

## Задачи и ТЗ

### Задачи:

1. SQL-код: Каким образом можно получить все записи с привязанным NDC-атрибутом по запросу для первой таблицы начинающихся на 'title 1'. Данные должны быть сопряжены с NDC.
2. SQL-код: Подсчитать кол-во записей имеющих больше чем 2 одинаковых вхождений по значениям поля NDC.
3. Дать рекомендации по структуре таблиц и хранению данных (только рекомендации, вносить изменения в БД не нужно).
4. Возможно ли упростить структуру. Как бы Вы это решили в контексте 1-ой задачи?

Поясните путь решения и почему сделали так.

Условие: Входные данные (таблицы справочника) могут меняться раз в неделю.

На Yii-фреймворке версии 1.x:

— Без изменения действующей структуры БД приложенной к настоящему заданию—

5. Реализовать решение 1-ой задачи на базе AR-модели.
6. Реализовать решение 1-ой задачи при помощи DAO.
7. Реализовать интерфейс вывода результатов через CGridView. поставленной задачи с кэшированием данных

### Условия:

- Развернуть Yii-приложение через composer.
- Создать все необходимые миграции для выполнения задач 5-7.

## Ответы и комментарии

1. SQL-код: Каким образом можно получить все записи с привязанным NDC-атрибутом по запросу для первой таблицы начинающихся на 'title 1'. Данные должны быть сопряжены с NDC.

В формулировке 1-й задачи встречаются слова "для первой таблицы"... Возникает вопрос: какая из них "первая", какая - "вторая"? Но судя по указанию условия для "title 1", то возможно предположить, что "первой" считается таблица tb\_source. Тогда решение задачи может быть выражено следующим SQL-запросом:

```
select s.*, r.ndc
from tb_source s
inner join tb_rel r on r.cx = s.cx
where
    s.title like 'title 1%'
```

Использовано внутреннее соединение (INNER JOIN) по "общему" для таблиц полю `cx`. Запрос можно изменить эквивалентным образом, например, так (с использованием USING):

```
select s.*, r.ndc
from tb_source s
inner join tb_rel r using (cx)
where
    s.title like 'title 1%'
```

Или так (с помощью уже внешнего соединения LEFT JOIN):

```
select s.*, r.ndc from tb_source s
left join tb_rel r on r.cx = s.cx
where
    s.title like 'title 1%'
    and
    r.ndc is not null
```

Поскольку в задаче требуется, чтобы данные (result set) должны быть сопряжены с ("привязанным") атрибутом `ndc` (из второй таблицы), то запросы принимают именно такую форму.

**Важное замечание:** поле `cx` выбрано мной в качестве поля для соединения произвольным образом, так как мне не известна содержательная бизнес-логика приложения и реальные моделируемые таблицами отношения сущностей. Однако же с формальной точки зрения **будет также правильным** использовать не соединение по `r.cx = s.cx`, а, скажем, по `r.cx = s.rx`, т.е. мы формально имеем право написать соединение так:

```
select s.*, r.ndc
from tb_source s
```

```
inner join tb_rel r on r.cx = s.rx
where
s.title like 'title 1%'
```

Ничто нам не мешает произвести соединения таблиц по **разноименным** полям, имеющих при этом **одинаковые определения** (`VARCHAR(10) NULL DEFAULT NULL`, хотя и это требование не обязательно) `r.cx = s.rx`, но опять-таки это будет **произвольным** выбором разработчика. Result set же для этого запроса также будет возможным (но даст другие результаты), но будет ли он иметь смысл с **содержательной** точки зрения?! Это мне не известно.

## 2. SQL-код: Подсчитать кол-во записей, имеющих больше чем 2 одинаковых вхождений по значениям поля NDC.

Чтобы подсчитать кол-во записей, "имеющих больше чем 2 одинаковых вхождений по значениям поля NDC", то решением может быть такой запрос:

```
select r.*, count(*) as cnt
from tb_rel r
group by r.ndc
having cnt > 2
```

```
1 select r.*, count(*) as cnt
2 from tb_rel r
3 group by r.ndc
4 having cnt > 2
5
```

tb_rel (3x991)		
cx	ndc	cnt
7057	NDC10000-010	544
8570	NDC10000-0100	600
5420	NDC10000-01000	587
6543	NDC10000-0101	589
9063	NDC10000-0102	618
5773	NDC10000-0103	627

Суммарное же (общее) число всех таких записей можно узнать, выполнив запрос по полю cnt из предыдущего result set'a:

```
select sum(a.cnt)
from (
    select r.*, count(*) as cnt
    from tb_rel r
    group by r.ndc
    having cnt > 2
) as a
```

```
1 select sum(a.cnt) as total
2 from (
3     select r.*, count(*) as cnt
4     from tb_rel r
5     group by r.ndc
6     having cnt > 2
7 ) as a
```

Результат #1 (1x1)	
total	
593 936	

## 3. Дать рекомендации по структуре таблиц и хранению данных (только рекомендации, вносить изменения в БД не нужно).

В качестве "Дано" мы имеем 2 таблицы, `tb_rel` и `tb_source`, структура которых описывается следующими операторами CREATE TABLE:

```
CREATE TABLE `tb_rel` (
  `cx` VARCHAR(10) NULL DEFAULT NULL,
  `ndc` VARCHAR(20) NULL DEFAULT NULL
)
COLLATE='utf8_general_ci'
ENGINE=InnoDB;
```

```
CREATE TABLE `tb_source` (
  `cx` VARCHAR(10) NULL DEFAULT NULL,
  `rx` VARCHAR(10) NULL DEFAULT NULL,
  `title` VARCHAR(100) NULL DEFAULT NULL
)
COLLATE='utf8_general_ci'
ENGINE=InnoDB;
```

Каждая из таблиц имеет по 600 000 записей, например:

Хост: 127.0.0.1	База данных: virtualhealth	Таблица: tb_source
1 select count(*) from tb_source		
Результат #1 (1x1)		
count(*)	600 000	

Первое, что бросается в глаза: определение **структур** таблиц для хранения данных (и их типов) **не оптимально с точки зрения самих хранимых данных**. Выполним:

1 SELECT * FROM tb_rel PROCEDURE ANALYSE()									
Результат #1 (10x2)									
Field_name	Min_value	Max_value	Min_length	Max_length	Empties_or_zeros	Nulls	Avg_value_or_avg_length	Std	Optimal_fieldtype
virtualhealth.tb_rel.cx	100	9999	3	5	0	0	3.9105	(NULL)	SMALLINT(4) UNSIGNED NOT NULL
virtualhealth.tb_rel.ndc	NDC10000-010	NDC1999-0741	11	14	0	0	12.8997	(NULL)	VARCHAR(14) NOT NULL
1 SELECT * FROM tb_source PROCEDURE ANALYSE()									
Результат #1 (10x3)									
Field_name	Min_value	Max_value	Min_length	Max_length	Empties_or_zeros	Nulls	Avg_value_or_avg_length	Std	Optimal_fieldtype
virtualhealth.tb_source.cx	100	99999	3	6	0	0	4.8914	(NULL)	MEDIUMINT(5) UNSIGNED NOT NULL
virtualhealth.tb_source.rx	100	99999	3	6	0	0	4.8913	(NULL)	MEDIUMINT(5) UNSIGNED NOT NULL
virtualhealth.tb_source.title	title 100	title 200	9	9	0	0	9.0000	(NULL)	ENUM('title 100','title 101','title 102','title 103','title 104','b...

Оптимально (в данных условиях!) было бы использовать целочисленные значения (как пример, **INT** или **UNSIGNED INT**, но возможны и MEDIUMINT или SMALLINT, предлагаемые анализатором) для соответствующих полей, исходя из структуры отчета и бизнес-логики (по крайней мере это можно реализовать для полей tb\_rel.cx, tb\_source.cx и tb\_source.rx). Более точную оценку оптимального размера целочисленного поля для этих полей можно дать после ознакомления с требованиями бизнес-логики.

Далее. В данных условиях **ни одна из таблиц вообще не имеет индексов**, что является существенным bottleneck для выполняемых SELECT-выборок. Количество индексов, их простота или составность, а также порядок вхождения полей в индекс выбираются, исходя из бизнес-логики и основных структур SELECT-запросов, которые будут использоваться. В данных условиях можно было бы добавить, например, **минимум** такие индексы для таблицы tb\_source:

**INDEX `cx` (`cx`),**  
**INDEX `title` (`title`).**

И для таблицы tb\_rel **минимум** такие:

**INDEX `cx` (`cx`).**

В данных условиях (и как следствие) ни одна таблица также **не имеет первичных и/или внешних ключей**, необходимых для нормализации данных и поддержания связей и организации возможной ссылочной целостности между таблицами (если мы вообще их рассматриваем как "связанные").

Обе таблицы, однако, имеют некое "**общее**" поле `cx` с идентичным определением типа и размерности данных и больше никакого другого (где бы совпадали определения полей, кроме tb\_source.rx, о котором я упоминал выше в "важном замечании").

На таблицах **невозможна поддержка первичных ключей**, так как есть множество дублей строк:

1 select count(*) - count(DISTINCT r.cx, r.ndc)	
2 from tb_rel r	
Результат #1 (1x1)	
count(*) - count(DISTINCT r.cx, r.ndc)	17 677
1 SELECT COUNT(DISTINCT r.cx, r.ndc) / COUNT(*)	
2 AS 'unique',	
3 1 - (COUNT(DISTINCT r.cx, r.ndc) / COUNT(*)	
4 AS 'non-unique'	
5 FROM tb_rel r;	
Результат #1 (2x1)	
unique	non-unique
0,9705	0,0295

Только для таблицы `tb_source` возможно (**только при данных условиях**) создание первичного ключа **по всем трем полям** (т.е. получается только составной Primary key), но его смысл и структура возможно не рациональны и не будут иметь смысла в контексте нормализации данных или поддержки связей между таблицами и ссылочной целостности данных, если последняя хоть как-то содержательно обоснована.

Другой рекомендацией по хранению данных может служить оптимизация **содержимого** полей таблиц. В полях `ndc` таблицы `tb_rel` и поля `title` таблицы `tb_source` имеются данные (строки), которые имеют префикс, начинающийся с "NDC..." и "title " соответственно. Поскольку эти префиксы используются во всех строках, имеет смысл убрать их для **оптимизации объема** хранимых данных.

### Дополнительные замечания

Из предыдущих замечаний также следует, что отношения между сущностями, моделируемых с помощью данных таблиц, не могут отражать **отношения one-to-many** или **many-to-many** в точном смысле.

Как следствие отсутствия декларации связей между таблицами (первичных и внешних ключей) не удастся смоделировать **полноценные relational AR-модели** в контексте использования этих таблиц в Yii-application:

- `BELONGS_TO` (не понятно, "кто кому" принадлежит, т.к. единственный "общий" атрибут ``cx`` не симметричен и не задает функцию однозначного соответствия между моделями),
- `HAS_ONE` (однозначно не проходит по тем же причинам),
- `MANY_MANY` (тоже нет, т.к. нет связующей, ассоциативной или junction-таблицы).
- Единственное отношение, которое может быть использовано с большой натяжкой и оговорками (и то без возможного содержательного смысла) - это `HAS_MANY`, если представить, что записи из `tb_rel` связаны со "многими" записями из таблицы `tb_source` по "**общему**" (**в кавычках**) полю ``cx``, и наоборот (для таблицы `tb_source`). Поэтому в последнем случае это "отношение" всегда будет декартовым произведением (комбинацией) каждой записи из таблицы `tb_rel` и `tb_source`, если условие соединения записывается по соединяющему полю ``cx``: **`tb_rel.cx = tb_source.cx`**. Для единичной же AR-модели Yii (т.е. экземпляра AR-класса) это будет означать, что она может иметь множество related AR-моделей, построенных на основе другой таблицы.  
Например, каждая запись в таблице `tb_source`, у которой `cx='100'` будет иметь "много" related-записей в таблице `tb_rel`, у которых атрибут `cx` также равен '100'. **Каждая из 6** имеющихся записей из таблицы **`tb_source`** (`select count(*) from tb_source s where s.cx = '100'` дает 6) будет иметь по **52** (`select count(*) from tb_rel r where r.cx = '100'` дает 52) related-записи в таблице **`tb_rel`**. И наоборот.

Из документации мы знаем:

Установка связей производится внутри метода `relations()` класса `CActiveRecord`. Этот метод возвращает массив с конфигурацией связей. Каждый элемент массива представляет одну связь в следующем формате:

```
'VarName'=>array('RelationType', 'ClassName', 'ForeignKey', ...дополнительные параметры)
```

где `VarName` — имя связи, `RelationType` указывает на один из четырёх типов связей, `ClassName` — имя AR-класса, связанного с данным классом, а `ForeignKey` обозначает один или несколько внешних ключей, используемых для связи. Кроме того, можно указать ряд дополнительных параметров, о которых будет рассказано позже.

### Тогда откуда мы будем брать Foreign Keys и Primary keys?!

Правильный ответ - "ниоткуда". Поскольку в данной структуре таблиц тестового задания **не определены никакие внешние ключи или первичные ключи**, то, можно конечно, попробовать написать некий "костыль", который будет эмулировать такие отношения

В документации также ясно сказано, что:

AR опирается на правильно определённые первичные ключи таблиц БД. Если в таблице нет первичного ключа, то требуется указать в соответствующем классе AR столбцы, которые будут использоваться как первичный ключ. Сделать это можно путём перекрытия метода `primaryKey()`:

```
public function primaryKey()
{
    return 'id';
    // Для составного первичного ключа следует использовать массив:
    // return array('pk1', 'pk2');
}
```

Поэтому можно, конечно, попробовать сэмулировать AR-отношения, с использованием, например, такого кода (по сути - "костыля"):

```

public function primaryKey()
{
    return 'cx';
}

public function getTableSchema()
{
    $table = parent::getTableSchema();

    $table->columns['cx']->isForeignKey = true;
    $table->foreignKeys['cx'] = array('TbRel', 'cx');

    return $table;
}

public function relations()
{
    return array(
        'rels' => array(
            self::HAS_MANY,
            'TbRel',
            'cx',
            'joinType' => 'LEFT JOIN',
            'together' => true
        )
    );
}

```

Но от такого решения нужно отказался по причине, что это нарушает смысл и принципы Relational AR.

#### 4. Возможно ли упростить структуру. Как бы Вы это решили в контексте 1-ой задачи?

*Поясните путь решения и почему сделали так.*

*Условие: Входные данные (таблицы справочника) могут меняться раз в неделю.*

Поскольку у нас используется реляционная модель, то слова "упростить структуру" можно понимать как использование логики нормализации таблиц (или, наоборот, денормализации, где это уместно и выгодно впоследствии), возможной декомпозицией их полей и т.п.

В контексте первой задачи мы извлекали данные из таблицы `tb_source` с привязанным атрибутом `NDC` из второй таблицы и пытались связать таблицы по произвольному полю `sx` (хотя могли их связывать и иным образом - ничто этому не мешает, поскольку нам не известна содержательная логика, стоящая за этим).

Если таблицы называются как таблицы "справочника", тогда непонятно, почему в этом "справочнике" столь много дубликатов? Ведь суть справочника как правило состоит в поддержании уникальных значений (элементов), на которые затем можно ссылаться по их первичному ключу. Примеры: каталоги геообъектов (стран), различные категориальные справочники и т.п.

Если поле ``ncd`` в таблице `tb_rel` - есть элемент справочника, то в этой таблице желательно использовать идентификатор этого `ncd`. Так же можно поступить и с полем ``sx``.

То есть, вообще можно выполнить декомпозицию таблицы `tb_rel` на несколько посредствующих или связующих таблиц, которые бы связывали элементы, на которые ссылаются поля этой таблицы (т.е. `ncd` и `sx`), как многие-ко-многим.

Тогда решение первой задачи опиралось бы на извлечение данных (`ncd`) через связывание с посредствующей (junction) таблицей и через нее уже со справочной. Аналогично имело бы смысл поступить и с атрибутом ``sx`` и/или ``gx`` из таблицы `tb_source`.

Я не совсем до конца понял постановку этой задачи, т.к. не хватает дальнейшей определенности и содержательных пояснений и вопрос этот больше контекстный, т.е. адресован человеку, который уже знает background этой задачи с содержательной стороны. Т.е. нормализация, переструктурирование таблиц, декомпозиция полей, выделение новых связующих таблиц и т.д. прежде всего зависит от понимания **содержательной** логики этой структуры БД. При формальном же подходе, и при прочих равных условиях можно сказать, что да, **описанные вещи** (рекомендации по прощению структуры) **делать, конечно, надо**.

Насчет условия, что "входные данные справочника могут меняться раз в неделю", мне также не совсем понятно. Если имеется ввиду отслеживание истории изменений справочника, обращение к отслеживаемым его "снимкам", или связанным с ним по времени сущностям и т.п., то конечно, надо вводить дополнительные поля хотя бы в виде временных меток (таймстемпов), чтобы впоследствии осуществлять по ним выборки, сортировки и отслеживания.

#### 5. На Yii-фреймворке версии 1.x:

*Без изменения действующей структуры БД  
приложенной к настоящему заданию*

*5. Реализовать решение 1-ой задачи на базе  
AR-модели.*

*6. Реализовать решение 1-ой задачи при  
помощи DAO.*

*7. Реализовать интерфейс вывода  
результатов через CGridView. поставленной  
задачи с кэшированием данных*

Реализовано в коде проекта.