Лабораторная работа № 2

Триггеры в распределённой базе данных

Цель работы

Получить навыки обеспечения целостности распределённой базы данных с помощью триггеров.

Обеспечение целостности данных средствами SQL

Достаточно простой способ повысить производительность СУБД – разнести её отдельные единицы по логическим структурам организации.

Поскольку в дальнейшем будет рассматриваться пример реализации такого механизма в информационной системе городского спорта, целесообразно коротко описать эту информационную систему.

Спортивная инфраструктура города представлена спортивными сооружениями различного типа: спортивные залы, манежи, стадионы, корты и т.д. Каждая из категорий спортивных сооружений обладает атрибутами, специфичными только для неё: стадион характеризуется вместимостью, корт – типом покрытия. Спортсмены под руководством тренеров занимаются отдельными видами спорта, при этом один и тот же спортсмен может заниматься несколькими видами спорта и в рамках одного и того же вида спорта может тренироваться у нескольких тренеров. Все спортсмены объединяются в спортивные клубы, при этом каждый из них может выступать только за один клуб. Организаторы соревнований проводят состязания по отдельным видам спорта на спортивных сооружениях города. По результатам участия в соревнованиях производится награждение спортсменов.

Таким образом, логически у спортивной инфраструктуры города можно выделить, как минимум, два отдела: отдел городского спорта и отдел спортивных организаций (см. рисунок 1). Отдел городского спорта в данном случае занимается организацией соревнований, учётом спортивных организаций и сооружений. Отдел спортивных организаций занимается учётом внутренней структуры каждой из организаций.

Краткое описание сущностей:

`Sport` – виды спорта, занятия и соревнования по которым проводятся в спортивных организациях и сооружениях города.
`Sport_Organization` – спортивная организация.
`Sport_ln_Organization` – ассоциативная сущность, связывает виды спорта со спортивными организациями.
`Sport_Building` – спортивные сооружения.
`Club` – спортивные клубы.
`Attendants` – спортсмены, члены спортивных организаций, дочерний тип сущности Person.
`Workers` – работники спортивных организаций.
`Person` – запись о человеке, базовый тип для сущностей Attendants и Workers.
`Experience` – спортивные разряды спортсменов.
`Competition` – проводимые соревнования.
`Participants` – участники соревнований.

Таким образом, для описанной выше предметной области к отделу городского спорта можно отнести следующие сущности: **`Sport`**, **`Sport_Organization`**, **`Sport_ln_Organization`**, **`Sport_Building`**. К отделу спортивных организаций будут отнесены все оставшиеся сущности.

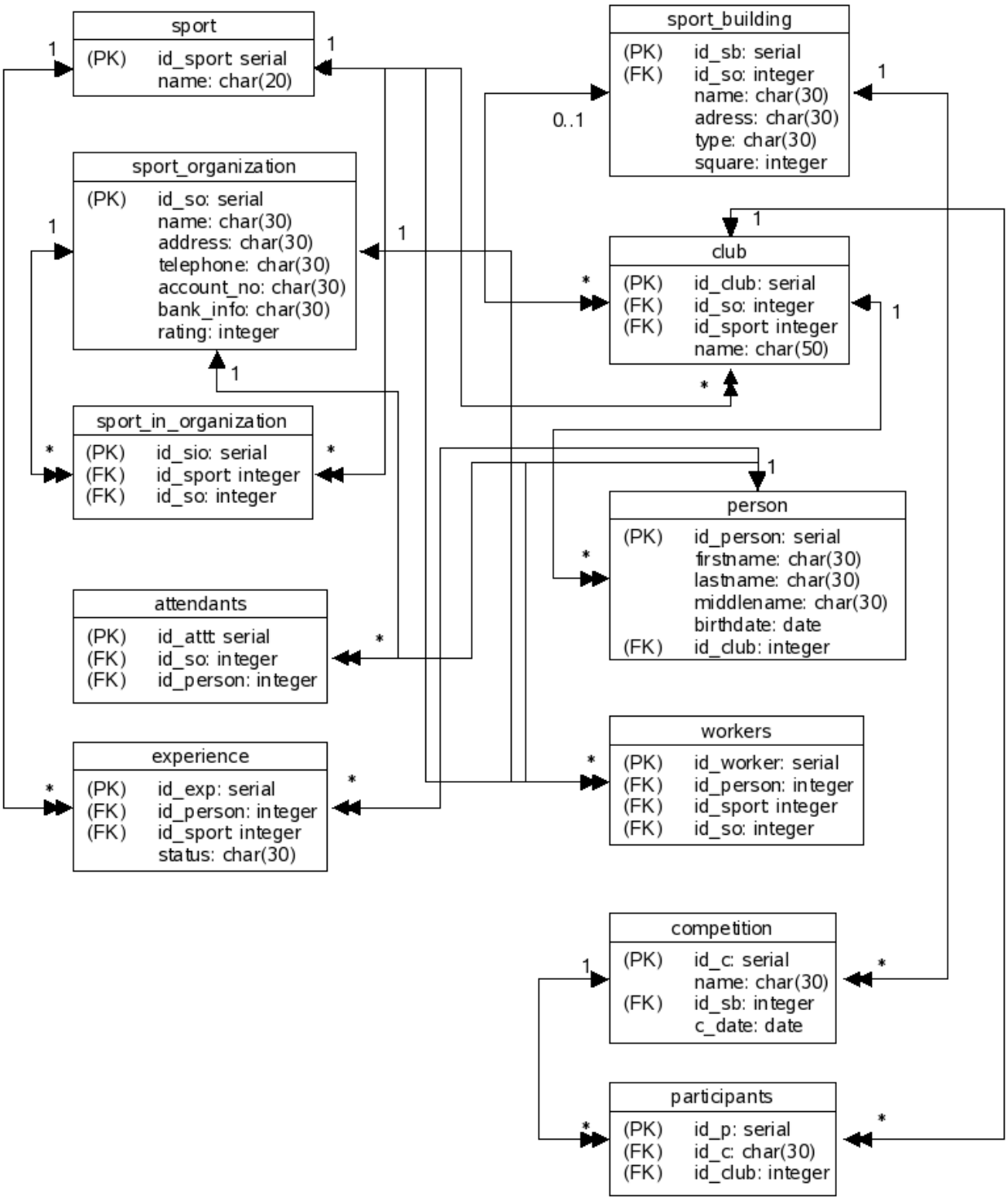

Рисунок 1 – ER-диаграмма структуры базы данных системы городского спорта

Для реализации механизма поддержания целостности данных можно воспользоваться стандартными средствами языка структурных запросов SQL. А именно, к описанию сущностей (таблиц) на языке SQL следует добавить серию триггеров, которые при добавлении, изменении и удалении записей будут поддерживать целостность рассматриваемой базы данных системы городского спорта.

Обеспечение целостности данных средствами СУБД PostgreSQL

Рассматриваемое распределение данных на примере системы городского спорта является разбиением отдельных таблиц исходной нераспределённой СУБД на два подразделения организации: отдел городского спорта и отдел спортивных организаций соответственно.

Физическое представление этого разбиения будет выглядеть как два отдельных сервера с установленными СУБД PostgreSQL, хранящими лишь те таблицы нераспределённой базы данных, которые соответствуют каждому из отделов организации.

Поскольку PostgreSQL не позволяет создавать внешние ссылки на таблицы из разных СУБД, это создаёт определённую угрозу целостности данных. Например, отдел городского спорта закрыл на ремонт одно из спортивных сооружений, а отдел спортивных организаций продолжает хранить запись о будущих соревнованиях, которые должны проводиться в уже закрытом на ремонт спортивном сооружении. В таком случае соревнования необходимо или отменить, или перенести в другое спортивное сооружение. Поскольку внешние ключи не могут ссылаться на таблицы из другой базы данных, то СУБД больше не заботится об обеспечении целостности, и эту работу приходится делать программистам баз данных.

Для каждой из таблиц двух подразделений следует создать триггеры на операции вставки, изменения и удаления данных. Именно таким способом можно будет поддерживать целостность рассматриваемого разбиения.

Для того чтобы подключиться к другой базе данных, можно использовать библиотеку **DBLink** для СУБД PostgreSQL. Следует отметить, что данная библиотека не включена по умолчанию в пакет PostgreSQL и должна устанавливаться отдельно. С порядком установки и руководством по использованию этой библиотеки можно ознакомиться на сайте [1].

Описание использования триггеров в СУБД PostgreSQL

Как уже говорилось ранее, для поддержания целостности данных в распределённой модели можно воспользоваться механизмом триггеров. Сначала необходимо создать саму таблицу, для которой впоследствии будут дописываться триггеры:

```
drop table if exists sport cascade;
create table sport
(
 id_sport serial not null,
 name     character(20) not null,
 primary key (id_sport)
);
```

SQL-оператор **drop table** удаляет ранее созданную таблицу. Модификатор **if exists** означает, что если таблицы с указанным именем не существует, то SQL-сервер в данном случае не сообщит об ошибке и дальше продолжит выполнение запросов. Модификатор **cascade** означает, что вместе с удаляемой таблицей будут также удалены все прикреплённые к ней данные, например, индексы таблицы или привязанные к ней триггеры.

Оператор языка SQL **create table** создаёт новую таблицу с указанным именем. Далее в круглых скобках следует список полей таблицы. После списка полей, как правило, следуют ограничения полей, которые создаются на основе предметной области информационной системы. В нашем случае таким ограничением служит указание на то, что поле **id_sport** является первичным ключом таблицы.

Теперь можно рассмотреть создание триггеров и необходимых хранимых процедур средствами PostgreSQL. Сначала нужно определить понятия. ***Триггер*** – это хранимая процедура, вызов которой обусловливается совершением некоторых действий над таблицей, например: вставка, изменение или удаление данных. ***Хранимая процедура***, в свою очередь, – это скомпилированный упорядоченный набор SQL-операторов, который был загружен на сервер СУБД и выполняется по требованию пользователя.

Далее приводится пример создания хранимой процедуры. Для создания следующей хранимой процедуры используется встроенный в СУБД PostgreSQL процедурный язык PL/PgSQL (PL – *Procedural Language*). Следует отметить, что этот процедурный язык не устанавливается по умолчанию на вновь созданную базу данных и поэтому для его использования администратор СУБД обязан подключить его.

```
drop function if exists PROC_Is_Id_Used(text, text, text, integer);
create function PROC_Is_Id_Used(conname text, table_name text,
                                id_name text, id_value integer)
returns void as $$
```

```
declare
  query text;
begin
  query := 'select count(*) as cnt from split2.' || table_name
           || ' where ' || id_name || '=' || id_value;
  perform public.dblink_open(conname, 'data_cursor', query);
  if (select * from public.dblink_fetch(conname, 'data_cursor', 1)
      as (cnt integer)
     ) > 0
  then
    perform public.dblink_disconnect(conname);
    raise exception 'Cannot delete from, table is referenced!';
  end if;
  perform public.dblink_close(conname, 'data_cursor');
end;
$$ language plpgsql;
```

Оператор языка SQL **drop function** удаляет ранее созданную функцию с указанием её сигнатуры. Под сигнатурой функции подразумеваются её название и список типов аргументов. Таким образом, может существовать несколько хранимых процедур с одним и тем же названием, но с различными списками аргументов. В случае если при создании хранимых процедур и названия, и списки аргументов двух любых процедур совпадают, сервер СУБД сообщит об ошибке.

Оператор **create function** объявляет о создании новой хранимой процедуры с указанной сигнатурой. После этого объявления следует тело функции, которое обособляется попарным оператором **$$**. После закрывающего оператора **$$** следует указание используемого языка в хранимой процедуре. Как и в любом другом процедурном языке программирования, внутри хранимых процедур SQL-программист может объявить некоторые локальные переменные. Локальные переменные объявляются в отдельном блоке **declare**. Сам текст хранимой процедуры заключен в секции между операторами **begin** и **end**.

Вызов хранимой процедуры в СУБД PostgreSQL осуществляется операторами **perform** и **select**. Различие между этими операторами заключается в том, что оператор **perform** используется с теми хранимыми процедурами, которые не возвращают какого-либо результата, т.е. с типом возвращаемого значения **void**; а оператор **select** используется в противном случае.

Описанная выше хранимая процедура проверяет, существуют ли записи в таблице **table_name**, у которых внешний ключ с названием **id_name** принимает значение **id_value**. Данная процедура используется в триггере на удаление данных (операция **delete from**) для тех таблиц, на которые существуют внешние ссылки, целостность которых необходимо обеспечить. В случае если таковую целостность обеспечить не удаётся, то операция удаления данных отменяется и хранимая процедура выбрасывает исключение, определённое программистом. Для создания пользовательского исключения используется оператор языка PL/PgSQL – **raise exception**.

Пример триггера на удаление данных для таблицы **Sport**:

```
drop function if exists PROC_Table_Sport_On_Delete();
create function PROC_Table_Sport_On_Delete()
returns trigger as $$
begin
  perform public.dblink_connect('child_connection',
                                'dbname=split2 user=zif password=123qwe');
  perform PROC_Is_Id_Used('child_connection', 'club',
                          'id_sport', old.id_sport);
  perform PROC_Is_Id_Used('child_connection', 'experience',
                          'id_sport', old.id_sport);
  perform PROC_Is_Id_Used('child_connection', 'workers',
                          'id_sport', old.id_sport);
  perform public.dblink_disconnect('child_connection');
  return new;
end;
$$ language plpgsql;

create trigger TRIGGER_Table_Sport_On_Delete
  before delete on sport
  for each row execute procedure PROC_Table_Sport_On_Delete();
```

Как видно из примера, триггер – это хранимая процедура, привязанная к некоторому событию, связанному с изменением данных в таблице. Таким образом, процедура **PROC_Table_Sport_On_Delete()** возвращает результат типа **trigger**, это и указывает на то, что данная процедура будет использоваться в качестве триггера.

Триггеры имеют несколько предопределённых переменных для операций изменения и удаления данных (**update** и **delete from** соответственно) – это переменные с именами **old** и **new**, для операции вставки данных **insert** – доступна только переменная **new**. Переменная **new** указывает на запись обновлённых данных, которые ещё не были внесены в таблицу. Переменная **old** указывает на запись данных предыдущей версии, т.е. тех данных, которые либо обновляются, либо удаляются. Поскольку этот триггер используется при возникновении события удаления данных, доступны обе переменные. Триггер должен возвращать переменную **new** в случае успешного обновления данных.

Конструкция языка SQL **create trigger** создаёт триггер с соответствующим именем и привязывает его к указанной таблице для вызова триггер-процедуры при возникновении обозначенных событий. Ключевое слово **before** в данной конструкции означает, что триггер-функция будет вызвана непосредственно перед обновлением данных. Если указан модификатор **after**, то триггер-функция будет вызвана сразу после обновления данных.

Таблица 1 – Краткий обзор функций библиотеки DBLink

Функция	Описание функции	Описание аргументов
dblink_connect (text connname, text connstr)	Осуществляет подключение к удалённой базе данных	**conname** – идентификатор подключения к серверу СУБД; **connstr** – строка параметров подключения к серверу СУБД
dblink_disconnect (text connname)	Закрывает установленное подключение к удалённому серверу СУБД	**conname** – идентификатор подключения к серверу СУБД
dblink_open (text connname, text cursorname, text sql)	Открывает курсор для доступа к данным результата **select**-запроса к удалённой базе данных	**conname** – идентификатор подключения к серверу СУБД; **cursorname** – идентификатор открытого курсора; **sql** – SQL-запрос к удаленной базе данных
dblink_fetch (text connname, text cursorname, int howmany) returns setof record	Получает **howmany** записей данных по открытому курсору **cursorname**	**conname** – идентификатор подключения к серверу СУБД; **cursorname** – идентификатор открытого курсора; **howmany** – количество возвращаемых записей
dblink_close (text connname, text cursorname)	Закрывает открытый курсор **cursorname**	**conname** – идентификатор подключения к серверу СУБД; **cursorname** – идентификатор открытого курсора

Далее приводится ещё несколько примеров хранимых процедур:

```
/*  хранимая процедура для проверки существования внешнего ключа таблицы sport с заданным
    значением
*/
drop function if exists PROC_Does_IdSport_Exist(text, integer);
create function PROC_Does_IdSport_Exist(conname text,
                                        id_sport integer)
returns void as $$
declare
  query text;
begin
  query := 'select count(*) as cnt from split1.sport
            where id_sport=' || id_sport;
  perform public.dblink_open(conname, 'data_cursor', query);
```

```
  if (select * from public.dblink_fetch(conname, 'data_cursor', 1)
      as (cnt integer)
     ) = 0
  then
    perform public.dblink_disconnect(conname);
    raise exception 'No such sport';
  end if;
  perform public.dblink_close(conname, 'data_cursor');
end;
$$ language plpgsql;

/*  хранимая процедура для проверки существования внешнего ключа таблицы sportorganization с
    заданным значением
*/
drop function if exists PROC_Does_IdSO_Exist(text, integer);
create function PROC_Does_IdSO_Exist(conname text, id_so integer)
returns void as $$
declare
  query text;
begin
  query := 'select count(*) as cnt from split1.sport_organization
            where id_so=' || id_so;
  perform public.dblink_open(conname, 'data_cursor', query);
  if (select * from public.dblink_fetch(conname, 'data_cursor', 1)
      as (cnt integer)
     ) = 0
  then
    perform public.dblink_disconnect(conname);
    raise exception 'No such sport organization';
  end if;
  perform public.dblink_close(conname, 'data_cursor');
end;
$$ language plpgsql;

/*  хранимая процедура для проверки существования внешнего ключа таблицы sportbuilding с за-
    данным значением
*/
drop function if exists PROC_Does_IdSB_Exist(text, integer);
create function PROC_Does_IdSB_Exist(conname text, id_sb integer)
returns void as $$
declare
  query text;
begin
  query := 'select count(*) as cnt from split1.sport_building
            where id_sb=' || id_sb;
  perform public.dblink_open(conname, 'data_cursor', query);
  if (select * from public.dblink_fetch(conname, 'data_cursor', 1)
      as (cnt integer)
     ) = 0
  then
    perform public.dblink_disconnect(conname);
    raise exception 'No such sport building';
  end if;
  perform public.dblink_close(conname, 'data_cursor');
end;
$$ language plpgsql;
```

Задание

Для созданной в лабораторной работе № 1 базы данных с оптимальным размещением таблиц по двум узлам создать набор триггеров для поддержания целостности базы данных.

I. Изучить способы создания триггеров для распределённой базы данных.
II. Исходя из предметной области базы данных, разработать набор триггеров, обеспечивающий целостность базы данных.

Пусть есть узел $\boldsymbol{A}$ с таблицами a, b, c, d и узел $\boldsymbol{B}$ с таблицами e, f, g, h. Для узла $\boldsymbol{A}$ написать триггеры, обеспечивающие обработку ситуации, когда на поле одной из таблиц узла $\boldsymbol{A}$ (a–d) имеется внешняя ссылка из какой-либо таблицы узла $\boldsymbol{B}$ (e–h):

1. Пусть из таблицы a предпринимается попытка удалить запись, на которую есть ссылка из таблицы e. Написать триггер, запрещающий удаление соответствующей строки из таблицы a.
2. Пусть в таблице b предпринимается попытка модифицировать запись, на которую есть ссылка из таблицы f. Написать триггер, запрещающий модификацию соответствующей строки из таблицы b, если модификация нарушает целостность базы данных, и разрешающий её, если целостность не нарушается.
3. Пусть из таблицы c предпринимается попытка удалить запись, на которую есть ссылки из таблиц g и h. Написать триггер, производящий каскадное удаление, т.е. удаление соответствующей строки из таблицы с узла $\boldsymbol{A}$ и всех строк из таблиц g и h узла $\boldsymbol{B}$, которые на неё ссылаются.
4. Пусть предпринимается попытка добавить запись в таблицу e, внешний ключ которой должен ссылаться на таблицу a. Написать триггер, запрещающий добавление такой записи, если она нарушает целостность базы данных.

III. Продемонстрировать работу триггеров.

Требования к оформлению отчёта

Отчёт по лабораторной работе должен включать в себя:

- титульный лист;
- краткое описание предметной области, для которой разработана база данных;
- ER-диаграммы баз данных для каждого узла;
- тексты всех триггеров с комментариями;
- подробное описание тестирования работы триггеров (нужно показать, что триггеры работают правильно);
- вывод по результатам проделанной работы.

Отчёт не должен содержать орфографических, пунктуационных и смысловых ошибок. Все его разделы должны быть выдержаны в едином стиле оформления.

Критерии оценивания качества работы

1. При возникновении ошибок триггеры должны подробно сообщить о причине произошедшего:
   1 – во всех случаях триггер сообщает подробную информацию о причине исключения;
   0 – иначе.
2. Наличие ER-диаграмм для каждого узла:
   1 – есть;
   0 – нет.
3. Содержание отчёта:
   1 – отчёт удовлетворяет всем требованиям;
   0 – отчёт удовлетворяет не всем требованиям.
4. Наличие комментариев в исходных кодах:
   1 – комментариев достаточно для документирования исходных кодов;
   0 – комментариев недостаточно.
5. Глубина понимания материала лабораторной работы каждым членом бригады:
   1 – быстрые и правильные ответы на все вопросы;
   0 – не на все вопросы ответы правильные и быстрые;
   л.р. не принимается – на половину вопросов ответы неправильные.

Список литературы

1. СУБД PostgreSQL [Электронный ресурс]. – Режим доступа: http://www.postgresql.org/.