Лабораторная работа №12

**Ограничения целостности**

*Цель лабораторной работы*: изучение и практическое применение ограничений целосности.

*Оборудование*: компьютерный класс, Microsoft Word, PostgreSQL.

*Теоретическая часть*:

Типы данных сами по себе ограничивают множество данных, которые можно сохранить в таблице. Однако для многих приложений такие ограничения слишком грубые. Например, столбец, содержащий цену продукта, должен, вероятно, принимать только положительные значения. Но такого стандартного типа данных нет. Возможно, вы также захотите ограничить данные столбца по отношению к другим столбцам или строкам. Например, в таблице с информацией о товаре должна быть только одна строка с определённым кодом товара.

Для решения подобных задач SQL позволяет вам определять ограничения для столбцов и таблиц. Ограничения дают вам возможность управлять данными в таблицах так, как вы захотите. Если пользователь попытается сохранить в столбце значение, нарушающее ограничения, возникнет ошибка. Ограничения будут действовать, даже если это значение по умолчанию.

***Ограничения-проверки***

Ограничение-проверка – наиболее общий тип ограничений. В его определении вы можете указать, что значение данного столбца должно удовлетворять логическому выражению (проверке истинности). Например, цену товара можно ограничить положительными значениями так:

CREATE TABLE products (

product\_no integer,

name text,

price numeric **CHECK (price > 0)**

);

Как вы видите, ограничение определяется после типа данных, как и значение по умолчанию. Значения по умолчанию и ограничения могут указываться в любом порядке. Ограничение-проверка состоит из ключевого слова CHECK, за которым идёт выражение в скобках. Это выражение должно включать столбец, для которого задаётся ограничение, иначе оно не имеет большого смысла.

Вы можете также присвоить ограничению отдельное имя. Это улучшит сообщения об ошибках и позволит вам ссылаться на это ограничение, когда вам понадобится изменить его. Сделать это можно так:

CREATE TABLE products (

product\_no integer,

name text,

price numeric **CONSTRAINT positive\_price** CHECK (price > 0)

);

То есть, чтобы создать именованное ограничение, напишите ключевое слово CONSTRAINT, а за ним идентификатор и собственно определение ограничения (если вы не определите имя ограничения таким образом, система выберет для него имя за вас).

Ограничение-проверка может также ссылаться на несколько столбцов. Например, если вы храните обычную цену и цену со скидкой, так вы можете гарантировать, что цена со скидкой будет всегда меньше обычной:

CREATE TABLE products (

product\_no integer,

name text,

price numeric CHECK (price > 0),

discounted\_price numeric CHECK (discounted\_price > 0),

**CHECK (price > discounted\_price)**

);

Первые два ограничения определяются похожим образом, но для третьего используется новый синтаксис. Оно не связано с определённым столбцом, а представлено отдельным элементом в списке. Определения столбцов и такие определения ограничений можно переставлять в произвольном порядке.

Про первые два ограничения можно сказать, что это ограничения столбцов, тогда как третье является ограничением таблицы, так как оно написано отдельно от определений столбцов. Ограничения столбцов также можно записать в виде ограничений таблицы, тогда как обратное не всегда возможно, так как подразумевается, что ограничение столбца ссылается только на связанный столбец. Ранее приведённый пример можно переписать и так:

CREATE TABLE products (

product\_no integer,

name text,

price numeric,

CHECK (price > 0),

discounted\_price numeric,

CHECK (discounted\_price > 0),

CHECK (price > discounted\_price)

);

Или даже так:

CREATE TABLE products (

product\_no integer,

name text,

price numeric CHECK (price > 0),

discounted\_price numeric,

CHECK (discounted\_price > 0 AND price > discounted\_price)

);

Это дело вкуса.

Ограничениям таблицы можно присваивать имена так же, как и ограничениям столбцов:

CREATE TABLE products (

product\_no integer,

name text,

price numeric,

CHECK (price > 0),

discounted\_price numeric,

CHECK (discounted\_price > 0),

**CONSTRAINT valid\_discount** CHECK (price > discounted\_price)

);

Следует заметить, что ограничение-проверка удовлетворяется, если выражение принимает значение true или NULL. Так как результатом многих выражений с операндами NULL будет значение NULL, такие ограничения не будут препятствовать записи NULL в связанные столбцы. Чтобы гарантировать, что столбец не содержит значения NULL, можно использовать ограничение NOT NULL, описанное в следующем разделе.

***Ограничения NOT NULL***

Ограничение NOT NULL просто указывает, что столбцу нельзя присваивать значение NULL. Пример синтаксиса:

CREATE TABLE products (

product\_no integer **NOT NULL**,

name text **NOT NULL**,

price numeric

);

Ограничение NOT NULL всегда записывается как ограничение столбца и функционально эквивалентно ограничению CHECK (*имя\_столбца* IS NOT NULL). Хотя у такой записи есть недостаток – назначить имя таким ограничениям нельзя.

Естественно, для столбца можно определить больше одного ограничения. Для этого их нужно просто указать одно за другим:

CREATE TABLE products (

product\_no integer NOT NULL,

name text NOT NULL,

price numeric NOT NULL CHECK (price > 0)

);

Порядок здесь не имеет значения, он не обязательно соответствует порядку проверки ограничений.

Для ограничения NOT NULL есть и обратное: ограничение NULL. Оно не означает, что столбец должен иметь только значение NULL, что конечно было бы бессмысленно. Суть же его в простом указании, что столбец может иметь значение NULL (это поведение по умолчанию). Ограничение NULL отсутствует в стандарте SQL и использовать его в переносимых приложениях не следует. Однако некоторые пользователи любят его использовать, так как оно позволяет легко переключать ограничения в скрипте. Например, вы можете начать с:

CREATE TABLE products (

product\_no integer NULL,

name text NULL,

price numeric NULL

);

и затем вставить ключевое слово NOT, где потребуется.

***Ограничения уникальности***

Ограничения уникальности гарантируют, что данные в определённом столбце или группе столбцов уникальны среди всех строк таблицы. Ограничение записывается так:

CREATE TABLE products (

product\_no integer **UNIQUE**,

name text,

price numeric

);

в виде ограничения столбца и так:

CREATE TABLE products (

product\_no integer,

name text,

price numeric,

**UNIQUE (product\_no)**

);

в виде ограничения таблицы.

Чтобы определить ограничение уникальности для группы столбцов, запишите его в виде ограничения таблицы, перечислив имена столбцов через запятую:

CREATE TABLE example (

a integer,

b integer,

c integer,

**UNIQUE (a, c)**

);

Такое ограничение указывает, что сочетание значений перечисленных столбцов должно быть уникально во всей таблице, тогда как значения каждого столбца по отдельности не должны быть (и обычно не будут) уникальными.

Можно назначить уникальному ограничению имя обычным образом:

CREATE TABLE products (

product\_no integer **CONSTRAINT must\_be\_different** UNIQUE,

name text,

price numeric

);

При добавлении ограничения уникальности будет автоматически создан уникальный индекс-B-дерево для столбца или группы столбцов, перечисленных в ограничении. Условие уникальности, распространяющееся только на некоторые строки, нельзя записать в виде ограничения уникальности, однако такое условие можно установить, создав уникальный [частичный индекс](https://postgrespro.ru/docs/postgrespro/9.5/indexes-partial).

Вообще говоря, ограничение уникальности нарушается, если в таблице оказывается несколько строк, у которых совпадают значения всех столбцов, включённых в ограничение. Однако два значения NULL при сравнении никогда не считаются равными. Это означает, что даже при наличии ограничения уникальности в таблице можно сохранить строки с дублирующимися значениями, если они содержат NULL в одном или нескольких столбцах ограничения. Это поведение соответствует стандарту SQL, но мы слышали о СУБД, которые ведут себя по-другому. Имейте в виду эту особенность, разрабатывая переносимые приложения.

***Первичные ключи***

Ограничение первичного ключа означает, что образующий его столбец или группа столбцов может быть уникальным идентификатором строк в таблице. Для этого требуется, чтобы значения были одновременно уникальными и отличными от NULL. Таким образом, таблицы со следующими двумя определениями будут принимать одинаковые данные:

CREATE TABLE products (

product\_no integer UNIQUE NOT NULL,

name text,

price numeric

);

CREATE TABLE products (

product\_no integer **PRIMARY KEY**,

name text,

price numeric

);

Первичные ключи могут включать несколько столбцов; синтаксис похож на запись ограничений уникальности:

CREATE TABLE example (

a integer,

b integer,

c integer,

**PRIMARY KEY (a, c)**

);

При добавлении первичного ключа автоматически создаётся уникальный индекс-B-дерево для столбца или группы столбцов, перечисленных в первичном ключе, и данные столбцы помечаются как NOT NULL.

Таблица может иметь максимум один первичный ключ (ограничений уникальности и ограничений NOT NULL, которые функционально почти равнозначны первичным ключам, может быть сколько угодно, но назначить ограничением первичного ключа можно только одно). Теория реляционных баз данных говорит, что первичный ключ должен быть в каждой таблице.

Первичные ключи полезны и для документирования, и для клиентских приложений. Например, графическому приложению с возможностями редактирования содержимого таблицы, вероятно, потребуется знать первичный ключ таблицы, чтобы однозначно идентифицировать её строки. Первичные ключи находят и другое применение в СУБД; в частности, первичный ключ в таблице определяет целевые столбцы по умолчанию для сторонних ключей, ссылающихся на эту таблицу.

***Внешние ключи***

Ограничение внешнего ключа указывает, что значения столбца (или группы столбцов) должны соответствовать значениям в некоторой строке другой таблицы. Это называется *ссылочной целостностью* двух связанных таблиц.

Пусть у вас уже есть таблица продуктов, которую мы неоднократно использовали ранее:

CREATE TABLE products (

product\_no integer PRIMARY KEY,

name text,

price numeric

);

Предположим, что есть таблица с заказами этих продуктов. Мы хотим, чтобы в таблице заказов содержались только заказы действительно существующих продуктов. Поэтому определим в ней ограничение внешнего ключа, ссылающееся на таблицу продуктов:

CREATE TABLE orders (

order\_id integer PRIMARY KEY,

product\_no integer **REFERENCES products (product\_no)**,

quantity integer

);

С таким ограничением создать заказ со значением product\_no, отсутствующим в таблице products (и не равным NULL), будет невозможно.

В такой схеме таблицу orders называют *подчинённой* таблицей, а products – *главной*. Соответственно, столбцы называют так же подчинённым и главным (или ссылающимся и целевым).

Предыдущую команду можно сократить так:

CREATE TABLE orders (

order\_id integer PRIMARY KEY,

product\_no integer **REFERENCES products**,

quantity integer

);

то есть, если опустить список столбцов, внешний ключ будет неявно связан с первичным ключом главной таблицы.

Внешний ключ также может ссылаться на группу столбцов. В этом случае его нужно записать в виде обычного ограничения таблицы. Например:

CREATE TABLE t1 (

a integer PRIMARY KEY,

b integer,

c integer,

**FOREIGN KEY (b, c) REFERENCES other\_table (c1, c2)**

);

Естественно, число и типы столбцов в ограничении должны соответствовать числу и типам целевых столбцов.

Ограничению внешнего ключа можно назначить имя стандартным способом.

Таблица может содержать несколько ограничений внешнего ключа. Это полезно для связи таблиц в отношении многие-ко-многим. Скажем, у вас есть таблицы продуктов и заказов, но вы хотите, чтобы один заказ мог содержать несколько продуктов (что невозможно в предыдущей схеме). Для этого вы можете использовать такую схему:

CREATE TABLE products (

product\_no integer PRIMARY KEY,

name text,

price numeric

);

CREATE TABLE orders (

order\_id integer PRIMARY KEY,

shipping\_address text,

...

);

CREATE TABLE order\_items (

product\_no integer REFERENCES products,

order\_id integer REFERENCES orders,

quantity integer,

PRIMARY KEY (product\_no, order\_id)

);

В последней таблице первичный ключ покрывает внешние ключи.

Внешние ключи запрещают создание заказов, не относящихся ни к одному продукту. Но что делать, если после создания заказов с определённым продуктом мы захотим удалить его? При попытке удаления продукта, на который ссылаются заказы (через таблицу order\_items), мы запрещаем эту операцию. Если же кто-то попытается удалить заказ, то удалится и его содержимое:

CREATE TABLE products (

product\_no integer PRIMARY KEY,

name text,

price numeric

);

CREATE TABLE orders (

order\_id integer PRIMARY KEY,

shipping\_address text,

...

);

CREATE TABLE order\_items (

product\_no integer REFERENCES products **ON DELETE RESTRICT**,

order\_id integer REFERENCES orders **ON DELETE CASCADE**,

quantity integer,

PRIMARY KEY (product\_no, order\_id)

);

Ограничивающие и каскадные удаления – два наиболее распространённых варианта. RESTRICT предотвращает удаление связанной строки. NO ACTION означает, что если зависимые строки продолжают существовать при проверке ограничения, возникает ошибка (это поведение по умолчанию). (Главным отличием этих двух вариантов является то, что NO ACTION позволяет отложить проверку в процессе транзакции, а RESTRICT — нет.) CASCADE указывает, что при удалении связанных строк зависимые от них будут так же автоматически удалены. Есть ещё два варианта: SET NULL и SET DEFAULT. При удалении связанных строк они назначают зависимым столбцам в подчинённой таблице значения NULL или значения по умолчанию, соответственно. Заметьте, что это не будет основанием для нарушения ограничений. Например, если в качестве действия задано SET DEFAULT, но значение по умолчанию не удовлетворяет ограничению внешнего ключа, операция закончится ошибкой.

Аналогично указанию ON DELETE существует ON UPDATE, которое срабатывает при изменении заданного столбца. При этом возможные действия те же, а CASCADE в данном случае означает, что изменённые значения связанных столбцов будут скопированы в зависимые строки.

Обычно зависимая строка не должна удовлетворять ограничению внешнего ключа, если один из связанных столбцов содержит NULL. Если в объявление внешнего ключа добавлено MATCH FULL, строка будет удовлетворять ограничению, только если все связанные столбцы равны NULL (то есть при разных значениях (NULL и не NULL) гарантируется невыполнение ограничения MATCH FULL). Если вы хотите, чтобы зависимые строки не могли избежать и этого ограничения, объявите связанные столбцы как NOT NULL.

Внешний ключ должен ссылаться на столбцы, образующие первичный ключ или ограничение уникальности. Таким образом, для связанных столбцов всегда будет существовать индекс (определённый соответствующим первичным ключом или ограничением), а значит проверки соответствия связанной строки будут выполняться эффективно. Так как команды DELETE для строк главной таблицы или UPDATE для зависимых столбцов потребуют просканировать подчинённую таблицу и найти строки, ссылающиеся на старые значения, полезно будет иметь индекс и для подчинённых столбцов. Но это нужно не всегда, и создать соответствующий индекс можно по-разному, поэтому объявление внешнего ключа не создаёт автоматически индекс по связанным столбцам.

***Ограничения-исключения***

Ограничения-исключения гарантируют, что при сравнении любых двух строк по указанным столбцам или выражениям с помощью заданных операторов, минимум одно из этих сравнений возвратит false или NULL. Записывается это так:

CREATE TABLE circles (

c circle,

EXCLUDE USING gist (c WITH &&)

);

При добавлении ограничения-исключения будет автоматически создан индекс того типа, который указан в объявлении ограничения.

Более подробная информация:

https://postgrespro.ru/docs/postgresql/17/index или

https://postgrespro.ru/docs/postgrespro/17/index

***Практическая часть*:**

***Указания к выполнению заданий:***

***1. Работа производится в созданных в предыдущих работах таблицах.***

***2. Для каждого пункта необходимо привести:***

* ***запрос, добавляющий ограничение;***
* ***формулировку смысла ограничения (в понятиях предметной области);***
* ***запрос, модифицирующий данные некорректно (не соответствуя ограничениям), и ответ сервера БД на этот запрос;***
* ***запрос, модифицирующий данные корректно, и ответ сервера на этот запрос.***

1. Напишите SQL-команду (**ALTER TABLE**), добавляющую в любую из ваших таблиц следующие ограничения целостности: **NOT NULL, UNIQUE, PRIMARY KEY**.
2. Напишите SQL-команду, добавляющую в таблицу (дочернюю), связанную с первой (базовой), ограничение **FOREIGN KEY**. Вы должны сделать это в трех вариантах, по-разному описывая действие при операции с первичным ключом базовой таблицы.
3. Добавьте в таблицу ограничение **CHECK**.
4. Приведите пример динамического контроля целостности.