КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
Институт экологии и природопользования
Кафедра моделирования экологических систем

А.К. Гильфанов, Т.В. Никоненкова, Е.А. Костерина

SQL-запросы в экологических информационных системах

Учебное пособие

2-ое издание, исправленное

Казань – 2018

УДК 574:002.52/.54(07)

Печатается по решению учебно-методической комиссии
Института экологии и природопользования
Казанского (Приволжского) федерального университета
Протокол №3 от 27 апреля 2018 г.

Рецензент
ст. преп. *Пилюгин А.Г.*

Гильфанов А.К. SQL-запросы в экологических информационных системах: учебное пособие / А.К. Гильфанов, Т.В. Никоненкова, Е.А. Костерина. – 2-ое изд., исправ. – Казань: Казан. ун-т, 2018. – 40 с.

Учебное пособие предназначено для использования при изучении дисциплины «Информатика» студентами направления подготовки бакалавров «Экология и природопользование» Института экологии и природопользования. Оно может быть использовано также студентами других естественнонаучных направлений подготовки бакалавров.

В учебном пособии приведены основные операторы языка SQL определения, манипулирования и выборки данных на примере флористической базы данных «Флора». Задания для самостоятельной работы даны для базы данных загрязнений атмосферного воздуха. Пособие составлено с целью развития у студентов навыков работы в современных информационных системах и применения их в экологии. Все примеры рассчитаны на использование системы PostgreSQL 8.4.

УДК 574:002.52/.54(07)

© Казанский федеральный университет, 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современных условиях использование информационных систем, основанных на реляционных базах данных, проникло практически во все области человеческой жизни. В экологии и природопользовании подобные системы развиваются в связи с накоплением большого количества биологического и экологического материала, который требует правильной организации хранения данных и доступа к ним. Примерами экологических информационных систем являются так называемые таксономические, геоботанические, флористические и др. базы данных. В частности, сотрудниками кафедры общей экологии Казанского университета создана флористическая база данных «Флора» [3], содержащая данные по растительности на территории Республики Татарстан. Особое место занимают базы данных экологического мониторинга. В настоящее время окружающая среда подвергается сильному антропогенному воздействию, и возникает необходимость в постоянном мониторинге загрязнений атмосферного воздуха, водных объектов, почв и др. Данные измерений при этом наиболее удобно хранить в виде баз данных, что сильно облегчает последующую их обработку.

Любая база данных создается и используется с помощью программного обеспечения, называемого системой управления базами данных (СУБД). Таких систем существует несколько. Каждая СУБД обладает своими особенностями, но при этом единственным стандартом работы с базами данных остается язык SQL (Structured Query Language), позволяющий создавать базы данных, таблицы, манипулировать данными, проводить их анализ. Порог вхождения в язык SQL не является высоким, и его освоение не требует навыков программирования, хотя для успешной работы некоторые знания в этой области были бы желательны. Поэтому изучение этого языка может быть рекомендовано студентам, не получающим специального программистского образования. Несмотря на независимость языка SQL от конкретной СУБД, пользователю необходимы также навыки работы в конкретной системе. Все примеры в настоящем пособии были отлажены в СУБД PostgreSQL 8.4.

Настоящее пособие предназначено для студентов, обучающихся по направлению подготовки бакалавров «Экология и природопользование». Целью пособия является развитие у студентов навыков работы с реляционными базами данных с помощью языка SQL. Включены темы, охватывающие основной цикл работы с базой данных: создание таблиц, занесение и удаление данных, индексирование, организация связей между таблицами, выборка и анализ данных и др. В каждом разделе содержится необходимый теоретический материал, предлагается выполнить примеры запросов, а также даны задания для самостоятельной работы. Все примеры операторов языка SQL показаны на основе структуры базы данных «Флора». В качестве самостоятельной работы в пособии предложены задания по базе данных загрязнений атмосферного воздуха.

При составлении пособия использовались перечень [1], база данных «Флора» [3], электронный ресурс и книга по SQL [2, 5], книга и документация по системе PostgreSQL [4, 6].

Автор выражает благодарность Прохорову Вадиму Евгеньевичу за любезно предоставленную базу данных «Флора».

1. Сведения о СУБД PostgreSQL.

Система управления базами данных (СУБД) – совокупность программных средств, предназначенная для ведения и использования базы данных. Во всех современных реляционных СУБД используется универсальный язык SQL (Structured Query Language), позволяющий создавать таблицы, модифицировать и управлять данными. Язык SQL стандартизирован, основные его конструкции поддерживаются всеми системами. Однако при реализации языка в некоторых системах вносятся изменения. Таким образом, нельзя быть уверенными, что отлаженный в одной СУБД запрос сработает в другой. В настоящем пособии все SQL-запросы запускались в системе PostgreSQL 8.4.

PostgreSQL – объектно-реляционная система управления базами данных. Работа над данной системой была начата в Калифорнийском университете в Беркли в конце 1970-х годов. В настоящее время, будучи мощным средством для работы с базами данных, PostgreSQL активно используется в различных отраслях деятельности: промышленности, коммерции, образовании, телекоммуникациях, в государственных учреждениях и др. Система является кроссплатформенной, т.е. существует в реализациях для различных операционных систем: Linux, Windows, Mac OS и др. PostgreSQL распространяется на условиях открытого исходного кода, т.е. пользователь имеет право свободно использовать, копировать и модифицировать программный комплекс.

Система PostgreSQL может быть установлена на компьютере двумя способами: с помощью установщика или сборкой из исходного кода. Рекомендуется использовать первый способ. Скачать установщик для операционной системы Windows можно по ссылке с официального сайта http://www.postgresql.org/download/windows/. При установке необходимо указать путь, по которому будет установлена программа и где будут храниться файлы базы данных. Далее нужно ввести пароль для суперпользователя системы (обычно, имя – postgres). Все остальные настройки, предлагаемые при установке, лучше оставить по умолчанию.

При инсталляции автоматически будет установлена программа pgAdmin III – программа с графическим интерфейсом, позволяющая работать с базами данных. Сама СУБД графической оболочки не имеет.

Для работы с базами данных необходимо подсоединиться к серверу PostgreSQL, как показано на рис. 1. При этом надо ввести пароль, указанный при

установке. Чтобы создать новую базу данных, нужно перейти в объект «Базы» и далее выбрать опции меню «Правка=>Новый объект=>Новая база данных», после чего ввести имя базы данных. Также могут быть установлены более тонкие настройки.

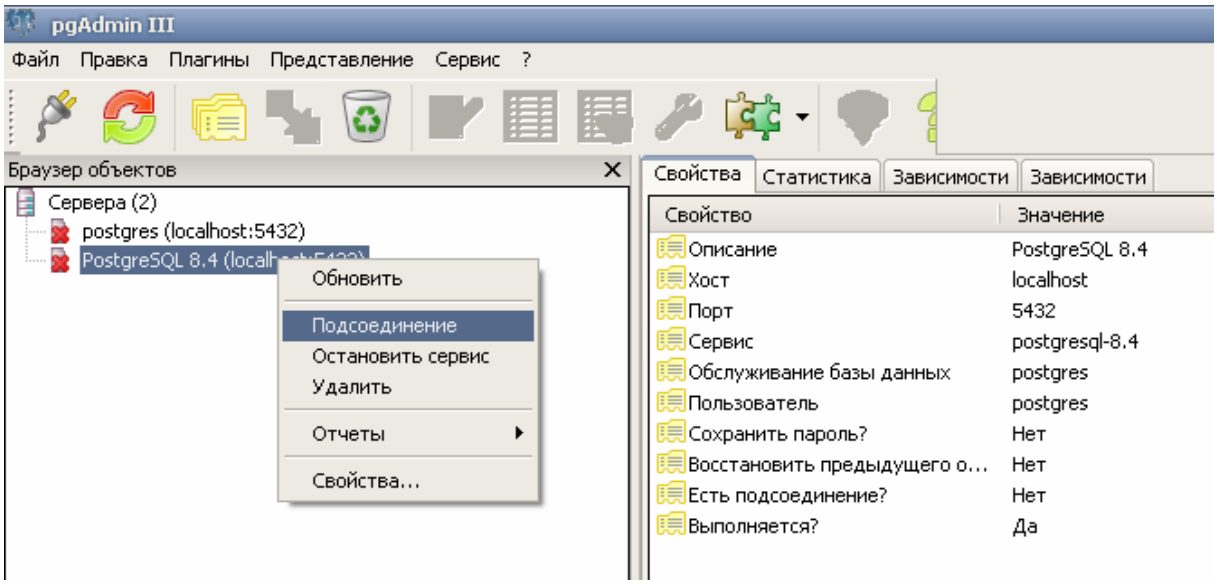


Рис. 1. Главное окно программы pgAdmin III.

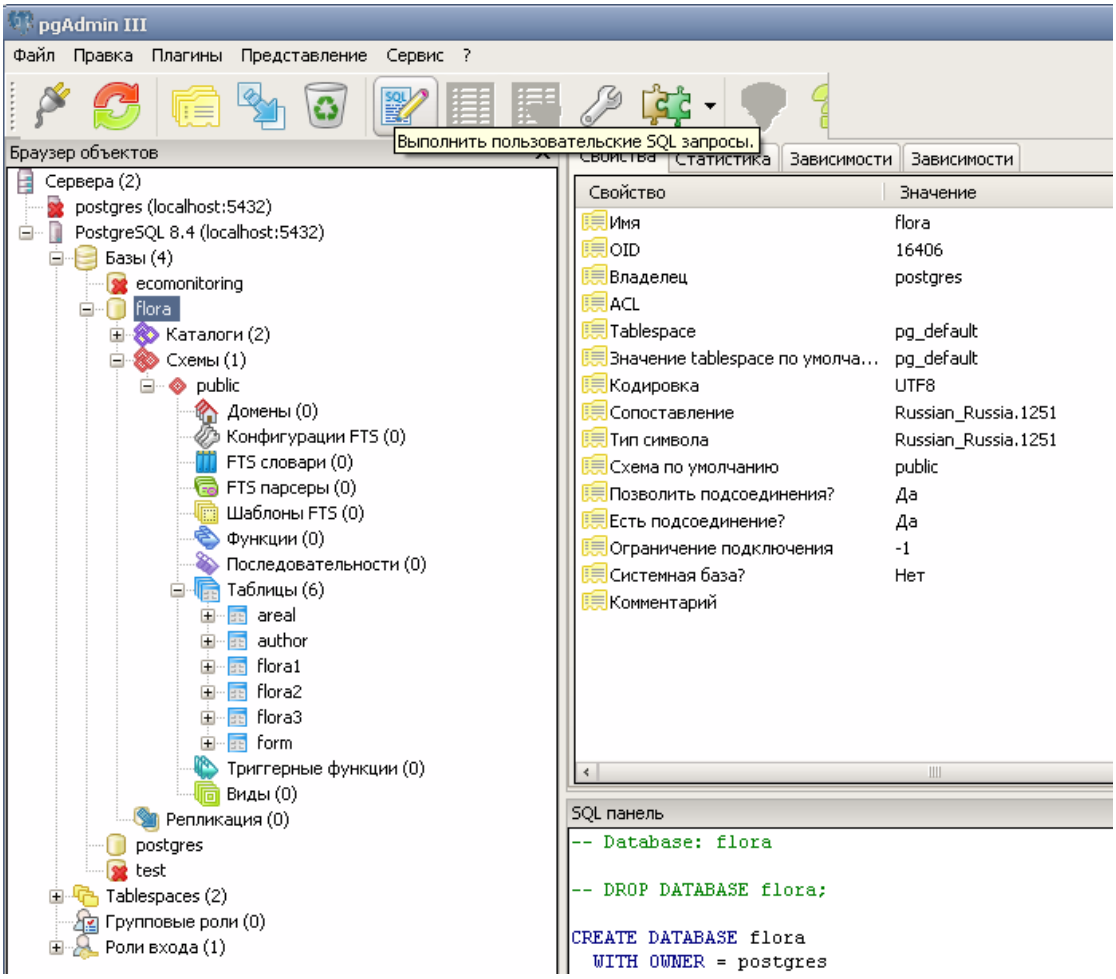


Рис. 2. Список объектов базы данных и их свойства.

После создания базы данных можно просмотреть объекты базы в списке (рис. 2). В правой части окна показаны свойства выбранного объекта, ниже – SQL-запрос, с помощью которого выбранный объект был создан. Таблицы базы данных можно увидеть в списке «Имя базы=>Схемы=>public=>Таблицы». По щелчку на значке рядом с именем таблицы выпадает список объектов таблицы (колонки, ограничения и др.). SQL-запросы можно запускать в специальном окне (рис. 3), которое появляется по щелчку на соответствующей иконке на панели инструментов.

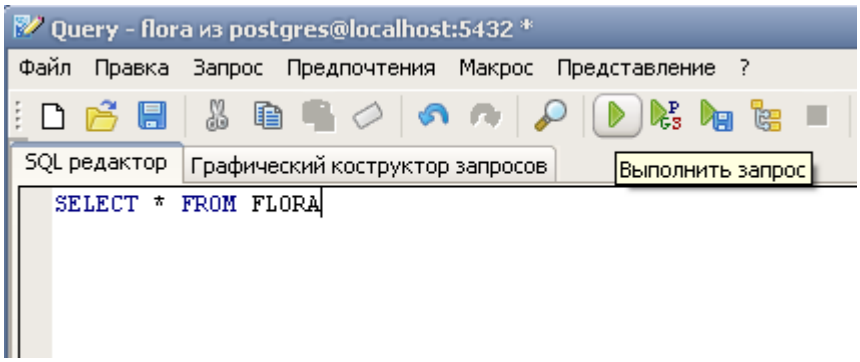


Рис. 3. Окно для редактирования и запуска SQL-запросов.

2. Типы данных в языке SQL. Оператор создания структуры таблицы `CREATE TABLE`. Задание значений по умолчанию, первичного ключа, связи между таблицами.

Теоретический материал

Таблица – базовый элемент реляционной базы данных. Столбцы таблицы называют полями, строки таблицы – записями. В одной базе данных может быть множество таблиц. Первичный ключ – поле или несколько полей, однозначно характеризующих запись таблицы. Создать таблицу можно с помощью следующего оператора SQL

```
CREATE TABLE таблица (
поле1 тип1 спецификатор1,
поле2 тип2 спецификатор2,
…
);
```

`таблица`, `поле1`, `поле2` – название таблицы и названия полей. Они могут состоять из символов латинского алфавита, цифр и знаков подчеркивания “_”, но не должны начинаться с цифры. При этом заглавные и строчные буквы не различаются. Каждое поле имеет определенный тип данных, который определяет множество значений данного поля и множество операций над ним. Тип поля должен соответствовать смыслу хранимых данных. В системе PostgreSQL можно использовать следующие типы

Название типа	Пояснение	Диапазон значений	Примеры значений
integer, int	целое число	-2147483648÷ +2147483647	1, 200, -3, 1000
numeric [(p,s)], decimal [(p,s)]	число с заданной точностью, p – общее количество знаков, s знаков после десятичной точки	до 131072 знаков до десятичной точки, до 16383 знаков после десятичной точки	1.0, 1.511, 3.141592, -10.34
real	число с плавающей точкой	6 знаков после десятичной точки	-1.236712E+01
double precision	число с плавающей точкой	15 знаков после десятичной точки	-5.6918917829E-02
serial	автоинкрементное целое положительное число	1÷2147483647	1, 2, 3, 4, 5
character(n) char(n), varchar(n)	строка из n символов		'abc', 'Иванов И.И.'
text	строка символов произвольной длины		
date	дата		1999-01-08, 1999-Jan-08
time	время	00:00:00÷24:00:00	04:06:06, 04:05
boolean, bool	логический	TRUE или FALSE	't', 'f', '0', '1'

Здесь перечислены только основные типы. Кроме них, система PostgreSQL поддерживает различные специфические типы (геометрические примитивы: точки, полигоны; сетевые адреса и др.).

Спецификаторы задают некоторые свойства полей. Предусмотрены следующие спецификаторы:

DEFAULT – задание значения поля по умолчанию;

CHECK – задание ограничения на значения поля;

NOT NULL – указание на то, что поле не может принимать пустое значение;

UNIQUE – указание на то, что поле может принимать только уникальные значения;

PRIMARY KEY – указание на то, что данное поле является первичным ключом;

REFERENCES – задание внешнего ключа, установка связи между таблицами.

Удалить таблицу можно оператором

```
DROP TABLE таблица
```

Примеры:

1. Создадим базу данных flora (процесс создания базы данных с помощью программы pgAdmin III описан на стр. 6).

2. Создадим таблицу – справочник растительных формаций.

```
CREATE TABLE form (
code numeric(2,0) PRIMARY KEY,
name character(20) NOT NULL
);
```

Таким образом, будет создана таблица `form` со всего двумя полями: `code` – код растительной формации (тип – целое число из двух знаков, первичный ключ таблицы), `name` – название растительной формации (тип – символьная строка из 20 символов, не может принимать пустое значение). Убедитесь, что таблица создана и появилась в списке.

3. Создадим таблицу – справочник типов ареала.

```
CREATE TABLE areal (
code numeric(2,0) PRIMARY KEY,
name character(65) NOT NULL
);
```

4. Создадим таблицу – справочник авторов.

```
CREATE TABLE author (
code numeric(2,0) PRIMARY KEY,
fam character(20) NOT NULL,
ns character(20),
dolzh character(50),
disser character(150)
);
```

В этой таблице уже четыре поля: `code` – код автора, первичный ключ таблицы, `fam` – фамилия автора, `ns` – ученая степень автора, `dolzh` – должность автора, `disser` – тема диссертации. Заметим, что для фамилии автора спецификатор `NOT NULL` имеет смысл, а вот поля `ns` и `dolzh` могут принимать пустые значения, так как у автора ученая степень и должность могут быть неизвестными. Пустое значение – это не пробелы или нули, это неизвестное или не введенное значение.

5. Создадим таблицу точек описаний. В ней будут храниться данные о точках, в которых делались описания растительных видов.

```
CREATE TABLE flora1 (
code numeric(5,0) PRIMARY KEY,
name character(30) NOT NULL,
form numeric(2,0) REFERENCES form (code),
author numeric(2,0) REFERENCES author (code),
date date,
memo text
);
```

В таблице предусмотрены поля: code – код точки описания (первичный ключ таблицы), name – название точки описания (например, «Раифа, выдел 1»), form – код растительной формации данной точки описания, author – код автора, который проводил описания на этой точке. Заметим, что для последних двух полей предусмотрен спецификатор REFERENCES. Таким образом, устанавливается связь этой таблицы с таблицами form и author. Это означает, что в таблицу form нельзя будет добавлять данные с кодами, которых нет в таблицах form и author. Если не сделать это ограничение, то может нарушиться так называемая ссылочная целостность данных. Обратите внимание, что эти правила появились в списке ограничений. Кроме того, в таблице есть поле date – дата описания и memo – примечание. Для примечания тип данных – строка произвольной длины.

6. Создадим таблицу – справочник растительных видов.

```
CREATE TABLE flora3 (
code numeric(4,0) PRIMARY KEY,
name character(40) NOT NULL,
fam character(10),
areal numeric(2,0) REFERENCES areal (code),
status boolean
);
```

В таблице предусмотрены поля: code – код вида (первичный ключ таблицы), name – русское название вида, fam – русское название семейства, areal – код типа ареала (можно вводить только те коды, которые есть в таблице areal), status – статус вида (поле логического типа, возможные значения: false – не занесен в Красную книгу, true – занесен в Красную книгу).

7. Создадим таблицу описаний. В ней будут храниться данные наблюдений – какие растительные виды встречались в точках описаний.

```
CREATE TABLE flora2 (
code_op numeric(5,0) REFERENCES flora1 (code),
code_vid numeric(4,0) REFERENCES flora3 (code),
tier numeric(1,0) CHECK (tier>0 AND tier<6),
dom numeric(1,0)
)
```

В таблице предусмотрены поля: code_op – код точки описаний (берутся из таблицы точек описаний flora1), code_vid – код вида (берутся из справочника видов flora3), tier – ярус (возможные значения: 1 – древостой 1, 2 – древостой 2, 3 – подлесок, 4 – травостой, 5 – мохово-лишайниковый), dom – степень присутствия вида в сообществе (возможные значения: 1 – доминант, 2 – содоминант, 3 – присутствует). Для поля tier сделано ограничение, введенные

значения должны удовлетворять неравенствам. Для поля dom ограничение добавим позже. Убедитесь, что все таблицы присутствуют в списке.

Задания для самостоятельной работы:

1. Создайте базу данных загрязнений атмосферного воздуха ecomonitoring.

2. Создайте таблицу объектов измерений objects со структурой

Имя поля	Тип поля	Описание	Примечание
code	числовой, общее количество знаков 3, после запятой – 0	код объекта	первичный ключ
name	строка из 20 символов	название объекта (город, населенный пункт)	не может принимать пустое значение

3. Создайте таблицу загрязняющих веществ veshestva со структурой

Имя поля	Тип поля	Описание	Примечание
code	числовой, общее количество знаков – 4, после запятой – 0	код вещества	первичный ключ
name	строка из 30 символов	название вещества	не может принимать пустое значение
class	числовой, общее количество знаков – 1, после запятой – 0	класс опасности	ограничение на значение: 1- чрезвычайно опасные; 2 - высоко опасные; 3 - опасные; 4 - умеренно опасные.
pdkmr	числовой, общее количество знаков – 7, после запятой – 5	предельно допустимая концентрация максимально-разовая, мг/ куб. м	
pdkss	числовой, общее количество знаков – 7, после запятой – 5	предельно допустимая концентрация среднесуточная, мг/ куб. м.	
source	числовой, общее количество знаков – 2, после запятой – 0	источник загрязнения по классификации	

4. Создайте таблицу измерений `measurements` со структурой

Имя поля	Тип поля	Описание	Примечание
`code_ob`	числовой, общее количество знаков – 3, после запятой – 0	код объекта	внешний ключ, связан с полем `code` таблицы `objects`
`code_vesh`	числовой, общее количество знаков – 4, после запятой – 0	код вещества	внешний ключ, связан с полем `code` таблицы `veshestva`
`date`	дата	дата измерения	
`conc1`	число с плавающей точкой	максимальная суточная концентрация, мг/ куб. м.	
`conc2`	число с плавающей точкой	среднесуточная концентрация, мг/ куб. м.	

3. Изменение структуры таблицы. Оператор `ALTER TABLE`.

Теоретический материал

Если при создании таблицы с помощью оператора `CREATE TABLE` была допущена ошибка, то можно удалить таблицу и создать ее заново. Однако это может оказаться крайне неудобно, так как в таблице могут быть данные, кроме того, таблица может иметь связь с другой таблицей, т.е. нарушится ссылочная целостность базы данных. В SQL предусмотрен оператор `ALTER TABLE`, с помощью которого можно

- добавить поле;
- удалить поле;
- добавить ограничение на значение поля;
- удалить ограничение на значение поля;
- изменить значение поля по умолчанию;
- изменить тип данных поля;
- переименовать поле;
- переименовать таблицу.

Общий формат оператора выглядит следующим образом.

```
ALTER TABLE таблица действие
```

Примеры:

1. В таблице авторов описаний `author` удалим поле `disser` – тему диссертации.

```
ALTER TABLE author DROP COLUMN disser
```

2. В таблице авторов описаний `author` добавим поля `imia` и `otch` – имя и отчество автора.

```
ALTER TABLE author ADD COLUMN imia character(15);
ALTER TABLE author ADD COLUMN otch character(20)
```

При добавлении в таблицу нового поля, также как и при создании таблицы, могут использоваться спецификаторы `PRIMARY KEY`, `CHECK` и др.

3. В таблице описаний `flora2` добавим ограничение на значение для поля `dom` – степень присутствия вида в сообществе. Оно должно принимать значение 1, 2 или 3.

```
ALTER TABLE flora2 ADD CHECK (dom>0 AND dom<4)
```

4. В таблице видов `flora3` изменим название поля `name` на `rus_name`.

```
ALTER TABLE flora3 RENAME COLUMN name to rus_name
```

5. В таблице видов `flora3` изменим название поля `fam` на `famrus`.

```
ALTER TABLE flora3 RENAME COLUMN fam to famrus
```

6. Для этого же поля увеличим количество символов с 10 до 17.

```
ALTER TABLE flora3 ALTER COLUMN famrus TYPE character(17)
```

С помощью этого оператора также можно изменить тип поля на любой другой. Проверьте названия и типы всех полей базы данных и, если допущены ошибки, исправьте их.

7. Проверьте, правильно ли названы все таблицы и, если допущена ошибка, переименуйте таблицу следующим образом

```
ALTER TABLE имя_таблицы RENAME TO новое_имя
```

Задания для самостоятельной работы:

1. В таблице загрязняющих веществ `veshestva` удалите поле `source` – источник загрязнения по классификации.
2. В таблицу загрязняющих веществ `veshestva` добавьте поле `notation` – обозначение вещества. Тип – строка из 6 символов.
3. В таблице `measurements` измените имена полей `conc1` и `conc2` на `concmax` и `concss` соответственно.

4. В таблице измерений measurements измените тип полей concmax и concss на числовой, общее количество знаков – 7, после запятой – 5.

5. В таблицах veshestva и measurements для полей pdkmr, pdkss, concmax, concss добавьте ограничение: значение должно быть больше или равно нулю (>= 0).

4. Добавление, изменение, удаление данных в таблице. Операторы INSERT, UPDATE, DELETE.

Теоретический материал

После создания таблицы в ней еще нет данных. Добавлять данные можно с помощью оператора INSERT. Общий формат оператора

```
INSERT INTO таблица (поле1, поле2, …) VALUES (значение1, значение2, …)
```

При этом в таблицу можно добавить одну или более записей. Также и данные можно вводить для всех полей (тогда список полей можно не перечислять) или только для некоторых.

Если при вводе данных допущена ошибка или данные нуждаются в изменении, то можно воспользоваться оператором UPDATE. Общий формат оператора

```
UPDATE таблица SET поле1=значение1, поле2=значение2, … WHERE условие
```

Таким образом, для всех записей, удовлетворяющих условию после ключевого слова WHERE, перечисленные поля примут указанные значения. При этом условие может выполняться для нескольких записей, тогда для всех них произойдет изменение данных. При необходимости выполнить изменение только для одной записи, в условии должно быть поле с уникальным значением, например, первичный ключ. Если условие не указывать, то изменение данных выполнится для всех записей.

Для удаления данных предусмотрен оператор DELETE. Его синтаксис подобен синтаксису оператору UPDATE

```
DELETE FROM таблица WHERE условие
```

Будут удалены все записи, для которых истинно условие. Если условие не указывать, будут удалены все записи таблицы.

Примеры:

1. Добавим запись в таблицу form – справочник растительных формаций.

```
INSERT INTO form (code, name) VALUES (1, 'березняк')
```

2. Добавим сразу несколько записей в эту же таблицу одним оператором.

```
INSERT INTO form VALUES
(2, 'дубрава'),
(3, 'степь каменистая'),
(4, 'луг пойменный'),
(5, 'луг остепененный'),
(6, 'болото');
```

3. Добавим несколько записей в таблицу `areal` – справочник типов ареала.

```
INSERT INTO areal VALUES
(1, 'Заносный'),
(2, 'Еврозападноазиатский'),
(3, 'Европейский'),
(4, 'Приатлантический'),
(5, 'Эндемик Поволжья'),
(6, 'Голарктический');
```

4. Добавим данные в таблицу `author` – справочник авторов.

```
INSERT INTO author (code, fam, imia, otch, ns, dolzh) VALUES
(1, 'Прохоров', 'Вадим', 'Евгеньевич', 'к.б.н.', 'доцент'),
(2, 'Галеева', 'Алия', 'Фаридовна', 'д.б.н.', 'профессор'),
(3, 'Айзатова', 'Марина', 'Андреевна', 'к.б.н.',
'ассистент'),
(4, 'Вафина', 'Гульнара', 'Азатовна', NULL, 'лаборант'),
(5, 'студенты', NULL, NULL, NULL, NULL);
```

Обратите внимание, что для некоторых записей вводится значение `NULL` – пустое значение.

5. Добавим данные в таблицу `flora1` – таблица точек описаний.

```
INSERT INTO flora1 VALUES
(1, 'Раифа, выд.1', 4, 1, '2009-Jul-15', 'N55.90947,
E48.70576'),
(2, 'Раифа, выд.2', 3, 5, '2009-Jul-29', 'N55.90846,
E48.71936'),
(3, 'Саралы, выд.1', 6, 5, '2010-Jun-28', 'Саралинский
участок ВКГПБЗ, площадь 0.8 га'),
(4, 'Камское устье', 1, 3, '2010-Jul-05', 'Сев.-вост.
склон'),
(5, 'Высокая гора', 2, 2, '2011-Jun-30', NULL),
(6, 'Саралы, выд.2', 1, 5, '2011-Jul-21', 'Саралинский
участок ВКГПБЗ, северный склон'),
(7, 'Раифа, выд.3', 5, 1, '2011-Aug-01', 'N55.88958,
E48.71877'),
```

```
(8, 'Мензелинский р-н', 3, 3, '2012-Jul-08', NULL),
(9, 'Раифа, выд.4', 1, 5, '2012-Jul-18', NULL),
(10, 'Саралы, выд.3', 3, 2, '2012-Jun-25', 'Саралинский участок ВКГПБЗ, площадь 0.6 га');
```

6. Добавим данные в таблицу `flora3` – справочник видов.

```
INSERT INTO flora3 VALUES
(1, 'Нивяник иркутский', 'Астровые', 1, false),
(2, 'Клевер луговой', 'Бобовые', 2, false),
(3, 'Донник лекарственный', 'Бобовые', 3, false),
(4, 'Береза повислая', 'Березовые', 2, false),
(5, 'Акация желтая', 'Бобовые', 1, false),
(6, 'Лядвенец жигулевский', 'Бобовые', 5, true),
(7, 'Ольха пушистая', 'Березовые', 2, false),
(8, 'Девясил германский', 'Астровые', 2, true),
(9, 'Щавель густой', 'Гречиховые', 3, false),
(10, 'Дуб черешчатый', 'Буковые', 3, false);
```

7. Добавим данные в таблицу `flora2` – таблицу описаний.

```
INSERT INTO flora2 (code_op, code_vid, tier, dom) VALUES
(1,2,4,1), (1,1,4,2), (1,9,4,3), (1,6,4,3),
(2,1,4,1), (2,4,2,1), (2,10,2,2),
(3,9,4,1),
(4,4,3,1), (4,4,1,1), (4,10,3,2),
(5,10,1,1), (5,4,1,3),
(6,4,1,1), (6,5,4,1),
(7,2,4,2), (7,3,4,1), (7,8,4,3),
(8,4,2,1), (8,10,2,2),
(9,4,3,1), (9,2,4,1),
(10,4,2,1), (10,10,2,2), (10,5,4,1);
```

8. Проверим, как работают правила ссылочной целостности. Попробуем добавить в таблицу `flora2` данные с кодами, не представленными в таблицах `flora1` и `flora3`.

```
INSERT INTO flora2 (code_op, code_vid, tier, dom) VALUES
(11,11,4,1)
```

СУБД должна выдать сообщение «`Key (code_op)=(11) is not present in table "flora1"`», что означает отсутствие данных с кодом описания 11 в таблице `flora1`.

9. Изменим в таблице `form` название растительной формации «болото» на «болото низинное» (`code`=6).

```
UPDATE form SET name='болото низинное' WHERE code=6
```

10. Удалим из таблицы `areal` запись с названием типа ареала «Голарктический» (`code`=6), этот тип ареала не используется в базе данных.

```
DELETE FROM areal WHERE code=6
```

11. Проверим, как работают правила ссылочной целостности при удалении данных. Попробуем удалить из таблицы `areal` запись с названием типа ареала «Заносный» (`code`=1).

```
DELETE FROM areal WHERE code=1
```

СУБД должна выдать сообщение «`Key (code)=(1) is still referenced from table "flora3"`», что на запись с указанным типом ареала есть ссылка из таблицы `flora3`, таким образом, ее удалить нельзя.

Задания для самостоятельной работы:

1. Добавьте в таблицу объектов наблюдений `objects` записи.

code	name
1	Казань
30	Набережные Челны
35	Нижнекамск
5	Альметьевск
20	Зеленодольск

2. Добавьте в таблицу веществ `veshestva` записи.

code	name	notation	class	pdkmr	pdkss
330	Серы диоксид	SO2	3	0.5	0.05
337	Углерода оксид	CO	4	4.0	0.7
301	Азота диоксид	NO2	2	0.085	0.04
2909	Пыль неорганическая	PYLN	3	0.5	0.15
303	Аммиак	NH3	4	0.2	0.04
1325	Формальдегид	FORM	2	0.035	0.003
1071	Фенол	FEN	2	0.01	0.003
703	Бенз(а)пирен	BENP	1		0.00001

3. Добавьте в таблицу веществ `measurements` записи.

code_ob	code_vesh	date	concmax	concss
1	337	15.04.2011	2.0	0.8
1	301	15.04.2011	0.5	0.08
1	303	15.04.2011	0.02	0.01
1	2909	15.04.2011	0.4	0.14
30	337	19.04.2011	2.0	0.5
30	301	19.04.2011	0.04	0.02

30	1071	19.04.2011	0.04	0.015
35	337	21.04.2011	1.5	0.3
35	301	21.04.2011	0.05	0.02
35	1325	21.04.2011	0.1	0.024
1	337	20.10.2011	1.1	0.5
1	301	20.10.2011	0.3	0.14
1	303	20.10.2011	0.08	0.02
1	2909	20.10.2011	0.7	0.2
30	337	24.10.2011	2.1	0.6
30	301	24.10.2011	0.05	0.03
30	1071	24.10.2011	0.01	0.009
35	337	26.10.2011	1.2	0.4
35	301	26.10.2011	0.06	0.02
35	1325	26.10.2011	0.06	0.01
1	337	10.04.2012	0.4	0.2
1	301	10.04.2012	0.32	0.15
1	303	10.04.2012	0.03	0.01
1	2909	10.04.2012	0.6	0.18
30	337	12.04.2012	1.8	0.5
30	301	12.04.2012	0.1	0.07
30	1071	12.04.2012	0.015	0.01
35	337	17.04.2012	2.0	0.8
35	301	17.04.2012	0.08	0.03
35	1325	17.04.2012	0.03	0.004
1	337	21.10.2012	0.5	0.2
1	301	21.10.2012	0.26	0.12
1	303	21.10.2012	0.04	0.01
1	2909	21.10.2012	0.2	0.1
30	337	25.10.2012	1.7	0.5
30	301	25.10.2012	0.06	0.04
30	1071	25.10.2012	0.009	0.006
35	337	29.10.2012	1.7	0.6
35	301	29.10.2012	0.07	0.02
35	1325	29.10.2012	0.12	0.031

4. Проверьте базу данных на ссылочную целостность. Попробуйте добавить в таблицу измерений запись с неизвестными кодами и удалить из таблицы объектов или веществ данные, используемые в третьей таблице.

5. Индексирование таблиц. Операторы `CREATE INDEX`, `DROP INDEX`.

Теоретический материал

Поиск данных – одна из самых востребованных операций с базами данных. Для улучшения производительности поиска осуществляется индексирование полей таблиц, по которым наиболее часто ищутся данные. Индекс представляет собой упорядоченный набор значений индексированного поля со ссылками на физическое размещение записей в исходной таблице. Индекс является самостоятельным объектом базы данных и хранится в отдельном файле. Общий формат оператора создания индекса

```
CREATE INDEX имя_индекса ON таблица (поле)
```

После имени поля могут быть указаны ключевые слова `ASC` или `DESC`, означающие сортировку по возрастанию или убыванию. По умолчанию происходит сортировка по возрастанию. Таким образом, поиск данных по индексированному полю в условии `WHERE` в запросе `SELECT` (стр. 17) будет производиться оптимальным методом.

В операторе может быть указан тип структуры индекса. По умолчанию создается наиболее универсальная структура – B-дерево. Индекс может быть составным, т.е. содержать результат сортировки по нескольким полям сразу. Также вместо имени поля в операторе создания индекса может быть указано некоторое выражение, примененное к полям. Например, соединение двух строковых полей.

Создавать индексы желательно перед добавлением данных в таблицу, так как сортировка больших таблиц требует серьезных временных затрат. Для полей, являющихся первичным ключом таблицы, т.е. созданных со спецификатором `PRIMARY KEY`, индекс создается автоматически. Одна таблица может иметь несколько индексов, но при операциях добавления, изменения и удаления данных обновляются и сами индексы, что приводит к замедлению этих операций. Поэтому рекомендуется индексировать только наиболее часто используемые в поиске поля.

Удалить индекс можно командой

```
DROP INDEX имя_индекса
```

Примеры:

1. Создадим индекс по названию растительной формации в таблице `form`.

```
CREATE INDEX form_name ON form (name)
```

Убедитесь, что индекс появился в списке.

2. Создадим индекс по названию типа ареала в таблице `areal`, но в порядке убывания (в данном случае порядок, обратный алфавитному).

```
CREATE INDEX areal_name ON areal (name DESC)
```

3. Создадим индекс по фамилии, имени и отчеству автора в таблице author.

```
CREATE INDEX author_fio ON author (fam, imia, otch)
```

4. Создадим индекс в таблице flora1 по названию точки описания.

```
CREATE INDEX flora1_name ON flora1 (name)
```

Поиск будет проводиться, в основном, по названию точки описания. Для поддержки ссылочной целостности можно проиндексировать код растительной формации и код автора. Дело в том, что при удалении или изменении данных в справочниках растительных формаций и авторов будет происходить проверка, не ссылаются ли на удаленные записи строки в таблице flora1. И для ускорения поиска при проверке указанные коды могут быть проиндексированы. Однако это может замедлить операции с записями в таблице flora1.

5. Создадим индексы в таблице flora3 по названию растительного вида и названию семейства.

```
CREATE INDEX flora3_rus_name ON flora3 (rus_name);
CREATE INDEX flora3_famrus ON flora3 (famrus)
```

6. Создадим индексы в таблице flora2 по коду точки описания и коду вида.

```
CREATE INDEX flora2_code_op ON flora2 (code_op);
CREATE INDEX flora2_code_vid ON flora2 (code_vid)
```

Таблица описаний является самой объемной из всех. Чаще всего поиск будет производиться по коду точки описания и коду вида.

7. Удалите индексы в таблицах form и areal. Данных в этих таблицах обычно мало, и при поиске можно обойтись и без индексов.

```
DROP INDEX form_name;
DROP INDEX areal_name
```

Задания для самостоятельной работы

1. Создайте индекс в таблице объектов наблюдений по названию объекта.

2. Создайте индекс в таблице загрязняющих веществ по названию вещества.

3. Создайте индекс в таблице загрязняющих веществ по классу опасности по убыванию.

4. Создайте индексы в таблице измерений по коду объекта и коду загрязняющего вещества.

6. Оператор выборки SELECT. Простые выборки. Исключение повторений. Сортировка записей.

Теоретический материал

SELECT – один из наиболее часто используемых операторов языка SQL. Он позволяет получать выборки из базы данных и преобразовывать их к нужному виду. В рамках одного оператора могут быть решены многие задачи анализа данных. В простом виде формат оператора выглядит следующим образом

```
SELECT поле1, поле2, …
FROM таблица
ORDER BY поле_сортировки
```

После SELECT указывается либо список полей, либо, при необходимости вывода всех полей, символ *. Также можно выводить значения выражений, используя операции над полями (арифметические операции над числовыми полями, конкатенация строк и т.п.). В полученной выборке могут присутствовать повторяющиеся записи. Для их исключения перед списком полей нужно указать ключевое слово DISTINCT. Однако использование DISTINCT может серьезно замедлить работу запроса.

В списке полей могут быть столбцы из разных таблиц, в этом случае таблицы должны быть указаны после FROM и имена полей должны уточняться именами таблиц: Имя_таблицы.имя_поля. При запросе к двум и более таблицам строится так называемое декартово произведение таблиц – каждая запись одной таблицы комбинируется с каждой записью другой таблицы.

В общем случае записи результата запроса никак не упорядочены. SELECT-запрос может заканчиваться ключевым словом ORDER BY, после которого указывается поле, по которому нужно провести сортировку. Способ упорядочивания задается параметрами ASC (по возрастанию, вариант по умолчанию) или DESC (по убыванию). Если поле, по которому проводится сортировка, проиндексировано, то индекс используется при построении результирующей выборки.

Примеры:

1. Выведем все данные из таблицы flora3.

```
SELECT * FROM flora3
```

2. Выведем список названий видов и семейств из таблицы flora3.

```
SELECT rus_name, famrus FROM flora3
```

3. Выведем список названий семейств из таблицы flora3.

```
SELECT famrus FROM flora3
```

В полученном списке существуют повторяющиеся записи, так как различные виды могут относиться к одному семейству.

4. Выведем список названий семейств из таблицы flora3, исключив дублирование данных.

```
SELECT DISTINCT famrus FROM flora3
```

5. Выведем список авторов, их научных степеней и должностей.

```
SELECT fam, imia, otch, ns, dolzh
FROM author
```

6. Выведем список авторов и их научных степеней и должностей, объединив фамилию, имя и отчество в одно поле и упорядочив по нему.

```
SELECT fam || ' ' || imia || ' ' || otch as fio, ns, dolzh
FROM author
ORDER BY fio
```

Поля fam, imia, otch соединяются в одну строку с помощью операции конкатенации строк ||. Дав полю псевдоним fio, можно провести по нему сортировку, включив его в ORDER BY. Заметим, что в результате запроса нет записи «студенты». Для этой записи поля imia и otch имеют значения NULL, а соединение строки и NULL дает NULL. Эту проблему можно решить, применив логические операторы, либо, в случае неизвестных имени и отчества, вводить пробел « », а не оставлять NULL.

7. Выведем список авторов и их научных степеней и должностей, объединив фамилию и инициалы в одно поле и упорядочив по нему.

```
SELECT fam || ' ' || substring(imia from 1 for 1) || '. ' ||
       substring(otch from 1 for 1) || '.' as fio, ns, dolzh
FROM author
ORDER BY fio
```

Для того, чтобы преобразовать имя и отчество в инициалы, используется функция substring(строка from n to m), возвращающая подстроку строки-параметра с n-го символа по m-й символ.

8. Выведем названия растительных формаций и типов ареала из таблиц form и areal.

```
SELECT form.name, areal.name
FROM form, areal
```

В результате получено достаточно большое количество записей – каждая запись первой таблицей комбинируется с каждой записью второй таблицы. Такая выборка называется декартовым произведением двух таблиц.

Задания для самостоятельной работы

1. Вывести все данные из таблицы загрязняющих веществ.

2. Вывести классы опасности представленных в таблице веществ, исключив повторяющиеся записи.

3. Вывести список названий объектов мониторинга, упорядочив по алфавиту.

4. Вывести список названий, обозначений и среднесуточную ПДК загрязняющих веществ, упорядочив по убыванию ПДК.

5. Вывести список кодов объектов, кодов веществ, дат измерений, среднесуточную концентрацию, упорядочив по возрастанию коду объекта, убыванию кода вещества.

7. Выборка по условию WHERE.

Теоретический материал

С помощью оператора SELECT можно получить записи, удовлетворяющие некоторому условию. Для этого нужно добавить в текст запроса ключевое слово WHERE. В общем виде запрос с условием выглядит следующим образом

```
SELECT поле1, поле2, …
FROM таблица
WHERE условие
ORDER BY поле_сортировки
```

В качестве условия могут быть:

- простые условия типа поле>значение. При этом можно использовать операции сравнения > (больше), < (меньше), >= (больше или равно), <= (меньше или равно), = (проверка на равенство) и <> (проверка на неравенство).
- составные условия типа поле1>значение1 AND поле2<значение2. При этом можно использовать логические операции AND (логическое И), OR (логическое ИЛИ), NOT (логическое отрицание).
- проверка на диапазон поле BETWEEN значение1 AND значение2. Проверяется, попадает ли значение поля в указанный диапазон. Границы диапазона (значение1 и значение2) при этом включаются.
- принадлежность множеству поле IN (значение1, значение2,…). Проверяется, принадлежит ли значение поля перечисленному множеству.
- проверка на неизвестное значение поле IS NULL или поле IS NOT NULL. Проверяется, содержит ли столбец значение NULL.

- сравнение с шаблоном `поле LIKE шаблон`. Проверяется, соответствует ли значение поля шаблону.

В шаблоне могут присутствовать следующие символы:

`%` – вместо символа может быть любое количество произвольных символов;

_ – вместо символа будет подставлен один произвольный символ.

Примеры шаблонов:

Шаблон	Пояснение	Выражения, соответствующие шаблону
`'%ра%'`	строки, в которых есть слог «ра»	`'Саралы','Высокая гора'`
`'__ра%'`	строки, в которых третий и четвертый символы «р» и «а»	`'Саралы'`
`'%Ра%'`	строки, в которых есть слог «Ра»	`'Раифа'`

Эти два символа также могут быть включены в шаблон как значащие символы, в этом случае перед ними нужно поставить две обратные косые черты (`'\\%'`, `'\\_'`).

Примеры:

1. Выведем названия растительных видов, занесенных в Красную книгу.

```
SELECT rus_name
FROM flora3
WHERE status=TRUE
```

2. Выведем названия растительных видов, относящихся к семейству «Бобовые».

```
SELECT rus_name, famrus
FROM flora3
WHERE famrus='Бобовые'
```

3. Выведем названия растительных видов, не относящихся к семейству «Бобовые». Упорядочим по названию семейства.

```
SELECT rus_name, famrus
FROM flora3
WHERE famrus<>'Бобовые'
ORDER BY famrus
```

4. Выведем названия растительных видов, относящихся к семейству «Астровые» или «Бобовые». Упорядочим по названию семейства.

```
SELECT rus_name, famrus
FROM flora3
```

```
WHERE famrus IN ('Астровые','Бобовые')
ORDER BY famrus
```

5. Выведем названия, даты и примечания для описаний, проводившихся в 2011 году.

```
SELECT name, date, memo
FROM flora1
WHERE date>='2011-Jan-01' AND date<='2011-Dec-31'
```

6. Выполним тот же запрос, используя ключевое слово BETWEEN. Упорядочим по названию точки описания.

```
SELECT name, date, memo
FROM flora1
WHERE date BETWEEN '2011-Jan-01' AND '2011-Dec-31'
ORDER BY name
```

7. Выведем названия, даты и примечания для описаний, проводившихся в любом году, кроме 2011-го.

```
SELECT name, date, memo
FROM flora1
WHERE date<'2011-Jan-01' OR date>'2011-Dec-31'
```

8. Выполним тот же запрос, используя ключевое слово BETWEEN. Упорядочим по названию точки описания.

```
SELECT name, date, memo
FROM flora1
WHERE date NOT BETWEEN '2011-Jan-01' AND '2011-Dec-31'
ORDER BY name
```

9. Выведем названия, даты и примечания для описаний, у которых в названии есть слог «ра». Упорядочим по названию точки описания.

```
SELECT name, date, memo
FROM flora1
WHERE name LIKE '%pa%'
ORDER BY name
```

10. Выведем названия, даты и примечания для описаний, у которых в названии есть слог «ра» или «Ра». Упорядочим по названию точки описания.

```
SELECT name, date, memo
FROM flora1
WHERE name LIKE '%pa%' OR name LIKE '%Pa%'
ORDER BY name
```

11. Выведем фамилии, имена и отчества авторов, у которых ученая степень неизвестна.

```
SELECT fam, imia, otch, ns
FROM author
WHERE ns IS NULL
```

12. Выведем фамилии, имена и отчества авторов, у которых ученая степень известна.

```
SELECT fam, imia, otch, ns
FROM author
WHERE ns IS NOT NULL
```

Задания для самостоятельной работы

1. Вывести названия, максимально-разовую и среднесуточную ПДК веществ второго класса опасности.

2. Вывести названия, максимально-разовую и среднесуточную ПДК веществ первого и второго класса опасности. Использовать ключевое слово `IN`.

3. Вывести данные измерений в 2011 году.

4. Вывести название, максимально-разовую и среднесуточную ПДК для веществ, у которых в формуле есть кислород.

5. Вывести названия веществ, у которых неизвестно максимально-разовое значение ПДК.

8. Соединение таблиц. Внутреннее, левое и правое внешние соединения `INNER JOIN`, `LEFT OUTER JOIN`, `RIGHT OUTER JOIN`.

Теоретический материал

Соединение таблиц – возможность на основе двух и более таблиц получать одну. В реляционных базах данных таблицы бывают связаны по ключевым полям. С помощью ключей обычно и производится соединение таблиц. В SQL можно выполнять следующие соединения.

```
SELECT таблица1.поле1, таблица2.поле2
FROM таблица1 INNER JOIN таблица2 ON условие_соединения
```

Естественное (внутреннее) соединение. Каждая запись первой таблицы соединяется с каждой записью второй таблицы, но в результате выдаются только те записи, для которых удовлетворяется условие соединения. Обычно в условиях присутствуют внешние и первичные ключи. Использование ключей также ускоряет работу запроса.

```
SELECT таблица1.поле1, таблица2.поле2
FROM таблица1 LEFT OUTER JOIN таблица2
ON условие_соединения
```

Левое внешнее соединение. Каждая запись первой таблицы соединяется с каждой записью второй таблицы, но в результате выдаются записи, для которых удовлетворяется условие соединения, и все остальные записи первой таблицы, не имеющие ни одного совпадения по условию соединения.

```
SELECT таблица1.поле1, таблица2.поле2
FROM таблица1 RIGHT OUTER JOIN таблица2
ON условие_соединения
```

Правое внешнее соединение. Каждая запись первой таблицы соединяется с каждой записью второй таблицы, но в результате выдаются записи, для которых удовлетворяется условие соединения, и все остальные записи второй таблицы, не имеющие ни одного совпадения по условию соединения.

Соединять можно более двух таблиц. В этом случае лучше отделять блоки `JOIN` скобками, чтобы установить порядок соединения.

Примеры:

1. Выведем список названий точек описаний и соответствующих растительных формаций.

```
SELECT flora1.name, form.name
FROM flora1 INNER JOIN form ON flora1.form=form.code
```

2. Выведем список названий растительных видов и соответствующих типов ареала. Типы ареала, для которых нет видов в справочнике, также должны быть в списке.

```
SELECT flora3.rus_name, areal.name
FROM flora3 RIGHT OUTER JOIN areal
ON flora3.areal=areal.code
```

Как видно из результата, выведены все записи, удовлетворяющие условию соединения, а также запись из второй таблицы, не имеющая ни одного совпадения по условию, т.е. приведен тип ареала, для которого нет данных в справочнике растительных видов.

3. Выведем список всех авторов и точек описаний, на которых они работали. Авторы, не проводившие описаний, также должны быть в списке.

```
SELECT author.fam, author.imia, flora1.name
FROM author LEFT OUTER JOIN flora1
ON author.code=flora1.author
```

Выведены все записи, удовлетворяющие условию соединения, а также запись из первой таблицы, не имеющая ни одного совпадения по условию, т.е. приведен автор, для которого нет данных в таблице описаний.

4. Выведем данные описаний в следующем виде: название точки описания, название вида, ярус, степень присутствия вида в сообществе.

```
SELECT flora1.name, flora3.rus_name, tier, dom
FROM (flora1 INNER JOIN flora2
ON flora1.code=flora2.code_op) INNER JOIN flora3
ON flora2.code_vid=flora3.code
```

Таким образом, используя внешние ключи `flora2`, в запросе соединяются три таблицы.

5. Выведем названия точек и даты описаний, относящихся к растительной формации «березняк».

```
SELECT f1.name, f1.date, form.name
FROM flora1 as f1 INNER JOIN form ON f1.form=form.code
WHERE form.name='березняк'
```

Для сокращения записи для таблицы `flora1` вводится псевдоним `f1`.

6. Выведем названия точек и даты описаний, которые проводили студенты. Упорядочим по названию точки описания.

```
SELECT f1.name, f1.date, author.fam
FROM flora1 as f1 INNER JOIN author ON f1.author=author.code
WHERE author.fam='студенты'
ORDER BY f1.name
```

7. Выведем данные описаний, проводившихся в Раифе, в следующем виде: название точки описания, дата описания, название вида, ярус, степень присутствия вида в сообществе.

```
SELECT f1.name, f1.date, f3.rus_name, tier, dom
FROM (flora1 as f1 INNER JOIN flora2 as f2
ON f1.code=f2.code_op)
INNER JOIN flora3 as f3 ON f2.code_vid=f3.code
WHERE f1.name LIKE 'Раифа%'
```

8. Выведем данные описаний видов, занесенных в Красную книгу, в следующем виде: название точки описания, дата описания, название вида.

```
SELECT f1.name, f1.date, f3.rus_name
FROM (flora1 as f1 INNER JOIN flora2 as f2
ON f1.code=f2.code_op)
INNER JOIN flora3 as f3 ON f2.code_vid=f3.code
WHERE f3.status=true
```

9. Выведем данные описаний, проводившихся Прохоровым в 2009 году, в следующем виде: название точки описания, фамилия автора, название вида, ярус, степень присутствия вида в сообществе.

```
SELECT f1.name, f1.date, f3.rus_name, tier, dom
FROM ((flora1 as f1 INNER JOIN author
ON f1.author=author.code)
INNER JOIN flora2 as f2 ON f1.code=f2.code_op)
INNER JOIN flora3 as f3 ON f2.code_vid=f3.code
WHERE author.fam='Прохоров' AND f1.date BETWEEN
                              '2009-Jan-01' AND '2009-Dec-31'
```

В данном примере необходимо соединение четырех таблиц, так как данные по авторам содержатся в таблице `author`.

Задания для самостоятельной работы:

1. Вывести список объектов и даты измерений на этих объектах. Исключить повторяющиеся записи.

2. Вывести список всех объектов и даты измерений на этих объектах. Исключить повторяющиеся записи.

3. Вывести данные мониторинга загрязнений в следующем виде: название объекта, название вещества, дата измерения, максимальная концентрация, среднесуточная концентрация.

4. Вывести данные мониторинга загрязнений для случаев превышения среднесуточной концентрации над среднесуточной ПДК в следующем виде: название объекта, название вещества, дата измерения, среднесуточная концентрация / среднесуточная ПДК. Упорядочить по названию объекта и дате измерения по возрастанию.

5. Вывести данные мониторинга по Казани для случаев более чем трехкратного превышения максимальной суточной концентрации над максимально-разовой ПДК в следующем виде: название объекта, название вещества, дата измерения, максимальная суточная концентрация / максимально-разовая ПДК. Упорядочить по значению отношения концентраций по убыванию.

9. Использование агрегатных функций. Группировка `GROUP BY`.

Теоретический материал

По всей таблице с помощью агрегатных функций можно проводить некоторые обобщающие вычисления. Общий формат использования агрегатных функций

```
SELECT функция(выражение) FROM таблица
```

В качестве выражения обычно бывает имя поля, но может быть арифметическое выражение. Агрегатные функции:

SUM() – вычисляется сумма значений в столбце;

AVG() – вычисляется среднее арифметическое значений в столбце;

MAX() – находится наибольшее значение в столбце;

MIN() – находится наименьшее значение в столбце;

COUNT() – вычисляется количество значений в столбце.

Функции SUM(), AVG() применимы только к числовым полям. При вычислении функций игнорируются пустые значения (NULL), кроме использования COUNT(*). В этом случае подсчитываются все записи таблицы, в том числе и пустые. В столбце могут быть повторяющиеся значения. Чтобы функция их не учитывала, нужно использовать DISTINCT.

Агрегатные функции часто применяются вместе с группировкой с помощью GROUP BY.

```
SELECT поле1, функция(поле2)
FROM таблица
GROUP BY поле1
```

В этом случае система группирует записи с одинаковыми значениями поля и вычисляет функцию для каждой группы. Ключевой фразой в запросе для использования GROUP BY является фраза «для каждого». Результат запроса может быть отфильтрован с помощью HAVING.

```
SELECT поле1, функция(поле2)
FROM таблица
GROUP BY поле1
HAVING условие
```

Отличие HAVING от WHERE состоит в том, что WHERE отбирает записи до группировки, а HAVING – после, т.е. в последнем случае выдаются группы, удовлетворяющие условию. Это может быть условие, содержащее имя поля, по которому проводится группировка, или условие, содержащее какую-либо агрегатную функцию.

Примеры:

1. Выведем дату последнего описания.

```
SELECT MAX(date) FROM flora1
```

2. Выведем количество записей в справочнике видов.

```
SELECT COUNT(*) FROM flora3
```

3. Выведем количество семейств, по которым представлены виды в справочнике.

```
SELECT COUNT(DISTINCT famrus) FROM flora3
```

Заметим, что использование DISTINCT в данном случае является обязательным, так как разные виды могут относиться к одному семейству, и без DISTINCT мы получим количество видов, а не семейств.

4. Для каждого семейства выведем количество видов этого семейства в справочнике.

```
SELECT famrus, COUNT(code)
FROM flora3
GROUP BY famrus
```

5. Для каждого вида выведем количество описаний в таблице flora2.

```
SELECT flora3.rus_name, COUNT(flora2.code_vid)
FROM flora2 INNER JOIN flora3 ON flora2.code_vid=flora3.code
GROUP BY flora3.rus_name
```

6. Выведем названия видов, для которых больше двух описаний в таблице flora2.

```
SELECT flora3.rus_name, COUNT(flora2.code_vid)
FROM flora2 INNER JOIN flora3 ON flora2.code_vid=flora3.code
GROUP BY flora3.rus_name
HAVING COUNT(flora2.code_vid)>2
```

7. Выведем данные описаний для вида «Береза повислая» в следующем формате: название вида, ярус, степень присутствия в сообществе, количество описаний.

```
SELECT f3.rus_name, f2.tier, f2.dom, COUNT(f2.code_vid)
FROM flora2 as f2 INNER JOIN flora3 as f3
ON f2.code_vid=f3.code
WHERE f3.rus_name LIKE 'Береза%'
GROUP BY f3.rus_name, f2.tier, f2.dom
```

Таким образом, можно судить, в каком ярусе и с какой степенью присутствия в сообществе чаще всего встречается данный вид.

8. Для каждого автора выведем количество сделанных им описаний. Упорядочим по количеству описаний по убыванию.

```
SELECT author.fam, COUNT(flora2.code_vid) AS C
FROM (author INNER JOIN flora1 ON author.code=flora1.author)
INNER JOIN flora2 ON flora1.code=flora2.code_op
GROUP BY author.fam
ORDER BY C DESC
```

Чтобы упорядочить записи по результату агрегатной функции, ей дается псевдоним C, однако подобный запрос возможен и без использования псевдонима.

Задания для самостоятельной работы

1. Вывести количество записей в таблице измерений.

2. Для каждого класса опасности вывести количество загрязняющих веществ.

3. Вывести наибольшее значение максимальной суточной концентрации диоксида азота в Казани.

4. Рассчитать для Казани среднее отношение среднесуточной концентрации загрязняющих веществ к среднесуточной ПДК.

5. Рассчитать для каждого города среднее отношение среднесуточной концентрации загрязняющих веществ к среднесуточной ПДК. Упорядочить по значению этого отношения.

6. Вывести для каждого города названия веществ, для которых среднее отношение среднесуточной концентрации загрязняющих веществ к среднесуточной ПДК больше или равно 1.

10. Использование подзапросов.

Теоретический материал

Многие задачи невозможно решить только с помощью одного оператора SELECT. Например, если какие-то величины в запросе еще не известны до выполнения запроса. В этом случае используются подзапросы. Обычно, это подзапросы в условиях WHERE и HAVING.

```
SELECT поле1, поле2,…
FROM таблица
WHERE поле операция_сравнения подзапрос
```

Подзапрос составляется как обычный запрос SELECT, за исключением того, что в нем нельзя применять сортировку ORDER BY. В подзапросе можно использовать имена полей таблицы внешнего запроса (с указанием таблицы).

Подзапрос может возвращать единственное значение или же множество значений. В последнем случае применяются операции IN, NOT IN, ALL, ANY, EXISTS, NOT EXISTS.

WHERE выражение IN подзапрос (WHERE выражение NOT IN подзапрос)

HAVING выражение IN подзапрос (HAVING выражение NOT IN подзапрос)

Получаются записи, для которых выражение входит (или не входит для NOT IN) во множество, возвращаемое подзапросом.

```
WHERE выражение операция_сравнения ALL подзапрос
WHERE выражение операция_сравнения ANY подзапрос
HAVING выражение операция_сравнения ALL подзапрос
HAVING выражение операция_сравнения ANY подзапрос
```

В случае ALL получаются записи, для которых условие выполняется для всех значений, возвращаемых подзапросом. В случае ANY получаются записи, для которых условие выполняется хотя бы для одного значения, возвращаемого подзапросом. Если подзапрос возвращает пустое значение, то для ALL условие считается выполненным, для ANY – не выполненным.

```
WHERE EXISTS подзапрос (WHERE NOT EXISTS подзапрос)
HAVING EXISTS подзапрос (HAVING NOT EXISTS подзапрос)
```

В этих случаях в условиях не используются какие-либо выражения. Запрос обрабатывается в зависимости от результата конструкции EXISTS подзапрос. EXISTS возвращает TRUE, если результатом подзапроса является хотя бы одна строка, и FALSE, если в результате подзапроса нет ни одного значения. NOT EXISTS возвращает TRUE и FALSE в противоположных EXISTS случаях.

Примеры:

1. Выведем название точки описания, последней по дате.

```
SELECT name, date
FROM flora1
WHERE date=(SELECT max(date) FROM flora1)
```

Эту задачу в рамках стандарта SQL невозможно решить без подзапроса, так как дата последнего описания неизвестна до выполнения запроса.

2. Выведем названия точек описаний, проводившихся в течение месяца до последнего описания.

```
SELECT name, date
FROM flora1
WHERE date>(SELECT max(date) FROM flora1)-30
```

3. Выведем названия точек и авторов для описаний, проводившихся в течение месяца до последнего описания.

```
SELECT flora1.name, author.fam, flora1.date
FROM flora1 INNER JOIN author ON flora1.author=author.code
WHERE flora1.date>(SELECT max(flora1.date) FROM flora1)-30
```

4. Выведем названия видов, которые отсутствуют в таблице описаний (т.е. виды, которые на данный момент нигде не встречались).

```
SELECT rus_name
FROM flora3
WHERE code NOT IN (SELECT code_vid FROM flora2)
```

5. Выведем названия видов, которые встречаются в Раифе и не встречаются в Саралах.

```
SELECT rus_name
FROM flora3
WHERE code IN (SELECT code_vid
FROM flora2 INNER JOIN flora1
ON flora1.code=flora2.code_op
WHERE flora1.name LIKE 'Раифа%')
AND code NOT IN (SELECT code_vid
FROM flora2 INNER JOIN flora1
ON flora1.code=flora2.code_op
WHERE flora1.name LIKE 'Саралы%')
```

Здесь делаются два подзапроса. В первом составляется множество видов, встречающихся в Раифе, во втором – множество видов, встречающихся в Саралах. Все виды, попадающие в первое множество и не попадающие во второе, удовлетворяют условию задания, т.е. встречаются в Раифе и не встречаются в Саралах.

6. Выведем названия видов, которые встречаются только в Раифе.

```
SELECT flora3.rus_name, flora1.name
FROM (flora1 INNER JOIN flora2
ON flora1.code=flora2.code_op)
INNER JOIN flora3 ON flora2.code_vid=flora3.code
WHERE flora3.rus_name NOT IN
(SELECT flora3.rus_name
FROM (flora1 INNER JOIN flora2
ON flora1.code=flora2.code_op)
INNER JOIN flora3 ON flora2.code_vid=flora3.code
WHERE flora1.name NOT LIKE 'Раифа%')
```

Заметим, что список видов, встречающихся в Раифе, получается без использования подзапроса. Однако слово «только» в задании требует сделать подзапрос. Ведь виды, встречающиеся в Раифе, могут встречаться и в других местах. В подзапросе составляется множество названий видов из всех остальных точек (не в Раифе). Таким образом, любой вид из таблицы описаний, не попадающий в это множество, встречается только в Раифе. Это и проверяется в условии внешнего подзапроса.

7. Выведем названия вида, который встречается чаще всего.

```
SELECT flora3.rus_name, COUNT(flora2.code_vid)
FROM flora2 INNER JOIN flora3 ON flora2.code_vid=flora3.code
GROUP BY flora3.rus_name
HAVING COUNT(flora2.code_vid)>= ALL
(SELECT COUNT(flora2.code_vid)
FROM flora2 INNER JOIN flora3 ON flora2.code_vid=flora3.code
GROUP BY flora3.rus_name)
```

В подзапросе для каждого вида рассчитывается количество сделанных описаний. Во внешнем запросе делается то же самое, но выводится только тот вид, для которого количество описаний равно или больше всех значений из множества, возвращаемого подзапросом. Это и будет вид, который встречается чаще всего.

8. Выведем фамилии, имена и отчества авторов, которые проводили описания (т.е. коды которых присутствуют в таблице точек описаний).

```
SELECT fam, imia, otch
FROM author
WHERE EXISTS (SELECT author
FROM flora1
WHERE author.code=flora1.author)
```

Таким образом, получается список авторов, для которых существует хотя бы одно совпадение кода с кодами авторов из таблицы точек описаний. Эта задача также может быть решена с помощью `IN`.

9. Аналогично выведем фамилии, имена и отчества авторов, которые не провели ни одного описания (т.е. коды которых отсутствуют в таблице точек описаний).

```
SELECT fam, imia, otch
FROM author
WHERE NOT EXISTS (SELECT author
FROM flora1
WHERE author.code=flora1.author)
```

Задания для самостоятельной работы

1. Вывести названия вещества с наименьшим среднесуточным ПДК.

2. Вывести названия объектов, по которым нет данных измерений.

3. Вывести названия веществ, по которым измерения проводились только в Казани.

4. Вывести название объекта, для которого среднее отношение среднесуточной концентрации загрязняющих веществ к среднесуточной ПДК наибольшее.

5. Для каждого объекта рассчитать индекс загрязнения атмосферы по формуле $\sum \frac{c_i}{\text{PDK}_i}$, где c_i – средняя концентрация i-го вещества за период.

6. Вывести названия веществ, по которым не проводились измерения. Использовать ключевое слово `EXISTS`.

ЛИТЕРАТУРА

1. Перечень и коды веществ, загрязняющих атмосферный воздух, СПб, НИИ охраны атмосферного воздуха, 2005 г.
2. Полякова Л.Н. Основы SQL. [Электронный ресурс]/ Л.Н. Полякова. – Режим доступа: http://www.intuit.ru/department/database/sql/. Дата обращения: 1.03.2018.
3. Прохоров В.Е. Информационная система «Флора»: учебно-методическое пособие / В.Е. Прохоров. – Казань: Изд-во Казан. гос. ун-та, 2009. -40 с.
4. Уорсли Дж. PostgreSQL. Для профессионалов / Дж. Уорсли, Дж. Дрейк. – СПб.: Питер, 2003. – 496 с.
5. Хернандес М.Дж. SQL-запросы для простых смертных: практическое руководство по манипулированию данными в SQL / М.Дж. Хернандес, Дж. Л. Вьескас. – М.: Изд-во «Лори», 2003. – 458 с.
6. PostgreSQL 9.1.3 Documentation. [Электронный ресурс]/ The PostgreSQL Global Development Group. – Режим доступа: http://www.postgresql.org/docs/9.2/interactive/index.html. Дата обращения: 1.03.2018.

ПРИЛОЖЕНИЕ

Структура базы данных «Флора»

flora1 – таблица точек описаний

Название поля	Тип поля	Описание	Примечание
code	numeric(5,0)	код описания	первичный ключ
name	character(30)	название точки описания	не может принимать пустое значение
form	numeric(2,0)	код растительной формации	внешний ключ, связан с полем code таблицы form
author	numeric(2,0)	код автора	внешний ключ, связан с полем code таблицы author
date	date	дата описания	
memo	text	примечания	

flora2 – таблица описаний

Название поля	Тип поля	Описание	Примечание
code_op	numeric(5,0)	код описания	внешний ключ, связан с полем code таблицы flora1
code_vid	numeric(4,0)	код вида	внешний ключ, связан с полем code таблицы flora3
tier	numeric(1,0)	ярус	1 - древостой 1, 2 - древостой 2, 3 - подлесок, 4 - травостой, 5 - мохово-лишайниковый
dom	numeric(1,0)	степень присутствия вида в сообществе	1 - доминант, 2 - содоминант, 3 – присутствует

flora3 – справочник видов

Название поля	Тип поля	Описание	Примечание
code	numeric(4,0)	код вида	первичный ключ
rus_name	character(40)	название вида	не может принимать пустое значение
famrus	character(17)	название семейства	
areal	numeric(2,0)	код ареала	внешний ключ, связан с полем code таблицы areal
status	boolean	Статус по Красной книге	

`form` – справочник растительных формаций

Название поля	Тип поля	Описание	Примечание
code	numeric(2,0)	код растительной формации	первичный ключ
name	character(20)	название растительной формации	не может принимать пустое значение

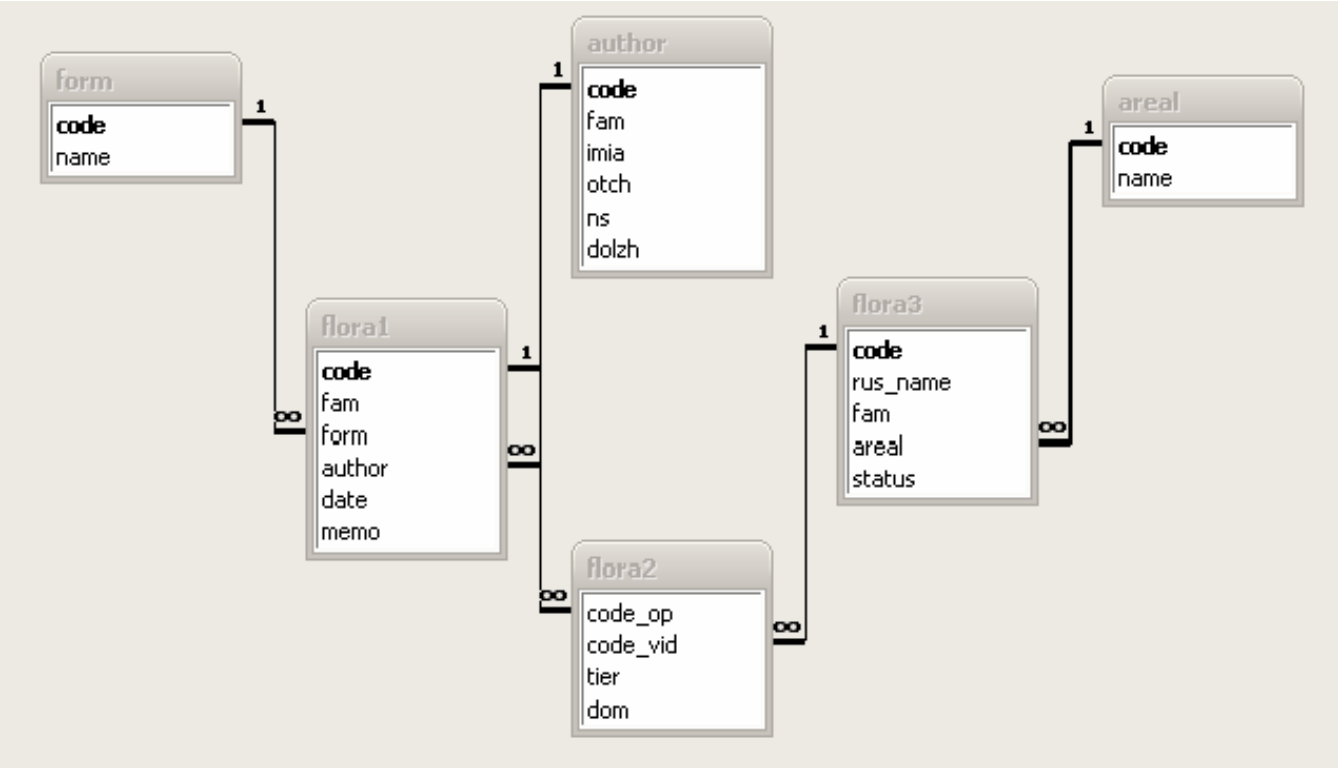
`areal` – справочник типов ареала

Название поля	Тип поля	Описание	Примечание
code	numeric(2,0)	код типа ареала	первичный ключ
name	character(65)	название типа ареала	не может принимать пустое значение

`author` – справочник авторов

Название поля	Тип поля	Описание	Примечание
code	numeric(2,0)	код автора	первичный ключ
fam	character(20)	фамилия автора	не может принимать пустое значение
imia	character(15)	имя автора	
otch	character(20)	отчество автора	
ns	character(20)	ученая степень	
dolzh	character(50)	должность	

Схема базы данных «Флора»



Структуры базы данных загрязнений атмосферного воздуха

objects – таблица объектов измерения

Название поля	Тип поля	Описание	Примечание
code	numeric(3,0)	код объекта	первичный ключ
name	character(20)	название объекта	не может принимать пустое значение

veshestva – справочник загрязняющих веществ

Название поля	Тип поля	Описание	Примечание
code	numeric(4,0)	код вещества	первичный ключ
name	character(30)	название вещества	не может принимать пустое значение
class	numeric(1,0)	класс опасности	ограничение на значение: 1- чрезвычайно опасные; 2 - высоко опасные; 3 - опасные; 4 - умеренно опасные.
pdkmr	numeric(7,5)	предельно допустимая концентрация, максимально-разовая, мг/ куб. м	ограничение на значение – больше или равно 0
pdkss	numeric(7,5)	предельно допустимая концентрация, среднесуточная, мг/ куб. м	ограничение на значение – больше или равно 0
notation	character(6)	обозначение (химическая формула)	

measurements – таблица данных измерений

Название поля	Тип поля	Описание	Примечание
code_ob	numeric(3,0)	код объекта	внешний ключ, связан с полем code таблицы objects
code_vesh	numeric(4,0)	код вещества	внешний ключ, связан с полем code таблицы veshestva
date	date	дата измерения	
concmax	numeric(7,5)	максимальная суточная концентрация, мг/ куб. м	ограничение на значение – больше или равно 0
concss	numeric(7,5)	среднесуточная концентрация, мг/ куб. м	ограничение на значение – больше или равно 0

Схема базы данных загрязнений атмосферного воздуха