МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Севастопольский государственный университет»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ по дисциплине

«Методы и средства хранения информации»

для студентов по направлению

09.03.02

«Информационные системы и технологи»

Часть 2

Севастополь
2019

УДК 004.056.053

Методические указания к выполнению лабораторных работ по дисциплине «Методы информационной оптимизации систем и процессов»/ Сост. ст. преп. А.Ю. Дрозин – Севастополь: Изд-во СевГУ, 2019. – 36с.

Методические указания рассмотрены и утверждены на заседании кафедры информационных систем (протокол №____от_____2019 года).

Допущено учебно-методическим центром СевГУ в качестве методических указаний.

Рецензент: _____

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА 5
«Исследование принципов партицирования баз данных»

1 ЦЕЛЬ РАБОТЫ

Исследовать способы партицирования таблиц баз данных и их влияние на скорость доступа с данным. Изучить основы партицирования на примере MySQL.

2 ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

2.1 Партицирование таблиц

Начиная с версии 5.1 MySQL поддерживает горизонтальное партицирование таблиц. Партицирование (partitioning) — это разбиение больших таблиц на логические части по выбранным критериям.

Наиболее частая ситуация, когда нагрузку обеспечивают всего несколько таблиц СУБД из всех имеющихся. Причем чтение в большинстве случаев приходится только на самую последнюю их часть (т.е. активно читаются те данные, которые недавно появились). Примером тому может служить блог (интернет-журнал событий, интернет-дневник,онлайн-дневник) - на первую страницу (это последние 5…10 записей) приходится 40…50% всей нагрузки, или новостной портал (суть одна и та же), или системы личных сообщений и т.п. Партицирование таблицы позволяет базе данных делать интеллектуальную выборку - сначала СУБД уточнит, какой партиции соответствует запрос (если это реально) и только потом сделает этот запрос, применительно к нужной партиции (или нескольким партициям). Таким образом, в рассмотренном случае, нагрузка распределяется на таблицу по ее партициям. Следовательно выборка типа:

```
SELECT *
FROM articles ORDER BY
id DESC LIMIT 10
```

будет выполняться только над последней партицией, которая значительно меньше всей таблицы.

2.2 Горизонтальное партицирование таблиц

Рассмотрим пример реализации горизонтального партицирования таблиц в MySQL. Пусть есть таблица вот такой структуры:

```
CREATE TABLE orders_range ( customer_surname
      VARCHAR(30), store_id INT,
      salesperson_id INT,
      order_date DATE, note
      VARCHAR(500)
      ) ENGINE = MYISAM
            PARTITION BY RANGE( YEAR(order_date) ) ( PARTITION p_old
                                    VALUES LESS THAN(2008), PARTITION
            p_2008 VALUES LESS THAN(2009), PARTITION p_2009 VALUES LESS
            THAN(MAXVALUE)
);
```

Таким образом первая «таблица» (или партиция) будет хранить данные за «архивный» период, до 2008-го года, вторая — за 2008-й год, и «третья» — все остальное.

При этом совершенно не надо переписывать/оптимизировать запросы:

```
select * from orders_range where order_date='2009- 08-01';
```

И вот что при этом происходит:

```
mysql> explain partitions select * from orders_range3 where order_date='2008-08-01';
+----+-------------+---------------+------------+--------+---------------+------+---------+------+------+-------+
| id | select_type | table         | partitions | type   | possible_keys | key | key_len | ref | rows | Extra |
+----+-------------+---------------+------------+--------+---------------+------+---------+------+------+-------+
| 1  | SIMPLE      | orders_range3 |     p_2008 | system |          NULL | NULL |    NULL | NULL |    1 |       |
+----+-------------+---------------+------------+--------+---------------+------+---------+------+------+-------+
1 row in set (0.00 sec)
```

Т.е. при выполнении этого запроса работа будет идти исключительно с «подтаблицей» p_2008.

Более того, ускорение достигается даже в случае выполнения запросов, затрагивающих все данные во всех партициях — ведь в этом случае сначала происходит первичная «обработка» таблиц поменьше, потом данные объединяются и производятся финальные вычисления. Так вот как раз «первые» этапы в данном случае будут происходить гораздо быстрее.

Главное преимущество в том, что партиции с «оперативными» данными (т.е. последними, по которым наиболее часто происходит выборка) имеют минимальный размер, и как следствие, могут постоянно находится в оперативной памяти.

Следующие способы «разделения» данных предоставляет MySQL:

1. RANGE. По диапазону значений.

```
PARTITION BY RANGE (store_id) ( PARTITION p0
VALUES LESS THAN (10),
```

```
PARTITION p1 VALUES LESS THAN (20), PARTITION p3
VALUES LESS THAN (30) );
```

2. LIST. По точному списку значений.

```
PARTITION BY LIST(store_id) (
PARTITION pNorth VALUES IN (3,5,6,9,17), PARTITION pEast
VALUES IN (1,2,10,11,19,20)
)
```

Разбивать на партиции необходимо либо исходя из соображений оптимизации выборки (что чаще), либо исходя из соображений оптимизации записи (реже). Соответственно, идеальный вариант — это когда таблица разбивается на максимально возможное количество партиций так, что бы 90% всех выборок происходило в пределах одной партиции.

3. HASH

```
PARTITION BY HASH(store_id)
PARTITIONS 4;
```

Пользователь никак не управляет партицированием, просто указывается, по какому полю строить хеш и сколько «подтаблиц» создавать. В данном случае гораздо быстрее происходит выборка по указанному полю. В некоторых случаях позволяет достигнуть «равномерного разброса» и ускорения записи данных.

4. KEY. Аналогично HASH, но по ключу.

```
PARTITION BY KEY(s1)
PARTITIONS 10;
```

Т.е. выборка по указанному ключевому полю происходит максимально эффективно.

Но тут так же следует определиться со способом партицирования. Хорошо подходит для счетчика посетителей, когда его логин является единственным идентификатором, по которому необходимо выбирать все остальные данные.

2.3 Вертикальное партицирование таблиц

В MySQL **нет** вертикального партицирования. Вертикальное партицирование — это когда разные столбцы (поля) находятся в разных «подтаблицах». Поскольку иногда это бывает полезно, вы можете достичь

этого самостоятельно, пусть даже не так прозрачно: разделить таблицу на две, связав их по первичному ключу.

Зачем это делать? Например, в таблице, где хранятся в основном числа и даты, есть одно поле VARCHAR (255) для комментариев, которое используется на порядок реже чем остальные поля. В случае если его вынести в другую таблицу, то мы получим фиксированный размер строки (MySQL сможет совершенно точно вычислять позицию нужной строки по индексу в файле данных). Таблица станет более устойчивой к сбоям в случае внештатных ситуаций (опять же, из-за фиксированного размера строки). Ну и существенно уменьшится сам размер таблицы.

3 ВАРИАНТ ЗАДАНИЯ И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Используя методику, описанную в приложении, создать простой HTTP + MySQL сервер и реализовать партицирование. Вариант таблицы и число строк для запуска скрипта приведены ниже:

Вариант	Название таблицы (в скобках указан перечень полей)	Кол-во строк в таблице
1	articles (id, title, price)	100, 500, 1000, 2000
2	blogs (id, title, posts_count)	200, 700, 1300, 1900
2	cities (id, title, short_code)	300, 800, 1400, 1700
3	countries (id, title, area)	100, 500, 1000, 2000
4	feedbacks (id, title, body, email)	200, 700, 1300, 1900
5	news (id, title, body, created)	300, 800, 1400, 1700
6	parameters (id, code, value)	100, 500, 1000, 2000
7	sessions (id, created, expired)	200, 700, 1300, 1900
8	transactions (id, amount, created)	300, 800, 1400, 1700
9	users (id, first_name, last_name, email)	100, 500, 1000, 2000

4 СОДЕРЖАНИЕ ОТЧЕТА

Отчет должен содержать следующие пункты:

- постановка задачи;
- тексты программ с комментариями;
- графики зависимости кол-ва строк и времени работы запросов
- выводы.

5 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое партицирование таблиц?
2. Опишите виды партицирования таблиц.

3. Расскажите о преимуществах и недостатках различных видов партицирования таблиц.
4. Для каких случаев оптимизации доступа к данным нужно применять партицирование таблиц?

ЛАБОРАТОРНАЯ РАБОТА 6
«Исследование принципов репликации данных»

1 ЦЕЛЬ РАБОТЫ

Исследовать способы репликации баз данных и их влияние на скорость доступа с данным. Изучить основы репликации данных на примере MySQL.

2 ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

2.1 Основные определения

Репликация — это автоматическое тиражирование изменений, сделанных на одном сервере, на все остальные сервера. Обычно при использовании репликации изменения записываются всегда на один и тот же сервер — его называют master, а остальные копии — slave. В большинстве СУБД есть встроенные или внешние средства для организации репликации. Различают синхронную репликацию — в этом случае запрос на изменение данных будет ожидать, пока данные будут скопированы на все сервера, и лишь потом завершится успешно — и асинхронную — в этом случае изменения копируются на slave-сервера с задержкой, зато запрос на запись завершается быстрее.

Multi-master репликация. Этот подход аналогичен предыдущему, однако тут можно производить изменение данных, обращаясь не к одному определенному серверу, а к любой копии базы. При этом изменения синхронно или асинхронно попадут на другие копии. Иногда такую схему называют термином «кластер базы данных».

Методы репликации в MySQL 8. Традиционный метод основан на мультиплицировании событий от двоичного журнала master и требует, чтобы файлы системного журнала и позиции в них были синхронизированы между master и slave. Более новый метод, основанный на глобальных операционных идентификаторах (GTID), является транзакционным и поэтому не требует работы с файлами системного журнала или позициями в пределах этих файлов, что очень упрощает много общих задач репликации.

2.2 Несколько MySQL серверов на одной машине

Для выполнения данной лабораторной работы потребуется несколько MySQL серверов. Их можно развернуть на нескольких компьютерах или запустить на одной машине. Ниже приведено руководство для запуска нескольких MySQL серверов на одном локальном компьютере.

MySQL хранит таблицы всех баз данных и служебную информацию в определенном директории. Помимо этого, каждый экземпляр сервера должен иметь уникальную конфигурацию. При стандартной установке MySQL эти

параметры имеют значения по умолчанию, зависящие от конкретной операционной системы. Для отработки репликации можно использовать глобальный сервер в качестве одного из партнеров по репликации. Однако, с целью унифицировать механизм запуска серверов в примерах ниже глобальный сервер не используется, а все необходимые экземпляры запускаются отдельно.

Для запуска нескольких серверов необходимо создать по одному файлу конфигурации для каждого из них, и указать в каждом следующие параметры, уникальные для каждого экземпляра:

- **datadir**: путь к служебному директорию для размещения данных;
- **port**: номер TCP порта для подключения к серверу (по умолчанию 3306);
- **pid-file**: путь к файлу, хранящему номер процесса сервера MySQL;
- **socket**: на Unix/OSX/Linux системах возможно подключение к серверу не только по протоколу TCP, но и с использованием локального Unix socket. Файл, указанный в этом параметре, задает место расположения socket-файла. Если даже этот протокол не используется, данный путь должен быть уникальным для конкретного экземпляра сервера в системах Unix/OSX/Linux;
- **mysqlx**: версия 8 MySQL поддерживает по умолчанию X Protocol помимо собственного. Он также требует уникального номера порта в файле конфигурации. Если протокол не используется, его можно отключить, указав 0, чтобы исключить конфликт при запуске нескольких экземпляров сервера.

Пути в вышеуказанном файле могут быть абсолютными или относительными. Если есть сомнения, то следует обратиться к документации или использовать абсолютные пути. При этом даже на Windows системах разделители в путях в файле конфигурации используют прямые слэши ‘/’, а не обратные ‘\’.

Помимо приведенных выше, файл конфигурации может содержать прочие параметры. Например, для настройки репликации файл конфигурации master может содержать следующие параметры:

- **server-id**: уникальный идентификатор сервера для репликации;
- **gtid_mode** = ON для указания, что используется транзакционный метод репликации;
- **enforce_gtid_consistency** = ON необходим для GTID репликации;
- **default_authentication_plugin** = mysql_native_password для совместимости MySQL сервера версии 8 с модулем nodejs.

Для запуска двух серверов с репликацией master-slave на Windows машине следует выполнить нижеследующие действия, исправив пути в случае необходимости. Для Unix/OSX/Linux все аналогично с точностью до путей.

1. Создать каталоги для данных серверов. Пусть это будет для примера

```
C:\labs\mysqld1
C:\labs\mysqld2
```

2. Создать файлы конфигурации серверов со следующим содержимым:

```
# C:\labs\server1.cnf
[mysqld]
datadir         = "C:/labs/mysqld1"
port            = 3305
pid-file        = "C:/labs/mysqld1.pid"
socket          = "C:/labs/mysqld1.sock"
mysqlx          = 0
server-id       = 1
gtid_mode       = ON
enforce_gtid_consistency = ON
default_authentication_plugin = mysql_native_password

# C:\labs\server2.cnf
[mysqld]
datadir         = "C:/labs/mysqld2"
port            = 3307
pid-file        = "C:/labs/mysqld2.pid"
socket          = "C:/labs/mysqld2.sock"
mysqlx          = 0
server-id       = 2
gtid_mode       = ON
enforce_gtid_consistency = ON
default_authentication_plugin = mysql_native_password
```

В данных файлах указаны номера портов 3305 и 3307, чтобы исключить конфликт с сервером по умолчанию на порту 3306. Если требуется большее количество серверов, то их файлы конфигурации строятся по аналогии, обеспечивая уникальные значения параметров и избегая конфликта портов с сервером по умолчанию, если он запущен.

3. Проинициализировать каталоги. Один из вариантов – скопировать аналогичный каталог сервера по умолчанию со всеми находящимися там данными. Но более правильно будет проинициализировать данные с помощью самого сервера, выполнив в консоли Windows следующие команды:

```
"C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld" --defaults-
file="C:/labs/server1.cnf" --initialize-insecure

"C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld" --defaults-
file="C:/labs/server2.cnf" --initialize-insecure
```

В примере используется небезопасная инициализация --initialize-insecure, подразумевающая создание пользователя MySQL root без пароля. Для реального применения должен использоваться метод --initialize, создающий для пользователя root случайный пароль.

4. Запустить master сервер и создать аккаунт пользователя, используемого для репликации, для чего выполнить команду:

```
"C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld" --defaults-
file="C:/labs/server1.cnf"
```

Эту консоль следует оставить открытой с работающим сервером, после чего подключиться к нему с помощью клиента из другого окна консоли и выполнить серию SQL команд для настройки репликации на мастер-сервере:

```
"C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql" --protocol=tcp --port=3305 -
-user=root
```

Появится приглашение для ввода:

```
mysql>
```

Выполнить команды:

```
DROP USER IF EXISTS repl;
CREATE USER repl IDENTIFIED BY 'replPassword';
GRANT REPLICATION SLAVE ON *.* TO repl;
```

Консоль с клиентом можно оставить открытой, чтобы позже можно было проверить статус репликации.

5. Запустить каждый из slave серверов и выполнить на нем настройки репликации, указав в командах SQL номер порта, имя пользователя для репликации и пароль только что запущенного master сервера.

Запустить сервер в отдельной консоли:

```
"C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld" --defaults-
file="C:/labs/server2.cnf"
```

Запустить клиента и выполнить команды:

```
"C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql" --protocol=tcp --port=3307 -
-user=root

STOP SLAVE;
CHANGE MASTER TO MASTER_HOST='localhost', MASTER_PORT=3305, MASTER_USER='repl',
MASTER_PASSWORD='replPassword', MASTER_AUTO_POSITION=1;
START SLAVE;
```

Эти консоли (с сервером и клиентом) также следует оставить открытыми.

6. Теперь можно проверить статус репликации на каждом из серверов, выполнив следующие команды SQL в консолях клиентов:

На клиенте, подключенном к master:

```
SHOW MASTER STATUS \G
```

```
SHOW SLAVE HOSTS \G
```

Должно быть выведено примерно следующее:

```
*************************** 1. row ***************************
            File: binlog.000001
        Position: 842
    Binlog_Do_DB:
 Binlog_Ignore_DB:
Executed_Gtid_Set: 9b39fd4b-bc4f-11e9-9e3b-c86000c9e27b:1-3
1 row in set (0.00 sec)

*************************** 1. row ***************************
 Server_id: 2
      Host:
      Port: 3307
 Master_id: 1
Slave_UUID: a421553b-bc4f-11e9-b169-c86000c9e27b
1 row in set (0.00 sec)
```

Видно, что к master с идентификатором 1 подключен slave с идентификатором 2, работающий на порту 3307.

На клиенте, подключенном к slave:

```
SHOW SLAVE STATUS \G
```

Должно быть выведено примерно следующее:

```
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: localhost
                  Master_User: repl
                  Master_Port: 3305
                Connect_Retry: 60
              Master_Log_File: binlog.000001
          Read_Master_Log_Pos: 842
               Relay_Log_File: Desktop-relay-bin.000002
                Relay_Log_Pos: 1050
        Relay_Master_Log_File: binlog.000001
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 842
              Relay_Log_Space: 1260
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 1
```

```
                Master_UUID: 9b39fd4b-bc4f-11e9-9e3b-c86000c9e27b
           Master_Info_File: mysql.slave_master_info
                  SQL_Delay: 0
        SQL_Remaining_Delay: NULL
    Slave_SQL_Running_State: Slave has read all relay log; waiting for more updates
         Master_Retry_Count: 86400
                Master_Bind:
    Last_IO_Error_Timestamp:
   Last_SQL_Error_Timestamp:
             Master_SSL_Crl:
         Master_SSL_Crlpath:
         Retrieved_Gtid_Set: 9b39fd4b-bc4f-11e9-9e3b-c86000c9e27b:1-3
          Executed_Gtid_Set: 9b39fd4b-bc4f-11e9-9e3b-c86000c9e27b:1-3
              Auto_Position: 1
       Replicate_Rewrite_DB:
               Channel_Name:
         Master_TLS_Version:
     Master_public_key_path:
      Get_master_public_key: 0
          Network_Namespace:
1 row in set (0.00 sec)
```

Если получены такие таблицы, то все хорошо, master-slave репликация настроена и готова к работе. С этого момента любые операции над master сервером, включая создание новых баз данных и модификацию таблиц, будут автоматически реплицироваться на slave серверы.

Если в первой строке будет состояние, отличное от "Waiting for master to send event", то что-то пошло не так. Следует еще раз проверить все настройки и файлы конфигурации.

После проверки можно завершить работу клиентов, подключенных к серверам, выполнив в них команду выхода и закрыв консоли:

```
QUIT
```

Когда запущенные серверы больше не нужны, можно завершить их работу, нажав в каждом `Ctrl-C` и закрыв окно после завершения работы сервера.

Если требуется просто запустить несколько серверов без репликации между ними, то в файлах конфигурации не нужны параметры `server-id`, `gtid_mode` и `enforce_gtid_consistency`. Также не следует выполнять приведенные выше команды для настройки репликации. Каждый сервер при этом будет работать независимо от остальных, включая собственный набор баз данных, их таблиц и пользовательских аккаунтов.

3 ВАРИАНТ ЗАДАНИЯ И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Используя методику, описанную в приложении, создать простой HTTP + MySQL сервер и реализовать репликацию. Вариант таблицы, кол-во SLAVE серверов и номера портов для SLAVE серверов приведены в таблице ниже:

№	Название таблицы	Кол-во SLAVE серверов	Номера портов SLAVE серверов
1	articles (id, title, price)	2	3307 3308
2	blogs (id, title, posts_count)	3	3310 3311 3312
2	cities (id, title, short_code)	2	3308 3309
3	countries (id, title, area)	3	3311 3312 3313
4	feedbacks (id, title, body, email)	2	3307 3308
5	news (id, title, body, created)	3	3310 3311 3312
6	parameters (id, code, value)	2	3308 3309
7	sessions (id, created, expired)	3	3311 3312 3313
8	transactions (id, amount, created)	2	3307 3308
9	users (id, first_name, last_name, email)	3	3310 3311 3312

4 СОДЕРЖАНИЕ ОТЧЕТА

Отчет должен содержать следующие пункты:

- постановка задачи;
- тексты программ с комментариями;
- состояния MASTER и SLAVE серверов до и после запуска NODE JS программы.
- выводы.

5 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое репликация?
2. Какие виды репликации существуют?
3. Расскажите о преимуществах и недостатках различных видов репликации данных.
4. Для каких случаев оптимизации доступа к данным нужно применять репликации данных?

ЛАБОРАТОРНАЯ РАБОТА 7
«Исследование принципов шардинга данных»

1 ЦЕЛЬ РАБОТЫ

Исследовать способы шардинга баз данных и их влияние на скорость доступа с данным. Изучить основы шардинга на примере MySQL.

2 ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

2.1 Определение понятия шардинг. Виды шардинга

Шардинг — разделение данных на уровне ресурсов. Концепция шардинга заключается в логическом разделении данных по различным ресурсам исходя из требований к нагрузке.

Рассмотрим пример. Пусть у нас есть приложение с регистрацией пользователей, которое позволяет писать друг другу личные сообщения. Допустим, что оно очень популярно и много людей им пользуются ежедневно. Естественно, что таблица с личными сообщениями будет намного больше всех остальных таблиц в базе (скажем, будет занимать 90% всех ресурсов). Зная это, мы можем подготовить для этой (только одной) таблицы выделенный сервер помощнее, а остальные оставить на другом (послабее). Теперь мы можем идеально подстроить сервер для работы с одной специфической таблицей, постараться уместить ее в память, возможно, дополнительно партицировать ее и т.д. Такое распределение называется **вертикальным шардингом.**

Что делать, если таблица с сообщениями стала настолько большой, что даже выделенный сервер под нее одну уже не спасает? Необходимо делать **горизонтальный шардинг** - т.е. разделение одной таблицы по разным ресурсам. Т.е. на разных серверах будет таблица с одинаковой структурой, но разными данными. Для нашего случая с сообщениями, мы можем хранить первые 10 миллионов сообщений на одном сервере, вторые 10 - на втором и т.д. Т.е. необходимо иметь критерий шардинга - какой-то параметр, который позволит определять, на каком именно сервере лежат те или иные данные.

2.2 Основы реализации вертикального шардинга

Обычно, в качестве параметра шардинга выбирают ID пользователя (или любой другой головной сущности приложения) - это позволяет делить данные по серверам равномерно и просто. Т.о. при получении личных сообщений пользователей алгоритм работы будет такой:

1. Определить, на каком сервере БД лежат сообщения пользователя, исходя из user_id

2. Инициализировать соединение с этим сервером

3. Выбрать сообщения

4. Задачу определения конкретного сервера можно решать двумя путями:

Хранить в одном месте хеш-таблицу с соответствиями “пользователь-сервер”. Тогда, при определении сервера, нужно будет выбрать сервер из этой таблицы. В этом случае узкое место - это большая таблица соответствия, которую нужно хранить в одном месте. Для таких целей очень хорошо подходят базы данных “ключ-значение”

Определять имя сервера с помощью числового (буквенного) преобразования. Например, можно вычислять номер сервера, как остаток от деления на определенное число (количество серверов, между которыми Вы делите таблицу). В этом случае узкое место - это проблема добавления новых серверов - придется делать перераспределение данных между новым количеством серверов.

Для шардинга не существует решения на уровне известных платформ, т.к. это весьма специфическая для отдельно взятого приложения задача.

Естественно, делая горизонтальный шардинг, возникает ограничение в возможности выборок, которые требуют пересмотра всей таблицы (например, последние записи в блогах людей будет достать невозможно, если таблица постов шардится). Такие задачи придется решать другими подходами. Например, для описанного примера, можно при появлении нового поста, заносить его ID в общий стек, размером в 100 элементом.

Горизонтальный шардинг имеет одно явное преимущество - он бесконечно масштабируем.

3 ВАРИАНТ ЗАДАНИЯ И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Используя методику, описанную в приложении, создать простой HTTP + MySQL сервер, использующий горизонтальный шардинг с разного количества серверов. Вариант таблицы приведен ниже:

Вариант	Таблица*	Серверов*	Строк*
1	articles (id, title, price)	3	4
2	blogs (id, title, posts_count)	4	5
2	cities (id, title, short_code)	3	4
3	countries (id, title, area)	4	5
4	feedbacks (id, title, body, email)	3	4
5	news (id, title, body, created)	4	5
6	parameters (id, code, value)	3	4
7	sessions (id, created, expired)	4	5
8	transactions (id, amount, created)	3	4
9	users (id, first_name, last_name, email)	4	5

В таблице используются следующие обозначения*:
- *Таблица* - название таблицы (в скобках указан перечень полей)
- *Серверов* - Число серверов, на которых размещаются данные
- *Строк* - Максимальное кол-во строк в таблице на каждом из серверов

4 СОДЕРЖАНИЕ ОТЧЕТА

Отчет должен содержать следующие пункты:
- постановка задачи;
- тексты программ с комментариями;
- результаты выполнения программ;
- выводы.

5 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое шардинг?
2. Какие есть виды шардинга?
3. Что такое вертикальный шардинг и в чем его преимущества?
4. Что такое горизонтальный шардинг и в чем его преимущества?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Э. Танненбаум, М. Ван Стеен. Распределенные системы. Принципы и парадигмы. СПб: Питер 2003 г.

2. Д. Саймино. Сети интранет: внутреннее движение. М. "Book Media Publisher". 1997 г.

3. Создание intranet. Официальное руководство Microsoft. “BHV – Санкт Петербург”. 1998 г.

ПРИЛОЖЕНИЕ А

Полезные приемы JavaScript для nodejs

Ниже описан ряд приемов, использованных в примерах ниже. Они подразумевают наличие общего представления о JavaScript и его конкретной реализации в движке nodejs, включая стандарты ES6 и ES7. По этой же причине рекомендуется использовать версию nodejs не ниже 10, так как в более ранних версиях не гарантируется поддержка всех описанных возможностей. Более подробно об этих и других возможностях языка можно прочитать в «Современном учебнике JavaScript» по ссылке https://learn.javascript.ru/.

‘use strict’; – использовать strict mode. В этом режиме предъявляются более жесткие требования к коду. Например, запрещено использовать необъявленные переменные. Использование режима strict выявляет ряд ошибок, которые могли бы остаться незамеченными, и рекомендуется для всего кода.

Использование const и let для определения констант или переменных. В отличие от var, доставшегося в наследство от ранних версий javascript для браузеров, эти ключевые слова не только подчеркивают различие между константами и переменными, но и имеют более логичную область видимости: от момента определения до конца блока (функции) аналогично таким языкам, как C. Использование var с его побочными эффектами рекомендуется только в браузерах или в отдельных специальных случаях.

Использование записи (args) => { body } для определения функций. Эта запись частично аналогична записи function(args) { body }, но имеет определенные различия в области видимости переменных. Помимо прочего, эта запись короче и часто используется для описания функций в контексте выражения.

Использование строк вида `some text and some ${variable + 5}`. В отличие от строк в одинарных (апостроф) и двойных кавычках, запись с обратными кавычками (backticks) позволяет включать значения переменных и выражений, заключенных в ${}, а также записывать многострочный текст в естественном виде.

Использование async/await в форме async function(args) { let x = await myAsyncFunction() }. Как известно, традиционный JavaScript активно использует callbacks в силу своей асинхронности. При необходимости выполнить последовательно ряд асинхронных вызовов приходится вкладывать callback-функции одну в другую, что приводит к трудночитаемому коду и сложности обработки ошибок. Частичным решением проблемы являются так называемые promises, упрощающие запись последовательных асинхронных вызовов в виде MyPromiseFn(args).then(a).then(b).catch(err). Использование ключевого слова await еще более упрощает запись, делая ее внешне похожей на обычные синхронные вызовы функций, возвращающие результаты после завершения процессов. При этом код по-прежнему остается асинхронным, а все события (например, запущенный web-сервер) продолжают работать, пока await ожидает результата. Кроме того, обрабатывать ошибки в любом месте цепочки становится возможным с помощью привычного try/catch вместо вложения обработчика в каждый callback.

Указанные возможности пока не работают в браузерах до широкого внедрения новых стандартов (хотя есть механизмы траскодирования такого кода в стандартный JavaScript).

Однако, их использование со стороны сервера под nodejs крайне рекомендуется с точки зрения читабельности кода и удобства его написания.

ПРИЛОЖЕНИЕ Б

Полезные модули npm для реализации web-сервера с MySQL

`nodejs` имеет ряд встроенных модулей, таких как http. Дополнительный функционал требует установки внешних модулей. Для этого обычно используется npm – менеджер модулей. Команда `npm install` <модуль1> <модуль2>… устанавливает указанные модули в текущем каталоге.

`mysql` – модуль для работы с MySQL сервером. Использует традиционные callbacks для реализации асинхронных вызовов функций. После выхода новой версии MySQL 8 модуль перестал с ним работать, выдавая ошибку аутентификации из-за изменения метода аутентификации по умолчанию. Пока модуль не обновлен, обойти проблему можно, добавив в файл конфигурации базы данных параметр:
`default_authentication_plugin = mysql_native_password`
Базы, созданные с его использованием, нормально работают с модулем `mysql`.

`mysql2` – альтернативная версия модуля для работы с MySQL. Помимо того, что по отзывам он значительно быстрее, чем `mysql`, он поддерживает ряд дополнительных возможностей. В частности, полезным является его promise wrapper, использующий promises вместо callback функций. Как следствие, с ними можно использовать синтаксис async/await, что особенно удобно при реализации сравнения длительностей исполнения запросов в циклах. Комментарий об аутентификации модуля `mysql` в равной степени относится и к `mysql2`.

`express` – популярный web-фреймворк для nodejs, позволяющий красиво и эффективно реализовывать web-сервисы. Некоторыми из многих его достоинств являются удобная маршрутизация запросов в зависимости от пути в URL, встроенная обработка параметров запроса и поддержка различных шаблонизаторов для генерации HTML страниц по шаблонам.

`pug` – один из поддерживаемых модулем express шаблонизаторов, ранее известный как jade. Позволяет формировать web-страницы по шаблону с использованием условий. Использование pug для генерации HTML позволяет акцентировать внимание на разработку логики приложения и структуры страницы вместо того, чтобы вручную подсчитывать количество открывающих и закрывающих HTML-тегов, снижающих читаемость исходного кода.

`hirestime` – модуль, позволяющий кроссплатформенно выполнять точные замеры времени выполнения фрагментов кода.

Указанные выше модули использованы при написании примеров. Дополнительную информацию по ним можно найти в документации на соответствующие модули.

ПРИЛОЖЕНИЕ В

Создание простейшего HTTP сервера на nodejs

nodejs позволяет легко создать собственный HTTP сервер приложений без использования популярных web-серверов, таких как Apache или nginx. Пример кода приведен ниже. Запустить его можно с помощью команды `node server.js`. После запуска сервер работает на порту 3000, открыть страницу можно по адресу http://127.0.0.1:3000.

 # File: server.js

```
'use strict';

const http = require('http');
const server_host = '127.0.0.1';
const server_port = '3000';

http.createServer((req, res) => {
    let data = "Hello world";
    res.writeHead(200, {
        'Content-Type': 'text/html',
        'Content-Length': data.length,
    });
    res.write(data);
    res.end();
}).listen(server_port, server_host);

console.log(`Server running at http://${server_host}:${server_port}/, press
Ctrl-C to exit`);
```

ПРИЛОЖЕНИЕ Г

Создание HTTP сервера на nodejs с использованием express и pug

Пример показывает, как сформировать корректный HTML страницы по шаблону с параметрами, передав в него заголовок, текст параграфа и массив JavaScript произвольной длины с данными для отображения в виде таблицы. При этом логика приложения отделена от его внешнего представления, а тегов HTML в коде нет вообще – это задача шаблонизатора.

Код примера приведен ниже. Подкаталог `views` содержит файл `index.pug` – шаблон для создания HTML страницы. Для работы примера требуется предварительно установить модули: `npm install express pug` в каталоге приложения.

Запуск сервера и просмотр страницы аналогичны предыдущему примеру.

 # File: server.js

```
'use strict';

// npm install express pug
const express = require('express');

const server_port = '3000';
const server_host = '127.0.0.1';

const app = express();

app.set('view engine', 'pug');
app.get('/', (req, res, next) => {
    let data = [
        [ 1, 'Item 1' ],
        [ 2, 'Item 2' ],
        [ 3, 'Item 3' ],
    ];
    res.render('index', {
        header: 'Example Express web server using Pug template engine',
        text: 'The table below is shown using data from JavaScript array:',
        table: data,
    })
});
app.listen(server_port, server_host, () => {
    console.log(`Server running at http://${server_host}:${server_port}/, press
Ctrl-C to exit`);
});
```

 # File: views/index.pug

```
html
  head
    title= 'Example web server'
  body
    h2= header
    p= text
    p
      table
        for rec in table
          tr
```

```
td= rec[0]
td= rec[1]
```

ПРИЛОЖЕНИЕ Д

Пример реализации партицирования в MySQL

В данном примере используется один экземпляр MySQL сервера, в котором создается база данных с двумя таблицами, одна из которых использует партицирование. Обе таблицы заполняются идентичными данными, а код nodejs сравнивает время выполнения идентичных запросов к каждой из таблиц.

Пример содержит 5 файлов.

`database.cnf` – файл конфигурации MySQL сервера. Он используется для создания базы в выделенном каталоге.

`database.sql` – файл для создания базы данных. С его помощью создается новая база, новый аккаунт пользователя и две идентичных таблицы (одна – с партицированием). Первая таблица заполняется случайными данными с помощью функций MySQL, размещенных в сохраненной процедуре. Потом эти данные копируются из первой во вторую таблицы.

`server.js` – файл программы nodejs, осуществляющей доступ к данным и формирующий web-страницу с результатами.

`views/results.pug` – файл шаблона для Pug, используемый для форматирования вывода.

`views/style.css` – файл с таблицей стилей для более красивого вывода.

Порядок запуска:

Создать файлы (для примера они расположены в директории C:\lab7).

Выполнить инициализацию каталога данных MySQL:

```
"C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld" --defaults-
file="C:/lab7/database.cnf" --initialize-insecure
```

Запустить сервер в отдельной консоли:

```
"C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld" --defaults-
file="C:/lab7/database.cnf"
```

Создать базу данных, импортировав файл database.sql с помощью клиента MySQL:

```
"C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql" --protocol=tcp --port=3305 -
-user=root <database.sql
```

Установить модули nodejs, использованные в программе:

```
npm install express pug mysql2 hirestime
```

Запустить сервер nodejs:

```
node server.js
```

Открыть web-страницу по адресу http://127.0.0.1:3000 и получить результат.

 # File: database.cnf

```
[mysqld]
datadir         = "C:/lab7/mysqld"
port            = 3305
pid-file        = "C:/lab7/mysqld.pid"
socket          = "C:/lab7/mysqld.sock"
mysqlx          = 0
default_authentication_plugin = mysql_native_password
```

 # File: database.sql

```
--
-- Скрипт создания базы данных с данными.
--

-- Удалить старую базу и аккаунт, если они существуют
DROP DATABASE IF EXISTS lab7;
DROP USER IF EXISTS lab7User;

-- Создать новую базу и аккаунт
CREATE DATABASE lab7;
CREATE USER lab7User IDENTIFIED BY 'lab7Password';
GRANT ALL PRIVILEGES ON lab7.* TO lab7User;

-- Использовать созданную базу
USE lab7;

-- Создать таблицу без партицирования
CREATE TABLE IF NOT EXISTS `transactions_no_part` (
    `id` INTEGER NOT NULL AUTO_INCREMENT,
    `amount` DOUBLE NOT NULL DEFAULT 0.0,
    `created` DATE NOT NULL,
    PRIMARY KEY (`id`)
) ENGINE = InnoDB;

-- Создать таблицу с партицированием
CREATE TABLE IF NOT EXISTS `transactions_part` (
    `id` INTEGER NOT NULL AUTO_INCREMENT,
    `amount` DOUBLE NOT NULL DEFAULT 0.0,
    `created` DATE NOT NULL,
    PRIMARY KEY (`id`)
) ENGINE = InnoDB
  PARTITION BY HASH(`id`)
  PARTITIONS 4;


-- Создать процедуру для генерации случайных данных
DELIMITER //
CREATE PROCEDURE `generate_data` (IN items_count INT)
BEGIN
    DECLARE i INT DEFAULT 0;
    WHILE i < items_count DO
        INSERT INTO `transactions_no_part` (`amount`, `created`)  VALUES (RAND()
* 1000, CONCAT('2012-01-0', CAST((rand()*8+1) AS UNSIGNED)));
        SET i = i + 1;
```

```
    END WHILE;
    -- Скопировать те же данные в партицированную таблицу
    INSERT INTO `transactions_part`
        SELECT * FROM `transactions_no_part`;
END//
DELIMITER ;

-- Заполнить таблицу данными
CALL generate_data(1000);

-- Удалить процедуру
DROP PROCEDURE `generate_data`;
```

 # File: server.js

```
#!/usr/local/bin/node

const express = require('express');
const mysql = require('mysql2/promise');
const hirestime = require('hirestime')

'use strict';

// Параметры web-сервера
const server_host = 'localhost';
const server_port = '3000';

// Параметры подключения к MySQL
const conn = { host: 'localhost', port: 3305, database: 'lab7', user: 'lab7User',
password : 'lab7Password' };

// Номер id для оценки времени
const test_id = 100;

// Тестовые запросы
const queries = [
    { table: 'transactions_no_part', sql: `SELECT SQL_NO_CACHE * FROM
transactions_no_part LIMIT 1` },
    { table: 'transactions_no_part', sql: `SELECT SQL_NO_CACHE * FROM
transactions_no_part WHERE id = ${test_id} LIMIT 1` },
    { table: 'transactions_part',    sql: `SELECT SQL_NO_CACHE * FROM
transactions_part LIMIT 1` },
    { table: 'transactions_part',    sql: `SELECT SQL_NO_CACHE * FROM
transactions_part WHERE id = ${test_id} LIMIT 1` },
];

async function run() {
    // Установить соединение с сервером
    const db = await mysql.createConnection(conn);

    // Создать web сервис
    const app = express();

    // Использовать pug (бывший jade) как HTML template движок
    app.set('view engine', 'pug');

    // По запросу http://server:port/ выполнить 4 тестовых выборки из двух таблиц
    app.get('/', wrapAsync(async (req, res, next) => {
        // Получить количество элементов в таблице (только для отчета)
        const [ rows ] = await db.execute(`SELECT COUNT(*) AS n FROM
${queries[0].table}`);
```

```
        const n = rows[0].n;

        // Выполнить тестовые выборки, накапливая результаты в массиве results
        let results = [];
        for (let i in queries) {
            let time = hirestime();
            let [ rows ] = await db.execute(queries[i].sql);
            let elapsed = time(hirestime.MS).toFixed(2);

            results.push({
                n: n,
                sql: queries[i].sql,
                table: queries[i].table,
                time: elapsed,
            });
        }

        // Отобразить страницу результатов с помощью pug по шаблону results.pug
        res.render('results', { header: `Query results`, results: results });
    }));

    // Запустить HTTP сервер
    app.listen(server_port, server_host, () => {
        console.log(`Server running at http://${server_host}:${server_port}/,
press Ctrl-C to exit`);
    });
}

// Вспомогательная функция для обработки возможных ошибок в асинхронных функциях.
// Переназначает exceptions на стандартный обработчик ошибок Express.
function wrapAsync(fn) {
    return function(req, res, next) {
        fn(req, res, next).catch(next);
    };
}

run();
```

 # File: results.pug

```
html
  head
    title= 'Lab7'
    style
      include style.css
  body
    h2= header
    table
      tr
        th= 'Records'
        th= 'SQL query'
        th= 'SQL table'
        th= 'Time (ms)'
        for res in results
          tr
            td= res.n
            td= res.sql
            td= res.table
            td= res.time
```

```
# File: style.css

table {
    border-collapse: collapse;
}
th, td {
    border: 1px solid blue;
    padding: 5px;
}
```

ПРИЛОЖЕНИЕ Е

Пример реализации репликации в MySQL

В данном примере используется два экземпляра MySQL сервера, один из которых является мастером репликации. Данные, изменяемые на этом сервере, автоматически реплицируются на slave сервер.

Для запуска данного примера необходимо предварительно настроить репликацию с одного master сервера на один или более slave серверов так, как было описано выше. Ниже подразумевается, что репликация уже настроена, master работает на порту 3305, а slave – на порту 3307. При желании можно добавить больше slave серверов, отредактировав файл server-slaves.js.

Назначение файлов:

`database.sql` – файл для создания схемы базы данных.

`server-master.js` – простейшее приложение nodejs, осуществляющее добавление записей в базу данных master посредством web-интерфейса. Каждое открытие страницы приводит к добавлению одной новой записи в базу данных.

`server-slaves.js` – приложение nodejs, при открытии страницы выполняющее выборку из всех баз данных (master и все slaves) и демонстрацию их содержимого.

`views/master.pug` и `views/slaves.pug` – шаблоны HTML страниц.

`views/style.css` – файл с таблицей стилей.

Порядок запуска:

Настроить репликацию двух серверов и создать остальные файлы.

Создать базу данных, импортировав файл database.sql с помощью клиента MySQL в базу данных на сервере master. Поскольку настроена репликация, копия этой базы будет автоматически создана на сервере slave.

```
"C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql" --protocol=tcp --port=3305 --user=root <database.sql
```

Установить модули nodejs, использованные в программе:

```
npm install express pug mysql2
```

Запустить сервер для добавления записей:

```
node server-master.js
```

Запустить сервер для просмотра записей во всех серверах:

```
node server-slaves.js
```

Открыть web-страницу по адресу http://127.0.0.1:3001 и убедиться, что оба сервера доступны и содержат базу с пустой таблицей записей.

Открыть web-страницу по адресу http://127.0.0.1:3000, добавив тем самым одну запись в базу master. Перечитав (обычно по F5) первую страницу, убедиться, что новая запись была добавлена в базу master и была реплицирована в базу slave. В целях упрощения кода автоматическое обновление страниц в данном примере не реализовано.

 # File: database.sql

```
--
-- Скрипт создания базы данных с данными.
--

-- Удалить старую базу и аккаунт, если они существуют
DROP DATABASE IF EXISTS lab8;
DROP USER IF EXISTS lab8User;

-- Создать новую базу и аккаунт
CREATE DATABASE lab8;
CREATE USER lab8User IDENTIFIED BY 'lab8Password';
GRANT ALL PRIVILEGES ON lab8.* TO lab8User;

-- Использовать созданную базу
USE lab8;


-- Создать таблицу
CREATE TABLE IF NOT EXISTS `transactions` (
    `id` INTEGER NOT NULL AUTO_INCREMENT,
    `amount` DOUBLE NOT NULL DEFAULT 0.0,
    `created` DATETIME NOT NULL DEFAULT NOW(),
    PRIMARY KEY (`id`)
) ENGINE = InnoDB;
```

 # File: server-master.js

```
#!/usr/local/bin/node

const express = require('express');
const mysql = require('mysql2/promise');

'use strict';

// Параметры web-сервера
const server_host = 'localhost';
const server_port = '3000';

// Параметры подключения к MySQL master
```

```
const conn = { host: 'localhost', port: 3305, database: 'lab8', user: 'lab8User',
password : 'lab8Password' };

async function run() {
    // Установить соединение с сервером
    const master = await mysql.createConnection(conn);

    // Создать web сервис
    const app = express();

    // Использовать pug (бывший jade) как HTML template движок
    app.set('view engine', 'pug');

    // По запросу http://server:port/ добавить новую запись в базу master
    app.get('/', wrapAsync(async (req, res, next) => {
        // Создать запись со случайным параметром amount
        let amount = (Math.random() * 1000).toFixed(3);
        await master.execute('INSERT INTO transactions SET amount = ?', [ amount
]);

        // Отобразить страницу с помощью pug по шаблону master.pug
        res.render('master', {
            header: `Writer to master at ${conn.host}:${conn.port}`,
            text: `New entry has been added to the master database with the
following amount: ${amount}`,
        });
    }));

    // Запустить HTTP сервер
    app.listen(server_port, server_host, () => {
        console.log(`Master writer running at
http://${server_host}:${server_port}/, press Ctrl-C to exit`);
    });
}

// Вспомогательная функция для обработки возможных ошибок в асинхронных функциях.
// Переназначает exceptions на стандартный обработчик ошибок Express.
function wrapAsync(fn) {
    return function(req, res, next) {
        fn(req, res, next).catch(next);
    };
}

run();
```

 # File: server-slaves.js

```
#!/usr/local/bin/node

const express = require('express');
const mysql = require('mysql2/promise');

'use strict';

// Параметры web-сервера
const server_host = 'localhost';
const server_port = '3001';

// Параметры подключения к MySQL
const conn = [
    { host: 'localhost', port: 3305, database: 'lab8', user: 'lab8User', password
```

```
: 'lab8Password' },
    { host: 'localhost', port: 3307, database: 'lab8', user: 'lab8User', password
: 'lab8Password' },
];

async function run() {
    // Установить соединение с серверами
    const master = await mysql.createConnection(conn[0]);
    const slave1 = await mysql.createConnection(conn[1]);

    // Создать web сервис
    const app = express();

    // Использовать pug (бывший jade) как HTML template движок
    app.set('view engine', 'pug');

    // По запросу http://server:port/ получить все записи из базы данных с
серверов
    app.get('/', wrapAsync(async (req, res, next) => {
        // Выбрать записи
        let rows = [];
        [ rows[0] ] = await master.execute('SELECT * FROM transactions ORDER BY
id');
        [ rows[1] ] = await slave1.execute('SELECT * FROM transactions ORDER BY
id');

        // Отобразить страницу с помощью pug по шаблону slaves.pug
        res.render('slaves', {
            header: `Database content from all servers`,
            names: [ 'master', 'slave1' ],
            conn: conn,
            rows: rows,
        });
    }));

    // Запустить HTTP сервер
    app.listen(server_port, server_host, () => {
        console.log(`Slave reader running at
http://${server_host}:${server_port}/, press Ctrl-C to exit`);
    });
}

// Вспомогательная функция для обработки возможных ошибок в асинхронных функциях.
// Переназначает exceptions на стандартный обработчик ошибок Express.
function wrapAsync(fn) {
    return function(req, res, next) {
        fn(req, res, next).catch(next);
    };
}

run();
```

 # File: master.pug

```
html
  head
    title= 'Lab8 - master'
    style
      include style.css
  body
```

```
    h2= header
    p= text
```

 # File: slaves.pug

```
html
  head
    title= 'Lab8 - slaves'
    style
      include style.css
  body
    h2= header
    each _, i in rows
      h3= 'MySQL ' + names[i] + ' at ' + conn[i].host + ':' + conn[i].port
      table
        tr
          th= 'id'
          th= 'amount'
          th= 'created'
          for rec in rows[i]
            tr
              td= rec.id
              td= rec.amount
              td= rec.created
```

 # File: style.css

```
table {
    border-collapse: collapse;
}

th, td {
    border: 1px solid blue;
    padding: 5px;
}
```

ПРИЛОЖЕНИЕ Ж

Пример реализации шардинга в MySQL

Данный пример демонстрирует реализацию шардинга в MySQL. В конкретном случае предполагается, что возможный диапазон номеров записей – от 0 до 11. Имеется 3 сервера, каждый из которых хранит, соответственно, записи с 0 по 3, с 4 по 7, с 8 по 11. Нужный номер сервера определяется по id записи, после чего осуществляется обращение к нужному серверу для получения данных.

Для запуска примера необходимо предварительно настроить 3 MySQL сервера. Ниже предполагается, что они работают на портах 3305, 3307 и 3308, соответственно. Каждый из серверов содержит одинаковую схему базы данных. Заполнение баз должно осуществляться с учетом id записи. В данном примере для этого используется сохраненная процедура generate_data, первым параметром получающая начальный номер записи, и вторым – количество записей, начиная с этого номера.

Назначение файлов:

`database.sql` – файл для создания схемы базы данных.

`server.js` – приложение nodejs, осуществляющее вывод всех записей со всех серверов, а также выборку конкретной записи по ее id с соответствующего сервера.

`views/index.pug` и `views/record.pug` – шаблоны HTML страниц для вывода полной таблицы и отдельной записи, соответственно.

`views/style.css` – файл с таблицей стилей.

Порядок работы:

Настроить три независимых MySQL сервера.

Импортировать схему базы данных и процедуру генерации данных из файла database.sql в каждый сервер:

```
"C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql" --protocol=tcp --port=3305 --user=root <database.sql
"C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql" --protocol=tcp --port=3307 --user=root <database.sql
"C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql" --protocol=tcp --port=3308 --user=root <database.sql
```

Заполнить каждую из трех баз данных записями в соответствии с диапазонами id (для разнообразия последняя из баз содержит меньшее количество записей). В примере использована команда echo для передачи команд клиентам MySQL. Вместо нее можно просто подключиться к соответствующему серверу и выполнить те же команды из консоли:

```
echo USE lab9; CALL generate_data(0, 4) | "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql" --protocol=tcp --port=3305 --user=root
```

```
echo USE lab9; CALL generate_data(4, 4) | "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql" --protocol=tcp --port=3307 --user=root
echo USE lab9; CALL generate_data(8, 3) | "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql" --protocol=tcp --port=3308 --user=root
```

Установить модули nodejs, использованные в программе:

```
npm install express pug mysql2
```

Запустить сервер nodejs:

```
node server.js
```

При открытии страницы по адресу http://127.0.0.1:3000 выводится полный список записей со всех серверов. Выбрав нужную ссылку, можно запросить конкретную запись по ее id. При этом будет показана данная запись при ее наличии и указано, с какого из серверов она была получена. Код также обрабатывает ошибки в случае отсутствия требуемой записи, а также при выходе id за пределы допустимого диапазона.

 # File: database.sql

```
--
-- Скрипт создания базы данных
--

-- Удалить старую базу и аккаунт, если они существуют
DROP DATABASE IF EXISTS lab9;
DROP USER IF EXISTS lab9User;

-- Создать новую базу и аккаунт
CREATE DATABASE lab9;
CREATE USER lab9User IDENTIFIED BY 'lab9Password';
GRANT ALL PRIVILEGES ON lab9.* TO lab9User;

-- Использовать созданную базу
USE lab9;


-- Создать таблицу
CREATE TABLE IF NOT EXISTS `transactions` (
    `id` INTEGER NOT NULL,
    `amount` DOUBLE NOT NULL DEFAULT 0.0,
    `created` DATETIME NOT NULL DEFAULT NOW(),
    PRIMARY KEY (`id`)
) ENGINE = InnoDB;


-- Создать процедуру для генерации случайных данных
DELIMITER //
CREATE PROCEDURE `generate_data` (IN start_id INT, IN items_count INT)
BEGIN
    DECLARE i INT DEFAULT start_id;
    WHILE i < start_id + items_count DO
        INSERT INTO `transactions` (`id`, `amount`)  VALUES (i, RAND() * 1000);
        SET i = i + 1;
    END WHILE;
```

```
END//
DELIMITER ;

# File: server.js
#!/usr/local/bin/node

const express = require('express');
const mysql = require('mysql2/promise');

'use strict';

// Параметры web-сервера
const server_host = 'localhost';
const server_port = '3000';

// Количество серверов и максимальное количество записей на сервере согласно
варианту
n_servers = 3
n_records = 4;

// Имена серверов для отображения
const server_names = [ 'server1', 'server2', 'server3' ];

// Параметры подключения к MySQL
const conn = [
    { host: 'localhost', port: 3305, database: 'lab9', user: 'lab9User', password
: 'lab9Password' },
    { host: 'localhost', port: 3307, database: 'lab9', user: 'lab9User', password
: 'lab9Password' },
    { host: 'localhost', port: 3308, database: 'lab9', user: 'lab9User', password
: 'lab9Password' },
];

async function run() {
    // Установить соединение с серверами
    let db = [];
    for (let i = 0; i < conn.length; i++) {
        db[i] = await mysql.createConnection(conn[i]);
    }

    // Создать web сервис
    const app = express();

    // Использовать pug (бывший jade) как HTML template движок
    app.set('view engine', 'pug');

    // По запросу http://server:port/ получить все записи из базы данных с
серверов
    app.get('/', wrapAsync(async (req, res, next) => {
        // Показать все записи
        let rows = [];
        for (let i = 0; i < db.length; i++) {
            [ rows[i] ] = await db[i].execute('SELECT * FROM transactions ORDER
BY id');
        }

        // Отобразить страницу с помощью pug по шаблону index.pug
        res.render('index', {
            header: `Database content from all servers`,
            names: server_names,
            conn: conn,
```

```
            rows: rows,
        });
    }));

    // По запросу http://server:port/id/<id> получить нужную запись с
соответствующего сервера
    app.get('/id/:id', wrapAsync(async (req, res, next) => {
        // Получить id из запроса
        let id = req.params.id;
        let max_id = n_servers * n_records - 1;

        // Вернуть ошибку, если вне допустимого диапазона
        if ((id < 0) || (id > max_id)) {
            res.render('record', {
                header: `Record ${id} is out of range`,
                msg: `Valid range is ${0}..${max_id}`,
                rows: [],
            });
            return;
        }

        // Найти номер сервера по id записи
        let server_id = Math.floor(id / n_records);

        // Использовать соответствующее подключение для выборки записи
        const [ rows ] = await db[server_id].execute('SELECT * FROM transactions
WHERE id = ?', [ id ]);

        // Отобразить страницу с помощью pug по шаблону record.pug
        if (rows.length) {
            res.render('record', {
                header: `Record ${id} was found on ${server_names[server_id]}`,
                rows: rows,
            });
        } else {
            res.render('record', {
                header:    `Record    ${id}    is    expected    to    be    on
${server_names[server_id]}, but was not found`,
                rows: rows,
            });
        }
    }));

    // Запустить HTTP сервер
    app.listen(server_port, server_host, () => {
        console.log(`Server  running  at  http://${server_host}:${server_port}/,
press Ctrl-C to exit`);
    });
}

// Вспомогательная функция для обработки возможных ошибок в асинхронных функциях.
// Переназначает exceptions на стандартный обработчик ошибок Express.
function wrapAsync(fn) {
    return function(req, res, next) {
        fn(req, res, next).catch(next);
    };
}

run();
```

 # File: index.pug

```
html
  head
    title= 'Lab9'
    style
      include style.css
  body
    h2= header
    each _, i in rows
      h3= 'MySQL ' + names[i] + ' at ' + conn[i].host + ':' + conn[i].port
      table
        tr
          th= 'id'
          th= 'amount'
          th= 'created'
          th= 'link'
          for rec in rows[i]
            tr
              td= rec.id
              td= rec.amount
              td= rec.created
              td
                a(href='/id/' + rec.id) view
```

 # File: record.pug

```
html
  head
    title= 'Lab9'
    style
      include style.css
  body
    h2= header
    if msg
      p= msg
    if rows.length
      table
        tr
          th= 'id'
          th= 'amount'
          th= 'created'
          for rec in rows
            tr
              td= rec.id
              td= rec.amount
              td= rec.created
```

 # File: style.css

```
table {
    border-collapse: collapse;
}

th, td {
    border: 1px solid blue;
    padding: 5px;
}

a {
    color: inherit;
```

```
    text-decoration: none;
}

a:hover {
    color: #F00;
    text-decoration: underline;
}
```