Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Дисциплина «Структуры и базы данных»

**ОТЧЕТ**

к практическому занятию

на тему:

**«Представления. Хранимые процедуры. Триггеры»**

|  |
| --- |
| Выполнил студент группы 983871  КАФТАНЧИКОВА Анастасия Олеговна |
|  |
| (подпись студента, дата) |
| Проверил ассистент ПИКС  Демидович Вероника Константиновна |
|  |
| (подпись преподавателя, дата) |

Минск 2022

**Цель:** Изучить и подробно описать, представления, хранимые процедуры, триггеры.

**Представления**

Основная структурная единица реляционных БД – таблицы, но язык SQL предоставляет еще один способ организации данных. Представление – это запрос на выборку, которому присваивается уникальное имя и который можно сохранять или удалять из БД как хранимую процедуру. Представления позволяют увидеть результаты сохраненного запроса так, как будто это полноценная таблица. MySQL, встретив в запросе ссылку на представление, ищет его определение в БД. Пользовательский запрос с участием представления преобразуется в эквивалентный запрос к исходным таблицам. Если определение представления простое, то каждая строка представления формируется «на лету». Если определение сложное, MySQL материализует представление в виде временной таблицы. Клиент, обращаясь к представлению, будет видеть только столбцы результирующей таблицы. Не имеет значения, сколько столбцов в исходной таблице и является ли запрос, лежащий в основе представления, одно- или многотабличным. Клиенту можно запретить обращаться к исходным таблицам, но снабдить привилегиями обращения к представлениям. На одном наборе таблиц можно создать гибкие системы доступа.

Преимущества представлений:

* безопасность – каждый пользователь имеет доступ к небольшому числу представлений, содержащих только ту информацию, которую ему позволено знать;
* простота запросов – с помощью представления можно извлечь данные из нескольких таблиц и представить их как одну таблицу (запрос ко многим таблицам заменяется однотабличным запросом к представлению);
* простота структуры – представления позволяют создать для каждого пользователя собственную структуру БД (отображаются данные, которые ему нужны);
* защита от изменений – таблицы и их структура могут постоянно изменяться и переименовываться; представления позволяют создавать виртуальные таблицы со старыми именами и структурой, позволяя избежать модификации приложений.

Недостатки представлений:

* производительность – представления создают видимость существования таблицы, и MySQL приходится преобразовывать запрос к представлению в запрос к исходным таблицам; если представление отображает многотабличный запрос, то простой запрос к представлению становится сложным объединением;
* ограничение на обновление – когда пользователь пытается обновить строки представления, MySQL необходимо обновить строки в исходных таблицах; это возможно только для простых представлений, сложные представления доступны только для выборки.

Поэтому не стоит везде применять представления вместо исходных таблиц. Создание представлений. Осуществляется при помощи следующего оператора:

CREATE

[OR REPLACE]

[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]

[DEFINER = user]

[SQL SECURITY { DEFINER | INVOKER }]

VIEW view\_name [(column\_list)]

AS select\_statement

[WITH [CASCADED | LOCAL] CHECK OPTION]

Оператор создает представление *view\_name* со столбцами, перечисленными в *column\_list*, на основании запроса *select\_statement*. Рассмотрим создание представления *active\_account*, которое будет отображать активные аккаунты для отделения №2 (здесь и далее используется пример базы данных из лабораторной работы №4):

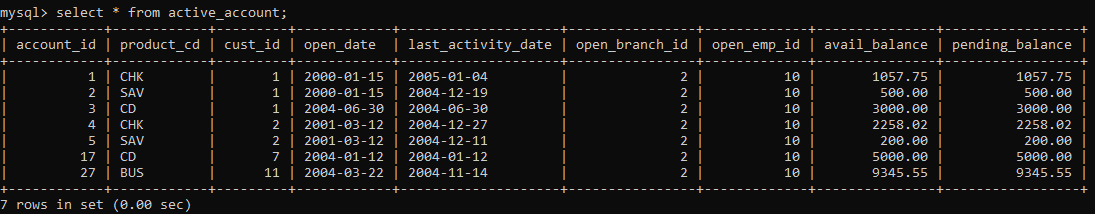
CREATE VIEW active\_account

AS SELECT account\_id, product\_cd, cust\_id, open\_date, last\_activity\_date, open\_branch\_id, open\_emp\_id, avail\_balance, pending\_balance

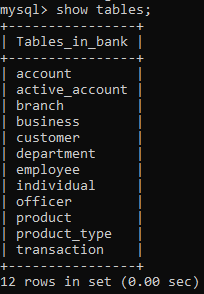
FROM account

WHERE open\_branch\_id = 2;

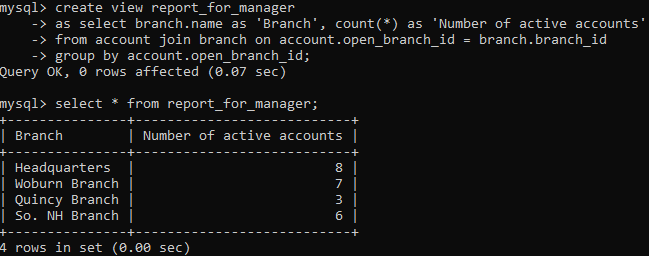
Для того, чтобы просмотреть результаты работы представления, используется оператор SELECT:



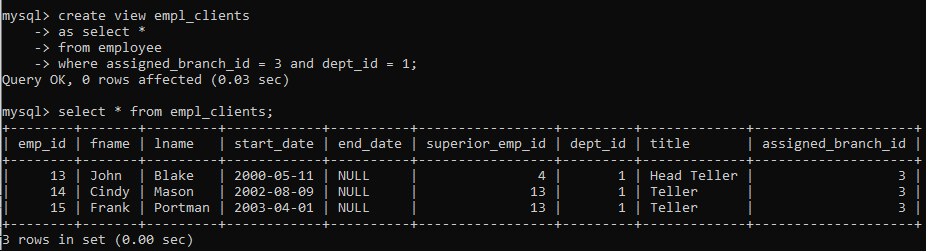
Представление рассматривается как полноценная таблица и может быть просмотрено в списке таблиц БД при помощи оператора SHOW TABLES:



Представления, которые отображают только определенные столбцы из таблицы, называются **вертикальными**. А если представление ограничивает просмотр определенных строк из таблицы, то оно называется **горизонтальным**. Пример вертикального представления представлен ниже:



Пример горизонтального представления:



На практике чаще всего используются смешанные представления, представляющие собой ограничение как на просматриваемые столбцы, так и на строки.  
 Стоит отметить, что в представлениях могут использоваться все те операторы, что и в обычном SELECT запросе.

Удаление представлений. Выполняется с помощью оператора:

DROP VIEW [IF EXISTS] view\_name [, view\_name] ... ;

Оператор позволяет уничтожить одно или несколько представлений, например:

**Хранимая процедура** – это скомпилированный набор SQL-предложений, сохраненный в базе данных как именованный объект и выполняющийся как единый фрагмент кода. Хранимые процедуры могут принимать и возвращать параметры. Когда пользователь создает хранимую процедуру, сервер компилирует ее и помещает в разделяемый кэш, после чего скомпилированный код может быть применен несколькими пользователями. Когда приложение использует хранимую процедуру, оно передает ей параметры, если таковые требуются, и сервер выполняет процедуру без перекомпиляции.

Хранимые процедуры позволяют повысить производительность приложений. Во-первых, по сравнению с обычными SQL-запросами, посылаемыми из клиентского приложения, они требуют меньше времени для подготовки к выполнению, поскольку они уже скомпилированы и сохранены. Во-вторых, сетевой трафик в этом случае также меньше, чем в случае передачи SQL-запроса, так как по сети передается меньшее количество данных.

Хранимые процедуры автоматически перекомпилируются, если с объектами, на которые они влияют, произведены какие-либо изменения; иными словами, они всегда актуальны. Как уже было сказано выше, хранимые процедуры могут принимать параметры, что позволяет разным приложениям использовать одну и ту же процедуру, применяя различные наборы входных данных.

Хранимые процедуры обычно используются для поддержки ссылочной целостности данных и реализации бизнес-правил. В последнем случае достигается дополнительная гибкость, поскольку, если бизнес-правила изменяются, можно изменить только текст процедуры, не изменяя клиентских приложений.

Для создания, изменения и удаления процедур существуют специальные SQL-предложения – CREATE PROCEDURE, ALTER PROCEDURE и DROP PROCEDURE.

CREATE

[DEFINER = user]

PROCEDURE sp\_name ([proc\_parameter[,...]])

[characteristic ...] routine\_body

proc\_parameter:

[ IN | OUT | INOUT ] param\_name type

characteristic:

COMMENT 'string'

| LANGUAGE SQL

| [NOT] DETERMINISTIC

| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }

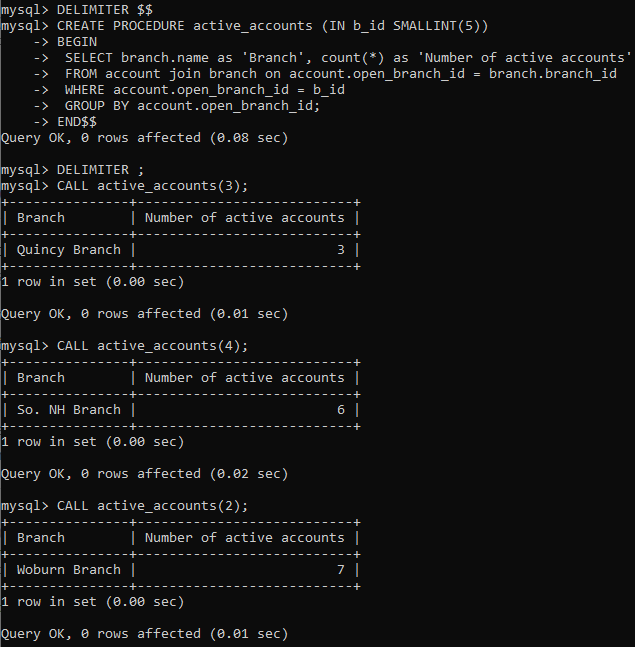
| SQL SECURITY { DEFINER | INVOKER }

routine\_body:

допустимый SQL-оператор

Как видно из определения оператора создания процедуры, она может принимать и возвращать параметры. Для передачи параметра в процедуру необходимо обозначить его как IN. Процедура может изменить значение переданного ей параметра, но данное изменение не будет видно вызывающей стороне. Чтобы показать, что процедура возвращает какой-либо параметр, его необходимо указать как OUT. Его начальное значение в процедуре равно NULL и видно вызывающей стороне. Если параметр указан как INOUT, то он инициализируется вызывающей стороной, может быть изменен процедурой, и любое изменение, внесенное процедурой, становится видимым вызывающей стороне, когда процедура вернет результат.

Создадим процедуру, которая будет предоставлять по запросу отчет о количестве открытых счетов в каком-либо отделении:



Стоит обратить внимание на следующий момент: так как процедура является набором других SQL-запросов, для разделения которых будут использоваться стандартный знак «;», то перед созданием процедуры необходимо переопределить данный стандартный разделить на другой, чтобы СУБД могла отличить начало и конец самой процедуры и запросов, находящихся внутри нее. Для переопределения стандартного разделителя используется оператор DELIMITER. После переопределения разделителя можно создавать процедуру.

Тело процедуры обернуто в конструкцию операторов BEGIN…END. В теле процедуры могут быть абсолютно любые команды SQL.

Для того, чтобы увидеть результаты работы процедуры используется оператор CALL, после которого указывается имя процедуры и, если есть, принимаемые параметры.

Для изменения параметров процедуры используется оператор ALTER PROCEDURE, который имеет следующий синтаксис:

ALTER PROCEDURE sp\_name [characteristic ...]

characteristic:

COMMENT 'string'

| LANGUAGE SQL

| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }

| SQL SECURITY { DEFINER | INVOKER }

Необходимо учитывать, что изменить тело процедуры через указанный выше оператор не удастся. Если нужно модифицировать сами запросы внутри процедуры, то нужно удалить уже созданную процедуру и заново ее создать с нужными изменениями.

Для удаления процедуры используется оператор DROP PROCEDURE:

DROP PROCEDURE [IF EXISTS] sp\_name

**Триггер** – эта та же хранимая процедура, но привязанная к событию изменения содержимого конкретной таблицы. Данная процедура не вызывается непосредственно, а исполняется при наступлении определенного события. Возможны три события, связанных с изменением содержимого таблицы, к которым можно привязать триггер:

* insert – вставка новых данных в таблицу;
* delete – удаление данных из таблицы;
* update – обновление данных в таблице.

Для создания триггера используется оператор CREATE TRIGGER, который имеет следующий синтаксис:

CREATE

[DEFINER = user]

TRIGGER trigger\_name

trigger\_time trigger\_event

ON tbl\_name FOR EACH ROW

[trigger\_order]

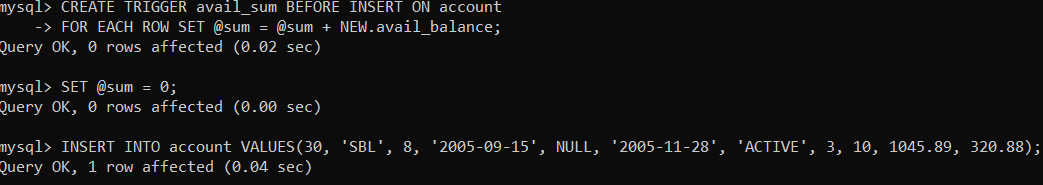
trigger\_body

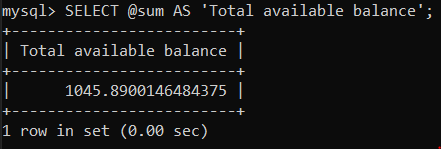
trigger\_time: { BEFORE | AFTER }

trigger\_event: { INSERT | UPDATE | DELETE }

trigger\_order: { FOLLOWS | PRECEDES } other\_trigger\_name

В теле триггера можно ссылаться на столбцы в таблице, связанной с триггером, используя псевдонимы OLD и NEW. OLD.col\_name ссылается на столбец существующей строки перед ее **обновлением** или **удалением**. NEW.col\_name ссылается на столбец новой строки для вставки или существующей строки после ее обновления. Пример приведен ниже:





Данный триггер будет увеличить сумму доступного баланса каждый раз, когда в таблицу account будут добавляться новые записи.

Для удаления триггера используется оператор DROP TRIGGER, синтаксис которого представлен ниже:

DROP TRIGGER [IF EXISTS] [schema\_name.]trigger\_name

**Выводы:** изучил и подробно описал, представления, хранимые процедуры, триггеры.