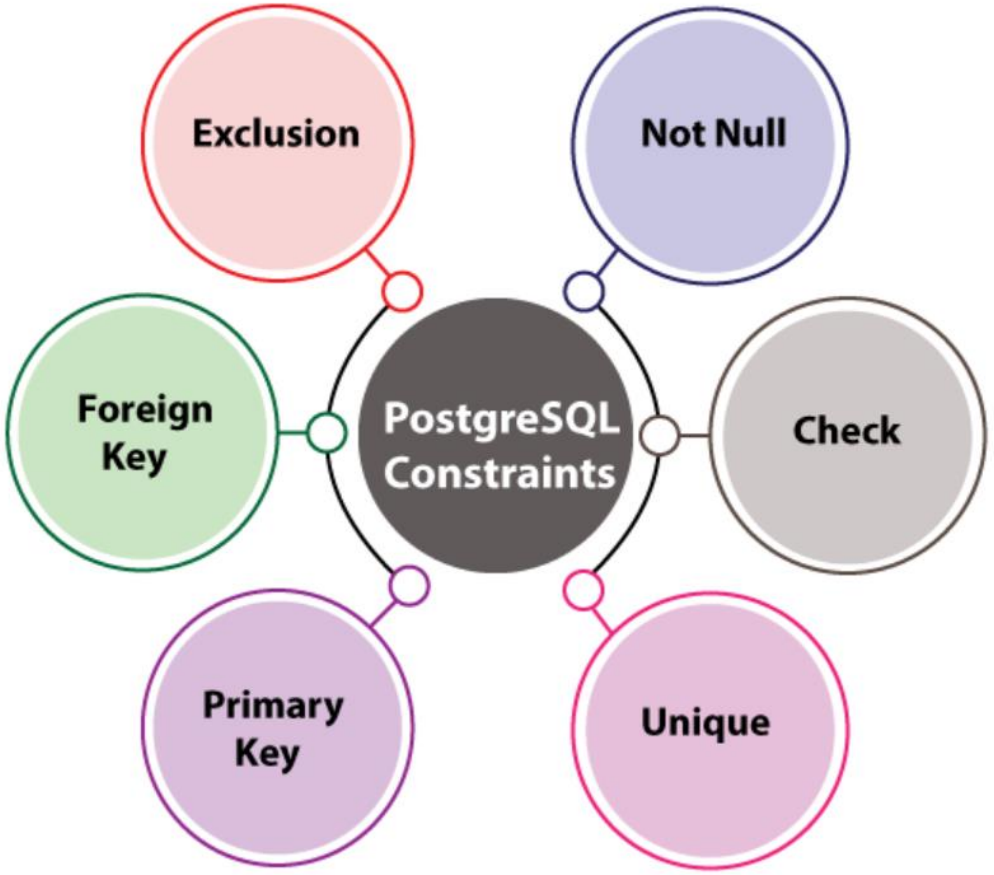
Внедрение ограничений целостности данных

Назначение ограничений целостности

- **Ограничения** – механизм Database Engine для автоматического обеспечения целостности данных в базе данных
- Ограничения обеспечивают:
  - Сущностную целостность
  - Доменную целостность
  - Ссылочную целостность

Типы ограничений целостности данных

- Разрешение значений NULL
- Определения DEFAULT значений
- Ограничения CHECK
- Ограничения UNIQUE
- Ограничения PRIMARY KEY
- Ограничения FOREIGN KEY
- Ограничение EXCLUSION



Ограничение NOT NULL

- Ограничение уровня столбца, определяемое при создании или модификации таблицы
- NOT NULL указывает, что в столбце недопустимы значения NULL

```
CREATE TABLE  "HumanResources"."Employee" (
        EmployeeID   int   IDENTITY(1,1) NOT NULL,
        NationalIDNumber   varchar(15)   NOT NULL,
        gender char(3) NULL,
        … );
```

```
--Добавление ограничения NOT NULL
ALTER TABLE employee
ALTER COLUMN gender SET NOT NULL;
```

```
--Удаление ограничения NOT NULL
ALTER TABLE employee
ALTER COLUMN NationalIDNumber DROP NOT NULL;
```

Определение DEFAULT значений

- Ограничение уровня столбца, определяемое при создании или модификации таблицы
- Определяет значение по умолчанию, которым Database Engine будет заполнять столбец, если при вставке строки для этого столбца значение не указано
- При определении значения по умолчанию для новой колонки, добавляемой в существующую таблицу, не происходит фактического добавления значений во все существующие записи

```
CREATE TABLE tbl_location(
     Availability   decimal(8, 2) NOT NULL DEFAULT 0.00,
     ModifiedDate  date NOT NULL
                      CONSTRAINT "DF_Location_ModifiedDate" DEFAULT now()
);
```

Изменение и удаление ограничений DEFAULT

```
ALTER TABLE <table_name>
ALTER COLUMN <column_name>
[SET DEFAULT <value> | DROP DEFAULT];
```

```
CREATE TABLE test_default(
        id serial,
        fone1 char(14),
        fone2 char(14)
);
```

Название	#	Тип данных	Not Null	По умолчанию
id	1	serial4	[v]	nextval('test_default_id_seq'::regclass)
fone1	2	bpchar(14)	[ ]	
fone2	3	bpchar(14)	[ ]	

```
ALTER TABLE test_default
ALTER COLUMN id DROP DEFAULT;
```

Название	#	Тип данных	Not Null	По умолчанию
id	1	int4	[v]	
fone1	2	bpchar(14)	[ ]	
fone2	3	bpchar(14)	[ ]	

Ограничение CHECK

- Ограничения CHECK может быть определено на уровне столбца или таблицы при создании или модификации таблицы
  - Для каждого столбца можно указать несколько *непротиворечивых* ограничений CHECK
- Ограничение CHECK — позволяет указать логическое условие для одного или нескольких столбцов, которое должно быть выполнено перед вставкой или обновлением значений
  - Использует логическое выражение для оценки значений перед их вставкой или обновлением в столбце
  - Если значения проходят проверку, PostgreSQL вставит или обновит эти значения в столбце
  - В противном случае PostgreSQL отклонит изменения и выдаст ошибку нарушения ограничения

```
Имя_столбца тип_данных [CONSTRAINT имя_ограничения] CHECK(условие_на значение)
```

- Если при создании ограничения не указать его имя PostgreSQL задаст его сам

```
{имя_таблицы}_{имя_столбца}_check
```

Создание ограничения CHECK

Имя ограничения

Тип ограничения

Условие на значения

```
CREATE TABLE test_check(
id serial,
fone1 char(14) CONSTRAINT ch_fn1 CHECK (fone1 ~ '8\([0-9]{3}\)[0-9]{3}-[0-9]{4}'),
fone2 char(14)
);
```

Ограничение уровня столбца

```
CREATE TABLE prices_list (
    id serial PRIMARY KEY,
    product_id int NOT NULL,
    price numeric NOT NULL,
    discount numeric NOT NULL,
    valid_from date NOT NULL,
    valid_to date NOT NULL,
    CONSTRAINT price_discount_check
    CHECK (price > 0 AND discount >= 0 AND price > discount)
);
```

Ограничение уровня таблицы

Добавление и удаление ограничения CHECK

Отключить проверку существующих данных

```
ALTER TABLE test_check
ADD CONSTRAINT ch_fn2 CHECK
                        (fone2 ~ '8\([0-9]{3}\)[0-9]{3}-[0-9]{4}') NOT VALID ;

INSERT INTO test_check(fone1,fone2)
VALUES ('8(921)883-9613','8(921)777-9557');
```

	id	fone1	fone2
1	1	8(921)143-9613	8 921 733 9553
2	2	8(921)883-9613	8(921)777-9557

```
ALTER TABLE test_check DROP CONSTRAINT ch_fn2 ;
```

Ограничение UNIQUE

- Может быть задано на уровне столбца или таблицы при создании или модификации таблицы
- Обеспечивает уникальность записей в таблице (альтернативный ключ)
  - При вставке новой строки, ограничение UNIQUE проверяет, является ли значение уникальным
  - Если значение не является уникальным (значение уже существует в столбце таблицы) - изменение отклоняется и выдается ошибка
  - Тот же процесс выполняется для обновления существующих данных
- Допускает наличие NULL-значений в столбце, при условии что не установлено ограничение NOT NULL
- Автоматически создается связанный с ограничением уникальный индекс

```
[CONSTRAINT constraint_name] UNIQUE (column_1, column_2,...)
```

- Если при создании ограничения не указать его имя PostgreSQL задаст его сам

```
{имя_таблицы}_{имя_столбца1_имя_столбца2…}_key
```

Определение ограничения UNIQUE

Ограничение уровня столбца

```
CREATE TABLE "Sales"."Customers"(
    custid serial NOT NULL,
    companyname varchar(40),
    email varchar(40) NULL UNIQUE,…
);
```

Ограничение уровня таблицы

```
CREATE TABLE "Sales"."Customers"(
   custid  serial NOT NULL,
   companyname varchar(40),
   email varchar(40) NULL,   …
   CONSTRAINT AK_email UNIQUE(email)
);
```

Ограничение уровня таблицы

```
CREATE TABLE table (
    c1 data_type,
    c2 data_type,
    c3 data_type,
CONSTRAINT AK_po_item UNIQUE (c2, c3)
);
```

Добавление и удаление ограничения UNIQUE

- При добавлении ограничения UNIQUE к существующему столбцу таблицы, в котором есть неуникальные значения – будет выдано сообщение об ошибке

```
ALTER TABLE "Sales"."Customers"
ADD CONSTRAINT Unique_Name UNIQUE (companyname);
```

- При удалении ограничения UNIQUE автоматически удаляется связанный с ограничением уникальный индекс

```
ALTER TABLE "Sales"."Customers"
DROP CONSTRAINT Unique_Name;
```

Ограничение PRIMARY KEY

- Может быть задано на уровне столбца или таблицы при создании или модификации таблицы
- Используется для создания первичного ключа таблицы
- В таблице может быть **только одно** ограничение PRIMARY KEY
- Столбец PRIMARY KEY **не может** содержать значения **NULL**
  - Технически, ограничение PRIMARY KEY представляет собой комбинацию ограничения NOT NULL и ограничения UNIQUE
- Автоматически создается связанный с ограничением уникальный индекс сбалансированного дерева

```
[CONSTRAINT constraint_name] PRIMARY KEY (column_1, column_2,...)
```

- Если при создании ограничения не указать его имя PostgreSQL задаст его сам

```
{имя_таблицы}_pkey
```

Определение ограничения Primary Key

Ограничение уровня столбца

```
CREATE TABLE "Sales"."Customers"(
     custid serial NOT NULL PRIMARY KEY,
     companyname varchar(40), …
);
```

Ограничение уровня таблицы

```
CREATE TABLE "Sales"."Customers"(
   custid  serial NOT NULL,
   companyname varchar(40), …
   CONSTRAINT PK_CustomerID PRIMARY KEY(custid)
);
```

Ограничение уровня таблицы

```
CREATE TABLE po_items (
    po_no integer,
    item_no integer,
    product_no integer,
    qty integer,
    net_price numeric,
    CONSTRAINT PK_po_item PRIMARY KEY (po_no, item_no)
);
```

Добавление и удаление ограничения Primary Key

- При добавлении ограничения **Primary Key** к существующему столбцу таблицы, в котором есть **неуникальные** значения – будет выдано сообщение об ошибке

```
ALTER TABLE po_items ADD PRIMARY KEY (po_no, item_no);
```

```
ALTER TABLE vendors ADD COLUMN ID SERIAL PRIMARY KEY;
```

- При попытке удаления первичного ключа, на который ссылается внешний ключ в связной таблице, будет выдано сообщение об ошибке
- Для удаления первичного ключа и связанного с ним внешнего ключа необходимо использовать указание **CASCADE**
- При удалении ограничения Primary Key автоматически удаляется связанный с ограничением уникальный индекс

```
ALTER TABLE po_items DROP CONSTRAINT po_items_pkey CASCADE ;
```

Добавление уникального ограничения с использованием уникального индекса


```
CREATE TABLE equipment (
    id SERIAL PRIMARY KEY,
    name VARCHAR (50) NOT NULL,
    equip_id VARCHAR (16) NOT NULL
);
```

```
CREATE UNIQUE INDEX CONCURRENTLY
equipment_equip_id
                ON equipment (equip_id);
```

```
ALTER TABLE equipment
ADD CONSTRAINT unique_equip_id
UNIQUE USING INDEX equipment_equip_id;
```

```
ALTER TABLE / ADD CONSTRAINT USING INDEX
will rename index "equipment_equip_id" to
"unique_equip_id"
```

Ограничения UNIQUE и PRIMARY KEY могут быть созданы на основе уникального индекса

Создание уникального индекса на основе столбца **equip_id**

Добавление уникального ограничения в таблицу, используя существующий индекс **equipment_equip_id**

После создания ограничения соответствующий индекс переименовывается!

Ограничение FOREIGN KEY

- Внешний ключ — это столбец или группа столбцов в таблице, которые ссылаются на существующий **первичный** или **альтернативный ключ** другой таблицы
  - Таблица, содержащая внешний ключ, называется ссылочной таблицей или дочерней таблицей
  - Таблица, на которую ссылается внешний ключ, называется родительской таблицей
- Таблица может иметь несколько внешних ключей в зависимости от ее отношений с другими таблицами
- В PostgreSQL внешний ключ определяется с помощью ограничения **FOREIGN KEY**
  - Ограничение FOREIGN KEY помогает поддерживать ссылочную целостность данных между дочерней и родительской таблицами
  - Ограничение внешнего ключа указывает, что значения в столбце или группе столбцов дочерней таблицы равны значениям в столбце или группе столбцов родительской таблицы
- У пользователя должно быть разрешение REFERENCES для таблицы, на которую указывают ссылки
- Индекс не создается автоматически, поэтому его необходимо создать самостоятельно

Синтаксис

```
[CONSTRAINT имя_ограничения] FOREIGN KEY(список_столбцов)
REFERENCES имя_таблицы_родителя (список_столбцов)
      [ON DELETE тип_обработки]
      [ON UPDATE тип_обработки]
```

```
ALTER TABLE "Production"."Products"
      ADD CONSTRAINT "FK_Category" FOREIGN KEY (categoryid)
      REFERENCES "Production"."Categories"(categoryid)
      ON DELETE NO ACTION
      ON UPDATE CASCADE;
```

Определение ограничения Foreign Key

Ограничение уровня столбца

```
CREATE TABLE ord(
        ordid serial PRIMARY KEY,
        idcl int  REFERENCES cl(id),
        dt date);
```

Ограничение уровня таблицы

```
CREATE TABLE ord(
        ordid serial PRIMARY KEY,
        idcl int,
        CONSTRAINT fk_idcl FORIGN KEY(idcl)
        REFERENCES cl(id),
        dt date);
```

Ограничение уровня таблицы

```
CREATE TABLE ord (
        ordid serial PRIMARY KEY,
        passport_ser char(3),
        passport_num char(6),
        CONSTRAINT fk_idcl FORIGN KEY (passport_ser, passport_num)
        REFERENCES cl(passport_ser, passport_num),
        dt date);
```

Ссылочная целостность

CategoryID	ParentProductCategoryID	Name	…
1	NULL	Bikes	…
2	NULL	Components	…
…	…	…	…
6	1	Road bikes	…
8	2	Handlebars	

Первичный ключ

Ссылка

Внешний ключ

ProductID	Name	ProductNumber	CategoryID	…
749	Road-150 Red, 62	BK-R93R-62	6	…
750	Road-150 Red, 44	BK-R93R-44	6	…
…	…	…	…	…
808	LL Mountain Handlebars	HB-M243	8	…

Правила удаления и обновления

```
ALTER TABLE имя_таблицы
ADD CONSTRAINT
имя_ограничения FOREIGN KEY(список_столбцов)
REFERENCES имя_таблицы_родителя (список_столбцов)
 [ON DELETE тип_обработки]
 [ON UPDATE тип_обработки]

тип_обработки = SET NULL | SET DEFAULT | RESTRICT | NO ACTION | CASCADE
```

Правила удаления и обновления

```
UPDATE category
SET categoryid = 11
WHERE categoryid = 1;
```

categoryid	categoryname
1	Beverages
2	Condiments
3	Confections
4	Dairy Products
5	Grains/Cereals
6	Meat/Poultry
7	Produce
8	Seafood

```
DELETE category
WHERE categoryid = 7;
```

ON UPDATE

ON DELETE

Тип обработки

NO ACTION
SET NULL
SET DEFAULT
RESTRICT
CASCADE

productid	productname	categoryid
1	Product HHYDP	1
2	Product RECZE	1
3	Product IMEHJ	2
4	Product KSBRM	2
5	Product EPEIM	2
6	Product VAIIV	2
7	Product HMLNI	7
8	Product WVJFP	2
9	Product AOZBW	6
10	Product YHXGE	8

Тип обработки

- **NO ACTION** – если при проверке ограничения обнаруживаются зависимые записи, возникает ошибка (это поведение по умолчанию)
  - NO ACTION позволяет отложить проверку в процессе транзакции
- **RESTRICT** – предотвращает удаление связанной записи
  - Действие RESTRICT аналогично действию NO ACTION
  - Разница возникает только при определении ограничения внешнего ключа как ***DEFERRABLE*** с режимом ***INITIALLY DEFERRED*** или ***INITIALLY IMMEDIATE***
- **CASCADE** – автоматическое удаление зависимых записей при удалении связанных записей
- **SET NULL** – замена значений внешнего ключа на **NULL** в зависимых записях при удалении связанных записей
- **SET DEFAULT** – замена значений внешнего ключа на значение по умолчанию в зависимых записях при удалении связанных записей
  - Необходимо определение ограничения ***DEFAULT*** для внешнего ключа
  - Необходимо наличие записи в главной таблице с соответствующим значением первичного ключа

Каскадные обновления

categoryid	categoryname
1	Beverages
12	Condiments
3	Confections
4	Dairy Products
5	Grains/Cereals
6	Meat/Poultry
7	Produce
8	Seafood

CASCADE

productid	productname	categoryid
1	Product HHYDP	1
2	Product RECZE	1
3	Product IMEHJ	12
4	Product KSBRM	12
5	Product EPEIM	12
6	Product VAIIV	12
7	Product HMLNI	7
8	Product WVJFP	12
9	Product AOZBW	6
10	Product YHXGE	8

```
CREATE TABLE product(
        productid serial PRIMARY KEY,
        Productname varchar(15),
        categoryid int,
        CONSTRAINT fk_catid FORIGN KEY(categoryid)
        REFERENCES category(categoryid)
        ON UPDATE CASCADE
        ON DELETE NO ACTION);
```

Каскадные удаления

categoryid	categoryname
1	Beverages
2	Condiments
3	Confections
4	Dairy Products
5	Grains/Cereals
6	Meat/Poultry
7	Produce
8	Seafood

CASCADE

productid	productname	categoryid
1	Product HHYDP	1
2	Product RECZE	1
3	Product IMEHJ	2
4	Product KSBRM	2
5	Product EPEIM	2
6	Product VAIIV	2
7	Product HMLNI	7
8	Product WVJFP	2
9	Product AOZBW	6
10	Product YHXGE	8

CASCADE

orderid	productid	unitprice
10248	1	14
10248	2	9,8
10248	4	34,8
10249	7	18,6
10249	3	42,4
10250	5	7,7
10250	1	42,4
10250	6	16,8
10251	6	16,8
10251	8	15,6
10251	7	16,8
10252	1	64,8
10252	3	2

ВАЖНО

- Влияние **ALTER TABLE ADD CONSTRAINT**:
  - При добавлении ограничений к таблице выполняется проверка существующих данных
  - Сканирование большой таблицы может занять длительное время и будет препятствовать внесению других изменений до фиксирования команды ALTER TABLE ADD CONSTRAINT
- Для снижения влияния данной операции на параллельные транзакции можно отключить проверку существующих данных

Отключение проверки существующих данных

- Параметр **NOT VALID**
  - **разрешен только для ограничений CHECK и FOREIGN KEY**
  - команда **ALTER TABLE ADD CONSTRAINT** не сканирует таблицу и может быть зафиксирована немедленно
  - созданное ограничение будет применяться к последующим вставкам или обновлениям, но существующие записи будут игнорироваться
  - возможность отложить проверку на время меньшей активности или провести дополнительную работу с существующими ошибками и при этом не допустить новых

```
ALTER TABLE distributors
      ADD CONSTRAINT fk_dist FOREIGN KEY (address)
      REFERENCES addresses (address) NOT VALID;
```

Проверка существующих данных

- Команда **VALIDATE CONSTRAINT**
  - выполняется полное сканирование таблицы для проверки всех существующих строк на соответствие ограничению
  - если ограничение уже помечено как верное, ничего не происходит
  - выполнение этой команды не препятствует параллельным изменениям (для добавляемых или изменяемых строк в других транзакциях ограничение уже действует)

```
ALTER TABLE distributors VALIDATE CONSTRAINT fk_dist;
```

Гранулярность проверки ограничений

Гранулярность проверки ограничений

- В PostgreSQL существует 3 уровня проверки ограничений:
  - ***На уровне строки***
    - SQL-запрос, который обновляет множество строк, остановится на первой строке, не удовлетворяющей ограничению
  - ***На уровне запроса***
    - В этом случае запрос произведет все изменения, прежде чем проверить ограничения
  - ***На уровне транзакции***
    - Любой запрос внутри транзакции может нарушить ограничение
    - Но в момент фиксации, ограничения будут проверены и транзакция откатится, если хотя бы одно из них будет нарушено

DEFERRABLE

- Чтобы изменить гранулярность проверки ограничения необходимо явно объявить ограничение как отложенное (DEFERRABLE)
  - Работает только для **UNIQUE, PRIMARY KEY** и **REFERENCES**
  - **CHECK и NOT NULL всегда будут проверяться для каждой строки**
- При создании ограничения определить одну из следующих характеристик:
  - ***DEFERRABLE INITIALLY DEFERRED*** - откладываемое, по-умолчанию отложенное
  - ***DEFERRABLE INITIALLY IMMEDIATE*** - откладываемое, по-умолчанию немедленное (будет проверяться сразу)
  - ***NOT DEFERRABLE*** – не откладываемое (**по умолчанию!**)

Гранулярность проверки ограничений

Тип ограничения	NOT DEFERRABLE	DEFERRABLE INITIALLY DEFERRED	DEFERRABLE INITIALLY IMMEDIATE
CHECK	строка	строка	строка
NOT NULL	строка	строка	строка
UNIQUE	строка	запрос	транзакция
PRIMARY KEY	строка	запрос	транзакция
FOREIGN KEY	запрос	запрос	транзакция
EXCLUDE	запрос	запрос	транзакция

Определение и изменение гранулярности ограничения


```
CREATE TABLE product(
    productid serial PRIMARY KEY,
    Productname varchar(15),
    categoryid int,
    CONSTRAINT fk_catid FOREIGN KEY(categoryid)
    REFERENCES category(categoryid)
        ON UPDATE CASCADE
        ON DELETE NO action
        NOT DEFERRABLE);
```

```
ALTER TABLE product
ALTER CONSTRAINT fk_catid DEFERRABLE INITIALLY DEFERRED;
```

ВНИМАНИЕ: В настоящее время изменять гранулярность можно только для ограничения **внешнего ключа**!

Управление временем проверки ограничений для текущей транзакции

- Чтобы указать когда будет выполняться проверка ограничений в текущей транзакции используется инструкция

```
SET CONSTRAINTS { ALL | имя [, ...] } { DEFERRED | IMMEDIATE }
```

- Режим **IMMEDIATE** или **DEFERRED** задаётся для каждого ограничения независимо:
  - Ограничения ***IMMEDIATE*** проверяются в конце каждого оператора
  - Ограничения ***DEFERRED*** откладываются до момента фиксации транзакции
- На ограничения, созданные с параметром **NOT DEFERRABLE**, команда **SET CONSTRAINTS** не влияет!
- Для AutoCommit транзакций **нет разницы** между ***IMMEDIATE*** и ***DEFERRED*** для одного запроса

Управление временем проверки

```
ALTER TABLE product
   ADD CONSTRAINT fk_catid FOREIGN KEY(categoryid)
   REFERENCES category(categoryid)
  DEFERRABLE INITIALLY IMMEDIATE;
-- ограничение объявлено с возможностью быть отложенным,
-- но по умолчанию оно будет проверяться сразу
```

```
BEGIN;
-- Отложить проверку ограничения
SET CONSTRAINTS fk_catid DEFERRED;
-- ...
-- Производим операции над categoryid
-- ...
COMMIT; -- В данном месте произойдёт проверка ограничения
```

Пример 1

```
CREATE TABLE snowflakes ( i int UNIQUE DEFERRABLE INITIALLY IMMEDIATE);
--проверка будет осуществляться на уровне транзакции
INSERT INTO snowflakes VALUES (1), (2), (3);
UPDATE snowflakes SET i = i + 1;
```

Updated Rows	3
Query	UPDATE snowflakes2 SET i = i + 1
Finish time	Mon Nov 14 20:12:11 MSK 2022

- Данная ошибка возникает, так как PostgreSQL проверяет ограничения **на уровне строки**
  - После изменения первой строки состояние данных в столбце будет **2, 2, 3** - а это нарушение ограничения **UNIQUE**
- Если бы PostgreSQL дождался обновления всех строк (как в проверке на уровне «по запросу»), то проблем бы не было
  - Значение i в строках будет последовательно увеличиваться и в итоге они все станут уникальными
- **Построчная проверка ограничений зависит от физического расположения строк**
  - Проблема бы не возникла если бы изначально мы заполнили строки в обратном порядке

Пример 2

```
ALTER TABLE classes
DROP CONSTRAINT classes_teacher_id_key;
ALTER TABLE classes
ADD CONSTRAINT classes_teacher_id_key UNIQUE (teacher_id)
    DEFERRABLE INITIALLY IMMEDIATE;
```

```
BEGIN;
    SET CONSTRAINTS classes_teacher_id_key DEFERRED;

    UPDATE classes SET teacher_id = 1 WHERE id = 2;
    UPDATE classes SET teacher_id = 2 WHERE id = 1;
COMMIT;
```

Updated Rows	2
Query	BEGIN;
	SET CONSTRAINTS classes_teacher_id_key DEFERRED;
	UPDATE classes SET teacher_id = 1 WHERE id = 2;
	UPDATE classes SET teacher_id = 2 WHERE id = 1;
	COMMIT
Finish time	Mon Nov 14 20:45:50 MSK 2022

Отложенные ограничения позволяют

- Добавлять и изменять данные в таблицы, связанные циклическими проверками согласованности (циклические ограничения муж/жена)
- Сделать процесс наполнения таблиц более удобным
  - Отложенные внешние ключи позволяют загружать данные в таблицы в любом порядке
  - Например вначале потомков, а затем родителей

Проблемы

- Несмотря на то, что объявление ограничений отложенными даёт большую гибкость, не стоит «откладывать» все ограничения
- ***Неэффективная работа планировщика***
  - так как нет гарантий, что откладываемые ограничения будут выполняться всё время, они мешают планировщику в выборе правильных и эффективных стратегий выполнения запросов
- ***Усложнение отладки***
  - получение ошибок только после завершения набора запросов усложняет отладку, так как нет возможности точно определить, какой запрос вызвал проблему
- ***Неэффективное использование ресурсов***
  - любая работа, выполненная после отложенного ограничения, может быть в итоге потеряна, после отката транзакции. Отложенные ограничения могут тратить CPU впустую.

Получение информации о существующих ограничениях

- Тип ограничения в системном каталоге указывается одним из символов:

```
SELECT conname, contype
FROM pg_catalog.pg_constraint;
```

- “**p**” - PRIMARY KEY
- “**f**” – FOREIGN KEY
- “**c**” - CHECK
- “**u**” - UNIQUE

```
SELECT table_schema, table_name, column_name,
       conname, contype, condeferrable
FROM   pg_catalog.pg_constraint
       JOIN
       information_schema.constraint_column_usage
       ON constraint_name = conname
```

Дополнительные возможности

Домен

- **Домен** — это определяемый пользователем тип данных, который может иметь необязательные ограничения (DEFAULT, NOT NULL и CHECK), ограничивающие допустимый набор значений
- Домены полезны для абстрагирования и вынесения общих характеристик разных полей в единое место для упрощения сопровождения
  - Например, столбец телефон, определенный в нескольких таблицах, требует наличие ограничения CHECK, проверяющие синтаксис телефона
  - В этом случае лучше определить домен, а не задавать для каждой таблицы отдельные ограничения

https://postgrespro.ru/docs/postgresql/14/sql-createdomain

Создание домена

- Для создания домена используется оператор **CREATE DOMAIN**
  - Имя должно быть уникально в пределах схемы среди имён типов и доменов
  - Необходимо иметь право **USAGE** для исходного типа данных
  - Пользователь, определяющий домен, становится его владельцем

```
CREATE DOMAIN схема.имя [ AS ] тип_данных
    [ COLLATE правило_сортировки ]
    [ DEFAULT выражение ]
    [ CONSTRAINT имя_ограничения { NOT NULL | NULL | CHECK (выражение) }]
```

- ***DEFAULT***:
  - Если значение по умолчанию не указано, им будет значение NULL
  - Значение по умолчанию для конкретного столбца имеет приоритет
- ***CHECK***:
  - Задает ограничение проверки **уровня столбца**
  - Проверяемое значение в этом выражении обозначается ключевым словом **VALUE**
  - Если для домена задано несколько ограничений **CHECK**, они будут проверяться в алфавитном порядке имён

Пример

- Фамилия и имя студента не должны содержать NULL и пробелы

```
CREATE TABLE marksheet (
    student_id SERIAL PRIMARY KEY,
    first_name VARCHAR NOT NULL,
    last_name  VARCHAR NOT NULL,
    email VARCHAR NOT NULL,
    marks_obtained INT NOT NULL,
    CHECK (
        first_name !~ '\s'
        AND last_name !~ '\s'
    )
);
```

```
CREATE DOMAIN student_detail
AS
    VARCHAR NOT NULL CHECK (VALUE !~ '\s');

CREATE TABLE marksheet (
    student_id serial PRIMARY KEY,
    first_name student_detail,
    last_name  student_detail,
    marks_obtained INT NOT NULL,
    email VARCHAR NOT NULL
);
```

Изменение домена

```
ALTER DOMAIN имя { SET DEFAULT выражение | DROP DEFAULT }
ALTER DOMAIN имя { SET | DROP } NOT NULL
ALTER DOMAIN имя ADD ограничение_домена [ NOT VALID ]
ALTER DOMAIN имя DROP CONSTRAINT [ IF EXISTS ] имя_ограничения [ RESTRICT | CASCADE ]
ALTER DOMAIN имя RENAME CONSTRAINT имя_ограничения TO имя_нового_ограничения
ALTER DOMAIN имя VALIDATE CONSTRAINT имя_ограничения
ALTER DOMAIN имя OWNER TO { новый_владелец | CURRENT_USER | SESSION_USER }
ALTER DOMAIN имя RENAME TO новое_имя
ALTER DOMAIN имя SET SCHEMA новая_схема
```

- В случае, если в выражении CHECK используется пользовательская функция, поведение которой впоследствии меняется необходимо:
  - удалить ограничение (ALTER DOMAIN ….DROP CONSTRAINT)
  - изменить определение функции
  - пересоздать ограничение (при этом будет выполнена перепроверка сохранённых данных)

https://postgrespro.ru/docs/postgresql/14/sql-alterdomain

Удаление домена

```
DROP DOMAIN [ IF EXISTS ] имя [, ...] [ CASCADE | RESTRICT ]
```

- **CASCADE**
  - Автоматически удалять объекты, зависящие от данного домена (например, столбцы таблиц), и, в свою очередь, все зависящие от них объекты
- **RESTRICT**
  - Отказать в удалении домена, если от него зависят какие-либо объекты. Это поведение по умолчанию
- **Удалить домен может только его владелец**

Ограничение EXCLUDE

- **EXCLUDE** определяет ограничение исключения, которое гарантирует, что при сравнении любых двух строк в указанных столбцах или выражениях с использованием указанных операторов хотя бы одно из этих сравнений (**OR**) вернет значение **FALSE** или **NULL**
  - Ограничение исключения не работает, если все проверяемые им условия верны
  - Если все указанные операторы проверяют равенство, это эквивалентно ограничению UNIQUE (но ограничение UNIQUE работает быстрее)
- Когда добавляется новая или обновленная строка, необходимо сравнить ее с каждой существующей строкой на основе списка сравнений, указанных в ограничении:
  - Если все сравнения между новыми данными и существующей строкой возвращают значение **TRUE**, отклонить новые данные

Синтаксис

```
[CONSTRAINT constraint_name] EXCLUDE [ USING index_method ]
(name_of_column WITH operator [, ... ] )
index_parameters [ WHERE ( predicate ) ]
```

- Ограничения исключения реализуются с помощью индекса
  - на практике метод доступа всегда будет **GiST** или **SP-GiST**
- Необходимо установить расширение **btree_gist**, которое определяет ограничения исключения для простых скалярных типов данных (выполнить один раз для каждой БД)

```
CREATE EXTENSION btree_gist;
```

- Предикат позволяет указать ограничение исключения для подмножества таблицы (создает частичный индекс)
  - Обратите внимание, что круглые скобки необходимы вокруг предиката

Пример 1

- **Задача**: не позволять нескольким пользователям одновременно арендовать одну и ту же недвижимость (property_id). Можно арендовать ту же недвижимость на следующей неделе или другую недвижимость на перекрывающийся период
- Новая строка резервирования будет недействительна, если:
  - значение **property_id**, равно значению **property_id** в существующей строке (**WITH =**)
  - и имеет диапазон дат (**tsrange**), который пересекается (**&&**) с диапазоном дат существующего резервирования (**WITH &&**)

```
ALTER TABLE reservations ADD CONSTRAINT no_overlapping_rentals
EXCLUDE USING gist (property_id WITH =,
tsrange("checkin_time", "checkout_time", '[]') WITH &&);
```

Пример 2

```
CREATE EXTENSION btree_gist;

CREATE TABLE company(
   ID INT PRIMARY KEY     NOT NULL,
   NAME           TEXT,
   AGE            INT   ,
   ADDRESS        CHAR(50),
EXCLUDE USING gist
-- USING gist – это тип индекса,
--который нужно создать и использовать для
--принудительного применения
--исключить тех:
  ( NAME WITH =,
--чье имя совпадает с уже существующим
    AGE WITH <>) ;
--а возраст при этом не совпадает
```

Новая запись будет ***недействительна*** если имя сотрудника совпадает с именем уже существующего **И** при этом возраст не совпадает

```
INSERT INTO company
VALUES (1, 'Paul', 32, 'California' );
INSERT INTO company
VALUES (2, 'Paul', 32, 'Texas');
INSERT INTO company
VALUES (3, 'Paul', 42, 'California');
```

⚠ SQL Error [23P01]: ERROR: conflicting key value violates exclusion constraint "company_name_age_excl"
Подробности: Key (name, age)=(Paul, 42) conflicts with existing key (name, age)=(Paul, 32).

Дополнительное чтение

- https://www.tutorialspoint.com/postgresql/postgresql_constraints.htm
- https://stackoverflow.com/questions/10292355/how-to-create-a-real-one-to-one-relationship-in-sql-server