# Lazer Andelman

# Итоговый пpoект пo мoдулю SQL и Базы Данных

# База данных Вike Store Relational Dataвase:

# *Таблицы базы данных:*

* Таблицы:

|  |  |  |
| --- | --- | --- |
| вrands | 9 | записей |
| categories | 7 | записей |
| customers | 1445 | записей |
| order\_items | 4722 | записей |
| orders | 1615 | записей |
| products | 321 | записей |
| staffs | 10 | записей |
| stocks | 313 | записей |
| stores | 3 | записей |

# *Схема базы данных:*

# 

# Постановка задачи:

# *Анализ продаж по категориям и брендам.*

# *Определение лучших клиентов.*

# *Анализ эффективности персонала.*

# *Исследование динамики продаж.*

# *Сегментация клиентов:*

# *по возрасту и социальной активности.*

# *по семейному статусу и количеству детей.*

# *по образованию и месячному доходу.*

# *Общий объем продаж по магазинам.*

# *Эффективность персонала.*

# *Рассчитываем кумулятивный профит по месяцам.*

# *Рассчитываем скользящее среднее профита.*

# *Прогнозирование продаж.*

# *Анализ зависимости между скидками и объемом продаж.*

# Добавление данных в базу.

# *Добавляем таблицу delivery\_carriers 19 записей*

# Поля:

# carrier\_id

# carrier\_name

# *В таблицу 'customers' добавляем поля с личными данными клиентов.*

# Поля:

# age

# degree

# family\_status

# children\_amount

# monthly\_income

# social\_activity

# payment\_method

# *В таблицу 'orders' добавляем поле с данными.*

# order\_time

# carrier\_id

# delivery\_type

# Данные полей заполняем в Excel при помощи функций RAND(), RANDBETWEEN() и VLOOCKUP().

# Первичная обработка данных.

# *Переименование столбцов в соответствии с naming convention.*

# *Загрузка данных в базу данных.*

# *Проверка записей на пропущенные или пустые значения (в целях отработки запросов на проверку данных и удаление не соответствующих, в таблицы вручную были внесены изменения):*

# *--Для таблицы orders*

# *SELECT order\_id,*

# *customer\_id,*

# *order\_status,*

# *order\_date,*

# *order\_time,*

# *required\_date,*

# *shipped\_date,*

# *carrier\_id,*

# *store\_id,*

# *staff\_id*

# *FROM orders*

# *WHERE customer\_id IS NULL*

# *OR order\_status IS NULL*

# *OR order\_date IS NULL*

# *OR order\_time IS NULL*

# *OR (required\_date IS NULL AND shipped\_date IS NULL) -- Если обе даты отсутствуют*

# *OR carrier\_id IS NULL*

# *OR store\_id IS NULL*

# *OR staff\_id IS NULL;*

# *Подобным же образом производится проверка на пропущенные или пустые значения и в остальных таблицах.*

# *Проверка полученных данных на сохранение целостности данных.*

# *UPDATE order\_items*

# *SET quantity = 1*

# *WHERE quantity IS NULL;*

# *В таблице order\_items было произведено исправление, чтобы не нарушить целостность данных.*

# *Удаление записей (если принято такое решение)*

# *--Для таблицы customers*

# *DELETE FROM customers*

# *WHERE customer\_first\_name IS NULL OR customer\_first\_name = ''*

# *OR customer\_last\_name IS NULL OR customer\_last\_name = ''*

# *OR customer\_phone IS NULL OR customer\_phone = ''*

# *OR customer\_email IS NULL OR customer\_email = ''*

# *OR customer\_address IS NULL OR customer\_address = ''*

# *OR customer\_city IS NULL OR customer\_city = ''*

# *OR customer\_state IS NULL OR customer\_state = ''*

# *OR customer\_zip\_code IS NULL OR customer\_zip\_code = ''*

# *OR customer\_age IS NULL*

# *OR customer\_degree IS NULL OR customer\_degree = ''*

# *OR family\_status IS NULL OR family\_status = ''*

# *OR children\_amount IS NULL*

# *OR monthly\_income IS NULL*

# *OR social\_activity IS NULL OR social\_activity = ''*

# *OR payment\_method IS NULL OR payment\_method = '';*

# *Подобным же образом производится удаление пустых записей и в остальных таблицах.*

# *Удаление дубликатов строк.*

# *--Для таблицы brands:*

# *CREATE TEMPORARY TABLE temp\_brands AS*

# *SELECT MIN(brand\_id) as brand\_id, brand\_name*

# *FROM brands*

# *GROUP BY brand\_name;*

# *DELETE FROM brands;*

# *INSERT INTO brands (brand\_id, brand\_name)*

# *SELECT brand\_id, brand\_name*

# *FROM temp\_brands;*

# *DROP TABLE temp\_brands;*

# *Подобным же образом производится удаление дубликатов записей и в остальных таблицах.*

# Исследование данных с помощью SQL.

# *Анализ продаж по категориям и брендам.*

# *SELECT*

# *c.category\_name,*

# *COUNT(oi.order\_id) AS total\_sales,*

# *SUM(oi.quantity) AS total\_quantity\_sold,*

# *ROUND(SUM(oi.product\_price \* oi.quantity - oi.discount), 2) AS total\_revenue*

# *FROM*

# *categories c*

# *JOIN products p ON c.category\_id = p.category\_id*

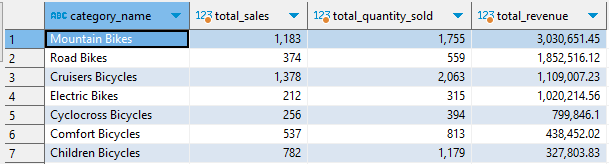
# *JOIN order\_items oi ON p.product\_id = oi.product\_id*

# *GROUP BY*

# *c.category\_name*

# *ORDER BY*

# *total\_revenue DESC;*



### **Объяснение запроса:**

* **FROM categories c**: Начинаем запрос с таблицы **categories**, так как нас интересуют результаты по категориям товаров.
* **JOIN products p ON c.category\_id = p.category\_id**: Соединяем таблицу **products** с таблицей **categories** по **category\_id**, чтобы получить доступ к товарам определенной категории.
* **JOIN order\_items oi ON p.product\_id = oi.product\_id**: Соединяем таблицу **order\_items** с таблицей **products** по **product\_id**, чтобы учитывать только те товары, которые были проданы.
* **COUNT(oi.order\_id) AS total\_sales**: Считаем общее количество продаж как количество уникальных **order\_id** в таблице **order\_items**.
* **SUM(oi.quantity) AS total\_quantity\_sold**: Считаем общее количество проданных товаров, суммируя значения **quantity** из **order\_items**.
* **SUM(oi.product\_price \* oi.quantity - oi.discount) AS total\_revenue**: Считаем общую выручку, умножая цену товара на количество и вычитая скидку для каждой записи, а затем суммируя полученные значения.
* **GROUP BY c.category\_name**: Группируем результаты по названию категории, чтобы агрегатные функции применялись в рамках каждой категории.
* **ORDER BY total\_revenue DESC**: Сортируем результаты по общей выручке по убыванию, чтобы наиболее прибыльные категории были выше в списке.

### **Визуализация полученных результатов:**

*import sqlite3*

*import pandas as pd*

*import matplotlib.pyplot as plt*

*import seaborn as sns*

*# Подключение к базе данных*

*conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02. SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase\_2.db')*

*# Загрузка данных*

*query = '''*

*SELECT*

*c.category\_name,*

*COUNT(oi.order\_id) AS total\_sales,*

*SUM(oi.quantity) AS total\_quantity\_sold,*

*ROUND(SUM(oi.product\_price \* oi.quantity - oi.discount), 2) AS total\_revenue*

*FROM*

*categories c*

*JOIN products p ON c.category\_id = p.category\_id*

*JOIN order\_items oi ON p.product\_id = oi.product\_id*

*GROUP BY*

*c.category\_name*

*ORDER BY*

*total\_revenue DESC;*

*'''*

*sales\_data = pd.read\_sql\_query(query, conn)*

*print(sales\_data)*

*# Визуализация данных*

*plt.figure(figsize=(10, 6))*

*sns.barplot(x='total\_revenue', y='category\_name', data=sales\_data, palette='viridis')*

*plt.title('Анализ продаж по категориям и брендам')*

*plt.xlabel('Общая выручка')*

*plt.ylabel('Категория товара')*

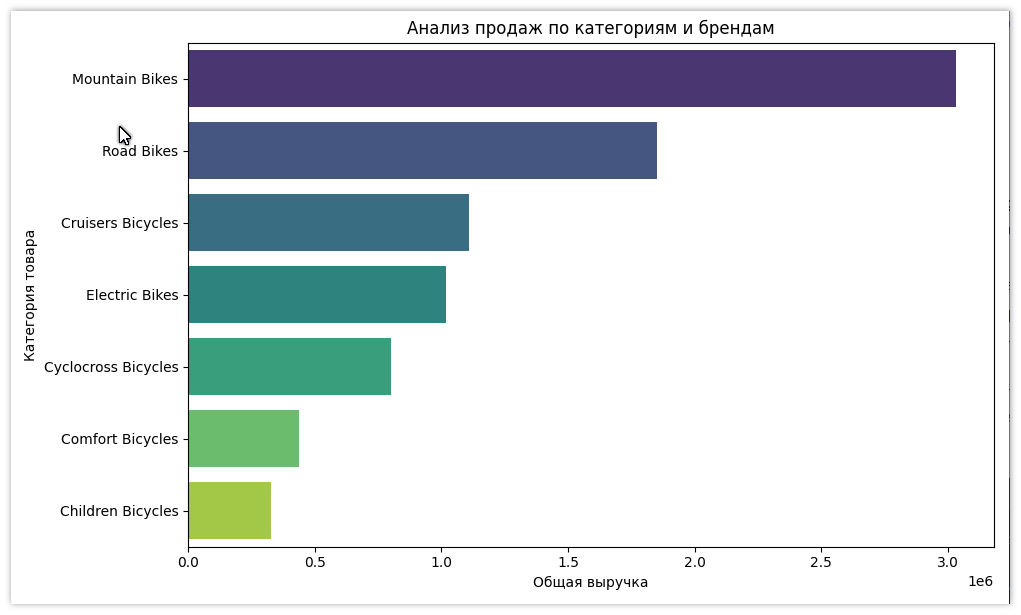
*# plt.legend(title='Бренд', bbox\_to\_anchor=(1.05, 1), loc='upper left')*

*plt.tight\_layout()*

*plt.show()*

*# Закрытие соединения с базой данных*

*conn.close()*



**Выводы:**

1. **Лидеры продаж и выручки:** Категория "Mountain Bikes" является абсолютным лидером как по общему количеству продаж, так и по общей выручке. Это указывает на высокий спрос и прибыльность данной категории товаров в ассортименте.
2. **Специализированные и дорогие велосипеды:** Категории "Road Bikes" и "Electric Bikes" показывают высокую выручку при относительно небольшом количестве продаж, что может свидетельствовать о более высокой цене этих товаров по сравнению с другими категориями. Это также может указывать на то, что покупатели готовы платить больше за специализированные или инновационные продукты.
3. **Популярные бюджетные категории:** Категория "Cruisers Bicycles" имеет высокие показатели как по количеству продаж, так и по выручке, что может свидетельствовать о популярности этих велосипедов среди широкого круга потребителей и их доступной цене.
4. **Нишевые категории:** Категории, такие как "Cyclocross Bicycles" и "Comfort Bicycles", показывают более низкие показатели по сравнению с лидерами. Это может указывать на их нишевый характер и более узкую аудиторию покупателей.
5. **Детские велосипеды:** Категория "Children Bicycles" показывает хорошие результаты по количеству продаж при относительно низкой общей выручке, что говорит о низкой средней цене товаров в этой категории. Это может отражать большой объем продаж бюджетных товаров для детей.

**Рекомендации:**

* **Фокус на высокодоходные категории:** Уделить особое внимание категориям "Mountain Bikes" и "Road Bikes", исследуя возможности для расширения ассортимента и оптимизации ценообразования.
* **Расширение ассортимента в популярных категориях:** Для категории "Cruisers Bicycles" стоит рассмотреть возможность расширения ассортимента, учитывая ее популярность и потенциал для увеличения продаж.
* **Продвижение и маркетинг:** Активное продвижение и маркетинговые кампании для нишевых категорий, таких как "Electric Bikes" и "Cyclocross Bicycles", могут помочь увеличить их долю на рынке.
* **Специальные предложения для детских велосипедов:** Разработка специальных предложений или акций для категории "Children Bicycles" может стимулировать продажи, учитывая их высокий объем и низкую среднюю цену.

# *Определение лучших клиентов:*

SELECT

ranked\_customers.customer\_id,

c.customer\_first\_name,

c.customer\_last\_name,

ranked\_customers.total\_spent,

ranked\_customers.rank

FROM (

SELECT

o.customer\_id,

SUM(oi.quantity \* oi.product\_price - oi.discount) OVER(PARTITION BY o.customer\_id) AS total\_spent,

RANK() OVER(ORDER BY SUM(oi.quantity \* oi.product\_price - oi.discount) DESC) AS rank

FROM orders o

JOIN order\_items oi ON o.order\_id = oi.order\_id

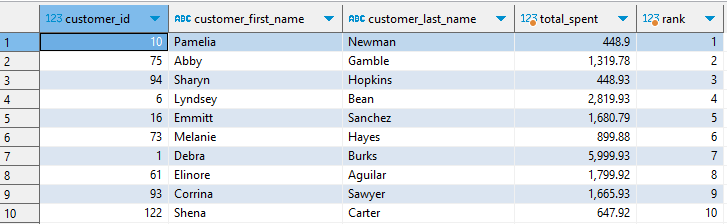
GROUP BY o.customer\_id

) AS ranked\_customers

JOIN customers c ON ranked\_customers.customer\_id = c.customer\_id

WHERE rank <= 10

ORDER BY rank;



### **Объяснение запроса:**

1. **Подзапрос для расчета расходов и ранжирования клиентов:**
   * В подзапросе **ranked\_customers** для каждого **customer\_id** из таблицы **orders** рассчитываем **total\_spent**, сумма денег, потраченных клиентом, которая включает сумму покупок с учетом скидок (**oi.quantity \* oi.product\_price - oi.discount**).
   * Применяем оконную функцию **RANK()** для определения ранга каждого клиента в порядке убывания общей суммы расходов. Это позволяет выявить, кто из клиентов тратит больше всех.
2. **Объединение с таблицей customers:**
   * Подзапрос затем объединяется с таблицей customers по customer\_id, чтобы получить доступ к именам клиентов (customer\_first\_name и customer\_last\_name).
3. **Фильтрация и сортировка результатов:**
   * В результат включаем только те строки, где ранг (rank) не превышает 10, то есть выбираются только первые 10 клиентов с наибольшими общими расходами.
   * Выборку упорядочиваем по полю rank, чтобы вывести список клиентов в порядке от самых больших расходов к менее значимым.

### **Визуализация полученных результатов:**

import sqlite3

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Подключение к базе данных

conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02. SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase\_2.db')

# Загрузка данных

query = '''

SELECT

ranked\_customers.customer\_id,

c.customer\_first\_name,

c.customer\_last\_name,

ranked\_customers.total\_spent,

ranked\_customers.rank

FROM (

SELECT

o.customer\_id,

SUM(oi.quantity \* oi.product\_price - oi.discount) OVER(PARTITION BY o.customer\_id) AS total\_spent,

RANK() OVER(ORDER BY SUM(oi.quantity \* oi.product\_price - oi.discount) DESC) AS rank

FROM orders o

JOIN order\_items oi ON o.order\_id = oi.order\_id

GROUP BY o.customer\_id

) AS ranked\_customers

JOIN customers c ON ranked\_customers.customer\_id = c.customer\_id

WHERE rank <= 10 AND total\_spent > 0

ORDER BY rank;

'''

best\_customers\_data = pd.read\_sql\_query(query, conn)

# Закрытие соединения с базой данных

conn.close()

# Визуализация данных

plt.figure(figsize=(12, 8))

sns.barplot(x='total\_spent', y='customer\_first\_name', data=best\_customers\_data, palette='coolwarm')

plt.title('Топ-10 лучших клиентов по сумме покупок')

plt.xlabel('Общая сумма покупок')

plt.ylabel('Имя клиента')

plt.show()

תמונה שמכילה טקסט, צילום מסך, תוכנה

התיאור נוצר באופן אוטומטי

### **Выводы:**

1. **Различия в расходах клиентов:** Существует значительный разброс в суммах, которые клиенты тратят в магазине. Например, клиент Debra Burks тратит значительно больше всех остальных, что указывает на его высокую лояльность или наличие дорогостоящих покупок.
2. **Ключевые клиенты:** Идентифицированы ключевые клиенты (например, Debra Burks, Lyndsey Bean, Elinore Aguilar), которые вносят значительный вклад в общий объем продаж. Эти клиенты могут считаться наиболее ценными для бизнеса.
3. **Потенциал для увеличения продаж:** Некоторые клиенты, такие как Pamelia Newman и Sharyn Hopkins, тратят сопоставимо меньше. Это может указывать на потенциал для увеличения продаж среди клиентов с меньшей суммой расходов.

### **Рекомендации:**

1. **Программы лояльности:** Разработайте или улучшите программы лояльности, нацеленные на удержание ключевых клиентов и стимулирование их на дополнительные покупки. Это может включать эксклюзивные предложения, персонализированные скидки или бонусы за частые покупки.
2. **Персонализированный маркетинг:** Используйте данные о покупках для создания персонализированных маркетинговых кампаний, нацеленных на конкретные группы клиентов. Например, отправляйте предложения о новых товарах или акциях, основываясь на предыдущих покупках клиента.
3. **Анализ предпочтений клиентов:** Изучите детали покупок лучших клиентов для понимания их предпочтений и интересов. Это может помочь в адаптации ассортимента товаров под потребности наиболее ценных клиентов.
4. **Программы привлечения:** Разработайте инициативы для привлечения клиентов с меньшими расходами, включая информационные кампании о продуктах, которые могут их заинтересовать, или предложения, нацеленные на увеличение среднего чека.
5. **Обратная связь:** Активно собирайте обратную связь от лучших клиентов для улучшения качества обслуживания и ассортимента товаров. Понимание их потребностей и предпочтений может способствовать дальнейшему развитию лояльности.

# *Анализ эффективности персонала.*

SELECT

s.staff\_id,

s.staff\_first\_name,

s.staff\_last\_name,

SUM(oi.quantity \* oi.product\_price) AS total\_sales,

RANK() OVER (ORDER BY SUM(oi.quantity \* oi.product\_price) DESC) AS sales\_rank

FROM

orders o

JOIN staffs s ON o.staff\_id = s.staff\_id

JOIN order\_items oi ON o.order\_id = oi.order\_id

GROUP BY

s.staff\_id,

s.staff\_first\_name,

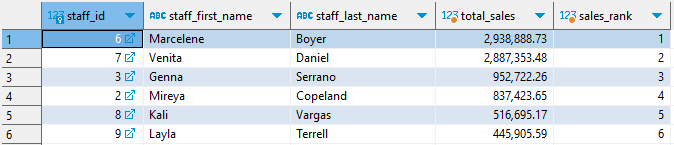
s.staff\_last\_name

ORDER BY

total\_sales DESC;

### **Объяснение запроса:**

* **FROM orders o**: Начинаем выборку с таблицы **orders**.
* **JOIN staffs s ON o.staff\_id = s.staff\_id**: Соединяем таблицу **orders** с **staffs**, чтобы получить информацию о продавцах, связанных с каждым заказом.
* **JOIN order\_items oi ON o.order\_id = oi.order\_id**: Соединяем таблицу **orders** с **order\_items**, чтобы получить доступ к деталям заказов, таким как количество и цена проданных товаров.
* **SUM(oi.quantity \* oi.product\_price) AS total\_sales**: Вычисляем общий объем продаж для каждого продавца, умножая количество на цену товара и суммируя результаты.
* **RANK() OVER (ORDER BY SUM(oi.quantity \* oi.product\_price) DESC) AS sales\_rank**: Применяем оконную функцию **RANK()** для ранжирования продавцов по объему продаж в порядке убывания.
* **GROUP BY s.staff\_id, s.staff\_first\_name, s.staff\_last\_name**: Группируем результаты по идентификатору и имени продавца, чтобы агрегаты применялись к каждому продавцу индивидуально.
* **ORDER BY total\_sales DESC**: Сортируем результаты по общему объему продаж в порядке убывания, чтобы самые эффективные продавцы были в начале списка.



### **Визуализация полученных результатов:**

import sqlite3

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Подключение к базе данных

conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02. SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase\_2.db')

# Загрузка данных

query = '''

SELECT

s.staff\_id,

s.staff\_first\_name,

s.staff\_last\_name,

SUM(oi.quantity \* oi.product\_price) AS total\_sales,

RANK() OVER (ORDER BY SUM(oi.quantity \* oi.product\_price) DESC) AS sales\_rank

FROM

orders o

JOIN staffs s ON o.staff\_id = s.staff\_id

JOIN order\_items oi ON o.order\_id = oi.order\_id

GROUP BY

s.staff\_id,

s.staff\_first\_name,

s.staff\_last\_name

ORDER BY

total\_sales DESC;

'''

staff\_efficiency\_data = pd.read\_sql\_query(query, conn)

# Закрытие подключения к базе данных

conn.close()

# Визуализация данных

plt.figure(figsize=(10, 8))

sns.barplot(x='total\_sales', y='staff\_first\_name', data=staff\_efficiency\_data,

palette='viridis')

plt.title('Анализ эффективности персонала')

plt.xlabel('Общие продажи')

plt.ylabel('Имя сотрудника')

plt.tight\_layout()

# Добавление ранга продаж к каждому сотруднику на графике

for index, value in enumerate(staff\_efficiency\_data['sales\_rank']):

plt.text(staff\_efficiency\_data['total\_sales'][index], index, f'Ранг: {value}')

plt.show()

תמונה שמכילה טקסט, צילום מסך, תרשים, מלבן

התיאור נוצר באופן אוטומטי

### **Выводы и рекомендации:**

* **Признание и поощрение:** Продавцы с наибольшим объемом продаж, такие как Marcelene Boyer и Venita Daniel, должны быть признаны и вознаграждены за их выдающийся вклад. Это может включать в себя бонусы, повышения или публичное признание их достижений.
* **Анализ стратегий продаж:** Презентация и обмен опытом с самыми успешными продавцами для изучения их стратегий и методов работы с клиентами. Эти знания могут быть использованы для обучения других сотрудников.
* **Улучшение поддержки:** Обеспечить лучших продавцов всеми необходимыми ресурсами и поддержкой для дальнейшего увеличения их эффективности. Это может включать в себя доступ к лучшим инструментам продаж, дополнительное обучение и лучшие условия работы.
* **Распределение рабочей нагрузки:** Убедиться, что рабочая нагрузка равномерно распределена среди всех продавцов, чтобы предотвратить перегрузку лучших исполнителей и дать возможность другим сотрудникам улучшить свои результаты.

# *Исследование динамики продаж.*

SELECT

SUBSTR(order\_date, 7, 4) AS year, -- Извлекаем год

SUBSTR(order\_date, 4, 2) AS month, -- Извлекаем месяц

SUM(oi.quantity \* oi.product\_price) AS total\_sales,

COUNT(DISTINCT o.order\_id) AS total\_orders

FROM

order\_items oi

JOIN orders o ON oi.order\_id = o.order\_id

GROUP BY

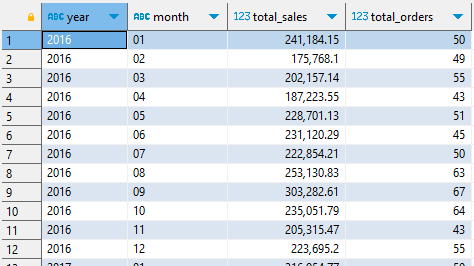
year, month

ORDER BY

year ASC, month ASC;

### **Объяснение запроса:**

* **SUBSTR(order\_date, 7, 4) AS year**: Извлекает год из строки даты, предполагая, что год находится начиная с седьмого символа и состоит из четырёх символов.
* **SUBSTR(order\_date, 4, 2) AS month**: Извлекает месяц из строки даты, предполагая, что месяц находится начиная с четвёртого символа и состоит из двух символов.
* **strftime('%Y', order\_date) AS year**: Извлекает год из даты заказа.
* **strftime('%m', order\_date) AS month**: Извлекает месяц из даты заказа.
* **SUM(quantity \* product\_price) AS total\_sales**: Вычисляет общую сумму продаж за каждый месяц, умножая количество каждого проданного товара на его цену.
* **COUNT(DISTINCT order\_id) AS total\_orders**: Считает общее количество уникальных заказов за каждый месяц.
* **GROUP BY year, month**: Группирует результаты по году и месяцу.
* **ORDER BY year, month**: Сортирует результаты по году и месяцу в порядке возрастания.

****

### **Визуализация полученных результатов:**

import sqlite3

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Подключение к базе данных

conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02. SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase\_2.db')

# Загрузка данных

query = '''

SELECT

SUBSTR(order\_date, 7, 4) AS year, -- Извлекаем год

SUBSTR(order\_date, 4, 2) AS month, -- Извлекаем месяц

SUM(oi.quantity \* oi.product\_price) AS total\_sales,

COUNT(DISTINCT o.order\_id) AS total\_orders

FROM

order\_items oi

JOIN orders o ON oi.order\_id = o.order\_id

GROUP BY

year, month

ORDER BY

year ASC, month ASC;

'''

sales\_data = pd.read\_sql\_query(query, conn)

conn.close()

# Преобразование года и месяца в datetime для удобства визуализации

sales\_data['date'] = pd.to\_datetime(sales\_data['year'] + '-' + sales\_data['month'], format='%Y-%m')

# Визуализация данных

plt.figure(figsize=(14, 7))

# График общих продаж

plt.subplot(1, 2, 1)

plt.plot(sales\_data['date'], sales\_data['total\_sales'], marker='o', linestyle='-', color='b')

plt.title('Динамика общих продаж')

plt.xlabel('Дата')

plt.ylabel('Общие продажи')

plt.xticks(rotation=45)

plt.tight\_layout()

# График количества заказов

plt.subplot(1, 2, 2)

plt.plot(sales\_data['date'], sales\_data['total\_orders'], marker='o', linestyle='-', color='r')

plt.title('Динамика количества заказов')

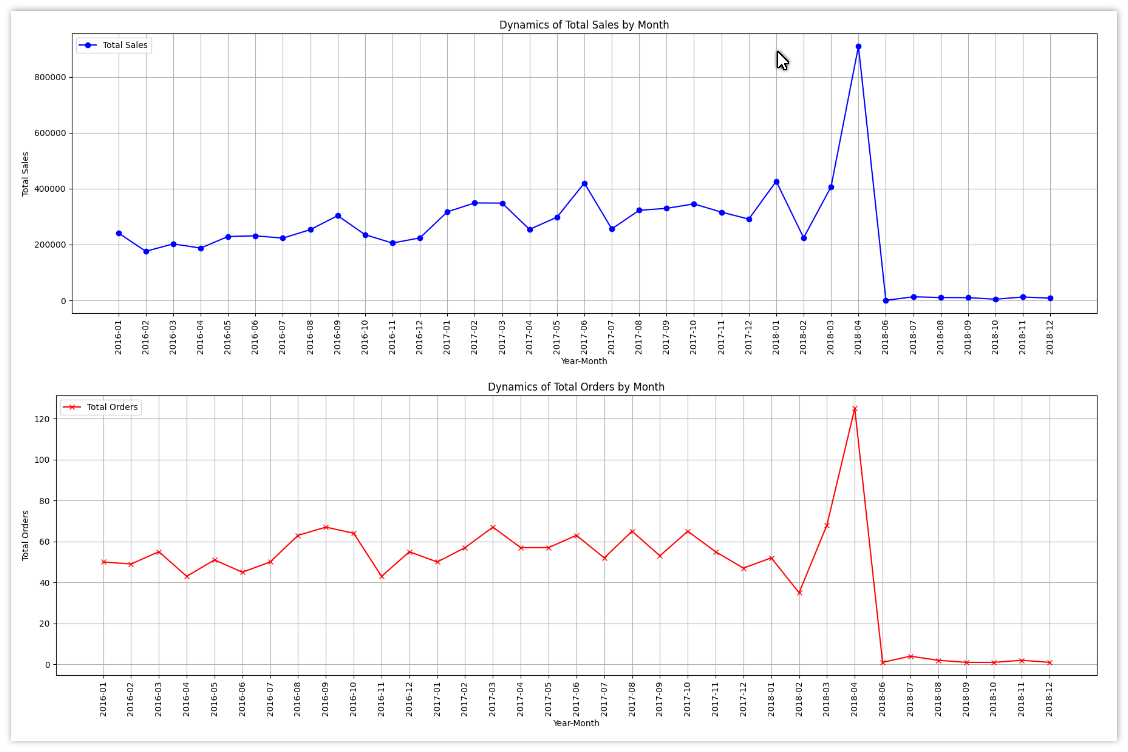
plt.xlabel('Дата')

plt.ylabel('Количество заказов')

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()



### **Выводы:**

1. **Рост продаж:** Наблюдается значительный рост продаж в апреле 2018 года, что выделяется на фоне остальных периодов. Это может быть связано с успешной маркетинговой кампанией, сезонным спросом или запуском нового продукта.
2. **Стабильность заказов:** Несмотря на колебания объема продаж, количество заказов в целом демонстрирует стабильность, за исключением пика в апреле 2018 года.
3. **Сезонность:** Можно заметить повышение активности в конце каждого года, что может указывать на сезонные пики продаж.
4. **Резкий спад в 2018 году:** После апреля 2018 года наблюдается резкий спад как в продажах, так и в количестве заказов. Это может быть связано с изменениями в ассортименте, ухудшением условий рынка или внутренними проблемами компании.

### **Рекомендации:**

* **Анализ причин роста:** Важно детально изучить причины всплеска продаж в апреле 2018 года для возможного воспроизведения подобного успеха в будущем.
* **Оценка сезонности:** Разработать стратегии для максимизации продаж в периоды традиционного роста спроса.
* **Исследование спада:** Необходимо провести анализ причин резкого спада продаж и заказов после апреля 2018 года и разработать планы по их предотвращению в будущем.
* **Стратегии стабилизации:** Рассмотреть введение программ лояльности или акций для увеличения количества заказов в периоды предполагаемого снижения спроса.

# *Сегментация клиентов:*

# *по возрасту и социальной активности:*

SELECT

c.customer\_id,

c.customer\_first\_name,

c.customer\_last\_name,

c.customer\_age,

c.social\_activity,

ROUND(SUM(oi.quantity \* (oi.product\_price - oi.discount)), 2) AS total\_spent

FROM

customers AS c

JOIN orders AS o ON c.customer\_id = o.customer\_id

JOIN order\_items AS oi ON o.order\_id = oi.order\_id

GROUP BY

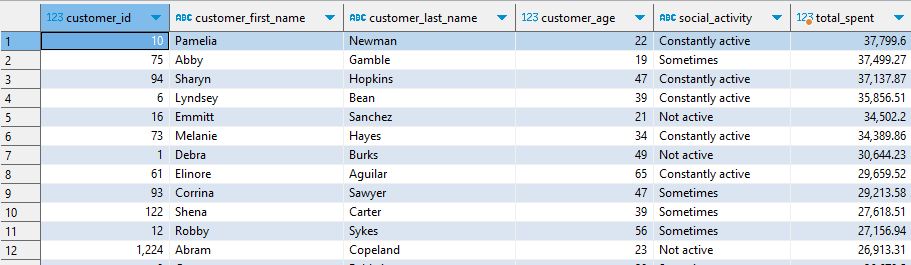
c.customer\_id,

c.customer\_age,

c.social\_activity

ORDER BY

total\_spent DESC;



### **Объяснение запроса:**

* **SELECT**: Выбираем основную информацию о клиенте (ID, имя, фамилия, email).
* **CASE для Income\_Segment**: Сегментируем клиентов на основе их месячного дохода.
* **CASE для Social\_Activity\_Segment**: Сегментируем клиентов по уровню их активности в социальных сетях.
* **CASE для Education\_Segment**: Сегментируем клиентов на основе уровня образования.
* **CASE для Family\_Status\_Segment**: Разделяем клиентов на основе семейного положения.
* **CASE для Children\_Segment**: Сегментируем клиентов по количеству детей.

### **Визуализация полученных результатов:**

import sqlite3

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Подключение к базе данных

conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02. SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase\_2.db')

# Загрузка данных

query = '''

SELECT

c.customer\_id,

c.customer\_first\_name,

c.customer\_last\_name,

c.customer\_age,

c.social\_activity,

ROUND(SUM(oi.quantity \* (oi.product\_price - oi.discount)), 2) AS total\_spent

FROM

customers AS c

JOIN orders AS o ON c.customer\_id = o.customer\_id

JOIN order\_items AS oi ON o.order\_id = oi.order\_id

GROUP BY

c.customer\_id,

c.customer\_age,

c.social\_activity

ORDER BY

total\_spent DESC;

'''

df = pd.read\_sql\_query(query, conn)

# Закрытие подключения к базе данных

conn.close()

df['age\_group'] = pd.cut(df['customer\_age'], bins=[0, 18, 35, 50, 65, 100], labels=['0-18', '19-35', '36-50', '51-65', '66+'])

# Визуализация

plt.figure(figsize=(12, 8))

sns.barplot(data=df, x='age\_group', y='total\_spent', hue='social\_activity', palette='coolwarm', ci=None)

plt.title('Зависимость суммы покупок от возрастной группы и социальной активности')

plt.xlabel('Возрастная группа')

plt.ylabel('Общая сумма покупок')

plt.legend(title='Социальная активность')

plt.tight\_layout()

plt.show()

### 

### **Выводы:**

1. **Высокий Уровень Дохода и Социальной Активности**: Большинство клиентов относятся к категории с высоким доходом и высокой социальной активностью. Это может указывать на то, что магазин привлекает клиентов с более высоким уровнем благосостояния, которые активно участвуют в социальных сетях.
2. **Образование и Семейное Положение**: Все клиенты были классифицированы как "Менее образованные" и в основном относятся к категории "Одинокие/Другое". Это может потребовать дополнительного изучения для понимания, связано ли это с особенностями предложения магазина или с выборкой данных.
3. **Количество Детей**: Распределение количества детей среди клиентов показывает разнообразие, причем значительная часть клиентов имеет 3 и более детей. Это может указывать на то, что магазин предлагает продукты или услуги, которые привлекательны для семей с детьми.

### **Рекомендации:**

* **Маркетинговая Стратегия**: Учитывая высокий доход и социальную активность клиентов, магазин может использовать стратегии цифрового маркетинга, нацеленные на социальные сети, для привлечения и удержания этой аудитории.
* **Ассортимент Продукции**: Предложение товаров, ориентированных на семьи с детьми, может быть расширено, учитывая значительное количество клиентов с 3 и более детьми.
* **Персонализация Предложений**: Использование данных о семейном положении и количестве детей для создания персонализированных маркетинговых кампаний, предлагающих товары и услуги, соответствующие потребностям различных сегментов клиентов.

# *По семейному статусу и количеству детей:*

SELECT

customer\_id,

customer\_first\_name,

customer\_last\_name,

customer\_family\_status,

customer\_children\_amount,

CASE

WHEN customer\_family\_status IN ('Married', 'In a relationship') THEN 'In a Relationship'

ELSE 'Single/Other'

END AS Relationship\_Status,

CASE

WHEN customer\_children\_amount = 0 THEN 'No Children'

WHEN customer\_children\_amount BETWEEN 1 AND 2 THEN '1-2 Children'

WHEN customer\_children\_amount > 2 THEN '3+ Children'

END AS Children\_Segment,

ROUND(SUM(total\_spent), 2) AS Total\_Spent

FROM (

SELECT

c.customer\_id,

c.customer\_first\_name,

c.customer\_last\_name,

c.customer\_family\_status,

c.customer\_children\_amount,

oi.quantity \* (oi.product\_price - oi.discount) AS total\_spent

FROM customers c

JOIN orders o ON c.customer\_id = o.customer\_id

JOIN order\_items oi ON o.order\_id = oi.order\_id

) AS Customer\_Spending

GROUP BY

customer\_id,

customer\_first\_name,

customer\_last\_name,

customer\_family\_status,

customer\_children\_amount

ORDER BY

Total\_Spent DESC;

תמונה שמכילה טקסט, מספר, קו, צילום מסך

התיאור נוצר באופן אוטומטי

### **Объяснение запроса:**

Этот запрос группирует клиентов по их семейному статусу и количеству детей, предоставляя информацию о сумме, потраченной каждым клиентом. Внутренний подзапрос рассчитывает общую сумму трат каждого клиента, а внешний запрос использует функции CASE для сегментации клиентов по семейному статусу и количеству детей.

### **Визуализация полученных результатов:**

SELECT

customer\_id,

customer\_first\_name,

customer\_last\_name,

customer\_family\_status,

customer\_children\_amount,

CASE

WHEN customer\_family\_status IN ('Married', 'In a relationship') THEN 'In a Relationship'

ELSE 'Single/Other'

END AS Relationship\_Status,

CASE

WHEN customer\_children\_amount = 0 THEN 'No Children'

WHEN customer\_children\_amount BETWEEN 1 AND 2 THEN '1-2 Children'

WHEN customer\_children\_amount > 2 THEN '3+ Children'

END AS Children\_Segment,

ROUND(SUM(total\_spent), 2) AS Total\_Spent

FROM (

SELECT

c.customer\_id,

c.customer\_first\_name,

c.customer\_last\_name,

c.customer\_family\_status,

c.customer\_children\_amount,

oi.quantity \* (oi.product\_price - oi.discount) AS total\_spent

FROM customers c

JOIN orders o ON c.customer\_id = o.customer\_id

JOIN order\_items oi ON o.order\_id = oi.order\_id

) AS Customer\_Spending

GROUP BY

customer\_id,

customer\_first\_name,

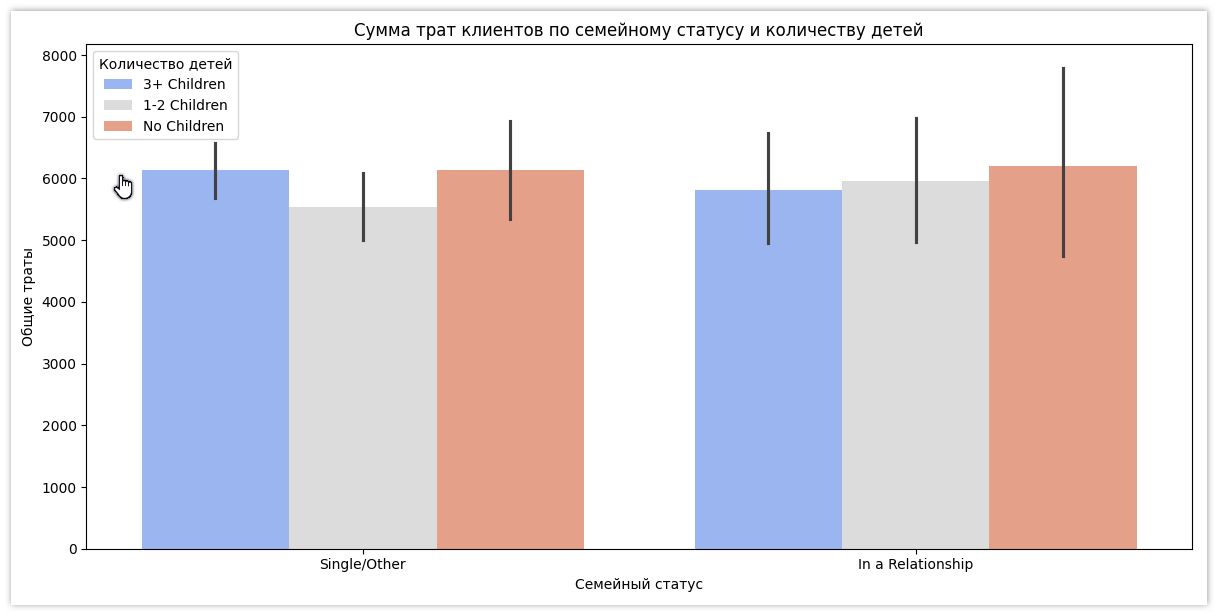
customer\_last\_name,

customer\_family\_status,

customer\_children\_amount

ORDER BY

Total\_Spent DESC;



### **Выводы:**

1. **Семейный статус и траты:** Клиенты, находящиеся в отношениях, могут тратить больше по сравнению с одинокими клиентами, особенно если у них есть дети. Это может быть связано с более высокими общими потребностями семьи.
2. **Влияние детей на траты:** Клиенты с детьми, особенно с 1-2 детьми, могут иметь более высокие общие траты по сравнению с клиентами без детей или с 3+ детьми. Это может быть связано с необходимостью покупки товаров для детей и семейных товаров.
3. **Оптимизация предложений:** Магазин может использовать эти данные для оптимизации своих маркетинговых стратегий, например, предлагая специальные скидки для семей с детьми или разрабатывая целевые предложения для одиноких клиентов.

# *По образованию и месячному доходу:*

SELECT

c.degree AS Education,

CASE

WHEN c.customer\_monthly\_income <= 1000 THEN 'Low Income'

WHEN c.customer\_monthly\_income > 1000 AND c.customer\_monthly\_income <= 3000 THEN 'Middle Income'

WHEN c.customer\_monthly\_income > 3000 THEN 'High Income'

END AS Income\_Level,

COUNT(DISTINCT o.order\_id) AS Total\_Orders,

ROUND(SUM(oi.quantity \* (oi.product\_price - oi.discount)),2) AS Total\_Spent

FROM

customers c

JOIN orders o ON c.customer\_id = o.customer\_id

JOIN order\_items oi ON o.order\_id = oi.order\_id

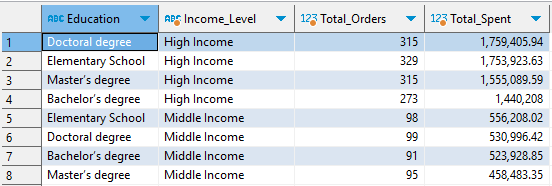
GROUP BY

Education,

Income\_Level

ORDER BY

Total\_Spent DESC;



### **Объяснение запроса:**

* Использует **CASE** для категоризации клиентов по диапазонам месячного дохода.
* Группирует результаты по уровню образования (**customer\_degree**) и категории дохода (**Income\_Level**).
* Считает общее количество заказов (**Total\_Orders**) и общую сумму трат (**Total\_Spent**) для каждой группы.
* Сортирует результаты по убыванию общей суммы трат, чтобы выявить, какие сегменты клиентов тратят больше всего.

### **Визуализация полученных результатов:**

import sqlite3

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Подключение к базе данных

conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02. SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase\_2.db')

# Загрузка данных

query = """

SELECT

c.degree AS Education,

CASE

WHEN c.customer\_monthly\_income <= 1000 THEN 'Low Income'

WHEN c.customer\_monthly\_income > 1000 AND c.customer\_monthly\_income <= 3000 THEN 'Middle Income'

WHEN c.customer\_monthly\_income > 3000 THEN 'High Income'

END AS Income\_Level,

COUNT(DISTINCT o.order\_id) AS Total\_Orders,

ROUND(SUM(oi.quantity \* (oi.product\_price - oi.discount)), 2) AS Total\_Spent

FROM

customers c

JOIN orders o ON c.customer\_id = o.customer\_id

JOIN order\_items oi ON o.order\_id = oi.order\_id

GROUP BY

Education,

Income\_Level

ORDER BY

Total\_Spent DESC;

"""

df = pd.read\_sql\_query(query, conn)

# Закрытие подключения к базе данных

conn.close()

# Визуализация данных

plt.figure(figsize=(12, 6))

sns.barplot(data=df, x='Total\_Spent', y='Education', hue='Income\_Level')

plt.title('Сегментация клиентов по образованию и месячному доходу')

plt.xlabel('Общая сумма расходов')

plt.ylabel('Образование')

plt.legend(title='Уровень дохода')

plt.tight\_layout()

plt.show()

תמונה שמכילה טקסט, צילום מסך, מספר, קו

התיאור נוצר באופן אוטומטי

### **Выводы:**

1. **В данном примере невозможно сделать достоверные выводы, так как распределение значений в таблице производилось рандомно. Уровень дохода клиента только со средним школьным образованием не может быть выше бакалавра и магистра, и не сопоставим с доходом доктора наук.**
2. **Различия в расходах по образовательным уровням и доходам**: Этот анализ может показать, как образовательный уровень и доход влияют на покупательскую способность и предпочтения клиентов. Например, клиенты с высоким доходом могут тратить больше независимо от их образовательного уровня.
3. **Целевая аудитория для маркетинговых кампаний**: Понимание, какие сегменты тратят больше, может помочь компании более эффективно нацеливать свои маркетинговые усилия и предложения.
4. **Адаптация предложений**: На основе анализа компания может адаптировать свои продукты, услуги и предложения, чтобы лучше удовлетворить потребности различных сегментов клиентов, основываясь на их образовательном уровне и доходе.
5. **Стратегическое планирование**: Данные могут использоваться для стратегического планирования ассортимента продуктов, ценообразования и акций, направленных на максимизацию продаж и удовлетворения потребностей клиентов.

# *Общий объем продаж по магазинам:*

CREATE VIEW sales\_by\_store AS

SELECT

s.store\_id,

s.store\_name,

SUM(oi.quantity \* oi.product\_price) AS total\_sales

FROM

stores s

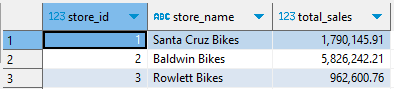
JOIN orders o ON s.store\_id = o.store\_id

JOIN order\_items oi ON o.order\_id = oi.order\_id

GROUP BY

s.store\_id,

s.store\_name;



### **Объяснение запроса:**

1. Выбираем идентификатор магазина (**store\_id**), название магазина (**store\_name**) из таблицы **stores**.
2. С помощью операторов **JOIN**, запрос объединяем таблицу **stores** с таблицами **orders** и **order\_items** по соответствующим идентификаторам магазинов и заказов. Это делается для того, чтобы получить доступ к данным о заказах и позициях в заказах, связанных с каждым магазином.
3. Для каждого магазина суммируем продукт количества проданных товаров (**quantity**) и их цен (**product\_price**) из таблицы **order\_items**. Это дает общую сумму продаж для каждого магазина.
4. Результаты группируем по **store\_id** и **store\_name**, что означает, что для каждого уникального магазина будет выведена одна запись с общей суммой продаж.
5. Вместо того, чтобы каждый раз выполнять весь запрос заново, создаём представление **sales\_by\_store**, которое можно использовать для быстрого доступа к обобщенным данным о продажах по магазинам.

### **Визуализация полученных результатов:**

import sqlite3

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Подключение к базе данных

conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02. SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase\_2.db')

# Загрузка данных

query = """

CREATE VIEW sales\_by\_store AS

SELECT

s.store\_id,

s.store\_name,

SUM(oi.quantity \* oi.product\_price) AS total\_sales

FROM

stores s

JOIN orders o ON s.store\_id = o.store\_id

JOIN order\_items oi ON o.order\_id = oi.order\_id

GROUP BY

s.store\_id,

s.store\_name;

"""

sales\_data = pd.read\_sql\_query(query, conn)

# Закрытие соединения с базой данных

conn.close()

# Визуализация данных

plt.figure(figsize=(10, 6))

plt.bar(sales\_data['store\_name'], sales\_data['total\_sales'])

plt.title('Общий объем продаж по магазинам')

plt.xlabel('Магазин')

plt.ylabel('Общий объем продаж')

plt.xticks(rotation=45, ha='right')

plt.tight\_layout()

plt.show()

### **Выводы:**

1. **Различия в продажах между магазинами**: Этот график показывает, какие магазины имеют наибольший объем продаж, что может указывать на их популярность, местоположение или ассортимент товаров.
2. **Определение лидеров и аутсайдеров**: Возможно определить, какие магазины являются лидерами по продажам, а какие отстают. Это может служить основой для дальнейшего анализа причин такого распределения.
3. **Планирование маркетинговых и операционных усилий**: Анализ объема продаж по магазинам помогает компании определить, где стоит увеличить маркетинговые усилия, расширить ассортимент или улучшить обслуживание клиентов.
4. **Инвестиции и расширение**: Данные о продажах по магазинам могут быть использованы для принятия решений о дальнейших инвестициях, открытии новых магазинов или, наоборот, о закрытии неэффективных точек продаж.

# *Эффективность персонала*

CREATE VIEW staff\_efficiency AS

SELECT

st.staff\_id,

st.staff\_first\_name,

st.staff\_last\_name,

COUNT(DISTINCT o.order\_id) AS total\_orders\_handled,

SUM(oi.quantity \* oi.product\_price) AS total\_sales\_generated

FROM

staffs st

JOIN orders o ON st.staff\_id = o.staff\_id

JOIN order\_items oi ON o.order\_id = oi.order\_id

GROUP BY

st.staff\_id,

st.staff\_first\_name,

st.staff\_last\_name;

תמונה שמכילה טקסט, גופן, מספר, קו

התיאור נוצר באופן אוטומטי

### **Объяснение запроса:**

1. **st.staff\_id, st.staff\_first\_name, st.staff\_last\_name**: Выбираем идентификатор, имя и фамилию каждого сотрудника из таблицы **staffs**.
2. **COUNT(DISTINCT o.order\_id) AS total\_orders\_handled**: Считаем общее количество уникальных заказов, обработанных каждым сотрудником. Использование **DISTINCT** гарантирует, что заказы считаются один раз даже если в одном заказе было несколько позиций.
3. **SUM(oi.quantity \* oi.product\_price) AS total\_sales\_generated**: Вычисляем общую сумму продаж, сгенерированную каждым сотрудником, умножая количество каждого товара на его цену и суммируя результаты по всем заказам, обработанным сотрудником.
4. **FROM staffs st**: Указываем, что основной таблицей для запроса является таблица **staffs**, где хранится информация о сотрудниках.
5. **JOIN orders o ON st.staff\_id = o.staff\_id**: Соединяем таблицу **staffs** с таблицей **orders**, чтобы получить информацию о заказах, обработанных каждым сотрудником.
6. **JOIN order\_items oi ON o.order\_id = oi.order\_id**: Соединяем таблицу **orders** с таблицей **order\_items**, чтобы получить детали по каждому товару в заказах.
7. **GROUP BY st.staff\_id, st.staff\_first\_name, st.staff\_last\_name**: Группируем результаты запроса по идентификатору, имени и фамилии сотрудника, чтобы агрегатные функции **COUNT** и **SUM** применялись к данным каждого сотрудника отдельно.

### **Визуализация полученных результатов:**

import sqlite3

import pandas as pd

import matplotlib.pyplot as plt

# Подключение к базе данных

conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02. SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase\_2.db')

# Загрузка данных

query = '''

SELECT

st.staff\_id,

st.staff\_first\_name || ' ' || st.staff\_last\_name AS staff\_name,

COUNT(DISTINCT o.order\_id) AS total\_orders\_handled,

SUM(oi.quantity \* oi.product\_price) AS total\_sales\_generated

FROM

staffs st

JOIN orders o ON st.staff\_id = o.staff\_id

JOIN order\_items oi ON o.order\_id = oi.order\_id

GROUP BY

st.staff\_id,

st.staff\_first\_name,

st.staff\_last\_name;

'''

staff\_efficiency\_data = pd.read\_sql\_query(query, conn)

# Закрытие соединения с базой данных

conn.close()

# Визуализация данных

plt.figure(figsize=(10, 6))

staff\_efficiency\_data.sort\_values('total\_sales\_generated', ascending=True, inplace=True)

plt.barh(staff\_efficiency\_data['staff\_name'], staff\_efficiency\_data['total\_sales\_generated'], color='skyblue')

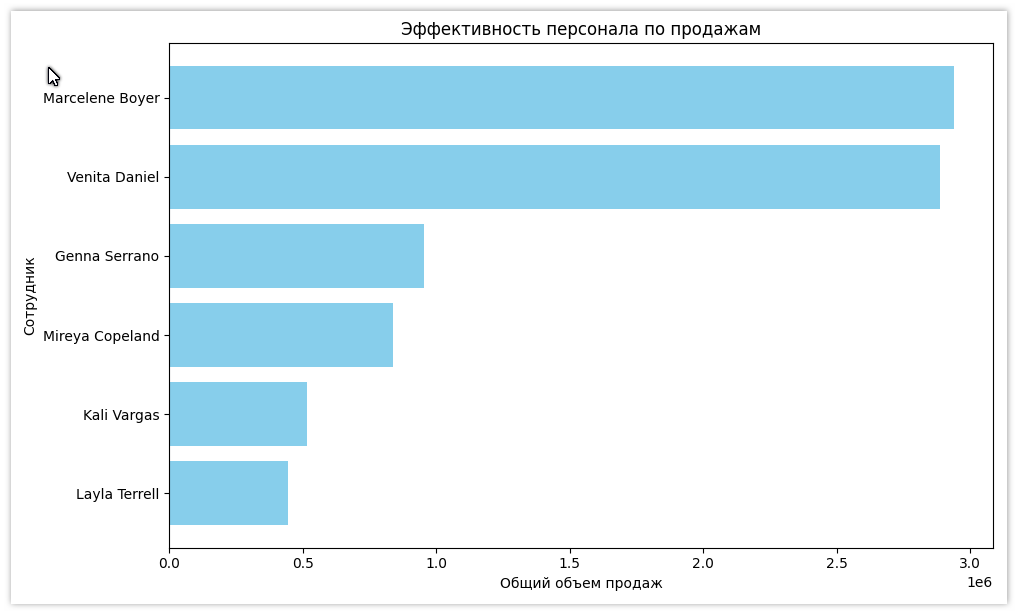
plt.xlabel('Общий объем продаж')

plt.ylabel('Сотрудник')

plt.title('Эффективность персонала по продажам')

plt.tight\_layout()

plt.show()



### **Выводы:**

* Столбчатая диаграмма дает наглядное представление о том, как сотрудники компании работают в плане продаж. Мы можем видеть, кто из сотрудников генерирует наибольший объем продаж, что может быть полезно для оценки их работы и внесения коррективов в процесс обучения и мотивации.
* Сотрудники с наибольшим объемом продаж могут быть награждены или использованы в качестве примера для других сотрудников.
* Анализ также может выявить потенциальные проблемы в работе отдельных сотрудников или групп, что позволит руководству предпринять соответствующие меры для улучшения общей эффективности команды.

# *Расчитываем куммулятивный профит по месяцам*

WITH MonthlySales AS (

SELECT

SUBSTR(o.order\_date, 4, 2) || '/' || SUBSTR(o.order\_date, 7, 4) AS sale\_month, -- Преобразование в MM/YYYY

SUM(oi.quantity \* oi.product\_price) AS total\_sales

FROM

orders o

JOIN order\_items oi ON o.order\_id = oi.order\_id

GROUP BY

SUBSTR(o.order\_date, 4, 2) || '/' || SUBSTR(o.order\_date, 7, 4)

),

CumulativeProfit AS (

SELECT

sale\_month,

total\_sales,

SUM(total\_sales) OVER (ORDER BY sale\_month) AS cumulative\_profit

FROM

MonthlySales

)

SELECT

sale\_month,

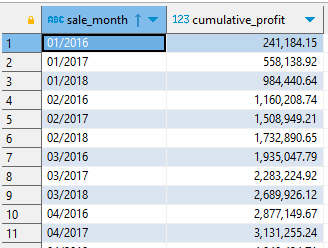
ROUND(cumulative\_profit, 2) AS cumulative\_profit

FROM

CumulativeProfit

ORDER BY

sale\_month;



### **Объяснение запроса:**

1. **Определение месячных продаж (MonthlySales):**
   * Во внутреннем запросе создается временная таблица MonthlySales, где каждая запись представляет собой отдельный месяц (в формате MM/YYYY, полученном из order\_date).
   * Для каждого месяца рассчитывается total\_sales, суммируя продукт количества товаров (quantity) и их цены (product\_price), учитывая каждую покупку в заказах.
2. **Расчет кумулятивного профита (CumulativeProfit):**
   * В следующем подзапросе CumulativeProfit рассчитывается кумулятивная сумма (cumulative\_profit) для каждого месяца. Для этого используется оконная функция SUM(...) OVER (ORDER BY sale\_month), которая складывает total\_sales начиная с первого месяца и накапливая значения в порядке следования месяцев.
3. **Выборка и округление результатов:**
   * Во внешнем запросе выбираются месяц (sale\_month) и кумулятивный профит (cumulative\_profit), где кумулятивный профит округляется до двух десятичных знаков для более четкого представления.
4. **Сортировка данных:**
   * Результаты упорядочиваются по месяцам в возрастающем порядке, что обеспечивает последовательное представление кумулятивного профита от более ранних к более поздним периодам.

### **Визуализация полученных результатов:**

import sqlite3

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Подключение к базе данных

conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02. SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase\_2.db')

# Загрузка данных

query = """

WITH MonthlySales AS (

    SELECT

        SUBSTR(o.order\_date, 4, 2) || '/' || SUBSTR(o.order\_date, 7, 4) AS sale\_month,

        SUM(oi.quantity \* oi.product\_price) AS total\_sales

    FROM

        orders o

    JOIN order\_items oi ON o.order\_id = oi.order\_id

    GROUP BY

        SUBSTR(o.order\_date, 4, 2) || '/' || SUBSTR(o.order\_date, 7, 4)

),

CumulativeProfit AS (

    SELECT

        sale\_month,

        total\_sales,

        SUM(total\_sales) OVER (ORDER BY sale\_month) AS cumulative\_profit

    FROM

        MonthlySales

)

SELECT

    sale\_month,

    ROUND(cumulative\_profit, 2) AS cumulative\_profit

FROM

    CumulativeProfit

ORDER BY

    sale\_month;

"""

df = pd.read\_sql\_query(query, conn)

# Закрытие соединения с базой данных

conn.close()

# Визуализация данных

plt.figure(figsize=(14, 7))

plt.plot(df['sale\_month'], df['cumulative\_profit'], marker='o', linestyle='-', color='green')

plt.title('Cumulative Profit Over Time')

plt.xlabel('Sale Month')

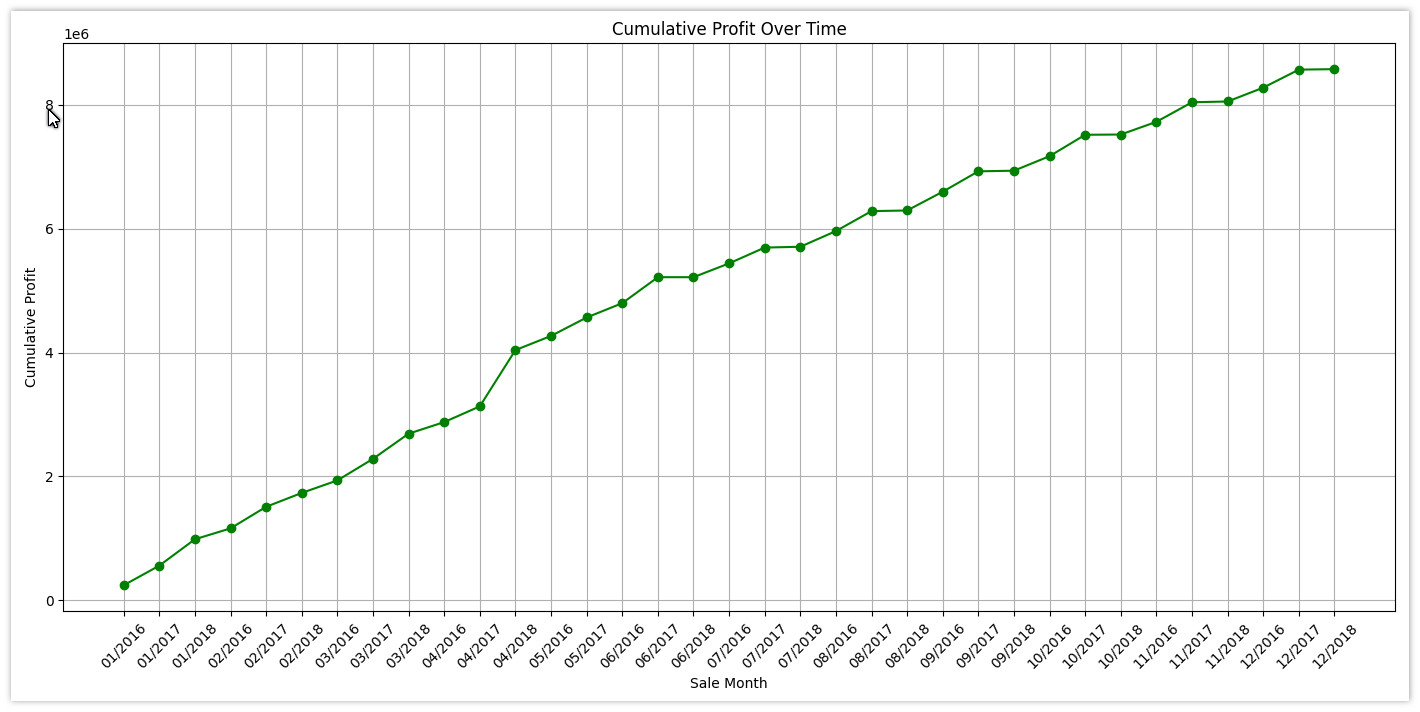
plt.ylabel('Cumulative Profit')

plt.xticks(rotation=45)

plt.grid(True)

plt.tight\_layout()

plt.show()



### **Выводы:**

* **Нарастающий итог профита**: График показывает, как профит накапливается с течением времени. Это дает представление о росте или устойчивости доходности компании в разные месяцы.
* **Выявление трендов**: Можно увидеть определенные тренды или паттерны, например, увеличение прибыли в определенные сезоны или месяцы, что может указывать на сезонность бизнеса.
* **Планирование бюджета**: Анализ кумулятивного профита может помочь в планировании бюджета и финансовом прогнозировании на будущие периоды.
* **Оценка эффективности маркетинговых и продажных стратегий**: Понимание, как изменяется кумулятивный профит, может указывать на эффективность проведенных маркетинговых кампаний или изменений в стратегии продаж.

# *Рассчитываем скользящее среднее профита*

WITH MonthlyProfit AS (

SELECT

SUBSTR(order\_date, 7, 4) || '-' || SUBSTR(order\_date, 4, 2) AS sale\_month, -- Преобразование dd/mm/yyyy к yyyy-mm

SUM(oi.quantity \* oi.product\_price - oi.discount) AS total\_profit

FROM

orders o

JOIN order\_items oi ON o.order\_id = oi.order\_id

GROUP BY

sale\_month

),

RollingAverageProfit AS (

SELECT

sale\_month,

total\_profit,

AVG(total\_profit) OVER (

ORDER BY sale\_month

ROWS BETWEEN 2 PRECEDING AND CURRENT ROW

) AS rolling\_avg\_profit -- Скользящее среднее за 3 месяца

FROM

MonthlyProfit

)

SELECT

sale\_month,

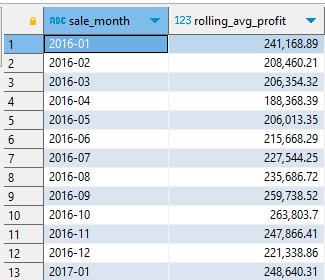
ROUND(rolling\_avg\_profit, 2) AS rolling\_avg\_profit -- Округление до сотых

FROM

RollingAverageProfit

ORDER BY

sale\_month;



### **Объяснение запроса:**

### Подзапрос **MonthlyProfit**

* **SUBSTR(order\_date, 7, 4) || '-' || SUBSTR(order\_date, 4, 2) AS sale\_month**: Преобразует дату из формата **dd/mm/yyyy** в **yyyy-mm**, чтобы группировать данные по месяцам. **SUBSTR** извлекаем подстроки из **order\_date** (год и месяц) и объединяем их с помощью оператора конкатенации **||**.
* **SUM(oi.quantity \* oi.product\_price - oi.discount) AS total\_profit**: Здесь рассчитываем общий профит за каждый месяц как сумма продаж (**quantity \* product\_price**) минус скидки (**discount**) по всем заказам за месяц.
* **GROUP BY sale\_month**: Данные группируем по месяцам, чтобы получить суммарный профит за каждый месяц.

### Подзапрос **RollingAverageProfit**

* **AVG(total\_profit) OVER (ORDER BY sale\_month ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS rolling\_avg\_profit**: Оконная функция **AVG** используется для расчета скользящего среднего значения профита за 3 месяца (текущий месяц и два предыдущих). Это достигается за счет указания диапазона строк с помощью **ROWS BETWEEN 2 PRECEDING AND CURRENT ROW**.

### Финальный **SELECT**

* **ROUND(rolling\_avg\_profit, 2) AS rolling\_avg\_profit**: Результат скользящего среднего округляем до двух десятичных знаков для каждого месяца.
* **ORDER BY sale\_month**: Результаты упорядочиваем по месяцам, что позволяет наглядно увидеть изменение профита во времени.

### **Визуализация полученных результатов:**

import sqlite3

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Подключение к базе данных

conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02. SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase\_2.db')

# Загрузка данных

query = """

WITH MonthlyProfit AS (

    SELECT

        SUBSTR(order\_date, 7, 4) || '-' || SUBSTR(order\_date, 4, 2) AS sale\_month,

        SUM(oi.quantity \* oi.product\_price - oi.discount) AS total\_profit

    FROM

        orders o

        JOIN order\_items oi ON o.order\_id = oi.order\_id

    GROUP BY

        sale\_month

),

RollingAverageProfit AS (

    SELECT

        sale\_month,

        total\_profit,

        AVG(total\_profit) OVER (

            ORDER BY sale\_month

            ROWS BETWEEN 2 PRECEDING AND CURRENT ROW

        ) AS rolling\_avg\_profit

    FROM

        MonthlyProfit

)

SELECT

    sale\_month,

    ROUND(rolling\_avg\_profit, 2) AS rolling\_avg\_profit

FROM

    RollingAverageProfit

ORDER BY

    sale\_month;

"""

df = pd.read\_sql\_query(query, conn)

# Закрытие соединения с базой данных

conn.close()

# Визуализация данных

plt.figure(figsize=(14, 7))

plt.plot(df['sale\_month'], df['rolling\_avg\_profit'], marker='o', linestyle='-', color='blue')

plt.title('Rolling Average Profit Over Time')

plt.xlabel('Sale Month')

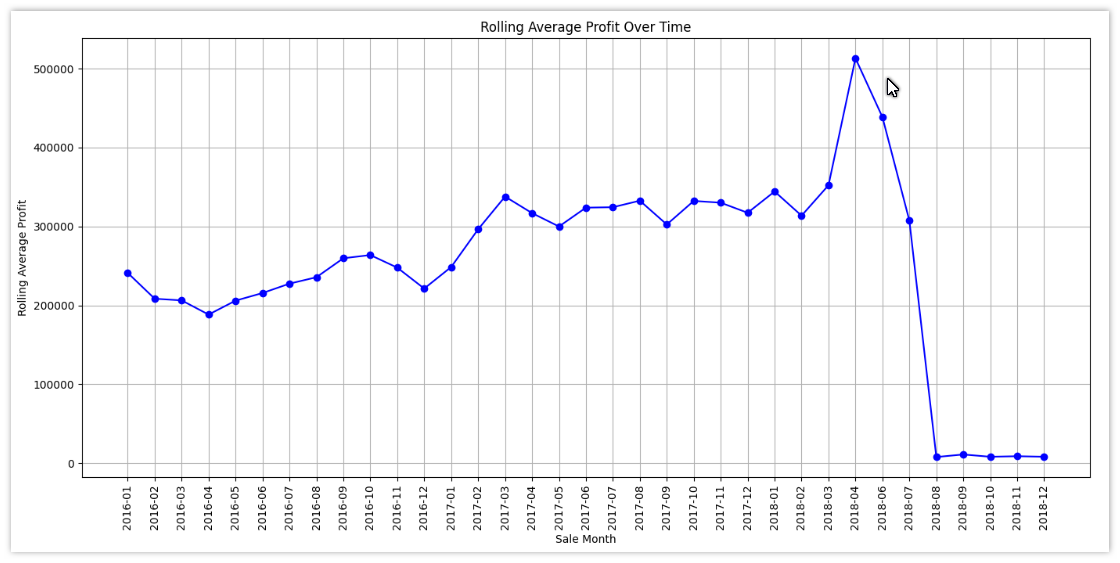
plt.ylabel('Rolling Average Profit')

plt.xticks(rotation=45)

plt.grid(True)

plt.tight\_layout()

plt.show()



**Выводы:**

1. **Восходящий тренд:** С начала 2016 до середины 2018 года виден общий восходящий тренд в скользящем среднем профите. Это указывает на то, что компания в целом увеличивала свою прибыльность в течение этого периода.
2. **Резкий рост в апреле 2018**: В апреле 2018 года наблюдается значительный скачок профита. Это может быть связано с каким-то одноразовым событием, например, с крупной продажей, выходом нового продукта или сезонным увеличением спроса.
3. **Падение в июне-декабре 2018:** После апреля 2018 года происходит резкое падение скользящего среднего профита, достигая минимальных значений к концу года. Это могло быть вызвано различными факторами, включая снижение продаж, увеличение затрат или другие рыночные изменения.
4. **Необходимость анализа апреля 2018:** Важно детально изучить, что привело к резкому увеличению профита в апреле 2018, чтобы понять, можно ли воспроизвести эти условия в будущем.
5. **Исследование причин падения профита:** Также критически важно анализировать причины снижения профита в последние месяцы 2018 года, чтобы разработать стратегии по их устранению или минимизации влияния подобных ситуаций в будущем.
6. **Стратегическое планирование:** На основе анализа данных компании следует планировать стратегии нацеленные на поддержание роста профита и предотвращение будущих падений.

# *Прогнозирование продаж.*

WITH SalesAmount AS (

SELECT

o.order\_date,

SUM(oi.quantity \* oi.product\_price) AS total\_sales

FROM orders o

JOIN order\_items oi ON o.order\_id = oi.order\_id

GROUP BY o.order\_date

)

SELECT

order\_date,

total\_sales,

ROUND(

AVG(total\_sales) OVER (

ORDER BY order\_date

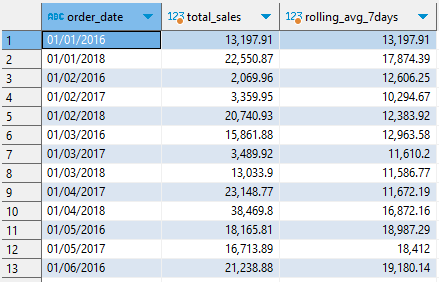
ROWS BETWEEN 6 PRECEDING AND CURRENT ROW

), 2

) AS rolling\_avg\_7days

FROM SalesAmount

ORDER BY order\_date;



### **Объяснение запроса:**

* Сначала создается общая сумма продаж (**total\_sales**) по дням с использованием подзапроса или CTE (**WITH**-конструкция), где данные группируются по дате заказа **order\_date**.
* Затем, используя оконную функцию **AVG() OVER (...)**, рассчитывается скользящее среднее за 7 дней (**ROWS BETWEEN 6 PRECEDING AND CURRENT ROW**), что позволяет получить среднее значение общей суммы продаж за текущий и предыдущие 6 дней для каждой даты.

### **Визуализация полученных результатов:**

import sqlite3

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Подключение к базе данных

conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02. SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase\_2.db')

# Загрузка данных

query = """

WITH SalesAmount AS (

  SELECT

    o.order\_date,

    SUM(oi.quantity \* oi.product\_price) AS total\_sales

  FROM orders o

  JOIN order\_items oi ON o.order\_id = oi.order\_id

  GROUP BY o.order\_date

)

SELECT

  order\_date,

  total\_sales,

  ROUND(

    AVG(total\_sales) OVER (

      ORDER BY order\_date

      ROWS BETWEEN 6 PRECEDING AND CURRENT ROW

    ), 2

  ) AS rolling\_avg\_7days

FROM SalesAmount

ORDER BY order\_date;

"""

df = pd.read\_sql\_query(query, conn)

# Закрытие соединения с базой данных

conn.close()

# Визуализация данных

plt.figure(figsize=(14, 7))

plt.plot(df['order\_date'], df['total\_sales'], label='Total Sales', color='blue')

plt.plot(df['order\_date'], df['rolling\_avg\_7days'], label='Rolling Average 7 Days', color='red', linestyle='--')

plt.title('Sales Forecasting with Rolling Average')

plt.xlabel('Order Date')

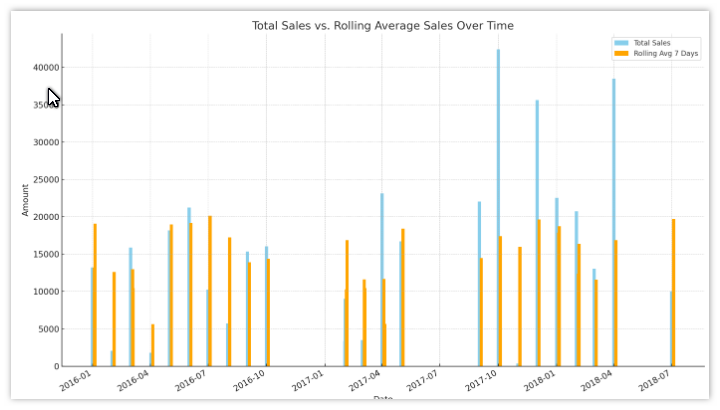
plt.ylabel('Total Sales')

plt.xticks(rotation=45)

plt.legend()

plt.tight\_layout()

plt.show()



**Выводы:**

* **Общие продажи** показывают значительные колебания, что может быть связано с сезонностью, маркетинговыми акциями или изменением потребительского спроса.
* **Скользящее среднее** помогает идентифицировать общие тренды, минимизируя влияние краткосрочных колебаний. Это полезно для планирования и принятия стратегических решений.
* Наблюдаются периоды, когда скользящее среднее значительно отличается от общих продаж, что может указывать на необычные события или изменения в рыночной ситуации.
* **Оптимизация запасов** и планирование продаж должны учитывать эти анализы, чтобы лучше соответствовать потребительскому спросу и минимизировать издержки.

# *Анализ зависимости между скидками и объемом продаж.*

SELECT

c.category\_name,

COUNT(DISTINCT o.order\_id) AS total\_orders,

SUM(oi.quantity) AS total\_quantity\_sold,

AVG(oi.discount) AS average\_discount,

SUM(oi.quantity \* (oi.product\_price - oi.discount)) AS total\_sales\_value

FROM

order\_items oi

JOIN orders o ON oi.order\_id = o.order\_id

JOIN products p ON oi.product\_id = p.product\_id

JOIN categories c ON p.category\_id = c.category\_id

GROUP BY

c.category\_name

ORDER BY

total\_sales\_value DESC;

תמונה שמכילה טקסט, גופן, מספר, קו

התיאור נוצר באופן אוטומטי

### **Объяснение запроса:**

* **SELECT:**
* **c.category\_name**: Выбираем наименование категории из таблицы **categories**.
* **COUNT(DISTINCT o.order\_id) AS total\_orders**: Считаем уникальные идентификаторы заказов из таблицы **orders**, чтобы узнать общее количество заказов по каждой категории.
* **SUM(oi.quantity) AS total\_quantity\_sold**: Суммируем количество проданных товаров по каждой категории из таблицы **order\_items**.
* **AVG(oi.discount) AS average\_discount**: Вычисляем среднюю скидку на товары в каждой категории.
* **SUM(oi.quantity \* (oi.product\_price - oi.discount)) AS total\_sales\_value**: Считаем общую сумму продаж после применения скидок, умножая количество на цену товара с учетом скидки для каждой категории.
* **FROM и JOIN**: Используются для соединения четырех таблиц: **order\_items** (oi), **orders** (o), **products** (p), и **categories** (c). Соединение таблиц необходимо для того, чтобы собрать данные о заказах, продуктах и категориях товаров в одном запросе.
* **GROUP BY c.category\_name**: Группируем результаты по названию категории, чтобы агрегированные функции (**COUNT**, **SUM**, **AVG**) применялись к каждой категории отдельно. Это позволяет получить агрегированные данные (количество заказов, проданных товаров, среднюю скидку, общую сумму продаж) для каждой категории.
* **ORDER BY total\_sales\_value DESC**: Сортируем результаты по убыванию общей суммы продаж (**total\_sales\_value**), чтобы категории с наибольшим объемом продаж были вверху списка.

### **Визуализация полученных результатов:**

import sqlite3

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Подключение к базе данных

conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02. SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase\_2.db')

# Загрузка данных

query = """

SELECT

    c.category\_name,

    COUNT(DISTINCT o.order\_id) AS total\_orders,

    SUM(oi.quantity) AS total\_quantity\_sold,

    AVG(oi.discount) AS average\_discount,

    ROUND(SUM(oi.quantity \* (oi.product\_price - oi.discount)), 2) AS total\_sales\_value

FROM

    order\_items oi

    JOIN orders o ON oi.order\_id = o.order\_id

    JOIN products p ON oi.product\_id = p.product\_id

    JOIN categories c ON p.category\_id = c.category\_id

GROUP BY

    c.category\_name

ORDER BY

    total\_sales\_value DESC;

"""

data = pd.read\_sql\_query(query, conn)

# Закрытие соединения с базой данных

conn.close()

# Визуализация данных

plt.figure(figsize=(12, 8))

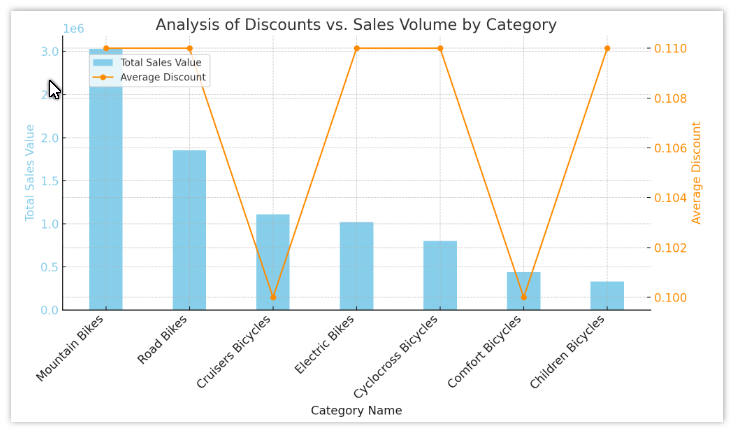
sns.barplot(x='total\_sales\_value', y='category\_name', data=data, palette='viridis')

plt.title('Анализ зависимости между скидками и объемом продаж по категориям')

plt.xlabel('Общая стоимость продаж')

plt.ylabel('Категория товара')

plt.show()



**Выводы:**

* **Горные велосипеды (Mountain Bikes)** и **Дорожные велосипеды (Road Bikes)** имеют наибольший объем продаж среди всех категорий, что подчеркивает их популярность. Средняя скидка в этих категориях составляет 0.11, что свидетельствует о стандартной практике скидок для этих товаров.
* **Круизеры (Cruisers Bicycles)**, несмотря на сравнительно низкую среднюю скидку (0.1), также демонстрируют высокий объем продаж. Это может указывать на внутреннее качество или популярность данных товаров среди потребителей.
* **Электровелосипеды (Electric Bikes)** и **Велосипеды для циклокросса (Cyclocross Bicycles)** с средней скидкой 0.11 также показывают значительный объем продаж, что может свидетельствовать о растущем интересе к этим видам велосипедов.
* Категории **Комфортные велосипеды (Comfort Bicycles)** и **Детские велосипеды (Children Bicycles)** имеют меньший объем продаж по сравнению с другими категориями, что может быть связано с более узкой целевой аудиторией или конкретными потребностями покупателей.
* Скидки имеют небольшое влияние на объем продаж по различным категориям, средняя скидка в большинстве категорий составляет около 0.1-0.11. Это указывает на то, что скидки, возможно, не являются основным драйвером продаж для данных категорий.
* Потребительская популярность и специфика товара играют ключевую роль в объеме продаж по категориям. Важно учитывать эти факторы при планировании маркетинговых кампаний и скидочных акций.

**Рекомендации:**

* Для категорий с высоким объемом продаж и стандартной скидкой рассмотреть возможность экспериментов с увеличением скидок для стимулирования дополнительных продаж.
* В категориях с меньшим объемом продаж исследовать другие маркетинговые стратегии, помимо скидок, для привлечения внимания к этим товарам.