

Univerzita Jana Evangelisty Purkyně v Ústí nad Labem (UJEP)

Přírodovědecká fakulta (PřF), Katedra informatiky (KI)

Seminární práce

OLAP a ClickHouse

Ondřej Švorc (F23209)

Aplikovaná informatika

OLAP a Data mining (KI/ODM)

27. 4. 2025

Obsah

Zkratky	3
Volba DBMS s OLAP podporou.....	3
Prerekvizity	3
Datová sada	5
Čištění datové sady	6
Datová kostka	6
Import datové sady	7
Tvorba schématu hvězdy	8
Komunikace s ClickHouse	9
Řezy datovou kostkou.....	17
Data mining	22
Závěr	24
Zdroje.....	24

Zkratky

DBMS = Database Management System

OLAP = Online Analytical Processing

Volba DBMS s OLAP podporou

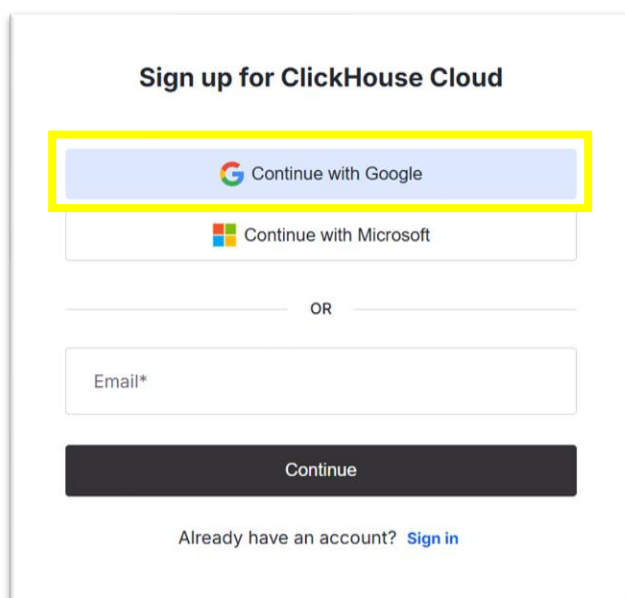
Pro tento projekt jsem zvolil produkt **ClickHouse Cloud**, především díky jednoduchosti použití ve srovnání s ostatními DBMS s OLAP podporou. Před zvolením ClickHouse Cloud jsem si krátce prohlédl a vyzkoušel Azure Analytics Service, Apache Druid a SSMS.

Prerekvizity


Díky tomu, že ClickHouse Cloud je webová aplikace, za jediné prerekvizity lze považovat:


1. Založený účet
2. Zvolené předplatné
3. Vytvořená cloudová služba pro provoz

Osobně jsem zvolil možnost registrace pomocí **Google účtu** a předplatné **Basic**. Při vytváření cloudové služby jsem ponechal název **My first Service**, vybral poskytovatele **AWS**, lokaci ve **Frankfurtu nad Mohanem** (zatím nejbližší možná) a **8 GB RAM**.



Sign up for ClickHouse Cloud

 Continue with Google


 Continue with Microsoft

OR

Email*

Continue

Already have an account? [Sign in](#)



Select a plan

ClickHouse Cloud offers a range of plans to cover your needs

Basic

Great for learning or starter projects. Limited storage, memory and features.

Start free trial

- ✓ Up to 1 TB storage per service
- ✓ Fixed memory, limited to 8 or 12 GiB
- ✓ 1 availability zone per service
- ✓ 1 daily backup retained per 1 day
- ✓ ClickPipes for data ingestion
- ✓ SOC 2 Type II and ISO 27001
- ✓ Support with 1 business day response time

Starting from \$25.3 per TB/month for storage and \$0.21811 per 8 GiB/hour for compute.

Scale (Recommended)

For production environments, data at scale, or professional use cases.

Start free trial

- ✓ Unlimited storage per service
- ✓ Vertical and horizontal scaling
- ✓ Multiple availability zones
- ✓ Configurable backups
- ✓ ClickPipes for data ingestion
- ✓ Private networking
- ✓ Compute-compute separation
- ✓ SOC 2 Type II and ISO 27001
- ✓ Support with 1 hour response time

Starting from \$25.3 per TB/month for storage and \$0.29846 per 8 GiB/hour for compute.

Enterprise


Maximum flexibility with premium features (SSO, CMEK, HIPAA and many more).

Start free trial

Everything in Scale, plus:

- ✓ Custom hardware profiles
- ✓ Private regions
- ✓ SAML / SSO support
- ✓ HIPAA compliance
- ✓ Transparent data encryption / CMEK
- ✓ Support with 30 minutes response time

Starting from \$25.3 per TB/month for storage and \$0.3903 per 8 GiB/hour for compute.




Create cloud infrastructure

Create an instance of ClickHouse that lives in the Cloud and can host multiple databases


Service name

My first service

Cloud provider

 AWS

Region

 Frankfurt (eu-central-1)

Can't find your region? [Request it](#)

Memory and scaling

Mini 8GB

8 GiB RAM per replica

1 replica

Mini 12GB

12 GiB RAM per replica

1 replica

Standard 3×16GB

16 ↔ 120 GiB RAM per replica

3 replicas

Custom configuration

Manually select your sizing

Create service

Back to plan options

Datová sada

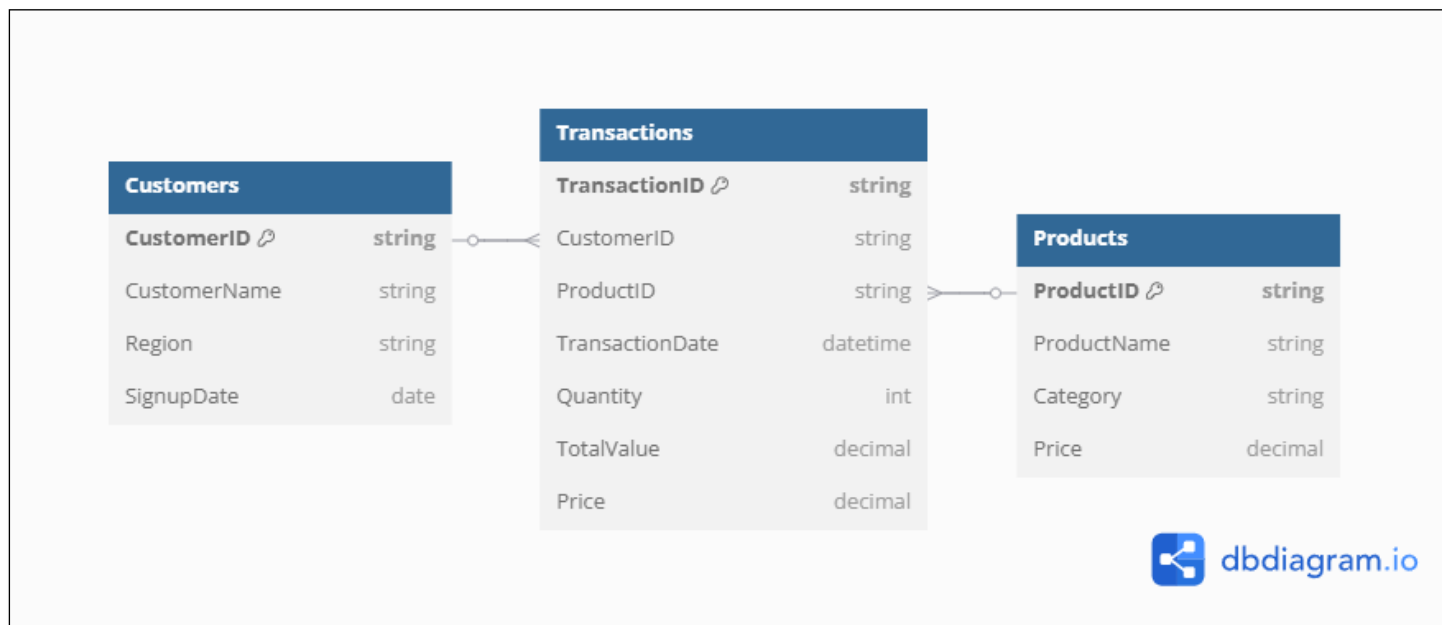
Zvolil jsem datovou sadu ze stránky **Kaggle** fiktivního online obchodu **EverMart Online**.¹

Soubory datové sady ve formátu CSV obsahují data o zákaznících, produktech a provedených transakcích.

- Customers.csv
- Products.csv
- Transactions.csv

Provedené transakce jsou duplicitně zapsány ve dvou souborech s mírně odlišnými formáty (Transactions.csv, EverMart_Online_Transactions.xlsx). CSV formát je vhodnější pro strojové zpracování, a proto nemá smysl XLSX soubor zahrnovat.

Pro lepší představu je zde ERD diagram, který reflektuje provázanost dat v CSV souborech.



¹ WAMBLES, Chad. Ecommerce Transactions [online]. [cit. 25. 3. 2025]. Dostupné z: <https://www.kaggle.com/datasets/chadwambles/ecommerce-transactions>

Čištění datové sady

Datová sada byla již po stažení vyčištěná, a proto nebylo nutné provádět její čištění.

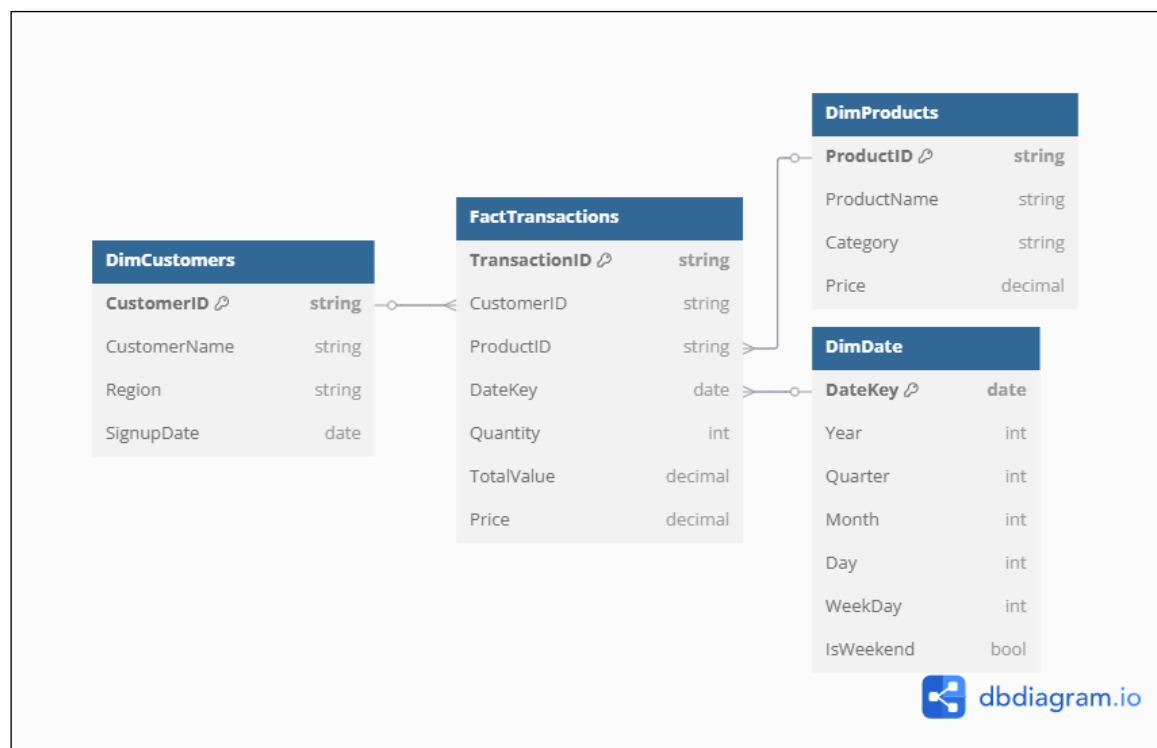
Datová kostka

Zvolil jsem **schéma hvězdy** (star schema) se třemi dimenzemi a jednou faktovou tabulkou.

- DimCustomers – zákazníci (celé jméno, kontinent, datum registrace)
- DimProducts – produkty (název, kategorie, cena)
- DimDate – datum transakce (rok, měsíc, den, víkend)
- FactTransactions – transakce (množství, celková cena, odkazy na dimenze)

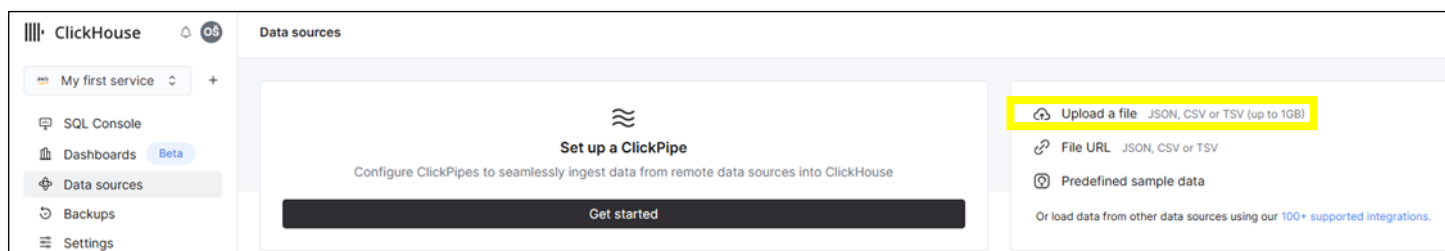
Zde jsou základní otázky, na jenž si můžeme na první pohled zodpovědět:

- Kdo nakupoval
- Co nakupoval
- Kdy nakoupil
- Kolik toho nakoupil
- Za kolik nakoupil



Import datové sady

V Původně jsem chtěl vložit CSV soubory přímo pomocí grafického rozhraní ClickHouse Cloud, ale tato možnost opakovaně selhala. Rozhodl jsem se tedy data naimportovat pomocí Python skriptu.



Zvažoval jsem využití knihovny **clickhouse-connect**² pro Python, která slouží jako rozhraní pro komunikaci s ClickHouse, ale nakonec jsem ji nepoužil, protože k mým účelům nebyla většina jejích funkcionalit vůbec potřeba; jednalo by se tak o zbytečnou závislost. Zvolil jsem tedy jednodušší řešení, a to použití přímých HTTP požadavků za pomoci knihovny **requests** Pythonu.

² CLICKHOUSE, Inc. ClickHouse Python Integrations [online]. [cit. 27. 4. 2025]. Dostupné z: <https://clickhouse.com/docs/integrations/python>

Tvorba schématu hvězdy

1. Smazání všech existujících tabulek.
2. Vytvoření surových tabulek pro import dat na základě CSV souborů.
3. Import dat ze souborů CSV do surových tabulek.
4. Vytvoření dimenzionálních tabulek.
5. Naplnění dimenzionálních tabulek daty z tabulek surových.
6. Vytvoření faktové tabulky.
7. Naplnění faktové tabulky daty.
8. Smazání surových tabulek.

```
1 def create_star_schema():
2     drop_tables_if_exist(ALL_TABLES)
3
4     create_raw_tables()
5     insert_csv_data_into_raw_tables()
6
7     create_dim_tables()
8     insert_into_dim_tables()
9
10    create_fact_table()
11    insert_into_fact_table()
12
13    drop_tables_if_exist(RAW_TABLES)
```


Komunikace s ClickHouse

Pro usnadnění opakovaného volání ClickHouse API jsem si na začátku skriptu definoval konstanty pro URL serveru, přihlašovací údaje a hlavičky HTTP požadavků.

```
1 # Tables.
2 RAW_TABLES = ["Customers", "Products", "Transactions"]
3 RAW_TABLES_DICT = { "Customers": "Customers.csv", "Products": "Products.csv", "Transactions": "Transactions.csv"}
4 DIM_TABLES = ["DimCustomers", "DimProducts", "DimDate"]
5 FACT_TABLES = ["FactTransactions"]
6 ALL_TABLES = RAW_TABLES + DIM_TABLES + FACT_TABLES
```

Pro přehlednost jsem si také nadeřinoval konstanty pro názvy surových tabulek, jejich přiřazení k CSV souborům, dimenzionální tabulky, faktovou tabulku a celkový seznam všech tabulek.

```
1 # Connection details for ClickHouse.
2 CLICKHOUSE_URL = "https://grv07xcd94.eu-central-1.aws.clickhouse.cloud:8443"
3 CLICKHOUSE_USER = "default"
4 CLICKHOUSE_PASSWORD = ""
5 AUTH = (CLICKHOUSE_USER, CLICKHOUSE_PASSWORD)
6 HEADERS = {"Content-Type": "text/plain"}
```

Funkce **run_sql** slouží k odesílání SQL příkazů na server ClickHouse, kde se SQL příkaz vyhodnotí a vykoná. Automaticky sestaví a odešle požadavek, zkontroluje jeho úspěšnost a vypíše stručnou informaci o výsledku.

```
1 def run_sql(sql: str) -> None:
2     query_summary = sql.strip().split("\n")[0][:80].strip()
3     response = requests.post(
4         url=f"{CLICKHOUSE_URL}/",
5         params={"query": sql},
6         auth=AUTH,
7         headers=HEADERS
8     )
9
10    if response.ok:
11        print(f"✅ Query OK: {query_summary}")
12    else:
13        print(f"❌ Query failed: {query_summary}\n{response.text}")
```

Funkce **drop_tables_if_exist** postupně smaže všechny tabulky uvedené v seznamu, pokud existují.

Tím zajišťuje, že při opětovném spuštění skriptu nedojde ke konfliktu kvůli existujícím tabulkám, a každý import proběhne vždy na čisté databázové struktuře.

```
1 def drop_tables_if_exist(tables: list[str]):
2     for table in tables:
3         try:
4             run_sql(f"DROP TABLE IF EXISTS {table}")
5         except Exception as e:
6             print(f"⚠ Warning: Could not drop table {table} → {e}")
```

Funkce **create_raw_tables** vytváří tři surové tabulky (Customers, Products, Transactions) podle struktury CSV souborů. Tyto tabulky slouží jako první krok před rozdělením dat do dimenzí a faktové tabulky.

```
1 def create_raw_tables():
2     # Customers
3     run_sql("""
4     CREATE TABLE Customers (
5         CustomerID String,
6         CustomerName String,
7         Region String,
8         SignupDate Date
9     ) ENGINE = MergeTree()
10    ORDER BY CustomerID;
11    """)
12
13    # Products
14    run_sql("""
15    CREATE TABLE Products (
16        ProductID String,
17        ProductName String,
18        Category String,
19        Price Decimal(10, 2)
20    ) ENGINE = MergeTree()
21    ORDER BY ProductID;
22    """)
23
24    # Transactions
25    run_sql("""
26    CREATE TABLE Transactions (
27        TransactionID String,
28        CustomerID String,
29        ProductID String,
30        TransactionDate DateTime,
31        Quantity Int32,
32        TotalValue Decimal(10, 2),
33        Price Decimal(10, 2)
34    ) ENGINE = MergeTree()
35    ORDER BY TransactionID;
36    """)
```

Funkce `insert_csv_data_into_raw_tables` načítá obsah CSV souborů a vkládá jej do odpovídajících tabulek.

```
1 def insert_csv_data_into_raw_tables():
2     for table, path in RAW_TABLES_DICT.items():
3         insert_url = f"{CLICKHOUSE_URL}/?query=INSERT INTO {table} FORMAT CSVWithNames"
4
5         with open(path, "rb") as csv_file:
6             csv_data = csv_file.read()
7
8         response = requests.post(
9             url=insert_url,
10            data=csv_data,
11            auth=AUTH,
12            headers=HEADERS,
13        )
14
15        if response.ok:
16            print(f"✅ Query OK: Inserted into {table}")
17        else:
18            print(f"❌ Query failed: Failed to insert into {table}: {response.status_code}\n{response.text}")
```

Funkce **create_dim_tables** vytvoří dimenzionální tabulky.

```
1 def create_dim_tables():
2     # Customer dimension
3     run_sql("""
4     CREATE TABLE DimCustomers (
5         CustomerID String,
6         CustomerName String,
7         Region String,
8         SignupDate Date
9     ) ENGINE = MergeTree()
10    ORDER BY CustomerID;
11    """)
12
13    # Product dimension
14    run_sql("""
15    CREATE TABLE DimProducts (
16        ProductID String,
17        ProductName String,
18        Category String,
19        Price Decimal(10, 2)
20    ) ENGINE = MergeTree()
21    ORDER BY ProductID;
22    """)
23
24    # Date dimension
25    run_sql("""
26    CREATE TABLE DimDate (
27        DateKey Date,
28        Year UInt16,
29        Quarter UInt8,
30        Month UInt8,
31        Day UInt8,
32        WeekDay UInt8,
33        Hour UInt8,
34        IsWeekend UInt8
35    ) ENGINE = MergeTree()
36    ORDER BY (DateKey, Hour);
37    """)
```

Funkce **insert_into_dim_tables** vloží data ze surových tabulek do dimenzionálních.

```
1 def insert_into_dim_tables():
2     run_sql("INSERT INTO DimCustomers SELECT DISTINCT * FROM Customers;")
3     run_sql("INSERT INTO DimProducts SELECT DISTINCT * FROM Products;")
4     run_sql("""
5     INSERT INTO DimDate
6     SELECT
7         toDate(TransactionDate) AS DateKey,
8         toYear(TransactionDate) AS Year,
9         toQuarter(TransactionDate) AS Quarter,
10        toMonth(TransactionDate) AS Month,
11        toDayOfMonth(TransactionDate) AS Day,
12        toDayOfWeek(TransactionDate) AS WeekDay,
13        toHour(TransactionDate) AS Hour,
14        if(toDayOfWeek(TransactionDate) IN (6, 7), 1, 0) AS IsWeekend
15    FROM Transactions
16    GROUP BY DateKey, Year, Quarter, Month, Day, WeekDay, Hour, IsWeekend;
17    """)
```

Funkce **create_fact_table** vytvoří faktovou tabulku.

```
1 def create_fact_table():
2     run_sql("""
3     CREATE TABLE FactTransactions (
4         TransactionID String,
5         CustomerID String,
6         ProductID String,
7         DateKey Date,
8         Quantity Int32,
9         TotalValue Decimal(10, 2),
10        Price Decimal(10, 2)
11    ) ENGINE = MergeTree()
12    ORDER BY TransactionID;
13    """)
```

Funkce **insert_into_fact_table** vloží a přetransformuje data z tabulky Transactions.

```
1 def insert_into_fact_table():
2     run_sql("""
3     INSERT INTO FactTransactions
4     SELECT
5         TransactionID,
6         CustomerID,
7         ProductID,
8         toDate(TransactionDate) AS DateKey,
9         Quantity,
10        TotalValue,
11        Price
12     FROM Transactions;
13     """)
```


Řezy datovou kostkou

Pro demonstraci práce s datovou kostkou jsem vytvořil následující čtyři řezy (a dva bonusové).

```
1 -- Řez 1: Celkové tržby podle roku
2 SELECT Year, SUM(TotalValue) AS Revenue
3 FROM FactTransactions
4 JOIN DimDate ON FactTransactions.DateKey = DimDate.DateKey
5 GROUP BY Year
6 ORDER BY Year;
```

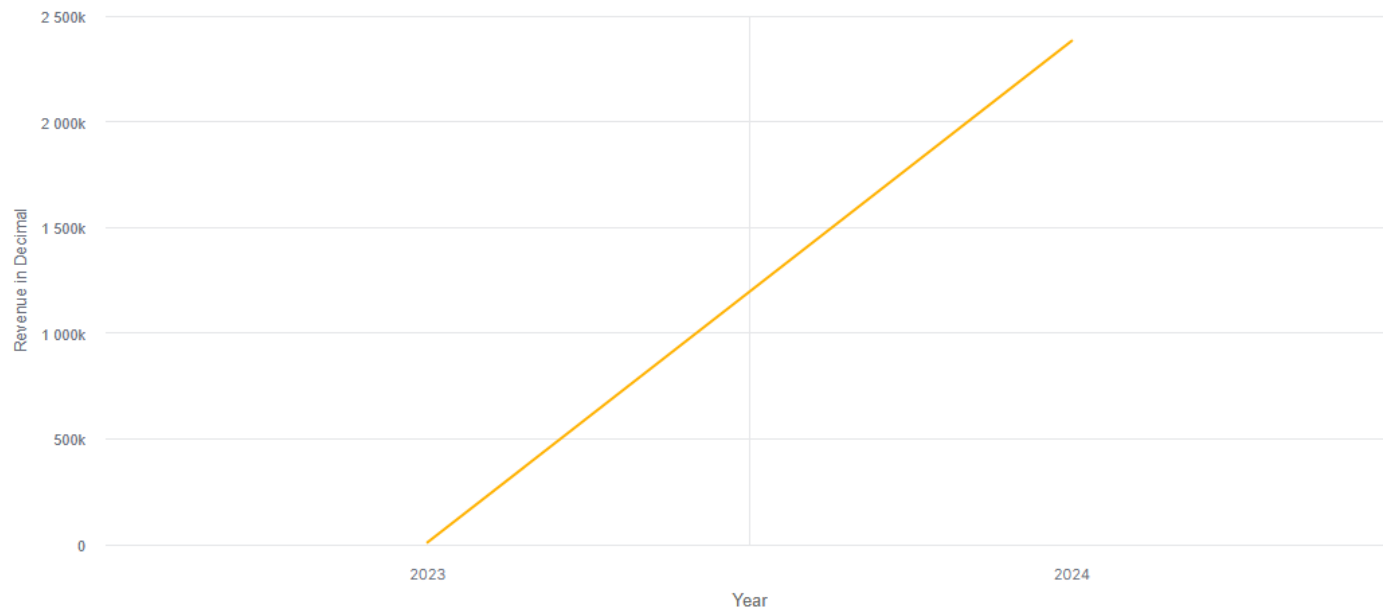
```
1 -- Řez 2: Počet transakcí podle regionu
2 SELECT Region, COUNT(*) AS TransactionCount
3 FROM FactTransactions
4 JOIN DimCustomers ON FactTransactions.CustomerID = DimCustomers.CustomerID
5 GROUP BY Region
6 ORDER BY TransactionCount DESC;
```

```
1 -- Řez 3: Průměrná cena produktu podle kategorie
2 SELECT Category, AVG(Price) AS AveragePrice
3 FROM FactTransactions
4 JOIN DimProducts ON FactTransactions.ProductID = DimProducts.ProductID
5 GROUP BY Category
6 ORDER BY AveragePrice DESC;
```

```
1 -- Řez 4: Tržby podle víkendu
2 SELECT IsWeekend, SUM(TotalValue) AS Revenue
3 FROM FactTransactions
4 JOIN DimDate ON FactTransactions.DateKey = DimDate.DateKey
5 GROUP BY IsWeekend
6 ORDER BY Revenue DESC;
```

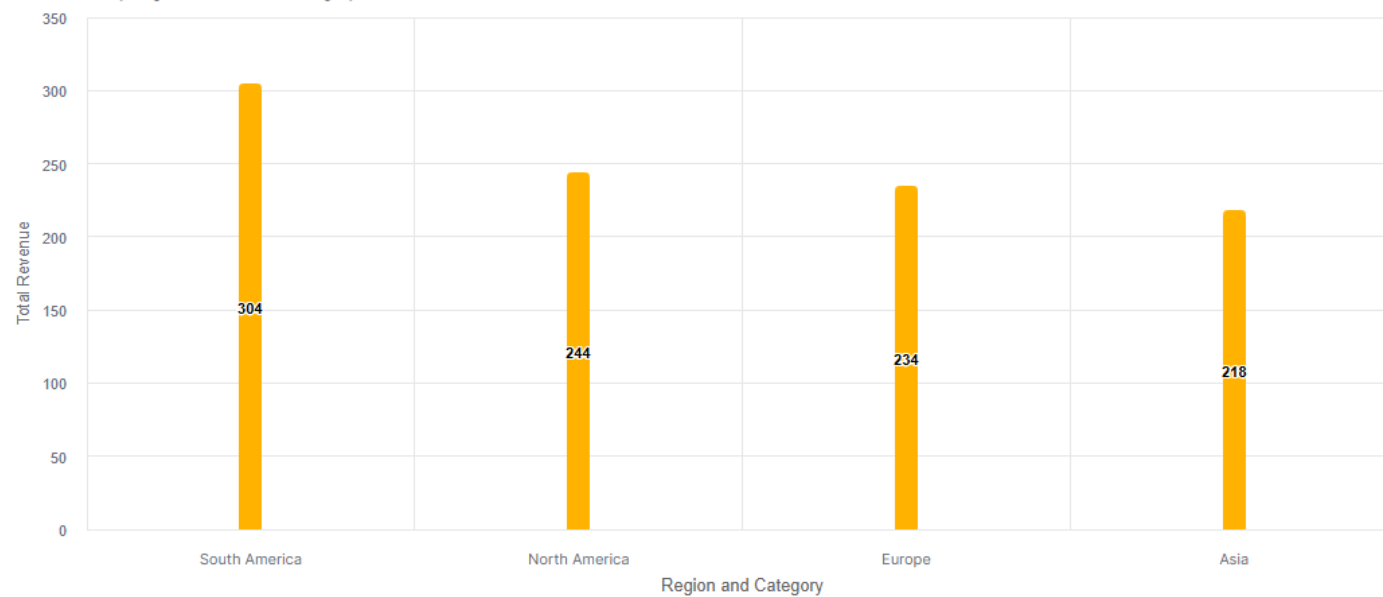
Celkové tržby podle roku

Annual Revenue

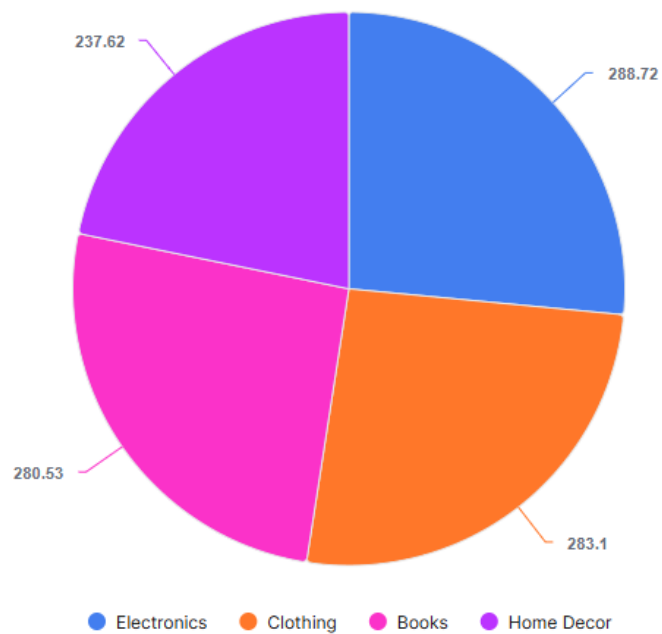


Počet transakcí podle regionu

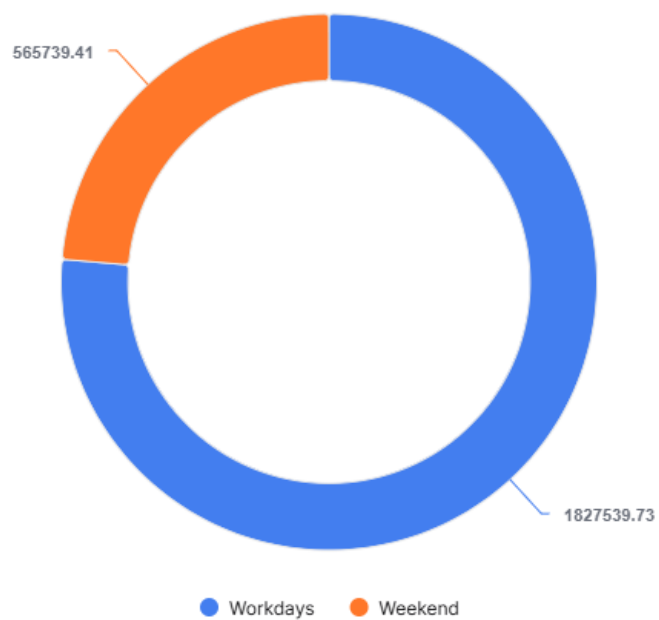
Total Revenue by Region and Product Category



Průměrná cena produktů podle kategorie
Average Price by Category



Tržby podle části týdne (víkend vs. pracovní dny)
Revenue by Weekend Status

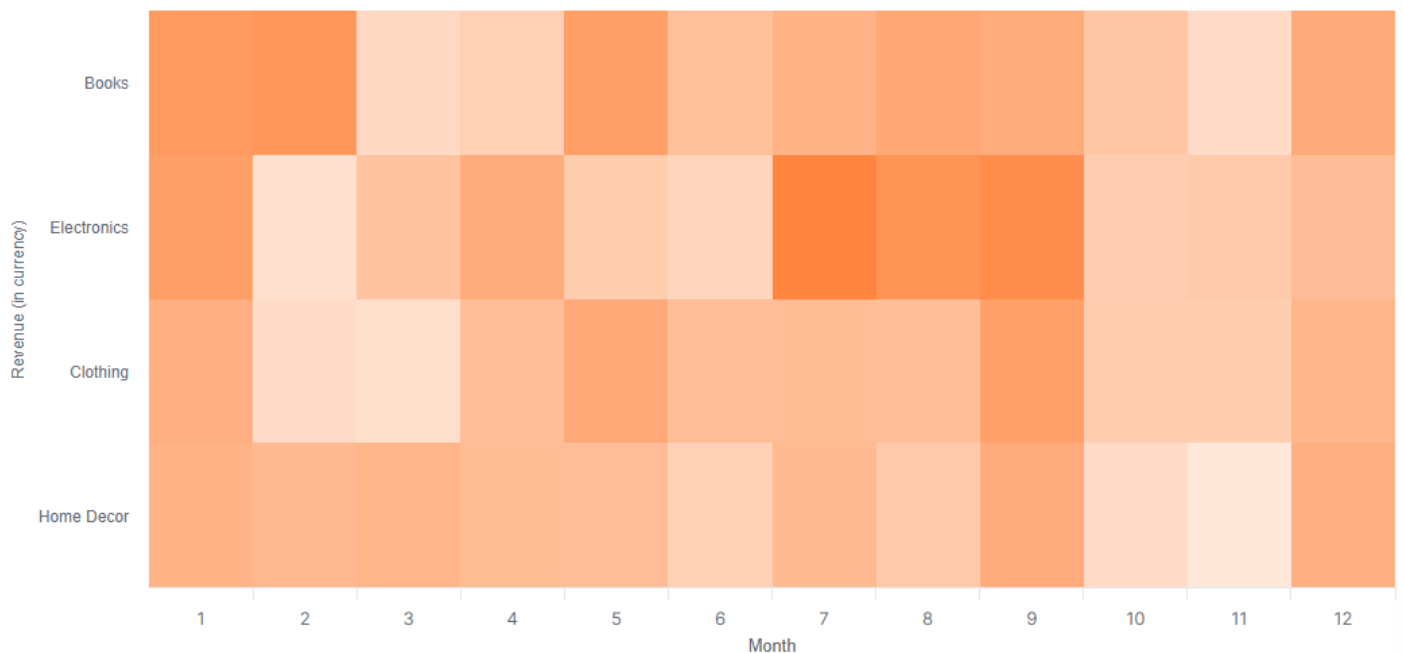


```

1 -- Řez 5: Tržby podle měsíce a kategorie produktu
2 SELECT Month, Category, SUM(TotalValue) AS Revenue
3 FROM FactTransactions
4 JOIN DimDate ON FactTransactions.DateKey = DimDate.DateKey
5 JOIN DimProducts ON FactTransactions.ProductID = DimProducts.ProductID
6 GROUP BY Month, Category
7 ORDER BY Month, Revenue DESC;

```

Tržby podle měsíce a kategorie produktu
Monthly Revenue by Category

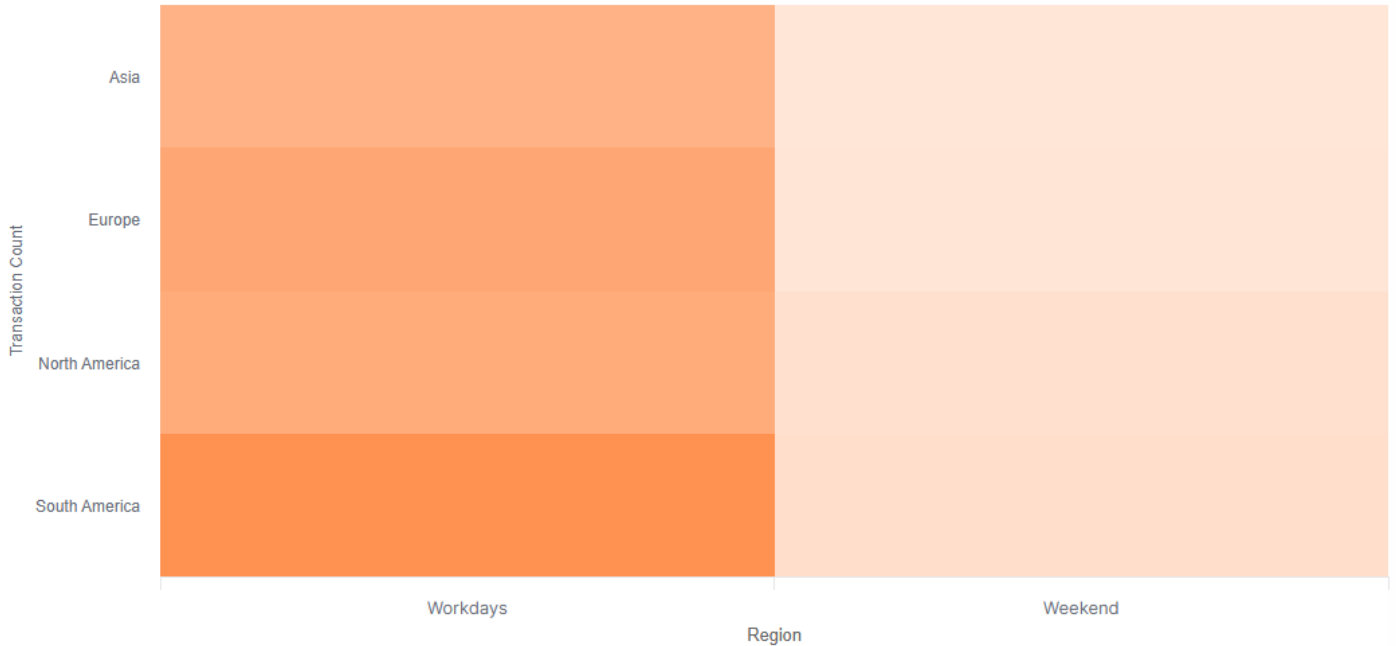


```

1 -- Řez 6: Počet transakcí podle kontinentu a víkendu
2 SELECT Region, CASE WHEN IsWeekend = 1 THEN 'Weekend' ELSE 'Workdays' END AS DayType, COUNT(*) AS TransactionCount
3 FROM FactTransactions
4 JOIN DimCustomers ON FactTransactions.CustomerID = DimCustomers.CustomerID
5 JOIN DimDate ON FactTransactions.DateKey = DimDate.DateKey
6 GROUP BY Region, DayType
7 ORDER BY Region, TransactionCount DESC;

```

Počet transakcí podle kontinentu a víkendu
Transaction Count by Region and Day Type

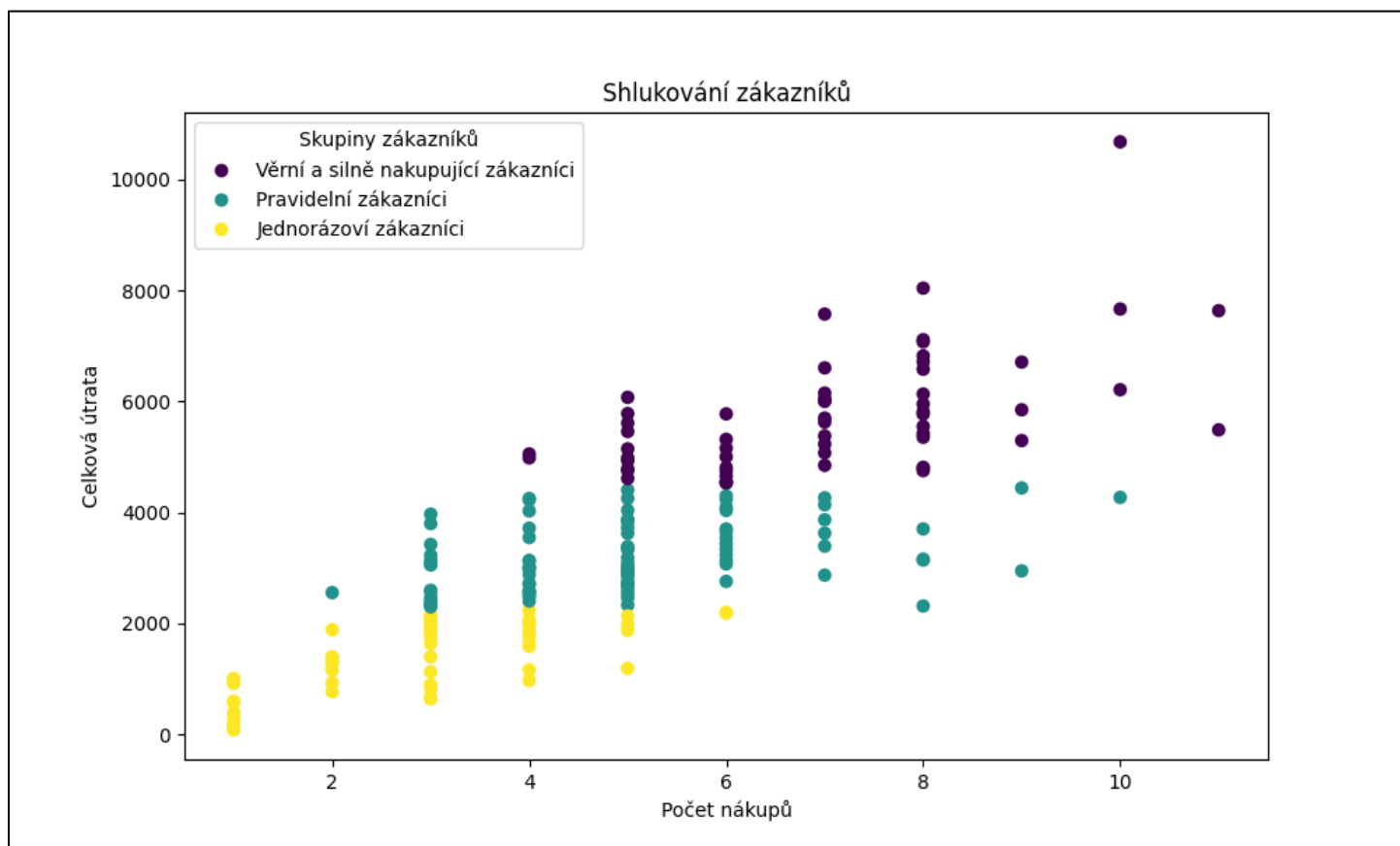


Data mining

ClickHouse nepodporuje nativní funkce pro data mining, a proto jsem se rozhodl tento krok realizovat pomocí Pythonu. V rámci úlohy jsem zvolil **shlukování (clustering)** metodou **K-means**³, kdy jsem zákazníky rozdělil do tří skupin podle počtu nákupů a celkové útraty. Pro přípravu dat jsem v ClickHouse vygeneroval agregovaný dataset (CustomerID, PurchaseCount, TotalSpent), který jsem následně exportoval do souboru CSV. V Pythonu jsem tento soubor načetl a pomocí připraveného skriptu provedl shlukování a vizualizaci výsledků.

```
1 import pandas as pd
2 from sklearn.cluster import KMeans
3 import matplotlib.pyplot as plt
4
5 data = pd.read_csv('customer_aggregates.csv')
6 X = data[['PurchaseCount', 'TotalSpent']]
7
8 kmeans = KMeans(n_clusters=3, random_state=42)
9 data['Cluster'] = kmeans.fit_predict(X)
10
11 groups = {
12     0: 'Věrní a silně nakupující zákazníci',
13     1: 'Pravidelní zákazníci',
14     2: 'Jednorázoví zákazníci'
15 }
16 data['GroupName'] = data['Cluster'].map(groups)
17
18 def print_customers_in_group(group_name: str):
19     selected_customers = data[data['GroupName'] == group_name]['CustomerID']
20     if selected_customers.empty:
21         print(f"Žádní zákazníci ve skupině: {group_name}")
22     else:
23         print(f"Zákazníci ve skupině '{group_name}':")
24         for cid in selected_customers:
25             print(f"{cid}")
26
27 plt.figure(figsize=(10, 6))
28 scatter = plt.scatter(
29     data['PurchaseCount'],
30     data['TotalSpent'],
31     c=data['Cluster'],
32     cmap='viridis'
33 )
34 plt.xlabel('Počet nákupů')
35 plt.ylabel('Celková útrata')
36 plt.title('Shlukování zákazníků')
37
38 handles, labels = scatter.legend_elements(prop='colors')
39 named_labels = [groups[i] for i in range(len(labels))]
40
41 plt.legend(handles, named_labels, title='Skupiny zákazníků')
42 plt.savefig('clustering.png')
43 plt.show()
44
45 print_customers_in_group('Věrní a silně nakupující zákazníci')
```

³ SCIKIT-LEARN Developers. KMeans clustering [online]. [cit. 27. 4. 2025]. Dostupné z: <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>



Pomocí funkce `print_customers_in_group` si lze vypsát, jací zákazníci se nachází v dané skupině, resp. jaký mají unikátní identifikátor.

Závěr

Pomocí ClickHouse Cloud jsem navrhl a realizoval datovou kostku nad fiktivní e-commerce datovou sadou. Zvolil jsem schéma hvězdy, provedl import dat přes vlastní Python skript s využitím ClickHouse API a vytvořil několik analytických řezů kostkou. V části data mining jsem ukázal jednoduché shlukování zákazníků metodou K-means v Pythonu.

Zdroje

WAMBLES, Chad. Ecommerce Transactions [online]. [cit. 25. 3. 2025]. Dostupné z: <https://www.kaggle.com/datasets/chadwambles/ecommerce-transactions>

CLICKHOUSE, Inc. ClickHouse Python Integrations [online]. [cit. 27. 4. 2025]. Dostupné z: <https://clickhouse.com/docs/integrations/python>

SCIKIT-LEARN Developers. KMeans clustering [online]. [cit. 27. 4. 2025]. Dostupné z: <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>