## Лабораторная работа № 7. MySql. Организация транзакций. Триггеры

|  |  |
| --- | --- |
| **Задание** | **Краткие теоретические сведения** |
| 1. Разработать сценарий, демонстрирующий свойства *транзакции* на примере базы данных **BaseMysql\_x**. | **Транзакция** − это механизм базы данных, позволяющий таким образом объединять несколько операторов, изменяющих базу данных, чтобы при выполнении этой совокупности операторов они или все выполнились или все не выполнились.  Если закончился товар с наименованием «Стол», то можно создать хранимую процедуру для осуществления простой транзакции и выполнить процедуру с помощью оператора Call.  **DELIMITER //**  **CREATE PROCEDURE TovarDel(tv varchar(20))**  **BEGIN**  **DECLARE EXIT HANDLER FOR SQLEXCEPTION**  **BEGIN**  **ROLLBACK; #откат транзакции в случае ошибки**  **END;**  **START transaction;**  **DELETE from Заказы where Наименование\_товара = tv;**  **DELETE from Товары where Наименование = tv;**  **COMMIT;**  **END//**  **DELIMITER ;**  **CALL TovarDel('Стол');**  COMMIT означает фиксацию всех изменений в транзакции. ROLLBACK означает отмену изменений, произошедших в результате выполнения транзакции. Однако в MySql не рекомендуется использовать хранимые процедуры для организации транзакций, так как в основном ошибки базы данных обрабатываются на стороне приложений на языке, например, PHP. |
| 2. Разработать сценарий, демонстрирующий свойство After-*триггера* на примере базы данных **BaseMysql\_x**. | **Триггер** – это особый вид хранимой процедуры, предназначенной для обработки событий UPDATE, DELETE и INSERT в базе данных. Они могут быть выполнены либо до (Before-триггеры), либо после события (After-триггеры).  Предположим, что каждый раз при добавлении строк в таблицу **Товары** нужно определять общую стоимость добавленных товаров в глобальной переменной **@sum**. Предварительно следует очистить эту переменную с помощью оператора: **set @sum = 0;**  Триггер для подсчета суммы:  **DELIMITER //**  **CREATE TRIGGER Insert\_sum After INSERT**  **ON Товары FOR EACH ROW**  **begin**  **SET @sum = @sum + NEW.Цена;**  **end; //**  **DELIMITER ;**  Если выполнить оператор добавления товаров  **INSERT into Товары (Наименование, Цена, Количество)**  **values ('Зеркало', 60, 10),**  **values ('Ковер', 260, 20);**  то можно вывести полученную сумму, использовав команду: **SELECT @sum;** |
| 3. Разработать сценарий, демонстрирующий работу Before-*триггера* на примере базы данных **BaseMysql\_x**. Создать After-триггер для этой же базы данных. | Пусть требуется при удалении строки из таблицы **Заказы** часть полей удаленной строки сохранить в таблице **Zakazy\_Deleted**. Для создания таблицы **Zakazy\_Deleted** служит скрипт:  **CREATE TABLE Zakazy\_Deleted**  **( Наименование\_удал varchar(20),**  **Цена\_удал real,**  **Заказчик\_удал varchar(20)**  **);**  Before-триггер, который срабатывает при удалении строки из таблицы Заказы:  **delimiter //**  **CREATE TRIGGER Trig\_Del After Delete**  **ON Заказы FOR EACH ROW**  **begin**  **INSERT into Zakazy\_Deleted (Наименование\_удал, Цена\_удал, Заказчик\_удал)**  **values (Old.Наименование\_товара, Old.Цена\_продажи, Old.Заказчик);**  **end; //**  **DELIMITER ;**  Здесь **Old** означает, что берутся старые значения полей. При изменении полей для новых значений определено ключевое слово **New**.  Для проверки работы триггера надо выполнить, например, команды:  **DELETE from Заказы Where Наименование\_товара = 'Диван';**  **SELECT \* from Zakazy\_Deleted;** |

[В начало практикума](#_Содержание)

## Лабораторная работа № 8. MySQL. Администрирование базы данных

|  |  |
| --- | --- |
| **Задание** | **Краткие теоретические сведения** |
| 1. Создать резервную копию базы данных **BaseMysql\_x**, внести изменения в базу и восстановить ее в прежнем виде с помощью Workbench. | Чтобы создать *резервную копию* базы данных надо в приложении Workbench перейти на вкладку **Administration** и выполнить команду **Data Export**.  На вкладке **Object Selection** выбрать объекты для копирования, отметить пункт **Export to Self-Contained File** и определить путь для размещения резервной копии.  Нажать кнопку **Start Export**. На вкладке **Export Progress** отображается результат.  Для *восстановления* базы данных на вкладке **Administration** надо выполнить команду **Data Import**, отметить пункт **Import from Self-Contained File**, найти нужные файлы, в поле **Default Target Schema** выбрать схему, куда будут импортированы данные из копии, или создать новую схему, и восстановить данные, нажав кнопку **Start Import**. |
| 2. Создать нового пользователя базы данных **BaseMysql\_x** с использованием графического интерфейса, определить для пользователя неограниченные права.  Создать другого пользователя с помощью языка SQL. | Чтобы создать нового пользователя нужно в приложении Workbench на панели **Administration** выполнить команду **Users and Privileges**, в появившемся окне нажать кнопку **Add Account**. На вкладке **Login** указать имя пользователя, пароль и т. д. На вкладке **Administrative Roles** определить роли и разрешения текущего пользователя.  После того, как все настройки завершены, нажать кнопку **Apply**.  Для реализации вышеуказанных функций в приложении Workbench также можно использовать операторы языка SQL.  Например, для определения имени пользователя, которому разрешено подключаться к серверу из подсети '**localhost**', и пароля пользователя служит оператор, общий вид которого:  **CREATE USER 'имя пользователя'@'localhost' IDENTIFIED BY 'пароль';**  Чтобы назначить вновь созданному пользователю неограниченные права доступа к базе данных, надо выполнить команду:  **GRANT ALL PRIVILEGES ON** **\* . \*** **TO** **'имя пользователя'@'localhost';**  Затем запустить команду обновления: **FLUSH PRIVILEGES;**  Новый пользователь получает те же права доступа в базе данных, что и пользователь **root**.  Все пользователи хранятся в базе данных в таблице **user**. Проверить наличие нового пользователя можно с помощью команды:  **SELECT user, host from user;** // вывод таблицы **user**  Чтобы войти в базу под новым именем необходимо зайти в интерфейс программы MySQL Workbench и создать новое подключение нажав на значок плюса.  В открывшемся окне ввести имя нового пользователя в поля **Connection Name** и **Username**.  После создания нового подключения нажать на него правой кнопкой мыши и в контекстном меню выполнить команду **Start Command Line Client**. |
| 3. Создать несколько пользователей базы данных **BaseMysql\_x** и определить для них разрешения.  Опробовать действие указанных разрешений. | Для безопасности лучше предоставлять разные разрешения (привилегии) для различных пользователей с помощью команды:  **GRANT [тип прав] ON [имя базы данных].[имя таблицы] TO ‘имя пользователя’@'localhost’;**  Эти разрешения [**тип прав**] могут быть любой комбинацией SELECT, INSERT, UPDATE, DELETE, INDEX, CREATE, ALTER, DROP, GRANT OPTION или ALL.  Например, если надо предоставить привилегии SELECT, INSERT, UPDATE и DELETE в таблице **Заказы** пользователю с именем **user1**, то следует выполнить команду:  **GRANT SELECT, INSERT, UPDATE, DELETE ON Заказы** **TO 'user1'@'localhost';**  Удалить пользователя можно, используя команду:  **DROP USER 'user1'@'localhost';**  Для смены пароля пользователя служит оператор:  **ALTER USER 'user1'@'localhost' IDENTIFIED BY 'New\_Password';** |
| 4. Для нескольких пользователей базы данных **BaseMysql\_x** определить роли и разрешения.  Опробовать их действие. | **Роль** – это именованный набор привилегий.  Для создания роли необходимо использовать оператор **CREATE ROLE.**  Например, создание двух ролей: **CREATE ROLE 'u\_read', 'u\_write';**  Для назначения привилегий ролям необходимо выполнить инструкцию GRANT, используя тот же синтаксис, что и для назначения привилегий учетным записям:  **GRANT SELECT ON Заказы TO 'u\_read';**  **GRANT INSERT, UPDATE, DELETE ON Заказы TO 'u\_write';**  Для создания учетной записи **rw\_user** ипредоставления ей доступа для чтения и записислужат операторы:  **CREATE USER 'rw\_user'@'localhost' IDENTIFIED BY 'rw\_userpass';**  **GRANT 'u\_read', 'u\_write' TO 'rw\_user'@'localhost';**  Роли нужно активировать:  **SET DEFAULT ROLE ALL TO 'rw\_user1'@'localhost';**  **Отзыв** роли осуществляется с помощью **REVOKE.**  Для **удаления** роли необходимо использовать инструкцию DROP ROLE:  **DROP ROLE 'u\_read', 'u\_write';** |
| 5. Ознакомиться с процедурой для подсчета количества пользователей базы данных.  Создать процедуру или функцию для базы данных **BaseMysql\_x**. | Количество пользователей может быть возвращено как значение из процедуры:  **DELIMITER //**  **CREATE PROCEDURE countusers()**  **BEGIN**  **DECLARE usercount int;**  **SELECT count(\*) INTO usercount FROM mysql.user;**  **SELECT usercount;**  **END //**  **DELIMITER ;**  Вызов процедуры: **CALL countusers;** |
| 6. Предоставить привилегию пользователю для работы с процедурой или функцией. Опробовать действие разрешения. | Пусть есть функция **GetBestPrice** и надо предоставить EXECUTE-доступ к ней пользователю с именем **user4**. Следует использовать оператор GRANT:  **GRANT EXECUTE ON FUNCTION GetBestPrice TO 'user4'@'localhost';**  Если надо предоставить всем пользователям возможность выполнить эту функцию, то используется оператор:  **GRANT EXECUTE ON FUNCTION GetBestPrice TO '\*'@'localhost';**  Аналогично для процедуры.  Для отмены привилегии на функцию или процедуру используется оператор:  **REVOKE EXECUTE ON [ PROCEDURE | FUNCTION ] object FROM user;** |

[В начало практикума](#_Содержание)