Распределенные запросы (несколько БД – ОДИН СЕРВЕР) MySQL

При создании SaaS-продукта (связанный с электронной торговлей), где каждый пользователь занимает свою собственную маленькую базу данных (MySQL). Можно гарантировать, что данные пользователя останутся исключительно его данными. SaaS-продукт завоёвывает популярность, вызывая интерес новых пользователей. У вас их уже 100 000, и неплохо бы выяснить, как все эти пользователи используют ваше ПО. Сколько продуктов ими добавлено, сколько сделано заказов их пользователями и т.д.? Чтобы получить все эти цифры, придётся делать запросы ко всем базам данных всех пользователей. Необходима одна большая таблица, внутри которой находились бы таблицы всех пользователей! Достаточно прописать запросы к половине этих таблиц, и вот у вас уже готовы (благодаря JOIN, GROUP BY и т.д.) запросы сразу ко всем таблицам в этой одной гигантской таблице.

Один из таких инструментов—это представления (VIEW), позволяющие по-новому взглянуть на основополагающие структуры данных. Для начала нужно создать представление с таблицами всех пользователей. Все эти таблицы нужно будет соединить друг с другом.

Именно ОБЪЕДИНЕНИЕ всех таблиц в одном VIEW мы и будем создавать. Проделывая всё это с каждой таблицей, мы добьёмся того, чтобы все данные баз данных были у нас под рукой. Как же будет выглядеть такое представление?

```
CREATE VIEW sample_table AS (
  SELECT "tenant  1" AS tenant, t.* FROM tenant  1.sample  table AS t
  UNION ALL
  SELECT "tenant_2" AS tenant, t.* FROM tenant_2.sample_table AS t
  ...
  UNION ALL
  SELECT "tenant  N" AS tenant, t.* FROM tenant  N.sample  table AS t
)
```

В каждом представлении ко всем колонкам исходных таблиц добавляется ещё одна колонка **арендатор (пользователь)** с именем БД пользователя, из которой берётся запись. Это, конечно, очень удобно: вы легко можете узнать, чьи это данные. Но надо быть осторожным при написании запросов JOIN, ведь к условиям JOIN тоже необходимо добавлять эту колонку:

```
SELECT
   pp.tenant, pp.name, ppg.num
FROM
   products__products AS pp
      LEFT JOIN
```

```
  products__product_gtins AS ppg ON (pp.id = ppg.product_id
    AND pp.tenant = ppg.tenant)
```

Теперь остаётся создать такое представление для всех таблиц в базе данных, но главное — надо использовать *хранимые процедуры.* Причины:

- нет зависимостей (в СУБД уже есть среда выполнения);
- обновлять представления не сложнее, чем запускать SQL-команду с любимого клиента.

Последовательность действий проста: сначала создаём *хранимую процедуру* для представления одной таблицы, затем выполняем итеративный обход всех таблиц, создавая представление для каждой из них.

Хранимая процедура для создания представления одной таблицы

```
CREATE PROCEDURE `update_table_view`(IN tbl_name VARCHAR(100), IN db_pattern_re VARCHAR(100))
BEGIN
  DECLARE all_dbs_view LONGTEXT;
  DECLARE all_dbs_done INT DEFAULT 0;
  DECLARE all_dbs_indx INT DEFAULT 0;
  DECLARE cur_tenant_db VARCHAR(100);

  -- (A)
  DECLARE all_dbs_cur CURSOR FOR SELECT `schema_name` FROM information_schema.schemata WHERE `schema_name` REGEXP db_pattern_re;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET all_dbs_done = 1;

  -- (B)
  SET all_dbs_view = CONCAT("CREATE VIEW ", tbl_name, " AS ");

  SET all_dbs_done = 0;
  SET all_dbs_indx = 0;

  -- (C)
  OPEN all_dbs_cur;
  all_dbs_loop: LOOP
```

```
    FETCH all_dbs_cur INTO cur_tenant_db;

    IF all_dbs_done = 1 THEN LEAVE all_dbs_loop; END IF;


    -- (D)

    IF all_dbs_indx > 0 THEN SET all_dbs_view = CONCAT(all_dbs_view, " UNION ALL "); END IF;

    SET all_dbs_view = CONCAT(all_dbs_view, "SELECT \"", cur_tenant_db, "\" AS tenant, t_", all_dbs_indx, ".* FROM `", cur_tenant_db, "`.`", tbl_name, "` AS t_", all_dbs_indx);

    SET all_dbs_indx = all_dbs_indx + 1;


  END LOOP all_dbs_loop;

  CLOSE all_dbs_cur;


  -- (E)

  SET @drop_view = CONCAT("DROP VIEW IF EXISTS ", tbl_name);

  PREPARE drop_view_stm FROM @drop_view; EXECUTE drop_view_stm; DEALLOCATE PREPARE drop_view_stm;


  -- (F)

  SET @all_dbs_view_v = all_dbs_view;

  PREPARE all_dbs_view_stm FROM @all_dbs_view_v; EXECUTE all_dbs_view_stm; DEALLOCATE PREPARE all_dbs_view_stm;

END
```

Хранимая процедура принимает два аргумента:

- *tbl_name*: имя таблицы, для которой создаётся VIEW;
- *db_pattern_re*: регулярное выражение для исключающей фильтрации баз данных, которые будут включены в VIEW.

A) сначала надо объявить курсор для итеративного обхода всех БД, включённых в представление. Чтобы отфильтровать только те БД, которые нам нужны, используем регулярное выражение *db_pattern_re*. Обработчик CONTINUE HANDLER позаботится о том, чтобы цикл остановился после итеративного обхода всех найденных БД.

B) инициализируем переменную *all_dbs_view*, содержащую нашу полную инструкцию CREATE VIEW. Переменная может оказаться довольно длинной (LONGTEXT), в зависимости от числа прошедших фильтрацию БД.

C) теперь открываем курсор и начинаем итеративный обход каждой прошедшей фильтрацию БД. Оператор IF выполнит проверку на наличие

переменной *all_dbs_done* на каждом этапе. При запуске обработчика CONTINUE HANDLER переменная будет иметь значение 1.

D) Первая итерация не требует ставить в начало UNION ALL, а вот все последующие будут ставить. Следующая строчка конкатенирует, то есть добавляет текущий оператор SELECT, содержащий все записи из таблицы БД конкретного пользователя:

```
SELECT "tenants_DB" AS tenant, t.* FROM tenants_DB.some_table AS t
```

E) прежде чем создавать VIEW, необходимо убедиться, что предыдущее (если оно существовало) удалено.

F) и, наконец, запускаем саму инструкцию, создающую VIEW.

Теперь можно выполнить процедуру создания представления VIEW одной таблицы:

```
CALL update_table_view("sample_table", "tenant_[0-9]+");
```

Хранимая процедура для итеративного обхода всех таблиц

```
CREATE PROCEDURE `update all views`(IN db first VARCHAR(100), IN db pattern re VARCHAR(100))

BEGIN

  DECLARE all_tbls_done INT DEFAULT 0;

  DECLARE cur  tbl VARCHAR(100);


  -- A

  DECLARE all_tbls_cur CURSOR FOR SELECT `table_name` FROM information_schema.tables WHERE table_schema = db_first;

  DECLARE CONTINUE HANDLER FOR NOT FOUND SET all  tbls  done = 1;


  SET all_tbls_done = 0;

  OPEN all_tbls_cur;

  -- B

  all  tbls  loop: LOOP

    FETCH all_tbls_cur INTO cur_tbl;

    IF all_tbls_done = 1 THEN LEAVE all_tbls_loop; END IF;


    -- C

    CALL update_table_view(cur_tbl, db_pattern_re);
```

```
  END LOOP all_tbls_loop;

  CLOSE all_tbls_cur;

END
```

Хранимая процедура принимает два аргумента:

- *db_first*: имя базы данных, которое будет использовано для вывода списка всех таблиц. Это может быть любая ваша БД;
- *db_pattern_re*: регулярное выражение для исключающей фильтрации баз данных, которые будут включены в VIEW.

A) объявляем курсор для итеративного обхода всех таблиц, найденных в базе данных *db_first*. Обработчик CONTINUE HANDLER позаботится о том, чтобы цикл остановился после итеративного обхода всех обнаруженных таблиц.

B) открываем курсор и начинаем итеративный обход каждой найденной таблицы. Оператор IF выполнит проверку на наличие переменной *all_tbls_done* на каждом этапе. При запуске обработчика CONTINUE HANDLER переменная будет иметь значение 1.

C) Получив значение текущей таблицы, хранимой в переменной *cur_tbl*, можно выполнить хранимую процедуру, которая создаст VIEW специально для *cur_tbl*.

При выполнении хранимой процедуры*:*

```
CALL update_all_views("tenant_1", "tenant_[0-9]+");
```

получаем:

- список всех представлений со всеми таблицами из базы данных *tenant_1;*
- ОБЪЕДИНЕНИЕ одинаковых таблиц всех арендаторов, чем и является, по сути, каждое представление.

Курсоры

Курсоры используются для прохождения по набору строк, возвращенному запросом, а также обработки каждой строки.

MySQL поддерживает курсоры в хранимых процедурах. Вот краткий синтаксис создания и использования курсора.

```
1 DECLARE cursor-name CURSOR FOR SELECT ...;       /*Объявление курсора и его заполнение */
2 DECLARE  CONTINUE HANDLER FOR NOT FOUND          /*Что делать, когда больше нет записей*/
3 OPEN cursor-name;                                /*Открыть курсор*/
4 FETCH cursor-name INTO variable [, variable];    /*Назначить значение переменной, равной текущему значению столбца*/
```

5 CLOSE cursor-name; /*Закрыть курсор*/

В этом примере мы проведем кое-какие простые операции с использованием курсора:

```
DELIMITER //

CREATE PROCEDURE `proc_CURSOR` (OUT param1 INT)
BEGIN
    DECLARE a, b, c INT;
    DECLARE cur1 CURSOR FOR SELECT col1 FROM table1;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET b = 1;
    OPEN cur1;

    SET b = 0;
    SET c = 0;

    WHILE b = 0 DO
        FETCH cur1 INTO a;
        IF b = 0 THEN
            SET c = c + a;
    END IF;
    END WHILE;

    CLOSE cur1;
    SET param1 = c;

END //
```

У курсоров есть три свойства, которые вам необходимо понять, чтобы избежать получения неожиданных результатов:

- Не чувствительный: открывшийся однажды курсор не будет отображать изменения в таблице, происшедшие позже. В действительности, MySQL не гарантирует то, что курсор обновится, так что не надейтесь на это.
- Доступен только для чтения: курсоры нельзя изменять.
- Без перемотки: курсор способен проходить только в одном направлении - вперед, вы не сможете пропускать строки, не выбирая их.

ЗАДАНИЕ:

Согласно вариантам, выданным преподавателем:

1. создать несколько БД (2 одинаковые базы данных под разными псевдонимами);

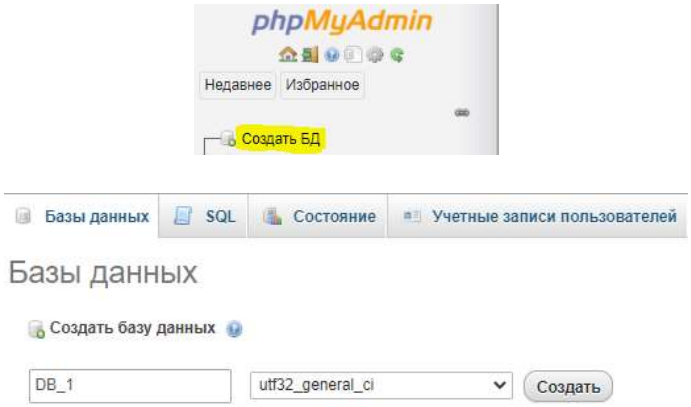

Рис. 1 – Пример создания базы данных в PHPMYADMIN

Замечание! Перед созданием Базы данных рекомендуется создать эскиз и на основе него создавать и видоизменять БД, например:

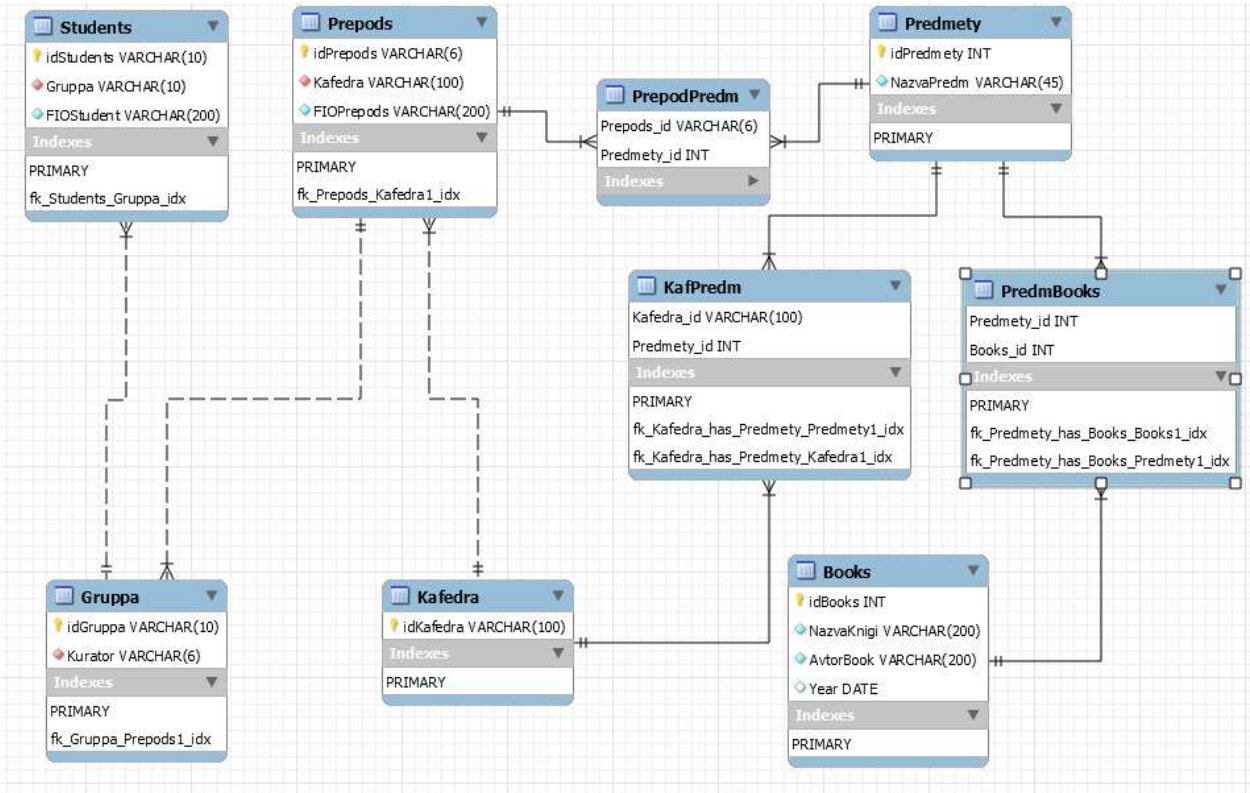

Рис. 3 – Пример создания эскиза базы данных в MySqlWorkbench

2. в каждой БД реализовать не менее 3-5 таблиц (во всех БД выполняется принцип целостности данных, наличие связей между таблицами обязательно);

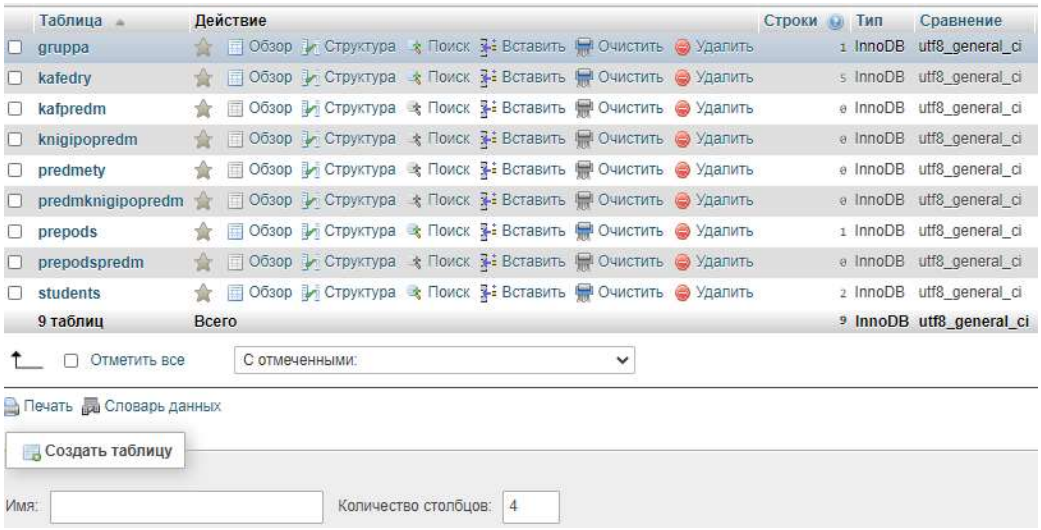

Рис.3 – Пример создания таблиц в тестовой базе данных

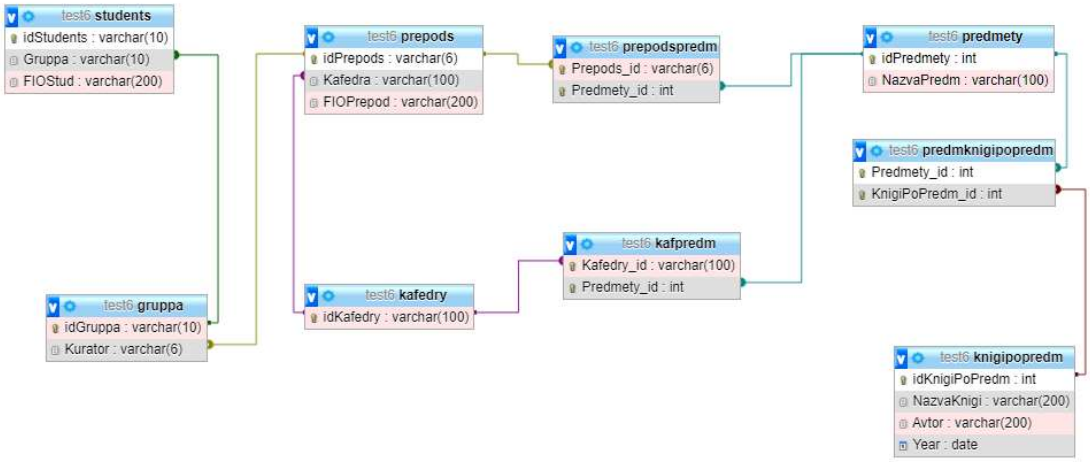

Рис. 4 – Пример базы данных в дизайнере PHPMYADMIN

3. Создать по 2-3 процедуры для каждой БД, предварительно сделав по 2-3 записи в каждой таблице;

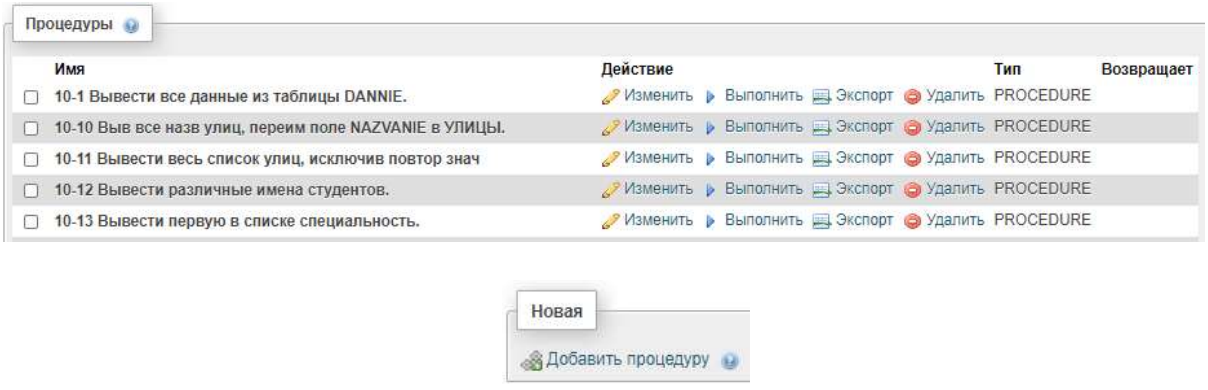

Рис. 4 – Пример создания процедур в базе данных

4. Создать объединенное представление всех БД.

Замечание! Сначало создайте примитивное представление, после чего комбинируйте и добавляйте в представление больше полей со всех таблиц, пока не получите представление со всеми полями всех таблиц.

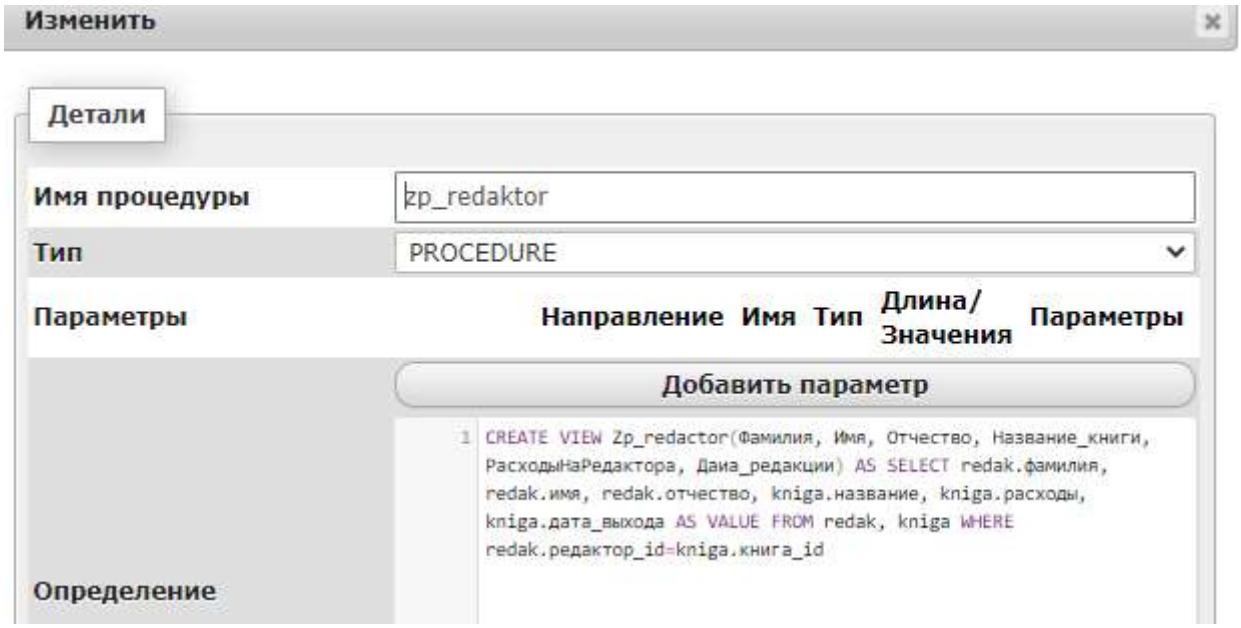

Рис. 4 – Пример создания представлений в базе данных

5. выбрать информацию из разных баз данных (в пределах одного сервера) – 3запроса.

Пример:

- Если в запросе таблица указывается с именем базы данных **DB1.table1**, то таблица выбирается из **DB1**, если просто **table1**, то - из активной базы данных.
- Общий принцип перекрестного запроса к двум базам в пределах одного MySQL-сервера:

```
SELECT t4.*, t7.*
FROM st_4.dannie AS t4
INNER JOIN st_7.dannie AS t7 ON t4.kod_student = t7.kod_student
```