8-11 классы

Программирование на Python

Презентация занятия

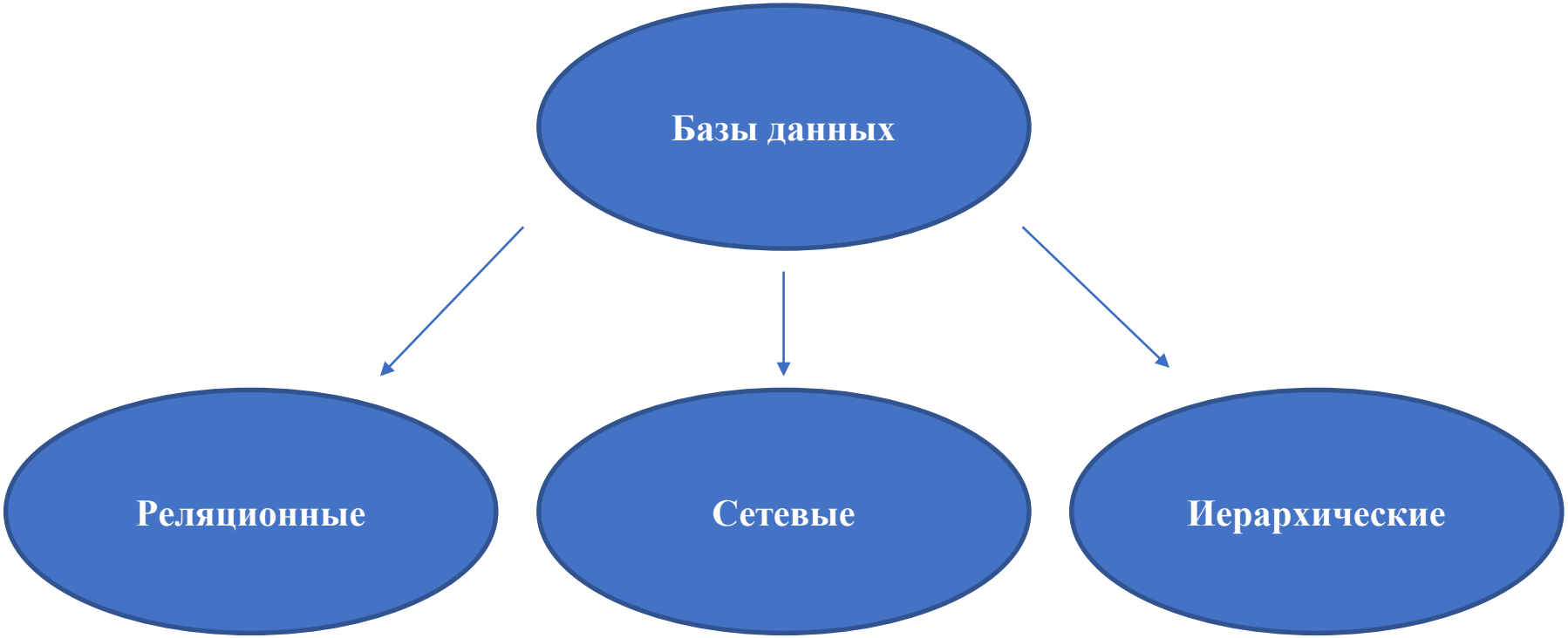
Системы хранения данных. Базы данных. Реляционные базы данных и их моделирование

50 занятие

инжиниум®
МГТУ им. Н.Э. Баумана

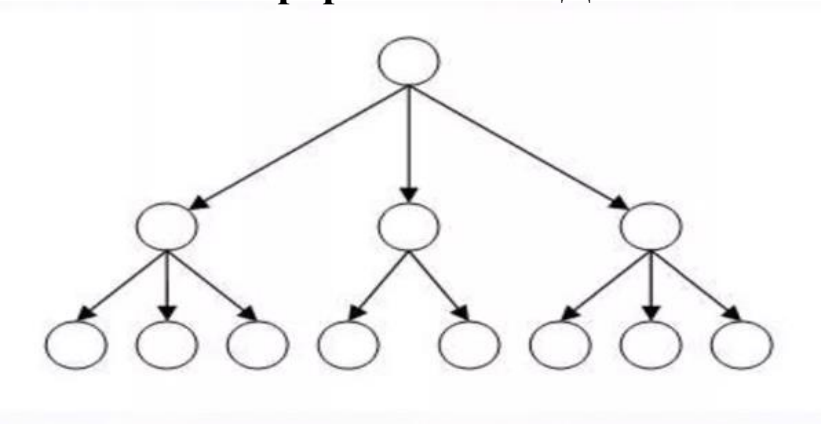
2023

Базы данных

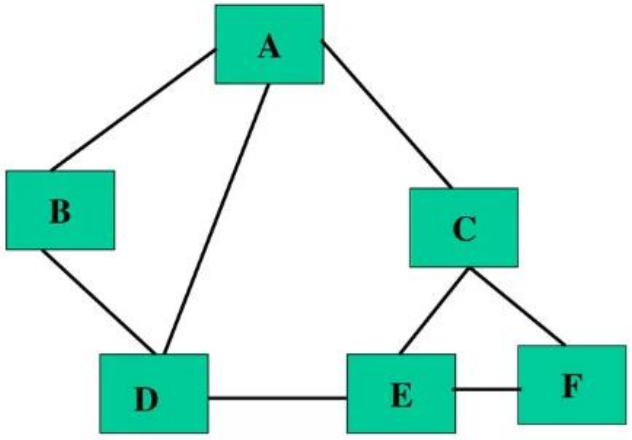


Базы данных

Иерархические БД



Сетевые БД



Реляционные БД

Реляционная БД. SQL

```
import sqlite3
```

Создание таблицы

```
CREATE [TEMP | TEMPORARY] TABLE [IF NOT EXISTS]
[<Название базы данных>.]<Название таблицы> (
   <Название поля1> [<Тип данных>] [<Опции>],
   [...,
   <Название поляN> [<Тип данных>] [<Опции>],]
   [<Дополнительные опции>]
);
```

Реляционная БД. SQL

В целях совместимости с другими базами данных, значение, указанное в параметре <Тип данных> , преобразуется в один из пяти классов родства:

- **INTEGER** — класс будет назначен, если значение содержит фрагмент "INT" в любом месте. Этому классу родства соответствуют типы данных INT, INTEGER, TINYINT, SMALLINT, MEDIUMINT, BIGINT и др.;
- **TEXT** — если значение содержит фрагменты "CHAR", "CLOB" или "TEXT". Например, TEXT, CHARACTER(30), VARCHAR(250), VARYING CHARACTER(100), CLOB и др. Все значения внутри круглых скобок игнорируются;
- **NONE** — если значение содержит фрагмент "BLOB" или тип данных не указан;
- **REAL** — если значение содержит фрагменты "REAL", "FLOA" или "DOUB". Например, REAL, DOUBLE, DOUBLE PRECISION, FLOAT;
- **NUMERIC** — если все предыдущие условия не выполняются, то назначается этот класс родства

SQLite поддерживает следующие типы данных:

- **NULL** — значение NULL;
- **INTEGER** — целые числа;
- **REAL** — вещественные числа;
- **TEXT** — строки;
- **BLOB** — бинарные данные.

Реляционная БД. SQL

В параметре <Опции> могут быть указаны следующие конструкции:

- NOT NULL [<Обработка ошибок>] — означает, что поле обязательно должно иметь значение при вставке новой записи. Если опция не указана, то поле может содержать значение NULL;
- DEFAULT <Значение> — задает для поля значение по умолчанию, которое будет использовано, если при вставке записи для этого поля не было явно указано значение. Пример:

```
sqlite> CREATE TEMP TABLE tmp3 (p1, p2 INTEGER DEFAULT 0);
sqlite> INSERT INTO tmp3 (p1) VALUES (800);
sqlite> INSERT INTO tmp3 VALUES (5, 1204);
sqlite> SELECT * FROM tmp3;
800|0
5|1204
sqlite> DROP TABLE tmp3;
```

В необязательном параметре <Дополнительные опции> могут быть указаны следующие конструкции:

- PRIMARY KEY (<Список полей через запятую>) [<Обработка ошибок>] — позволяет задать первичный ключ для нескольких полей таблицы;
- UNIQUE (<Список полей через запятую>) [<Обработка ошибок>] — указывает, что заданные поля могут содержать только уникальные значения;
- CHECK(<Условие>) — значение должно удовлетворять указанному условию.

SQL. Базовые методы

Вставка записей

Синтаксис

```
INSERT [OR <Алгоритм>] INTO [<Название базы данных>.]<Название таблицы>
[(<Поле1>, <Поле2>, ...)] VALUES (<Значение1>, <Значение2>, ...);
```

Пример

```
sqlite> CREATE TABLE user (
   ...> id_user INTEGER PRIMARY KEY AUTOINCREMENT,
   ...> email TEXT,
   ...> passw TEXT);
sqlite> CREATE TABLE rubr (
   ...> id_rubr INTEGER PRIMARY KEY AUTOINCREMENT,
   ...> name_rubr TEXT);
sqlite> CREATE TABLE site (
   ...> id_site INTEGER PRIMARY KEY AUTOINCREMENT,
   ...> id_user INTEGER,
   ...> id_rubr INTEGER,
   ...> url TEXT,
   ...> title TEXT,
   ...> msg TEXT);
```

```
sqlite> INSERT INTO user (email, passw)
   ...> VALUES ('unicross@mail.ru', 'password1');
sqlite> INSERT INTO rubr VALUES (NULL, 'Программирование');
sqlite> SELECT * FROM user;
1|unicross@mail.ru|password1
sqlite> SELECT * FROM rubr;
1|Программирование
sqlite> INSERT INTO site (id_user, id_rubr, url, title, msg)
   ...> VALUES (1, 1, 'http://wwwadmin.ru', 'Название', 'Описание');
```

SQL. Базовые методы

Вставка записей

Во всех этих примерах строковые значения указываются внутри одинарных кавычек. Однако бывают ситуации, когда внутри строки уже содержится одинарная кавычка. Попытка вставить такую строку приведет к ошибке:

```
sqlite> INSERT INTO rubr VALUES (NULL, 'Название 'в кавычках'');
Error: near "в": syntax error
```

Чтобы избежать этой ошибки, можно заключить строку в двойные кавычки или удвоить каждую одинарную кавычку внутри строки:

```
sqlite> INSERT INTO rubr VALUES (NULL, "Название 'в кавычках'");
sqlite> INSERT INTO rubr VALUES (NULL, 'Название ''в кавычках''');
sqlite> SELECT * FROM rubr;
1|Программирование
2|Название 'в кавычках'
3|Название 'в кавычках'
```

SQL. Базовые методы

Вставка записей

Если предпринимается попытка вставить запись, а в таблице уже есть запись с таким же значением первичного ключа (или значение индекса **UNIQUE** не уникально), то такая SQL-команда приводит к ошибке. Если необходимо, чтобы такие неуникальные записи обновлялись без вывода сообщения об ошибке, можно указать алгоритм обработки ошибок **REPLACE** после ключевого слова **OR**. Заменим название рубрики с идентификатором 2:

```
sqlite> INSERT OR REPLACE INTO rubr
   ...> VALUES (2, 'Музыка');
sqlite> SELECT * FROM rubr;
1|Программирование
2|Музыка
3|Название 'в кавычках'
```

Вместо алгоритма REPLACE можно использовать инструкцию REPLACE INTO. Инструкция имеет следующий формат:

```
REPLACE INTO [<Название базы данных>.]<Название таблицы>
[(<Поле1>, <Поле2>, ...)] VALUES (<Значение1>, <Значение2>, ...);
```

Заменим название рубрики с идентификатором 3:

```
sqlite> REPLACE INTO rubr VALUES (3, 'Игры');
sqlite> SELECT * FROM rubr;
1|Программирование
2|Музыка
3|Игры
```

SQL. Базовые методы

Обновление и удаление записей

```
UPDATE [OR <Алгоритм>] [<Название базы данных>.]<Название таблицы>
SET <Поле1>='<Значение>', <Поле2>='<Значение2>', ...
[WHERE <Условие>];
```

В качестве примера изменим название рубрики с идентификатором 3:

```
sqlite> UPDATE rubr SET name_rubr='Кино' WHERE id_rubr=3;
sqlite> SELECT * FROM rubr;
1|Программирование
2|Музыка
3|Кино
```

SQL. Базовые методы

Обновление и удаление записей

```
DELETE FROM [<Название базы данных>.]<Название таблицы>
[WHERE <Условие>];
```

Если условие не указано, то будут удалены все записи из таблицы. В противном случае удаляются только записи, соответствующие условию. В качестве примера удалим рубрику с идентификатором 3:

```
sqlite> DELETE FROM rubr WHERE id_rubr=3;
sqlite> SELECT * FROM rubr;
1|Программирование
2|Музыка
```

SQL. Базовые методы

Изменение свойств таблицы

В некоторых случаях необходимо изменить структуру уже созданной таблицы. Для этого используется инструкция **ALTER TABLE**. В **SQLite** инструкция **ALTER TABLE** позволяет выполнить лишь ограниченное количество операций. Например, нельзя изменить свойство поля или удалить его из таблицы.

```
ALTER TABLE [<Название базы данных>.]<Название таблицы>
<Преобразование>;
```

В параметре <Преобразование> могут быть указаны следующие конструкции:

- RENAME TO <Новое имя таблицы> — переименовывает таблицу. Изменим название таблицы user на users:

```
sqlite> .tables
rubr            sqlite_sequence  tmp1            user
site            table            tmp2
sqlite> ALTER TABLE user RENAME TO users;
sqlite> .tables
rubr            sqlite_sequence  tmp1            users
site            table            tmp2
```

SQL. Базовые методы

Изменение свойств таблицы

- `ADD [COLUMN] <Имя нового поля> [<Тип данных>] [<Опции>]` — добавляет новое поле после всех имеющихся полей. Обратите внимание на то, что в новом поле нужно задать значение по умолчанию, или значение `NULL` должно быть допустимым, т. к. в таблице уже есть записи. Кроме того, поле не может быть объявлено как `PRIMARY KEY` или `UNIQUE`. Добавим поле `iq` в таблицу `site`:

```
sqlite> ALTER TABLE site ADD COLUMN iq INTEGER DEFAULT 0;
sqlite> PRAGMA table_info(site);
0|id_site|INTEGER|0||1
1|id_user|INTEGER|0||0
2|id_rubr|INTEGER|0||0
3|url|TEXT|0||0
4|title|TEXT|0||0
5|msg|TEXT|0||0
6|iq|INTEGER|0|0|0
sqlite> SELECT * FROM site;
1|1|1|http://wwwadmin.ru|Название|Описание|0
```