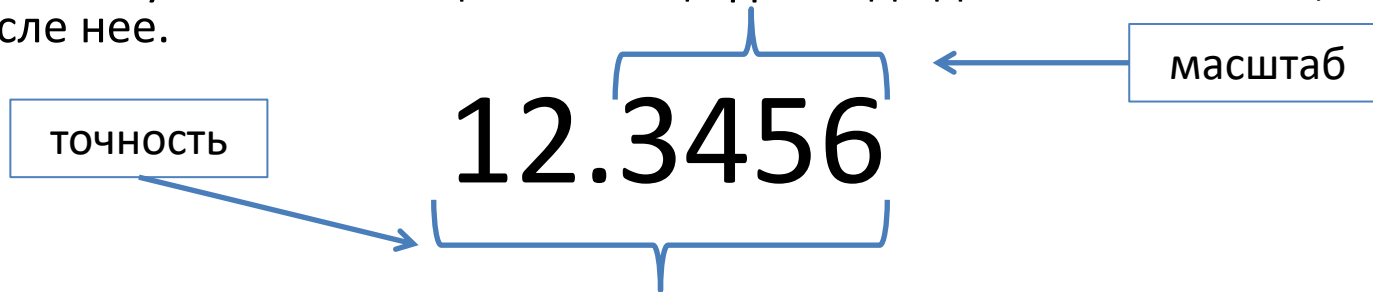


- целочисленные типы
- числа фиксированной точности
- типы данных с плавающей точкой
- последовательные типы (serial)

- В составе целочисленных типов находятся следующие представители: **smallint**, **integer**, **bigint**. Если атрибут таблицы имеет один из этих типов, то он позволяет хранить *только целочисленные* данные.
- При этом перечисленные типы различаются по количеству байтов, выделяемых для хранения данных.
- В PostgreSQL существуют псевдонимы для этих стандартизированных имен типов, а именно: **int2**, **int4** и **int8**.
- Число байтов отражается в имени типа.
- При выборе конкретного целочисленного типа принимают во внимание диапазон допустимых значений и затраты памяти.
- Зачастую тип **integer** считается оптимальным выбором с точки зрения достижения компромисса между этими показателями.

- Представлены двумя типами — **numeric** и **decimal**. Однако они являются идентичными по своим возможностям. Поэтому мы будем проводить изложение на примере типа `numeric`.
- Для задания значения этого типа используются два базовых понятия: **масштаб (scale)** и **точность (precision)**.
- Масштаб показывает число значащих цифр, стоящих справа от десятичной точки (запятой).
- Точность указывает общее число цифр как до десятичной точки, так и после нее.



- Например, у числа 12.3456 точность составляет 6 цифр, а масштаб — 4 цифры.
- Параметры этого типа данных указываются в скобках: `numeric(точность, масштаб)`. Например, `numeric(6, 2)`.

- Представителями типов данных с плавающей точкой являются **real** и **double precision**. Они представляют собой реализацию стандарта IEEE «Standard 754 for Binary Floating-Point Arithmetic».
- Тип данных **real** может представить числа в диапазоне, как минимум, от $1E-37$ до $1E+37$ с точностью не меньше 6 десятичных цифр.
- Тип **double precision** имеет диапазон значений примерно от $1E-307$ до $1E+308$ с точностью не меньше 15 десятичных цифр.
- При попытке записать в такой столбец слишком большое или слишком маленькое значение будет генерироваться ошибка.
- Если точность вводимого числа *выше допустимой*, то будет иметь место *округление* значения.
- А вот при вводе *очень маленьких чисел*, которые невозможно представить значениями, отличными от нуля, будет генерироваться ошибка потери значимости, или исчезновения значащих разрядов (**an underflow error**).

- Этот тип фактически реализован не как настоящий тип, а просто как удобная замена целой группы SQL-команд. Тип **serial** удобен в тех случаях, когда требуется в какой-либо столбец вставлять уникальные целые значения, например, значения суррогатного первичного ключа.
- Синтаксис для создания столбца типа serial таков:
CREATE TABLE tablename (colname SERIAL);

Эта команда эквивалентна следующей группе команд:

```
CREATE SEQUENCE tablename_colname_seq;  
CREATE TABLE tablename  
( colname integer NOT NULL  
    DEFAULT nextval( 'tablename_colname_seq' )  
);  
ALTER SEQUENCE tablename_colname_seq  
OWNED BY tablename.colname;
```

- Для пояснения вышеприведенных команд нам придется немного забежать вперед.
- Одним из видов объектов в базе данных являются так называемые **последовательности**. Это, по сути, *генераторы уникальных целых чисел*. Для работы с этими последовательностями-генераторами используются специальные функции. Одна из них — это функция **nextval**, которая как раз и получает очередное число из последовательности, имя которой указано в качестве параметра функции.
- В команде CREATE TABLE ключевое слово DEFAULT предписывает, чтобы СУБД использовала в качестве значения по умолчанию то значение, которое формирует функция nextval. Поэтому если в команде вставки строки в таблицу INSERT INTO не будет передано значение для поля типа serial, то СУБД обратится к услугам этой функции.

- В том случае, когда в таблице поле типа serial является суррогатным первичным ключом, тогда нет необходимости указывать явное значение для вставки в это поле.
- Кроме типа serial существуют еще два аналогичных типа: **bigserial** и **smallserial**. Им фактически, за кадром, соответствуют типы bigint и smallint.
- Практический совет. При выборе конкретного последовательного типа нужно учитывать предполагаемое число строк в таблице и *частоту удаления и вставки* строк, поскольку даже для небольшой таблицы может потребоваться большой диапазон, если операции удаления и вставки строк выполняются часто.

- Стандартные представители строковых типов — это **character varying(n)** и **character(n)**, где параметр указывает максимальное число символов в строке, которую можно сохранить в столбце такого типа.
- При работе с многобайтовыми кодировками символов, например, UTF-8, нужно учитывать, что речь идет именно *о символах*, а не о байтах.
- Если сохраняемая строка символов будет короче, чем указано в определении типа, то значение типа **character** будет *дополнено* пробелами до требуемой длины, а значение типа **character varying** будет сохранено так, как есть.
- Типы **character varying(n)** и **character(n)** имеют псевдонимы **varchar(n)** и **char(n)** соответственно. На практике, как правило, используют именно эти краткие псевдонимы.

- PostgreSQL дополнительно предлагает еще один символьный тип — **text**. В столбец этого типа можно ввести сколь угодно большое значение, конечно, в пределах, установленных при компиляции исходных текстов СУБД.
- Практический совет. Документация рекомендует использовать типы text и varchar, поскольку такое отличительное свойство типа character, как дополнение значений пробелами, на практике почти не востребовано. В PostgreSQL обычно используется тип text.
- Константы символьных типов в SQL-командах заключаются в одинарные кавычки:

```
SELECT 'PostgreSQL' ;
```

```
?column?
```

```
-----
```

```
PostgreSQL
```

```
(1 строка)
```

В том случае, когда в константе содержится символ одинарной кавычки или обратной косой черты, их необходимо удваивать. Например:

```
SELECT 'PGDAY' '17' ;
```

```
?column?
```

```
-----
```

```
PGDAY'17
```

```
(1 строка)
```

Можно использовать символы «\$» в качестве ограничителей. Это расширение, предлагаемое PostgreSQL. При этом уже не нужно удваивать никакие символы, содержащиеся в самой константе: ни одинарные кавычки, ни символы обратной косой черты. Например:

```
SELECT $$PGDAY'17$$;
```

```
?column?
```

```
-----
```

```
PGDAY'17
```

```
(1 строка)
```

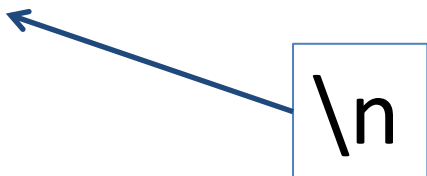
PostgreSQL предлагает еще одно расширение стандарта SQL.

Например, для включения в константу символа новой строки «\n» нужно сделать так:

```
SELECT E'PGDAY\n17' ;
```

```
?column?  
-----
```

```
PGDAY  +  
17  
(1 строка)
```

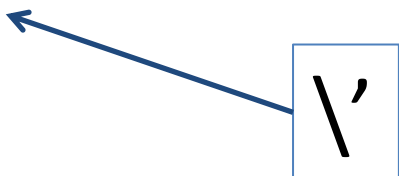


А для включения в содержимое константы символа обратной кавычки можно либо удвоить ее, либо сделать так:

```
SELECT E'PGDAY\'17' ;
```

```
?column?  
-----
```

```
PGDAY'17  
(1 строка)
```



- Язык SQL традиционно разделяется на две группы команд.
- Первая из них предназначена для определения данных, т. е. для создания объектов базы данных, таких, например, как таблицы.
- Вторая группа команд служит для выполнения различных операций с данными, таких, как вставка строк в таблицы, выполнение запросов к ним, обновление и удаление строк из таблиц.
- В этой лекции мы сосредоточимся на командах первой группы, т. е. на *определении данных*.

Описание атрибута	Имя атрибута	Тип данных	Тип PostgreSQL	Ограничения
Но зачетной книжки	record_book	Числовой	numeric(5)	NOT NULL
Ф. И. О.	name	Символьный	text	NOT NULL
Серия документа	doc_ser	Числовой	numeric(4)	
Номер документа	doc_num	Числовой	numeric(6)	

По мере рассмотрения ограничений будет становиться понятно назначение каждого из них в обеих таблицах.

Описание атрибута	Имя атрибута	Тип данных	Тип PostgreSQL	Ограничения
Но зачетной книжки	record_book	Числовой	numeric(5)	NOT NULL
Учебная дисциплина	subject	Символьный	text	NOT NULL
Учебный год	acad_year	Символьный	text	NOT NULL
Семестр	term	Числовой	numeric(1)	NOT NULL term = 1 OR term = 2
Оценка	mark	Числовой	numeric(1)	DEFAULT 5 mark >= 3 AND mark <= 5

- При работе с базами данных нередко возникают ситуации, когда то или иное значение является *типичным* для какого-то конкретного столбца.
- Например, если мы при проектировании таблицы «Успеваемость» (progress) знаем, что успехи студентов, как правило, заслуживают оценки «отлично», то в команде CREATE TABLE мы можем отразить этот факт с помощью ключевого слова DEFAULT:

```
CREATE TABLE progress
```

```
...
```

```
    mark numeric( 1 ) DEFAULT 5,
```

```
...
```

- Это ограничение бывает двух видов: ограничение уровня *атрибута* и уровня *таблицы*.
- Различие между ними только в синтаксическом оформлении: в обоих случаях в выражении могут содержаться обращения не только к одному, но также и к нескольким атрибутам таблицы.
- В первом случае ограничение CHECK является частью определения одного конкретного атрибута, а во втором случае оно записывается как самостоятельный элемент определения таблицы.

```
CREATE TABLE progress
( ...
  term numeric( 1 ) CHECK ( term = 1 OR term = 2 ),
  mark numeric( 1 ) CHECK ( mark >= 3 AND mark <= 5 ),
  ...
);
```


Каждое ограничение имеет имя. Мы можем задать его сами с помощью ключевого слова **CONSTRAINT**. Если же мы этого не сделаем, тогда СУБД сформирует имя автоматически.

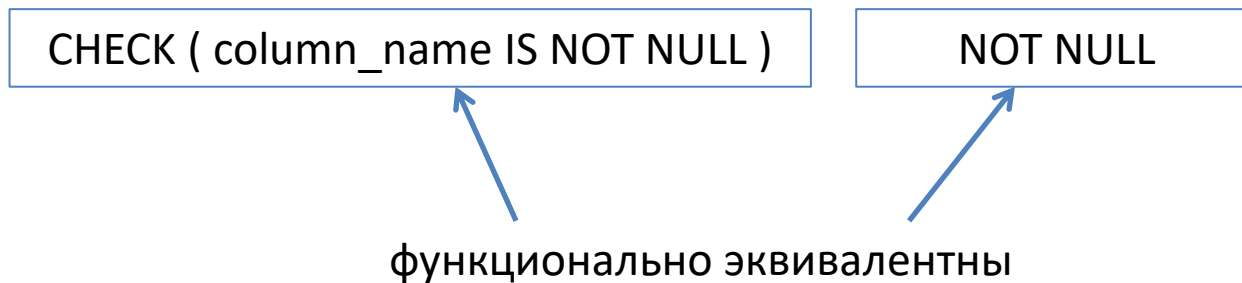
А вот ограничение уровня таблицы:

```
CREATE TABLE progress
( ...
  mark numeric( 1 ),
  CONSTRAINT valid_mark CHECK ( mark >= 3 AND mark <= 5 ),
  ...
);
```

A blue arrow points from a rectangular box containing the text 'имя ограничения' to the 'valid_mark' constraint name in the SQL code above.

имя ограничения

- Оно означает, что в столбце таблицы, на который наложено это ограничение, должны обязательно присутствовать какие-либо *определенные* значения.
- При разработке баз данных, исходя из логики конкретной предметной области, зачастую требуется использовать это ограничение.
- Как сказано в документации, оно функционально эквивалентно ограничению CHECK (column_name IS NOT NULL), но в PostgreSQL создание явного ограничения NOT NULL является более эффективным подходом.

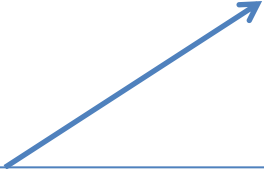


- Такое ограничение, наложенное на конкретный столбец, означает, что все значения, содержащиеся в этом столбце в различных строках таблицы, должны быть *уникальными*, т. е. не должны повторяться.
- Ограничение уникальности может включать в себя и *несколько* столбцов. В этом случае уникальной должна быть уже *комбинация их значений*.
- Когда в ограничение уникальности включается только один столбец, то можно задать ограничение непосредственно в определении столбца.


```
CREATE TABLE students  
( record_book numeric( 5 ) UNIQUE,  
  ...
```

Это ограничение можно было бы записать и так, дав ему осмысленное имя:

```
CREATE TABLE students  
( record_book numeric( 5 ),  
  name text NOT NULL,  
  ...  
  CONSTRAINT unique_record_book UNIQUE ( record_book ),  
  ...  
) ;
```



осмысленное имя
ограничения



имя уникального столбца
(столбцов)

Опять обратимся к таблице «Студенты» (students) и покажем, как можно создать ограничение уникальности, включающее более одного столбца.

```
CREATE TABLE students
( ...
  doc_ser numeric( 4 ),
  doc_num numeric( 6 ),
  ...
  CONSTRAINT unique_passport UNIQUE ( doc_ser, doc_num ),
  ...
);
```

ВАЖНО! При добавлении ограничения уникальности автоматически создается индекс на основе B-дерева для поддержки этого ограничения.

- Этот ключ является уникальным идентификатором строк в таблице.
- Ключ может быть как простым, т. е. включать только один атрибут, так и составным, т. е. включать более одного атрибута.
- При этом в отличие от уникального ключа, определяемого с помощью ограничения UNIQUE, *атрибуты, входящие в состав первичного ключа, не могут иметь значений NULL.*
- Таким образом, определение первичного ключа эквивалентно определению уникального ключа, дополненного ограничением NOT NULL.
- Однако не стоит в реальной работе заменять первичный ключ комбинацией ограничений UNIQUE и NOT NULL, поскольку теория баз данных требует наличия в каждой таблице именно первичного ключа.
- Первичный ключ является частью метаданных, его наличие позволяет другим таблицам использовать его в качестве уникального идентификатора строк в данной таблице.
- Это удобно, например, при создании внешних ключей, речь о которых пойдет ниже.
- Перечисленными свойствами обладает также и уникальный ключ.

Если первичный ключ состоит из одного атрибута, то можно указать его непосредственно в определении этого атрибута:

```
CREATE TABLE students  
( record_book numeric( 5 ) PRIMARY KEY,  
  ...  
);
```

А можно сделать это и в виде отдельного ограничения:

```
CREATE TABLE students  
( record_book numeric( 5 ),  
  ...  
  PRIMARY KEY ( record_book )  
);
```



атрибут(ы) первичного ключа

- В случае создания составного первичного ключа имена столбцов, входящих в его состав, перечисляются в выражении PRIMARY KEY через запятую:

PRIMARY KEY (column1, column2, ...)


- При добавлении первичного ключа автоматически создается *индекс* на основе *B-дерева* для поддержки этого ограничения.
- В таблице может быть любое число ограничений UNIQUE, дополненных ограничением NOT NULL, но первичный ключ может быть только один.
- PostgreSQL допускает и отсутствие первичного ключа, хотя строгая теория реляционных баз данных *не рекомендует так поступать*.

- Внешние ключи являются средством поддержания так называемой **ссылочной целостности** (referential integrity) между связанными таблицами.
- Напомним, что это означает, на примере таблиц «Студенты» (students) и «Успеваемость» (progress). В первой из них содержатся данные о студентах, а во второй — сведения об их успеваемости. Поскольку в процессе обучения студенты сдают целый ряд зачетов и экзаменов, то в таблице «Успеваемость» для каждого студента может присутствовать несколько строк.
- Для большинства из них это так и будет, хотя, в принципе, возможна ситуация, когда для какого-то студента в таблице «Успеваемость» не окажется ни одной строки (если, он, например, находится в академическом отпуске или был отчислен, не сдав ни одного экзамена).


- Конечно, должна быть возможность определить, какому студенту принадлежат те или иные оценки, т. е. какие строки в таблице «Успеваемость» с какими строками в таблице «Студенты» связаны.
- Для решения этой задачи не требуется в каждой строке таблицы «Успеваемость» повторять все сведения о студенте: номер зачетной книжки, фамилию, имя и отчество, данные документа, удостоверяющего личность.
- Достаточно включить в состав каждой строки таблицы «Успеваемость» лишь уникальный идентификатор строки из таблицы «Студенты». В нашем случае это будет номер зачетной книжки — `record_book`. Данный атрибут и будет являться *внешним ключом* таблицы «Успеваемость».
- Таким образом, получив строку из таблицы «Студенты», можно будет найти все соответствующие ей строки в таблице «Успеваемость», сопоставив значения атрибутов `record_book` в строках обеих таблиц. В результате мы сможем получить все строки таблицы «Успеваемость», связанные с конкретной строкой из таблицы «Студенты» по внешнему ключу.

- Таблица «Успеваемость» будет **ссылающейся** (referencing), а таблица «Студенты» — **ссылочной** (referenced).
- Обратите внимание, что *внешний ключ* ссылающейся таблицы ссылается на *первичный ключ* ссылочной таблицы. Допускается ссылка и на уникальный ключ, не являющийся первичным.
- В данном контексте для описания отношений между таблицами можно сказать, что таблица students является *главной*, а таблица progress — *подчиненной*.
- Создать внешний ключ можно в формате ограничения уровня атрибута следующим образом:

```
CREATE TABLE progress  
( record_book numeric( 5 )  
  REFERENCES students ( record_book ),  
...  
);
```



ссылочная (главная)
таблица



столбец (столбцы)
ссылочной таблицы

- Предложение REFERENCES создает ограничение ссылочной целостности и указывает в качестве ссылочного ключа атрибут record_book.
- Это означает, что в таблицу «Успеваемость» (progress) нельзя ввести строку, значение атрибута record_book которой отсутствует в таблице «Студенты» (students). Говоря простым языком, нельзя ввести запись об оценке того студента, информация о котором еще не введена в таблицу «Студенты».
- Поскольку внешний ключ в нашем примере ссылается на первичный ключ, можно использовать сокращенную форму записи этого ограничения, не указывая список атрибутов:

```
CREATE TABLE progress  
( record_book numeric( 5 ) REFERENCES students,  
  ...  
);
```



здесь нет списка столбцов

- Можно определить внешний ключ и в форме ограничения уровня таблицы:

```
CREATE TABLE progress
( record_book numeric( 5 ),
  ...
  FOREIGN KEY ( record_book )
    REFERENCES students ( record_book )
);
```

- **ВАЖНО!** Число атрибутов и их типы данных (и *домены!*) во внешнем ключе ссылающейся таблицы и в первичном ключе ссылочной таблицы должны быть согласованы.
- Ограничению внешнего ключа можно присвоить наименование, как и любому другому ограничению, с помощью ключевого слова CONSTRAINT.

- При наличии связей между таблицами, организованных с помощью внешних ключей, необходимо придерживаться *определенной политики* при выполнении операций удаления и обновления строк в ссылочных таблицах, т. е. в тех, *на которые* ссылаются другие таблицы.
- В нашем примере ситуация принятия «политического» решения возникает при удалении строк из таблицы «Студенты» (students).
- Тогда возникает закономерный вопрос: что делать со строками в таблице «Успеваемость» (progress), которые ссылаются на удаляемую строку в таблице «Студенты» (students)?
- Возможны несколько вариантов.

1. *Удаление связанных строк* из таблицы «Успеваемость» (progress).
2. *Запрет удаления строки* из таблицы «Студенты» (students), если в таблице «Успеваемость» (progress) есть хотя бы одна строка, ссылающаяся на удаляемую строку в таблице «Студенты»
3. *Присваивание* атрибутам внешнего ключа в строках таблицы «Успеваемость» значения *NULL*. (Если нет ограничения NOT NULL.)
4. *Присваивание* атрибутам внешнего ключа в строках таблицы «Успеваемость» (progress) значения *DEFAULT*, если оно, конечно, было предписано при создании таблицы.

Пример. При удалении какого-то отдела в большой организации его сотрудники временно переподчиняются другому отделу, код которого как раз и указывается в предложении DEFAULT.

- Удаление связанных строк из таблицы «Успеваемость» (progress), что означает, что при отчислении студента будет удаляться вся история его успехов в учебе.
- Эта операция называется **каскадным удалением** и для ее реализации в определение внешнего ключа добавляются ключевые слова ON DELETE CASCADE. Например:

```
CREATE TABLE progress
( record_book numeric( 5 ),
  ...
  FOREIGN KEY ( record_book )
    REFERENCES students ( record_book )
    ON DELETE CASCADE
);
```


Запрет удаления строки из главной таблицы (1)

- Запрет удаления строки из таблицы «Студенты» (students), если в таблице «Успеваемость» (progress) есть хотя бы одна строка, ссылающаяся на удаляемую строку в таблице «Студенты».
- Для реализации такой политики в определение внешнего ключа добавляются ключевые слова ON DELETE RESTRICT или ON DELETE NO ACTION. Если в определении внешнего ключа не предписано конкретное действие, то по умолчанию используется NO ACTION.
- Оба эти варианта означают, что если в ссылающейся таблице, т. е. «Успеваемость», есть строки, ссылающиеся на удаляемую строку в таблице «Студенты», то операция удаления будет отменена, и будет выведено сообщение об ошибке.
- Отличие между этими двумя вариантами лишь в том, что при использовании NO ACTION *можно отложить проверку* выполнения ограничения на более поздний строк в рамках транзакции, а в случае RESTRICT проверка выполняется немедленно.

Запрет удаления строки из главной таблицы (2)

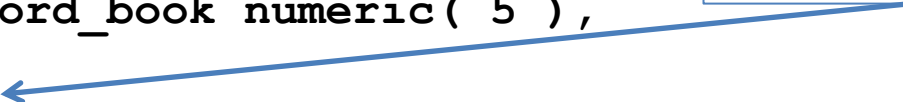
Поэтому если бы внешний ключ определили таким образом:

```
CREATE TABLE progress
( record_book numeric( 5 ),
  ...
  FOREIGN KEY ( record_book )
    REFERENCES students ( record_book ) ON DELETE RESTRICT
);
```

или таким:

```
CREATE TABLE progress
( record_book numeric( 5 ),
  ...
  -- по умолчанию NO ACTION
  FOREIGN KEY ( record_book )
    REFERENCES students ( record_book )
);
```

комментарий



то при попытке удаления строки из таблицы «Студенты» и наличии в таблице «Успеваемость» строк, связанных с ней, операция удаления была бы отменена с выводом сообщения об ошибке.

- Присваивание атрибутам внешнего ключа в строках таблицы «Успеваемость» значения NULL.
- Для реализации этого подхода необходимо, чтобы на атрибуты внешнего ключа не было наложено ограничение NOT NULL.
- Оформляется этот вариант так:

```
CREATE TABLE progress
( record_book numeric( 5 ),
  ...
  FOREIGN KEY ( record_book )
    REFERENCES students ( record_book )
    ON DELETE SET NULL
);
```

- Присваивание атрибутам внешнего ключа в строках таблицы «Успеваемость»
- (progress) значения DEFAULT, если оно, конечно, было предписано при создании таблицы.
- Оформляется этот вариант так (значение во фразе DEFAULT взято произвольное):

```
CREATE TABLE progress
( record_book numeric( 5 ) DEFAULT 12345,
  ...
  FOREIGN KEY ( record_book )
    REFERENCES students ( record_book )
    ON DELETE SET DEFAULT
);
```

- Важно учитывать, что если в ссылочной таблице нет строки с *тем же значением* ключевого атрибута, которое было предписано во фразе DEFAULT при создании ссылающейся таблицы, то будет иметь место нарушение ограничения ссылочной целостности и операция удаления не будет выполнена.

- При выполнении операции UPDATE используются эти же варианты подходов по отношению к обеспечению ссылочной целостности.
- Аналогом каскадного удаления является каскадное обновление:

```
CREATE TABLE progress  
( record_book numeric( 5 ),  
  ...  
  FOREIGN KEY ( record_book )  
    REFERENCES students ( record_book )  
    ON UPDATE CASCADE  
);
```

- В этом случае измененные значения ссылочных атрибутов копируются в ссылающиеся строки ссылающейся таблицы, т. е. новое значение атрибута record_book из строки таблицы «Студенты» будет скопировано во все строки таблицы «Успеваемость», ссылающиеся на обновленную строку.

Прежде чем создавать таблицы, создайте базу данных edu из среды операционной системы:

```
createdb -U postgres edu
```

Подключитесь к ней:

```
psql -d edu -U postgres
```

```
CREATE TABLE students
( record_book numeric( 5 ) NOT NULL,
  name text NOT NULL,
  doc_ser numeric( 4 ),
  doc_num numeric( 6 ),
  PRIMARY KEY ( record_book )
);
```

```
CREATE TABLE progress
( record_book numeric( 5 ) NOT NULL,
  subject text NOT NULL,
  acad_year text NOT NULL,
  term numeric( 1 ) NOT NULL
    CHECK ( term = 1 OR term = 2 ),
  mark numeric( 1 ) NOT NULL
    CHECK ( mark >= 3 AND mark <= 5 )
    DEFAULT 5,
  FOREIGN KEY ( record_book )
    REFERENCES students ( record_book )
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```