать при изменении схемы БД]

## Организация ссылок

Ссылка в таблице представляется полем, ссылающимся на ключ DbKey (ForeignKey) и имеющим CONSTRAINT ON DELETE SET NULL.

### Решение проблемы базовых ссылок

В таблице ключей объектов db\_key сохраняется идентификатор реального типа объекта – real\_type\_id.

### Организация обратных ссылок (BackReference)

Обратные ссылки всегда парные, т.е. при наличии в классе А поля-ссылки на экземпляр класса Б с указанием имени поля-обратной ссылки, в классе Б обязательно будет поле-ссылка на экземпляр класса А с указанным именем.

Автоматическое изменение связанных полей-ссылок. Хранимая процедура - два варианта использования:

* триггер на вставку/удаление/изменение записей;
* отслеживать на стороне клиента и вызывать вручную.

Аналогично для списков ссылок, только другая процедура.

### Пример 2

Пусть дана иерархия классов, отображенная на рисунке 2. Таблицы 3 и 4 отражают содержание служебных таблиц type и field соответственно для текущей иерархии. Полная схема базы данных представлена на рисунке 3.

Таблица 3. Пример 2: содержание таблицы type

|  |  |  |  |
| --- | --- | --- | --- |
| id | name | table\_name | base\_type\_id |
| 0 | DbObject | db\_obj | NULL |
| 1 | User | user | 0 |
| 2 | Admin | admin | 1 |
| 3 | Group | group | 0 |

Таблица 4. Пример 2: содержание таблицы field

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| id | owner\_type\_id | name | column\_name | type\_name | type\_id | back\_ref\_field\_id | Is\_list |
| 0 | 1 | Login | login | string | NULL | NULL | FALSE |
| 1 | 1 | Email | email | string | NULL | NULL | FALSE |
| 2 | 1 | Rating | rating | int | NULL | NULL | FALSE |
| 3 | 2 | Group | grp | Group/NULL | 3 | 5 | FALSE |
| 4 | 3 | Title | title | string | NULL | NULL | FALSE |
| 5 | 3 | Admin | admin | Admin/NULL | 2 | 3 | FALSE |
| 6 | 3 | IsPublic | is\_public | bool | NULL | NULL | FALSE |

Рисунок 2. Пример 2: диаграмма классов

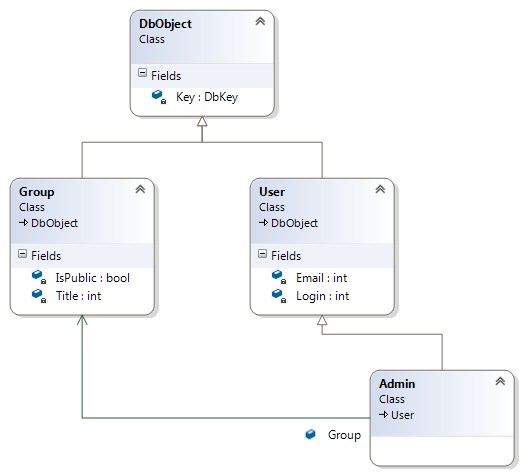


Рисунок 3. Пример 2: схема базы данных [TODO]

### Решение проблемы циклических зависимостей при получении данных

Для решения проблемы зацикливания при получении данных об объектах, содержащих перекрестные ссылки, или получении объекта, ссылающегося на самого себя, вводится вспомогательная таблица watched\_keys, в которую заносятся ключи всех просмотренных во время обхода объектов. Перед каждым переходом по ссылке проводится проверка на вхождение ключа в таблицу просмотренных. Если ключ присутствует в таблице, то переход по ссылке не осуществляется. Если ключ в таблице отсутствует, то он добавляется в нее и происходит переход по ссылке – получение полей вложенного объекта-записи. После выполнения запроса таблица очищается.

## Организация списков

### Списки типов-значений

Любые списки типов-значений представляются в виде BLOB-ов. [TODO]

### Список ссылок

Для всех списков ссылок введена пара таблиц: таблица, содержащая списки и таблица с их элементами. Вложенные списки ссылок (List<List<RefType>>) не поддерживаются.

# Порядок работы с объектами в БД

\* Ситуация:

Класс Б расширяет класс А. Мы добавили в базу объект класса Б, получили его ключ. Позже создали объект базового класса А, дали ему полученный ключ и сохранили в базу.

При сохранении объекта в БД:

1. объект с уже существующим ключом

=> обновить устанавливаемые поля

1. объект с уникальным ключом - объекта не существует

=> создать новый объект

\* Два объекта в базе не могут иметь ссылки на один и тот же List. При попытке сохранения в базу двух объектов, имеющих таковые, List копируется таким образом, что у каждого из объектов остается своя копия списка.

\* Dictionary не может связывать объекты БД - представляет собой просто пары "ключ-значение" для полей-значений => BLOB.

\* [TODO: Чтение объектов] SELECT `field` as `\_field\_name` FROM …;

\* [TODO: Запись объектов]

\* [TODO: Поиск объектов]

\* [TODO: Работа со списками объектов]

# Проектирование административной (вспомогательной) базы данных

[TODO: translate comments from 'MySQL prepare.sql']

`db\_list`:

\* name VARCHAR(64)

\* access TINYINT =

(

0 - ReadWrite

1 - ReadOnly

2 – NoAccess

)

## Получение списка баз данных

SELECT `name` FROM `nz\_admin\_db`.`db\_list`;

## Изменение списка баз данных

### Удаление базы данных из списка

Для удаления определенного списка баз данных необходимо заполнить таблицу db\_rem\_list именами удаляемых баз, после чего вызвать хранимую процедуру remove\_databases(). При попытке удалить несуществующую базу данных ошибок не возникает.

[TODO: перемещать запись удаляемой БД в таблицу db\_to\_cleanup; реально удалять по требованию – cleanup\_removed\_db\_list()]

### Добавление базы данных в список

Для создания новых баз данных необходимо заполнить таблицу db\_add\_list именами новых баз, после чего вызвать хранимую процедуру add\_databases(). При попытке создать уже существующую базу данных ошибок не возникает. Права доступа по умолчанию для новой базы данных – ReadWrite.

Вызов хранимой процедуры alter\_database\_list() после заполнения вышеуказанных таблиц произведет сначала удаление, а потом добавление новых баз данных.

[TODO: TRUNCATE удаляемую и заново создаваемую БД во избежание конфликтов]