The background image shows a dimly lit museum gallery. Several large Renaissance-style paintings are displayed on the walls. In the center, a person is walking away from the camera, looking at the art. The lighting is focused on the paintings, creating a dramatic effect.

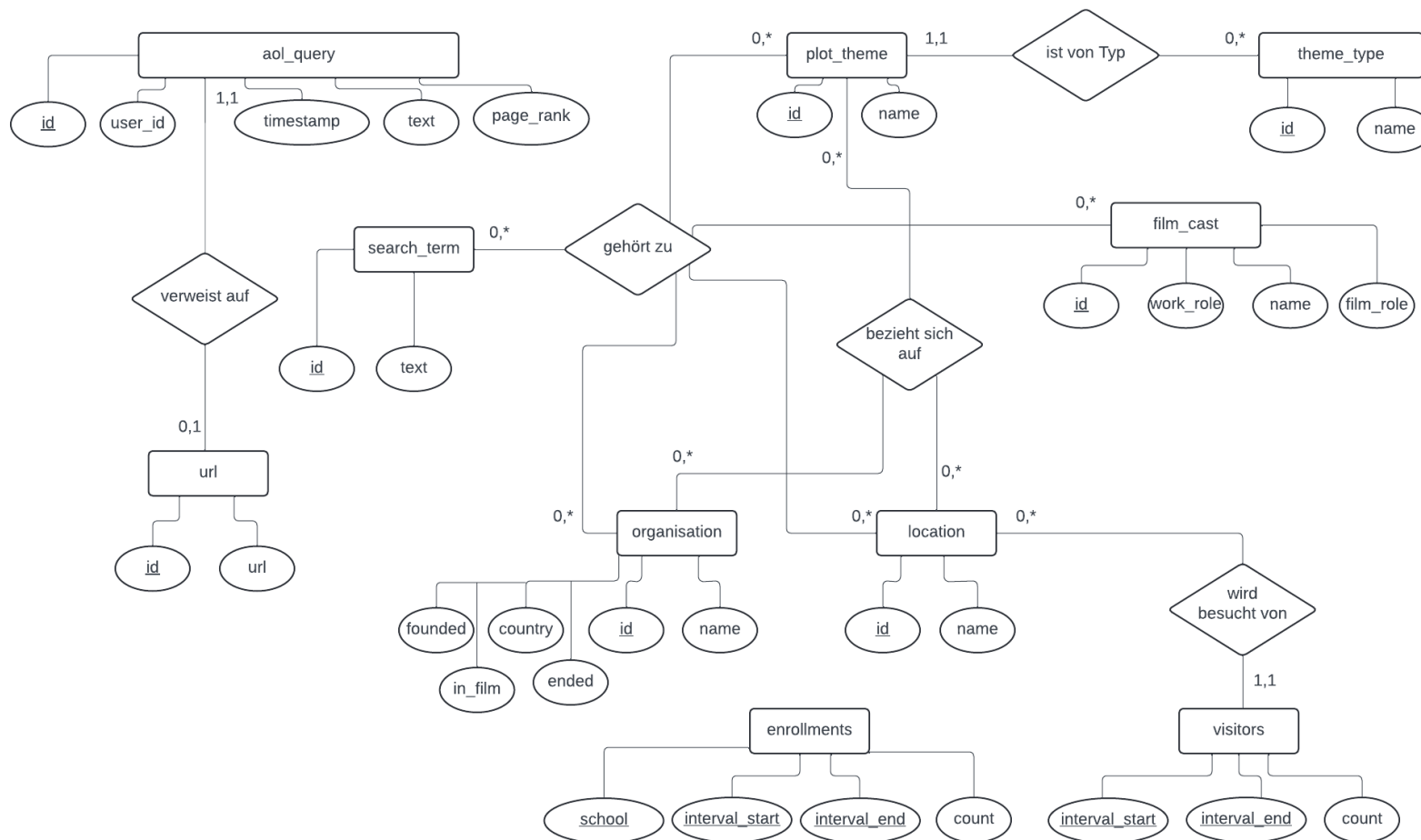
Serhii Berehovi, Anselm Nehls

The Da Vinci Code

Auf der Suche nach der *Suche nach dem Code*...

Aufgabe 3

Unsere Datenbank



RDM der Datenbank

- aol_query(id, user_id, timestamp, text, page_rank, *url_id*)
- url(id, url)
- search_term(id, text)
- theme_type(id, name)
- plot_theme(id, name, *theme_type_id*)
- location(id, name)
- organisation(id, name, country, founded, ended, in_film)
- film_cast(id, name, work_role, film_role)
- visitors(location_id, interval_start, interval_end, count)

*Unterstrichen = primary key

***Fett** = foreign key

RDM der Datenbank

- enrollments(school, interval start, interval end, count)
- search_term_plot_theme(**search term id**, **plot theme id**)
- search_term_organisation(**search term id**, **organisation id**)
- search_term_location(**search term id**, **location id**)
- search_term_cast(**search term id**, **cast id**)
- plot_theme_location(**plot theme id**, **location id**)
- plot_theme_organisation(**plot theme id**, **organisation id**)

*Unterstrichen = primary key

***Fett** = foreign key

Erste Schritte zur Visualisierung

Die von uns entworfene Datenbank ist sehr gut für die Visualisierung geeignet. 💪 😎

Erste Schritte zur Visualisierung

Python code und Jupyter Notebook

An dieser Stelle haben wir entschieden,
Python mit Pandas und Plotly für Visualisierung und
Jupyter Notebook als Entwicklungsumgebung
zu benutzen.

Dadurch ist unser Code gut strukturiert und aufgeteilt.

Einlesen von Daten - Query

Frage: Interesse an den Mitwirkenden des Films

select

```
fc.name,  
count(q.text) as cnt,  
q.timestamp::date as d
```

from

```
aol_query as q
```

join

```
film_cast as fc
```

on

```
q.text LIKE '%' || lower(fc.name) || '%'
```

group by

```
fc.name,  
d
```

order by

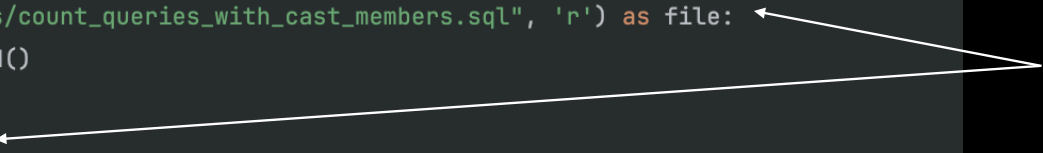
```
fc.name asc,  
d asc
```

Einlesen von Daten - Python

Python wird über psycopg2 mit der PostgreSQL Datenbank interagiert.

```
1 import psycopg2
2
3 conn = psycopg2.connect(
4     host="localhost",
5     database="postgres",
6     user="serhii",
7     password="password"
8 )
9
10 cur = conn.cursor()
11
12 with open("../work/queries/count_queries_with_cast_members.sql", 'r') as file:
13     sql_script = file.read()
14
15 cur.execute(sql_script)
16
17 records = cur.fetchall()
18
19 cur.close()
20 conn.close()
```

Lesen und Ausführen der
Query von der vorherigen
Folie



Daten vorbereiten - Python

Module importieren, DataFrame erstellen und die Daten für Visualisierung vorbereiten.

```
1 import pandas as pd
2 import plotly.graph_objects as go
3 import datetime
```

```
1 # DataFrame erstellen
2 df = pd.DataFrame(records, columns=['actor', 'queries', 'date'])
3
4 # 'date' in datetime umwandeln
5 df['date'] = pd.to_datetime(df['date'])
6
7 # Gruppieren von Daten nach Datum und Akteuren
8 grouped_df = df.groupby(['date', 'actor']).sum().reset_index()
9
10 # unique Schauspieler und Dates finden
11 actors = grouped_df['actor'].unique()
12 dates = grouped_df['date'].unique()
13
14 # Daten für eine interaktive Grafik vorbereiten
15 data = []
16 hover_text_dict = {}
17
18 Executed at 2024.07.01 19:33:19 in 8ms
```

Die Daten für Grafik einlesen und die Spalten für jeden Schauspieler erstellen.

```
1  # Füllen eines Wörterbuchs zum Speichern von Hover-Text nach Datum
2  for date in dates:
3      daily_data = grouped_df[grouped_df['date'] == date]
4      total_queries = daily_data['queries'].sum()
5      hover_text_parts = {}
6      for _, row in daily_data.iterrows():
7          percent = (row['queries'] / total_queries) * 100 if total_queries != 0 else 0
8          hover_text_parts[row['actor']] = f"({percent:.2f}%)"
9      hover_text_dict[date] = hover_text_parts
10
11
12  # Spalten für jeden Schauspieler erstellen
13  for actor in actors:
14      actor_data = grouped_df[grouped_df['actor'] == actor]
15      hover_text = [hover_text_dict[date][actor] for date in actor_data['date']]
16
17      data.append(go.Bar(
18          x=actor_data['date'],
19          y=actor_data['queries'],
20          name=actor,
21          text=hover_text,
22          # hovertextsrc="date",
23          marker=dict(line=dict(width=0.5))
24      ))
```

Wann wurde der Film publiziert?

Publikationsdatum von Code Da'Vinci Film zum Diagramm hinzufügen.

```
1  # Hinzufügen einer vertikalen Linie für das Veröffentlichungsdatum vom Film "The Da Vinci Code"
2  release_date = datetime.date(2006, 5, 19)
3  release_line = go.Scatter(
4      x=[release_date, release_date],
5      y=[0, grouped_df['queries'].max()],
6      mode='lines',
7      name='The Da Vinci Code Release',
8      line=dict(color='red', width=4, dash='dash'),
9      hoverinfo='skip' # Убираем hover для линии
10 )
```

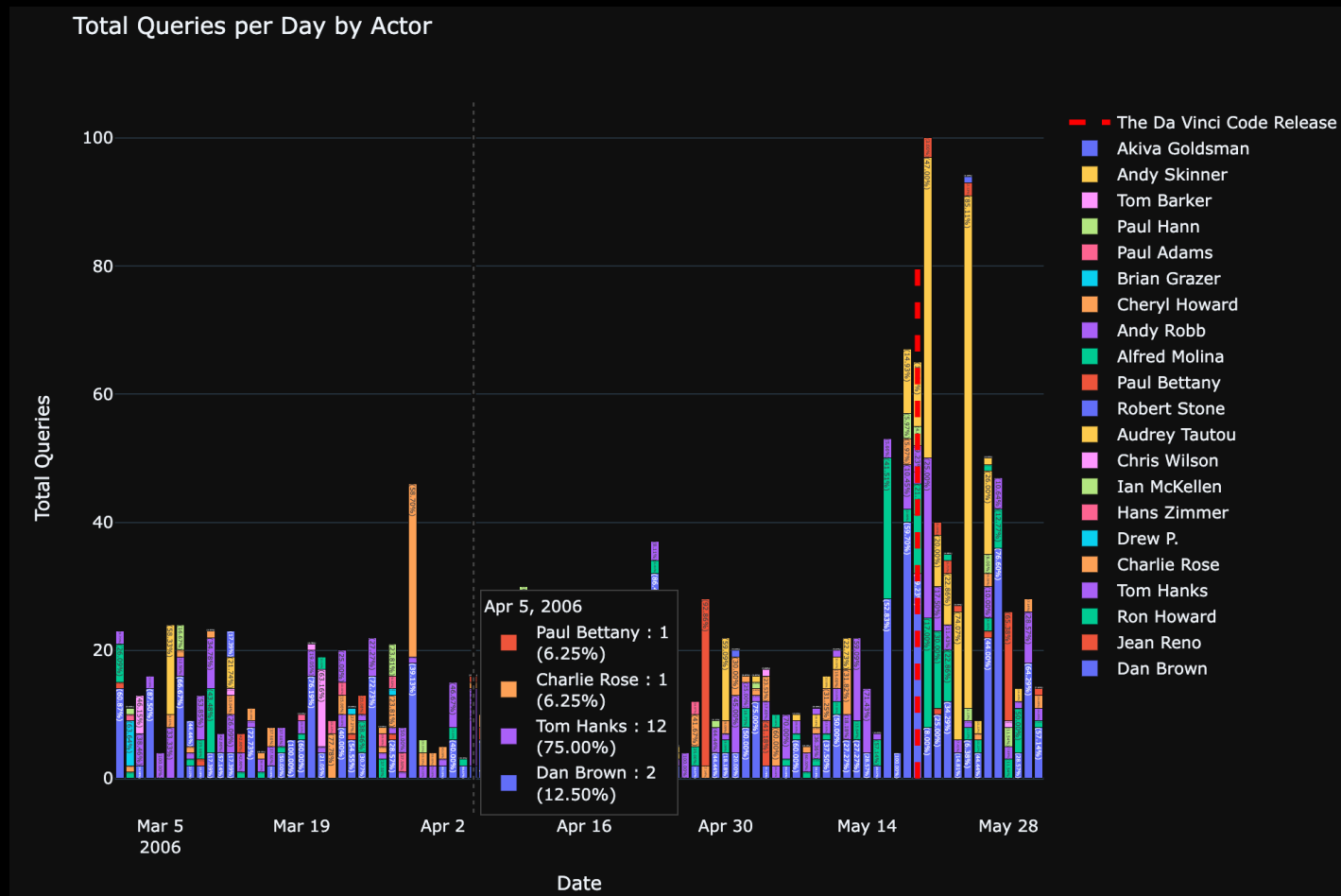
Rote Strichlinie am Publikationstag hinzugefügt.

Endlich mit Python visualisieren!!

Die Grafik für tägliche Darstellung wird eingestellt und gezeichnet.

```
1  # Einstellungen für das Diagrammlayout
2  layout = go.Layout(
3      title='Total Queries per Day by Actor',
4      xaxis=dict(title='Date'),
5      yaxis=dict(title='Total Queries'),
6      barmode='stack',
7      hovermode='x unified',
8      height=700
9  )
10
11 # Eine Grafik erstellen und Daten hinzufügen
12 fig = go.Figure(data=data, layout=layout)
13 fig.add_trace(release_line)
14
15 fig.show()
```

Erste Ergebnisse - visualisiert



Wie erwartet, haben die Nutzer kurz vor und nach der Veröffentlichung des Films deutlich häufiger nach den Schauspielern gegoogelt, die in dem Film mitgespielt haben.

Noch eine kleine Darstellung

Durchschnittliche Anfragen pro Woche

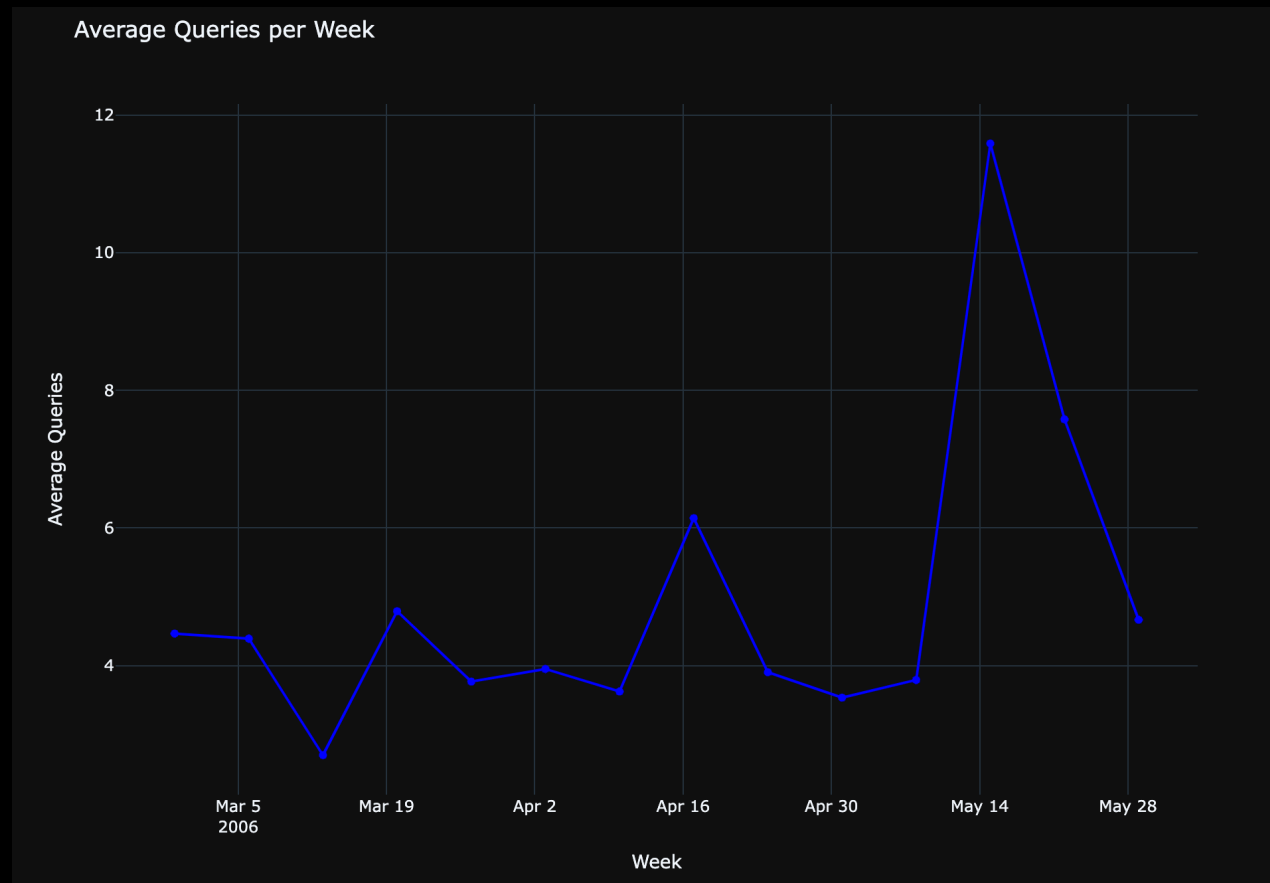
Datenvorbereitung

```
1 df['week'] = df['date'].dt.to_period('W').apply(lambda r: r.start_time)
2 weekly_avg = df.groupby('week')['queries'].mean().reset_index()
3
4 # Erstellen eines zweiten Diagramms
5 fig2 = go.Figure()
6
7 Executed at 2024.07.01 19:37:19 in 13ms
```

Die Grafik wird eingestellt und gezeichnet.

```
1 fig2.add_trace(go.Scatter(
2     x=weekly_avg['week'],
3     y=weekly_avg['queries'],
4     mode='lines+markers',
5     name='Average Weekly Queries',
6     line=dict(color='blue', width=2)
7 ))
8
9 # Einstellungen für das zweite Diagrammlayout
10 layout2 = go.Layout(
11     title='Average Queries per Week',
12     xaxis=dict(title='Week'),
13     yaxis=dict(title='Average Queries'),
14     hovermode='x unified',
15     height=700
16 )
17
18 fig2.update_layout(layout2)
19
20 fig2.show()
21
22 Executed at 2024.07.01 19:37:21 in 10ms
```

Durchschnittliche Anfragen pro Woche - visualisiert!



Fortsetzung der Untersuchung

Nach der Analyse der Google-Suchanfragen für Schauspieler aus dem Film "The Da Vinci Code" (2006) möchten wir unsere Forschung weiter ausbauen, um die umfassenderen Auswirkungen des Films zu untersuchen.

Durch die Erweiterung unserer Forschung auf diese Bereiche hoffen wir, ein umfassenderes Bild der kulturellen und wirtschaftlichen Auswirkungen des Films zu zeichnen.

Einfluss des Films auf den Tourismus

Ziel: Untersuchung, wie der Film den Tourismus an den im Film gezeigten Orten beeinflusst hat.

Methode: Analyse von Besucherzahlen und Tourismuseinnahmen vor und nach der Veröffentlichung des Films.

Datenquellen: Tourismusbehörden, historische Besucherzahlen, Buchungen und Reiseberichte.

Interesse an Kunstwerken und Verschwörungstheorien

Ziel: Erforschung des Anstiegs des Interesses an Kunstwerken und Verschwörungstheorien, die im Film thematisiert wurden.

Methode: Analyse von Google-Suchtrends, Bibliotheksausleihen und Verkaufszahlen von Büchern und Kunstführern.

Datenquellen: AOL search data leak 2006.