Проектирование базы данных

Цель занятия

После освоения темы вы:

- научитесь выполнять сложные запросы и делать подзапросы, объединять таблицы из базы данных, группировать данные в таблице;
- узнаете, как работают агрегатные функции и как использовать оконные функции;
- узнаете термины нормализации;
- узнаете основные принципы проектирования БД, применяемые в работе программ и приложений;
- узнаете об основных функциях и методах модуля sqlite3.

План занятия

1. Сложные запросы
2. Функции SQL
3. Нормализация и тестирование базы данных
4. Принципы проектирования базы данных
5. Модуль sqlite3

Используемые термины

Сложные запросы — запросы с использованием SELECT, которые обращаются к двум и более таблицам, или в запросе используется несколько условий.

Вложенный запрос — это запрос на выборку, который используется внутри инструкции SELECT, INSERT, UPDATE или DELETE или внутри другого вложенного запроса.

Агрегатная функция в SQL — функция, которая выполняет вычисление на наборе значений и возвращает одиночное значение.

Оконная функция в SQL — функция, которая работает с выделенным набором строк (окном, партицией) и выполняет вычисление для этого набора строк в отдельном столбце.

Партиции (окна из набора строк) — это набор строк, указанный для оконной функции по одному из столбцов или группе столбцов таблицы.

Нормализация — это процесс организации данных в базе данных, включающий создание таблиц и установление отношений между ними в соответствии с правилами, которые обеспечивают защиту данных и делают базу данных более гибкой, устраняя избыточность и несогласованные зависимости.

Нормальная форма — требование, предъявляемое к структуре таблиц для устранения избыточных функциональных зависимостей между атрибутами (полями таблиц) в БД.

Аномалия — ситуация в таблице БД, которая приводит к противоречию в БД, либо существенно усложняет обработку БД.

Модуль sqlite3 — это API к СУБД SQLite, он обеспечивает интерфейс SQL.

Исключения — ошибки, которые возникают при исполнении кода.

Конспект занятия

1. Сложные запросы

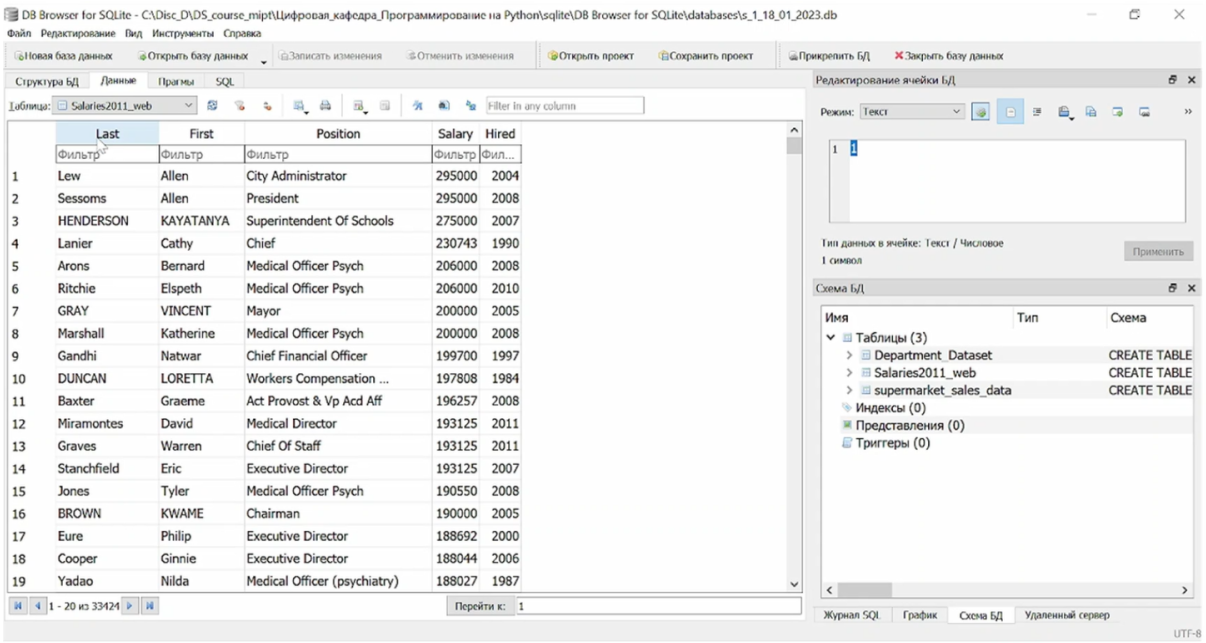
Операторы для написания сложных запросов

Оператор	Значение
• DISTINCT	Позволяет выбрать уникальные значения по какому-то столбцу
• Логические операторы AND, OR и NOT	Подходят для выборки по нескольким условиям
• SELECT TOP	Выборка из первых n строк
• LIKE	Позволяет проверить, что заданное строковое поле таблицы соответствует шаблону

• IN	Выбирают значения из набора значений для какого-то поля
• BETWEEN	Используется для того, чтобы выбрать строки, для которых заданное поле лежит в каком-то интервале

Практика. Знакомство с операторами

На практике познакомимся с применением рассмотренных операторов.
Работаем с данными по сотрудникам в DB Browser (рис. 1)



	Last	First	Position	Salary	Hired
1	Lew	Allen	City Administrator	295000	2004
2	Sessoms	Allen	President	295000	2008
3	HENDERSON	KAYATANYA	Superintendent Of Schools	275000	2007
4	Lanier	Cathy	Chief	230743	1990
5	Arons	Bernard	Medical Officer Psych	206000	2008
6	Ritchie	Elspeth	Medical Officer Psych	206000	2010
7	GRAY	VINCENT	Mayor	200000	2005
8	Marshall	Katherine	Medical Officer Psych	200000	2008
9	Gandhi	Natwar	Chief Financial Officer	199700	1997
10	DUNCAN	LORETTA	Workers Compensation ...	197808	1984
11	Baxter	Graeme	Act Provost & Vp Acd Aff	196257	2008
12	Miramontes	David	Medical Director	193125	2011
13	Graves	Warren	Chief Of Staff	193125	2011
14	Stanchfield	Eric	Executive Director	193125	2007
15	Jones	Tyler	Medical Officer Psych	190550	2008
16	BROWN	KWAME	Chairman	190000	2005
17	Eure	Philip	Executive Director	188692	2000
18	Cooper	Ginnie	Executive Director	188044	2006
19	Yadao	Nilda	Medical Officer (psychiatry)	188027	1987

Рисунок 1. Окно DB Browser.

Есть поля Имя, Фамилия сотрудника, позиция сотрудников компании, зарплата, год найма.

Допустим, нас интересует список уникальных позиций компании. Строк более 3000, и просматривать все позиции – не самый лучший вариант.

```
SELECT DISTINCT (Position) FROM Salaries2011_web;
```

Теперь ответим на вопрос, сколько сотрудников занимают позицию, например, City Administrator и, например, Chief Financial Officer.

```
SELECT DISTINCT (Position) FROM Salaries2011_web
WHERE Position NOT in ('City Administrator', 'Chief Financial Officer');
```

Рассмотрим, как используется оператор LIKE в документации.

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

Найдем сотрудников, у которых имя начинается с буквы A и заканчивается на s.

```
SELECT DISTINCT (Position) FROM Salaries2011_web
WHERE Last LIKE ‘A%s’;
```

Выберем сотрудников, у которых зарплата лежит в диапазоне от 90000$ до 150000$.

```
SELECT DISTINCT (Position) FROM Salaries2011_web
WHERE Salary BETWEEN 90000 and 150000;
```

Выведем список сотрудников, которые были наняты в 2000, 2004, 2006 годах.

```
SELECT DISTINCT (Position) FROM Salaries2011_web
WHERE Hired in (2000, 2004, 2006);
```

Объединение таблиц

Join — объединение таблиц.

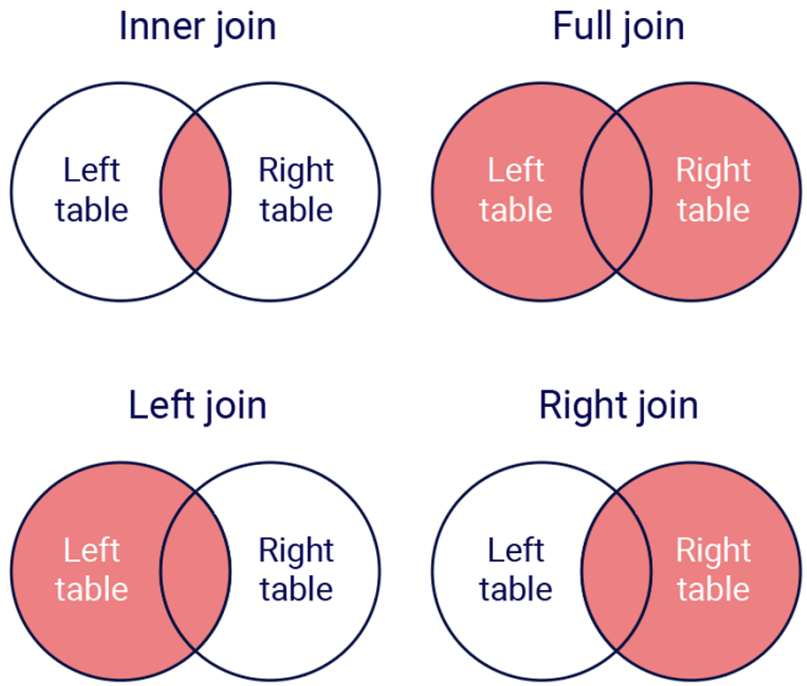


Рисунок 2. Типы объединения таблиц

Практика. Объединение таблиц

В практической части рассмотрим объединение таблиц на двух таблицах (рис. 3).

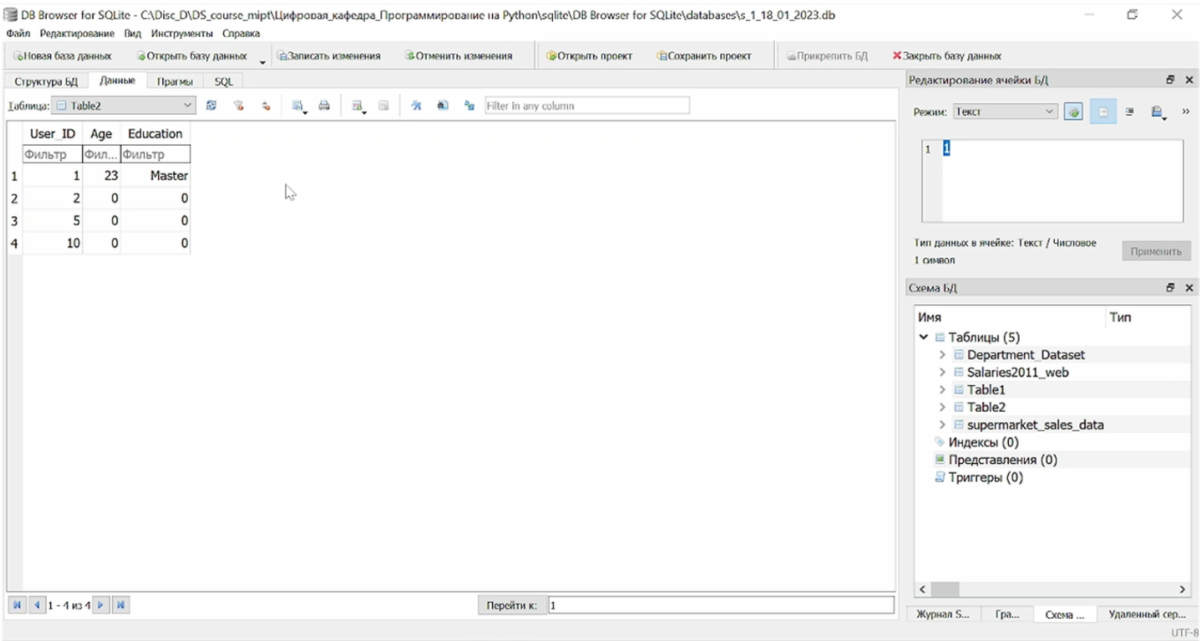


Рисунок 3. Окно DB Browser.

В обеих таблицах есть уникальный идентификатор пользователя (User_ID). Первая таблица содержит информацию об имени клиента и города. Во второй таблице информация о возрасте и образовании пользователя.

Начнем с внутреннего join.

```
SELECT Table1.User_ID, Table1.Name, Table2.User_ID, Table2.Age, Table2.Education
FROM Table1 INNER JOIN Table2
        ON Table1.User_ID = Table2.User_ID
```

Получаем строки с User_ID 1, 2, 5 из таблиц.

Если поменяем тип join на left, то получим строки с User_ID 1, 2, 4, 5:

```
SELECT Table1.User_ID, Table1.Name, Table2.User_ID, Table2.Age, Table2.Education
FROM Table1 LEFT JOIN Table2
        ON Table1.User_ID = Table2.User_ID
```

Вложенный запрос

Подзапрос может возвращать:

- одно значение — скалярный подзапрос;
- одну строку — строковый подзапрос;
- один столбец — столбцовый подзапрос;
- таблицу — табличный подзапрос.

Структура вложенного запроса

Структура запроса	Пример
SELECT поля_таблиц FROM список_таблиц WHERE конкретное_поле IN (SELECT поле_таблицы FROM таблица)	SELECT * FROM Transaction WHERE date = (SELECT MAX(date) FROM Transaction);

Практика. Вложенные запросы

В практической части рассмотрим вложенные запросы на примере данных по транзакциям (рис. 4)..

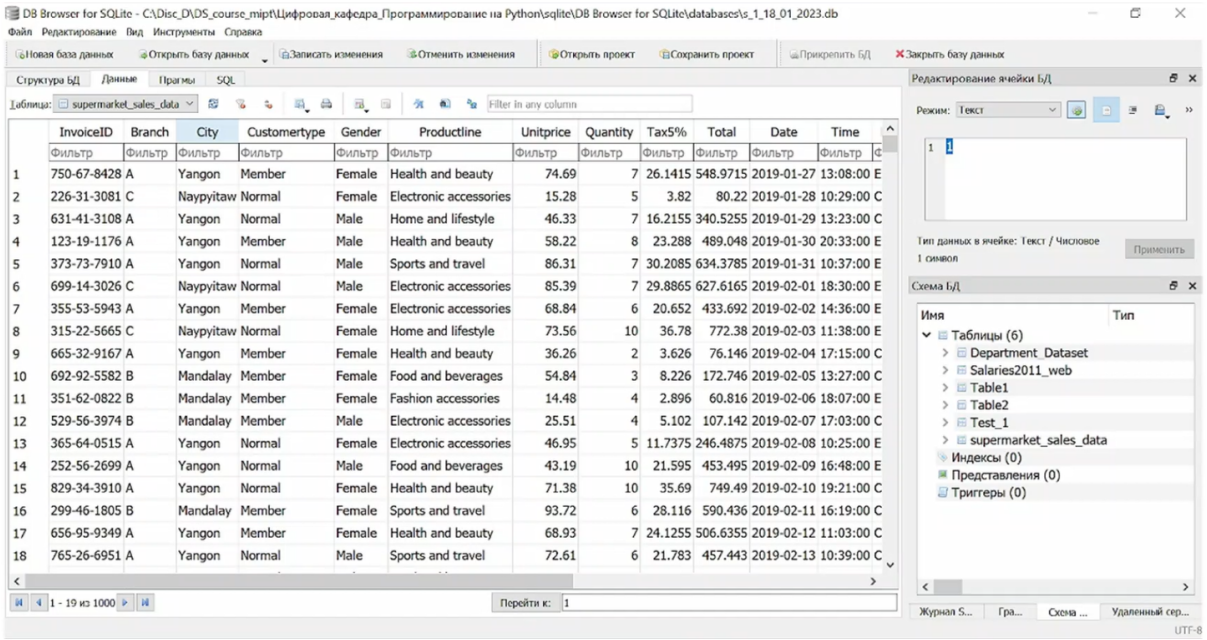


Рисунок 4. Окно DB Browser. Данные по транзакциям.

Допустим, нас интересуют транзакции, которые совершены за последнюю дату, которая присутствует в данных.

```
SELECT * FROM supermarket_sales_data
WHERE date - (SELECT MAX(date) FROM supermarket_sales_data);
```

Группировка данных

Структура запроса	Пример
SELECT ИМЕНА_СТОЛБЦОВ FROM ИМЯ_ТАБЛИЦЫ [WHERE УСЛОВИЕ] GROUP BY ИМЕНА_СТОЛБЦОВ	SELECT CustomerID, Country FROM Customers WHERE Country = “United Kindom” GROUP BY CustomerID;

Агрегатные функции:

- MIN
- MAX
- COUNT
- SUM
- AVG

MIN() возвращает минимальное значение столбца. Пример:

Code	Revenue	RegDate	Country	Company
id_109	4.5	12.01.2016	Australia	Company_1
id_005	5.1	5.09.2007	UK	Company_2
id_230	7.8	19.06.2022	Germany	Company_3
id_450	11.4	6.04.2019	UK	Company_4

```
SELECT MIN(Revenue) FROM Data;
 → 4.5
```

MAX() возвращает максимальное значение столбца:

```
SELECT MAX(Revenue) FROM Data;
 → 11.4
```

COUNT() возвращает количество записей, т.е. строк таблицы:

```
SELECT COUNT(CODE) FROM Data;
 → 4
```

Мы можем добавить дополнительное условие:

```
SELECT COUNT(CODE) FROM Data WHERE Revenue > 6;
 → 4
```

SUM() возвращает сумму значений столбца таблицы:

```
SELECT SUM(Revenue) FROM Data;
 → 28.8
```

AVG() возвращает среднее значение столбца:

```
SELECT AVG(Revenue) FROM Data;
 → 7.2
```

Практика. Группировка данных

Рассмотрим группировку данных по заданному полю и расчет агрегатных функций. Продолжим работать с данными по транзакциям (рис. 4). Напишем запрос, в котором сгруппируем транзакции по дате и рассчитаем количество транзакций за определенную дату, среднее значение суммы транзакций, суммарное количество купленного товара и минимальную и максимальную стоимость товаров, купленных за некоторую дату.

```
SELECT COUNT(InvoiceID), ROUND(AVG(Total), 2), SUM(Quantity),
MIN(Unitprice), MAX(Unitprice), Date
FROM supermarket_sales_data
GROUP BY Date;
```

2. Функции SQL

ROUND() — функция для округления десятичных чисел. Пример:

Code	Revenue	RegDate	Country	Company
id_109	4.5	12.01.2016	Australia	Company_1
id_005	5.1	5.09.2007	UK	Company_2
id_230	7.8	19.06.2022	Germany	Company_3
id_450	11.4	6.04.2019	UK	Company_4

```
SELECT ROUND(1087.48, 1) AS RValue;
 → 1087.50
```

UCASE() — перевод значений столбца к верхнему регистру:

```
SELECT UCASE(Country) FROM Data;

 → AUSTRALIA
    UK
    GERMANY
    UK
```

LCASE() — перевод значений столбца к нижнему регистру:

```
SELECT LCASE(Country) FROM Data;

→ australia
  uk
  germany
  uk
```

SUBSTRING(string, start, length) возвращает определенное количество символов текстового поля таблицы:

```
SELECT SUBSTRING(Code, 3, 3) FROM Data;

→ 109
  005
  230
  450
```

NOW() возвращает системное время и дату:

```
SELECT * FROM Data
WHERE RegDate > NOW();

→
```

Оконные функции

Оконные функции применяются для расчета скользящего среднего, подсчета накопительных итогов, ранжирования и ряда других задач.

Синтаксис оконных функций:

Запрос	Значение
• SELECT	list of columns, window functions
• FROM	table / joint tables / subquery

• WHERE	filtering clause
• GROUP BY	list of columns
• HAVING	aggregation filtering clause
• ORDER BY	list of columns / window functions

Оконную функцию можно добавить либо после оператора SELECT, либо после ORDER BY.

Рассмотрим отличие оконной функции от оператора GROUP BY. Есть таблица с успеваемостью студентов:

Имя	Предмет	Оценка
Петя	Математика	3
Петя	Русский язык	4
Петя	Физика	5
Маша	Математика	4
Маша	Русский язык	4
Маша	Физика	5

При применении GROUP BY по имени, получим таблицу только из двух строк:

Имя	Средняя оценка
Петя	4
Маша	4,3

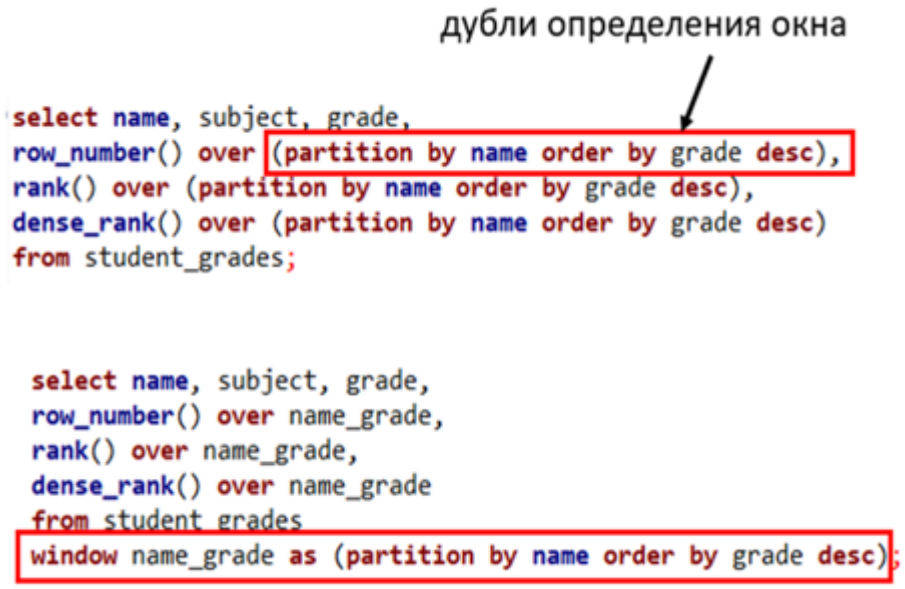
В результате получим среднюю оценку для каждого студента.

Если воспользуемся оконной функцией, то количество строк будет равно исходному, но добавится столбец «Средняя оценка»:

Имя	Предмет	Оценка	Средняя оценка
Петя	Математика	3	4
Петя	Русский язык	4	4
Петя	Физика	5	4
Маша	Математика	4	4,3
Маша	Русский язык	4	4,3
Маша	Физика	5	4,3

Оконная функция задается следующим образом:

FUNCTION_NAME(column_name)	Имя оконной функции одного из классов
[FILTER(WHERE filter_clause)]	Необязательное выражение фильтрации
OVER **(**	Ключевое слово определения оконной функции
PARTITION BY (column_names),	Определение партиций по колонкам
ORDER BY (column_names),	Сортировка вычисления оконной функции
[frame_clause] **)**	Указание фрейма для партиции



Эквивалентные записи одной и той же оконной функции

Типы оконных функций

Ranking (рейтинги) — оконная функция ранжирования.

row_number()	возвращает порядковый номер строки
dense_rank()	возвращает ранг строки
rank()	возвращает тоже ранг, но с пропусками
ntile(n)	разбивает все строки на n групп и возвращает номер группы, в которую попала строка

Value (соседние элементы и границы) — функции сравнения со смещением:

lag(value, offset)	возвращает значение value из строки, отстоящей на offset строк назад от текущей
lead(value, offset)	возвращает значение value из строки, отстоящей на offset строк вперед от текущей

first_value(value)	возвращает значение value из первой строки фрейма
last_value(value)	возвращает значение value из последней строки фрейма
nth_value(value, n)	возвращает значение value из n-й строки фрейма

Aggregate (сумма и среднее) — агрегатные функции:

min(value)	возвращает минимальное value среди строк, входящих в окно
max(value)	возвращает максимальное value
count(value)	возвращает количество value, не равных null
avg(value)	возвращает среднее значение по всем value
sum(value)	возвращает сумму значений value
group_concat(value, separator)	возвращает строку, которая соединяет значение value через разделитель separator

Рассмотрим применение оконных функций для расчета скользящих агрегатов (сумма и среднее в динамике).

Month	Revenue
1	345
2	120
3	200
4	250

5	402

Скользящее среднее рассчитывается по трем значениям:

- для текущего значения строки,
- предыдущего,
- последующего.

Для первой и последней строк нет предыдущего и последующего. Для них получаем расчет только из двух строк:

Month	Revenue	Rolling_avg
1	346	232.5
2	120	222
3	200	190
4	250	284
5	400	326

3. Нормализация и тестирование базы данных

Что такое нормализация

Рассмотрим пример, в котором нужно изменить таблицу. Казалось бы, с таблицей все хорошо, но это не так.

Проблема: таблица (отношение) описывает две сущности — поставку и поставщика.

Код поставщика	**Город**	**Код товара**	**Количество**
1029	Москва	10	200
3044	Санкт-Петербург	12	300
1023	Москва	25	100
1476	Владивосток	45	500

Могут возникать аномалии:

- Аномалия вставки. Мы не сможем добавить нового поставщика, если у него пока не было никаких поставок.
- Аномалия обновления.
- Аномалия удаления. Мы не можем удалить данные о поставщике, не затронув данные о поставке.

Решение: разбиение исходной таблицы на две.

Цель нормализации: исключить избыточное дублирование данных, которое является причиной аномалий, возникающих при добавлении, редактировании и удалении строк таблицы.

Нормальные формы

Всего нормальных форм шесть.

Первая нормальная форма (1НФ). Таблица находится в 1НФ, если все ее атрибуты являются простыми. Все используемые допустимые значения атрибутов должны содержать только скалярные значения.

Фирма	Модели
Dell	Inspiron, Latitude, Precision
Asus	ExpertBook, Zenbook
Acer	Aspire
Apple	MacBook Air

Для Dell и Asus в столбце «Модели» нескалярные значения. То есть таблица не является таблицей в первой нормальной форме, не все ее атрибуты простые.

Перейдем к первой нормальной форме:

Фирма	Модели
Dell	Inspiron
Dell	Latitude

Dell	Precision
Asus	ExpertBook
Asus	Zenbook
Acer	Aspire
Apple	MacBook Air

Вторая нормальная форма (2НФ). Таблица находится во 2НФ, если она находится в 1НФ, и каждый неключевой атрибут неприводимо зависит от Первичного Ключа (ПК).

Пример:

Фирма	Модели	Цена	Скидка
Dell	Inspiron	30000	5
Dell	Latitude	45000	5
Dell	Precision	46000	5
Asus	ExpertBook	70000	2
Asus	Zenbook	34000	2
Acer	Aspire	65000	10
Apple	MacBook Air	90000	15

Здесь нарушено требование второй нормальной формы. Разобьем на две таблицы — с информацией о цене и с информацией со скидкой.

Фирма	Модели	Цена
Dell	Inspiron	30000
Dell	Latitude	45000
Dell	Precision	46000
Asus	ExpertBook	70000

Asus	Zenbook	34000
Acer	Aspire	65000
Apple	MacBook Air	90000

Фирма	Скидка
Dell	5
Asus	2
Acer	10
Apple	15

Третья нормальная форма (3НФ). Таблица находится в 3НФ, когда находится во 2НФ и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.

Модели	Магазин	Телефон
Inspiron	Ленинградское шоссе	56-34-90
Latitude	ул. Беговая	50-21-76
Precision	БЦ Метрополис	32-09-08
ExpertBook	ул. Беговая	50-21-76

Эта таблица не находится в третьей нормальной форме. Нужно таблицу разбить на две — с информацией о номерах телефонов и с информацией о моделях в магазинах.

Магазин	Телефон
Ленинградское шоссе	56-34-90
ул. Беговая	50-21-76
БЦ Метрополис	32-09-08

Модели	Магазин
Inspiron	Ленинградское шоссе
Latitude	ул. Беговая
Precision	БЦ Метрополис
ExpertBook	ул. Беговая

Нормальная форма Бойса-Кодда (НФБК) — частная форма третьей нормальной формы.

Введение дополнительной формы связано с тем, что третья нормальная форма не подходит для следующих отношений:

- отношение имеет два или более потенциальных ключа;
- два и более потенциальных ключа являются составными;
- они пересекаются, то есть имеют хотя бы один общий атрибут.

Номер стоянки	Время начала	Время окончания	Тариф
1	9:30	10:30	Бережливый
1	11:00	12:00	Бережливый
1	14:00	15:30	Стандарт
2	10:00	12:00	Премиум-B
2	12:00	14:00	Премиум-B
2	15:00	18:00	Премиум-A

Чтобы таблица была в нормальной форме Бойса-Кодда, нужно разбить ее на две.

Тарифы:

Тариф	Номер стоянки	Имеет льготы
Бережливый	1	Да

Стандарт	1	Нет
Премиум-А	1	Да
Премиум-В	2	Нет

Бронирования:

Тариф	Время начала	Время окончания
Бережливый	9:30	10:30
Бережливый	11:00	12:00
Стандарт	14:00	15:30
Премиум-В	10:00	12:00
Премиум-В	12:00	14:00
Премиум-А	15:00	18:00

Четвертая нормальная форма (4НФ). Таблица находится в 4НФ, если она находится в НФБК, и все нетривиальные многозначные зависимости фактически являются функциональными зависимостями от ее потенциальных ключей.

Пятая нормальная форма (5НФ). Таблица находится в 5НФ, если она находится в 4НФ и отсутствуют сложные зависимые соединения между атрибутами.

Сотрудник	Проект	Направление
Иванов	Прогнозирование продаж	Планирование
Петров	Интернет-магазин	Бухгалтерия
Петров	Маркетинговые кампании	Маркетинг
Сидорова	Личный кабинет	Разработка
Волков	Оптимизация маршрутов	Логистика

В данной таблице получаются сложные связи.

Связь сотрудников и проектов:

Сотрудник	Проект
Иванов	Прогнозирование продаж
Петров	Интернет-магазин
Петров	Маркетинговые кампании
Сидорова	Личный кабинет
Волков	Оптимизация маршрутов

Связь сотрудников и направлений:

Сотрудник	Направление
Иванов	Планирование
Петров	Бухгалтерия
Петров	Маркетинг
Сидорова	Разработка
Волков	Логистика

Связь проектов и направлений:

Проект	Направление
Прогнозирование продаж	Планирование
Интернет-магазин	Бухгалтерия
Маркетинговые кампании	Маркетинг
Личный кабинет	Разработка
Оптимизация маршрутов	Логистика

Шестая нормальная форма (6НФ). Переменная отношения находится в шестой нормальной форме тогда и только тогда, когда она удовлетворяет всем нетривиальным зависимостям соединения.

Работники:

Таб. №	Время	Должность	Домашний адрес
6575	01-01-2000: 10-02-2003	слесарь	Ленина, 10
6575	11-02-2003: 15-06-2006	слесарь	Советская, 22
6575	16-06-2006: 05-03-2009	бригадир	Советская, 22

Таблица не находится в шестой нормальной форме. Нужно разбить исходную таблицу на две.

Должности работников:

Таб. №	Время	Должность
6575	01-01-2000: 10-02-2003	слесарь
6575	16-06-2006: 05-03-2009	бригадир

Домашние адреса работников:

Таб. №	Время	Домашний адрес
6575	01-01-2000: 10-02-2003	Ленина, 10
6575	16-06-2006: 05-03-2009	Советская, 22

Для чего нужно тестировать БД

Важный этап работы с базами данных — их тестирование. Оно нужно, чтобы проверить:

- целостность данных;
- достоверность данных;
- согласованность данных.

При тестировании БД нужно обратить внимание на следующие моменты:

- корректное отображение информации;
- корректное обновление записи в таблице;
- тестирование свойств ACID операций;
- проверить сохранность и целостность данных;
- проверить соответствие заданным бизнес-правилам.

Основные виды тестирования БД:

- Структурная проверка — тестирование схем БД таблиц, столбцов и т. д. Например, проверка типов данных в базе данных по значениям полей в клиентском приложении.

- Функциональная проверка – проверка работоспособности БД с точки зрения пользователя. Например, проверка соответствия бизнес-требованиям.

- Нефункциональная проверка — тестирование производительности БД в нагрузочных тестах, тестирование рисков до стресс-тестирования, а также аналитики минимальных требований системы. Например, многократный запуск наиболее часто используемой транзакции, чтобы увидеть производительность системы базы данных.

4. Принципы проектирования базы данных

Для чего нужно структурировать БД:

- экономит место на диске, так как не содержит лишней информации;
- поддерживает целостность и точность данных;
- обеспечивает удобный доступ к данным.

В проектировании БД можно выделить три основных этапа:

1. Концептуальное проектирование.
2. Логическое проектирование.
3. Физическое проектирование.

Концептуальное проектирование:

- Определение сущностей, т. е. объектов, которые существуют независимо от других.
- Определение связей между сущностями и их документирование.

- Определение атрибутов объектов, первичных ключей и т. д.

Логическое проектирование:

- Выбор модели данных (реляционная и т. д.).
- Определение набора таблиц в БД.
- Нормализация таблиц.
- Проверка возможности выполнения всех транзакций, предусмотренных пользователями.
- Определение требований поддержки целостности данных для того, чтобы не допустить ввод некорректной информации.

Физическое проектирование:

- Проектирование таблиц на основе выбранной СУБД.
- Реализация требуемых бизнес-правил.
- Проектирование физической организации БД.
- Проработка защиты БД от несанкционированного доступа.
- Мониторинг функционирования БД и ее настройки.

5. Модуль sqlite3

Что такое sqlite3

Модуль sqlite3 — это API к СУБД SQLite, он обеспечивает интерфейс SQL.

Преимущества модуля sqlite3:

- Не требуется установки сервера для работы с базой данных.
- Не нужно дополнительно устанавливать SQLite, все необходимое уже есть в модуле sqlite3.
- В модуле sqlite3 содержится большое число классов, функций и констант.

Недостатки:

- Не подходит для реализации доступа к базе данных с нескольких разных устройств.
- Ограничена многопоточность.

Для того чтобы подключиться к SQLite, нужно:

1. Воспользоваться методом **connect()** из модуля sqlite3 и передать в качестве аргумента название базы данных.
2. Создать объект **cursor** для выполнения SQLite-запросов из Python.

3. Отключиться от БД, т. е. закрыть объект **cursor** после завершения работы.
4. Перехватить исключение базы данных, если в процессе подключения произошла ошибка.

Исключения базы данных SQLite

Чтобы исполнение кода продолжалось, нужно обрабатывать исключения. Для этого нужно знать, какие они бывают и по каким причинам возникают. Обрабатываются ошибки через конструкцию except.

Типы исключений:

- sqlite3.Warning — по сути ошибкой не является, это предупреждение. Не приводит к прекращению исполнения кода.
- sqlite3.Error — общий тип исключений.
- sqlite3.DatabaseError — такая ошибка возникает в случае, если мы попытаемся открыть файл, который на самом деле базой данных не является.
- sqlite3.IntegrityError — возникает по причине интеграционных проблем в базе данных.
- sqlite3.ProgrammingError — возникает в случае ошибки в SQL запросе, когда, например, пытаемся создать таблицу с тем названием, которое уже есть, или же допустили ошибки в синтаксисе SQL запроса.
- sqlite3.OperationalError — возникает в случае разрыва соединения с базой данных или по другим техническим причинам.
- sqlite3.NotSupportedError — если мы используем те методы, которые не предусмотрены в базе данных.

Сохранение резервной копии базы данных из Python

Для сохранения резервной копии базы SQLite есть метод **connection.backup()**:

connection.backup(target, *, pages=0, progress=None, name="main", sleep=0.250)

target — экземпляр другого соединения, т. е. куда копируем БД.

pages — определяет, сколько страниц копируем за один раз (если 0 или отрицательное число, то вся БД копируется в один шаг).

name — определяет базу данных, резервную копию которой нужно сделать.

sleep — количество секунд между последовательными попытками сохранить оставшиеся страницы.