Uebung3

Benötigte Pakete:

```
- Oracle-Pandas: 'pip install pandas-oracle',
https://github.com/cwade/pandas_oracle
- cx_Oracle: siehe unten
- Seaborn, Pandas, Numpy, etc
```

Anleitung:

- Installiert `cx_Oracle`
- Ladet den Oracle Instant Client herunter: https://www.oracle.com/database/technologies/instant-client/downloads.html
- Führt die entsprechenden Installationsanweisungen durch

In [1]:

```
# Bei Bedarf auskommentieren
!python -m pip install cx_Oracle --upgrade
!python -m pip install pandas-oracle
```

```
Requirement already satisfied: cx_Oracle in /opt/anaconda3/envs/data/lib/python3.1
1/site-packages (8.3.0)
Requirement already satisfied: pandas-oracle in /opt/anaconda3/envs/data/lib/python
3.11/site-packages (2.1.4)
Requirement already satisfied: pandas in /opt/anaconda3/envs/data/lib/python3.11/si
te-packages (from pandas-oracle) (2.2.3)
Requirement already satisfied: cx-Oracle in /opt/anaconda3/envs/data/lib/python3.1
1/site-packages (from pandas-oracle) (8.3.0)
Requirement already satisfied: pyaml in /opt/anaconda3/envs/data/lib/python3.11/sit
e-packages (from pandas-oracle) (24.12.1)
Requirement already satisfied: numpy>=1.23.2 in /opt/anaconda3/envs/data/lib/python
3.11/site-packages (from pandas->pandas-oracle) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/anaconda3/envs/data/l
ib/python3.11/site-packages (from pandas->pandas-oracle) (2.9.0)
Requirement already satisfied: pytz>=2020.1 in /opt/anaconda3/envs/data/lib/python
3.11/site-packages (from pandas->pandas-oracle) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in /opt/anaconda3/envs/data/lib/pytho
n3.11/site-packages (from pandas->pandas-oracle) (2023.3)
Requirement already satisfied: PyYAML in /opt/anaconda3/envs/data/lib/python3.11/si
te-packages (from pyaml->pandas-oracle) (6.0.2)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/envs/data/lib/python3.11/
site-packages (from python-dateutil>=2.8.2->pandas->pandas-oracle) (1.16.0)
```

Pfad setzen

In [4]:

```
import cx_Oracle

# Pfad anpassen!!
path_to_instant_client = "/Users/luhangfang/Documents/Software/instantclient_23_3"
try:
    cx_Oracle.init_oracle_client(lib_dir=path_to_instant_client)
except cx_Oracle.ProgrammingError as e:
    if "already been initialized" not in str(e):
        raise
```

HU-VPN aktivieren oder in HU-Netzwerk (WLAN) einloggen.

Anleitungen: https://www.cms.hu-berlin.de/de/dl/netze/vpn

Verbindungsdaten:

Die Verbindungsdaten werden in `config.yml` gespeichert. Die Datein *muss* im gleichen Ordner liegen.

In [3]:

```
import cx_Oracle

import pandas_oracle.tools as pt
import numpy as np
import pandas as pd

# visualization
import seaborn as sns
import matplotlib.pyplot as plt
import folium
from folium import plugins
```

Verbindungsaufbau:

In [5]:

```
def run_query(query) :
    ## opening conn
    conn = pt.open_connection("config.yml")

    try:
        ## passing the conn object to the query_to_df
        df1 = pt.query_to_df(query, conn, 10000)
        return df1
    except Exception as e:
        print("An exception occurred")
        print(str(e))
    finally:
        ## close connection
        pt.close_connection(conn)
```

Wichtig: die Verbindungs zur DB muss geschlossen werden, wenn ein Fehler auftritt.

EXPLAIN PLAN

In [6]:

```
def explain_plan_query(query) :
    ## opening conn
    conn = pt.open_connection("config.yml")
    try:
        # There is small change
        #  ("+query+")" => " + query
        # query_explain = "EXPLAIN PLAN FOR ("+query+")"
        query_explain = "EXPLAIN PLAN FOR " + query
        pt.execute(query_explain, conn)
        query2 = "SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY())"

        df = pt.query_to_df(query2, conn, 10000)
        return df
    except Exception as e:
        print("An exception occurred:", str(e))
    finally:
        ## close connection
        pt.close_connection(conn)
```

In [7]:

```
query = "SELECT * FROM delays"
df = explain_plan_query(query)

print(df.to_string())
```

```
                                                                    PLAN_TABLE_OUTPUT
0                                                      Plan hash value: 3893133417
1
2  ----------------------------------------------------------------------------
3  | Id  | Operation         | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
4  ----------------------------------------------------------------------------
5  |   0 | SELECT STATEMENT  |        |  5332K|   472M| 22662   (2)| 00:00:01 |
6  |   1 |  TABLE ACCESS FULL| DELAYS |  5332K|   472M| 22662   (2)| 00:00:01 |
7  ----------------------------------------------------------------------------
```

1.Ursache der Verspätung

SQL

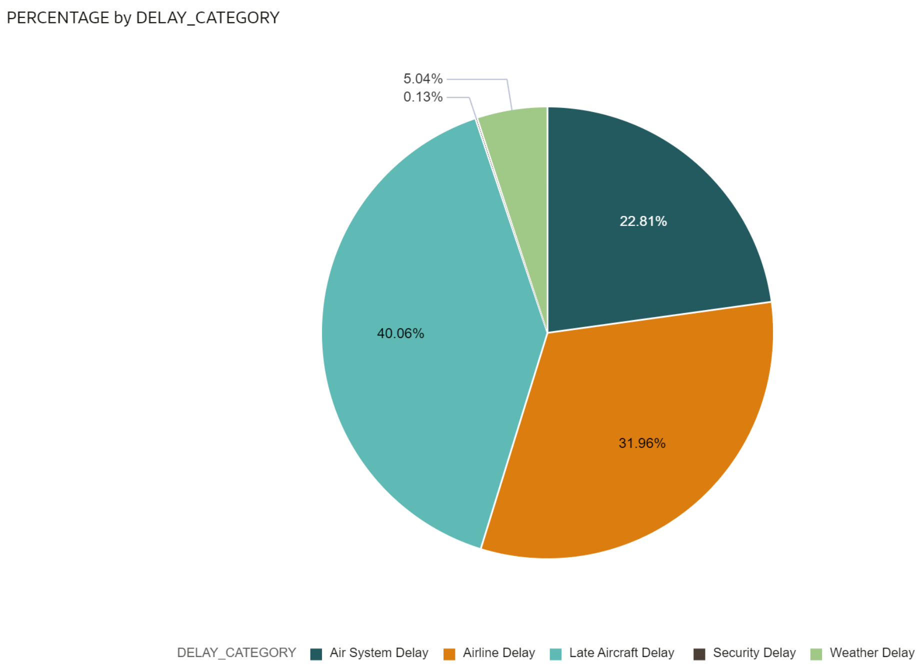
In [8]:

```
query1_1 = """
SELECT
    CASE
        WHEN delay_type = 'air_system_delay' THEN 'Air System Delay'
        WHEN delay_type = 'security_delay' THEN 'Security Delay'
        WHEN delay_type = 'airline_delay' THEN 'Airline Delay'
        WHEN delay_type = 'late_aircraft_delay' THEN 'Late Aircraft Delay'
        WHEN delay_type = 'weather_delay' THEN 'Weather Delay'
    END as delay_category,
    SUM(delay_minutes) as total_minutes,
    ROUND(SUM(delay_minutes) * 100.0 / SUM(SUM(delay_minutes)) OVER (), 2) as perce
FROM (
    SELECT 'air_system_delay' as delay_type, COALESCE(air_system_delay, 0) as delay
    UNION ALL
    SELECT 'security_delay', COALESCE(security_delay, 0) FROM delays
    UNION ALL
    SELECT 'airline_delay', COALESCE(airline_delay, 0) FROM delays
    UNION ALL
    SELECT 'late_aircraft_delay', COALESCE(late_aircraft_delay, 0) FROM delays
    UNION ALL
    SELECT 'weather_delay', COALESCE(weather_delay, 0) FROM delays
) unpivoted
GROUP BY delay_type
HAVING SUM(delay_minutes) > 0
ORDER BY total_minutes DESC
"""
df1_1 = run_query(query1_1)
df1_1
```

Out[8]:

	DELAY_CATEGORY	TOTAL_MINUTES	PERCENTAGE
0	Late Aircraft Delay	23767674	40.06
1	Airline Delay	18966945	31.96
2	Air System Delay	13533065	22.81
3	Weather Delay	2991008	5.04
4	Security Delay	77945	0.13

Visualisierung



EXPLAIN PLAN und Indices

In [9]:

```
query1_2 = """
CREATE INDEX idx_delay_facts_delays ON delays (
    air_system_delay,
    security_delay,
    airline_delay,
    late_aircraft_delay,
    weather_delay
)
"""


df1_3 = explain_plan_query(query1_1)
print(df1_3.to_string())
```

```
                                                             PLAN_TABLE_OUTP
UT
0                                                   Plan hash value: 38516895
23
1
2   ------------------------------------------------------------------------------
--
3   | Id  | Operation              | Name    | Rows  | Bytes | Cost (%CPU)| Time
|
4   ------------------------------------------------------------------------------
--
5   |   0 | SELECT STATEMENT       |         |   26M|   610M|   115K  (4)| 00:00:05
|
6   |   1 |  WINDOW SORT           |         |   26M|   610M|   115K  (4)| 00:00:05
|
7   |*  2 |   FILTER               |         |      |       |            |
|
8   |   3 |    HASH GROUP BY       |         |   26M|   610M|   115K  (4)| 00:00:05
|
9   |   4 |     VIEW               |         |   26M|   610M|   113K  (2)| 00:00:05
|
10  |   5 |      UNION-ALL         |         |      |       |            |
|
11  |   6 |       TABLE ACCESS FULL| DELAYS  | 5332K|    10M| 22630   (2)| 00:00:01
|
12  |   7 |       TABLE ACCESS FULL| DELAYS  | 5332K|    10M| 22630   (2)| 00:00:01
|
13  |   8 |       TABLE ACCESS FULL| DELAYS  | 5332K|    10M| 22630   (2)| 00:00:01
|
14  |   9 |       TABLE ACCESS FULL| DELAYS  | 5332K|    10M| 22630   (2)| 00:00:01
|
15  |  10 |       TABLE ACCESS FULL| DELAYS  | 5332K|    10M| 22630   (2)| 00:00:01
|
16  ------------------------------------------------------------------------------
--
17
18                               Predicate Information (identified by operation i
d):
19                               -------------------------------------------------
--
20
21                                                  2 - filter(SUM("DELAY_MINUTES")>
0)
```

3.2 Analyse:Region

3.2.1 SQL

In [12]:

```
query2_1 ="""
SELECT
    f.origin_state as STATE,
    f.origin_airport as AIRPORT,
    f.origin_city as City,
    f.origin_latitude/100000.0 as LATITUDE,
    f.origin_longitude/100000.0 as LONGITUDE,
    COUNT(*) as TOTAL_FLIGHTS,
    SUM(CASE WHEN d.departure_delay > 0 THEN 1 ELSE 0 END) as DELAYED_FLIGHTS,
    ROUND(SUM(CASE WHEN d.departure_delay > 0 THEN 1 ELSE 0 END) * 100.0 / COUNT(*)
    ROUND(AVG(CASE WHEN d.departure_delay > 0 THEN d.departure_delay ELSE NULL END)
FROM Flight f
JOIN Delays d ON f.flight_id = d.flight_id
GROUP BY
    f.origin_state,
    f.origin_airport,
    f.origin_city,
    f.origin_latitude,
    f.origin_longitude
HAVING COUNT(*) > 100
"""
df2_1 = run_query(query2_1)
df2_1
```

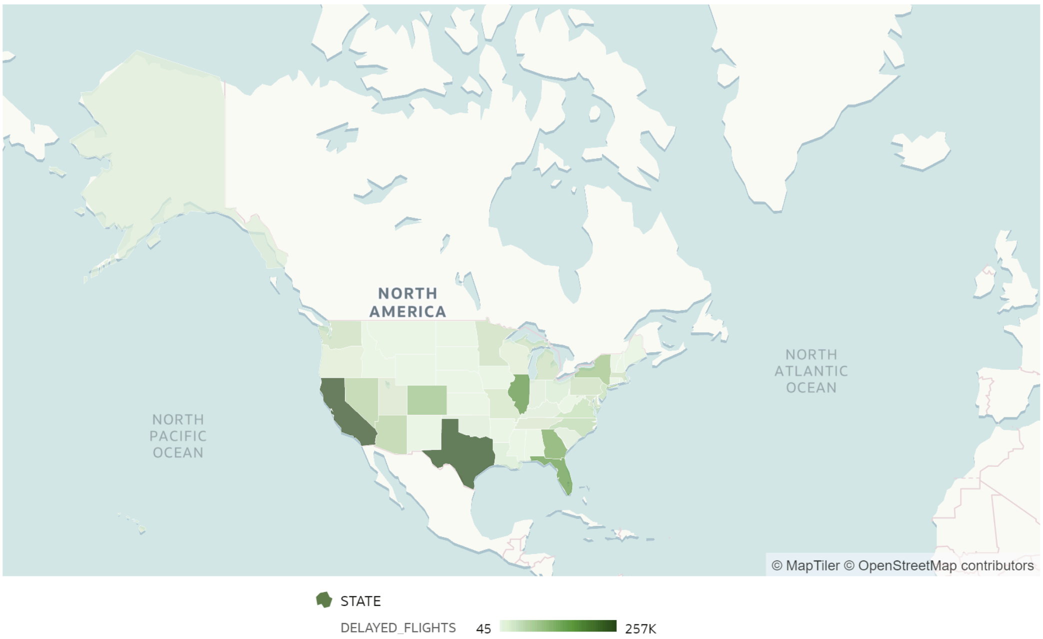
Out[12]:

	STATE	AIRPORT	CITY	LATITUDE	LONGITUDE	TOTAL_FLIGHTS	DELAYED
0	TN	McGhee Tyson Airport	Knoxville	35.81249	-83.99286	6963	
1	GA	Savannah/Hilton Head International Airport	Savannah	32.12758	-81.20214	7542	
2	NC	Albert J. Ellis Airport	Jacksonville	34.82916	-77.61214	1136	
3	TX	Easterwood Airport	College Station	30.58859	-96.36382	2311	
4	TX	William P. Hobby Airport	Houston	29.64542	-95.27889	52042	
...	...	...	...	...	...	...	
309	AK	Ketchikan International Airport	Ketchikan	55.35557	-131.71374	2319	
310	MI	Bishop International Airport	Flint	42.96550	-83.74346	4746	
311	UT	Canyonlands Field	Moab	38.75496	-109.75484	206	
312	MI	Kalamazoo/Battle Creek International Airport	Kalamazoo	42.23488	-85.55206	1845	
313	AK	Petersburg James A. Johnson Airport	Petersburg	56.80165	-132.94528	664	

314 rows × 9 columns

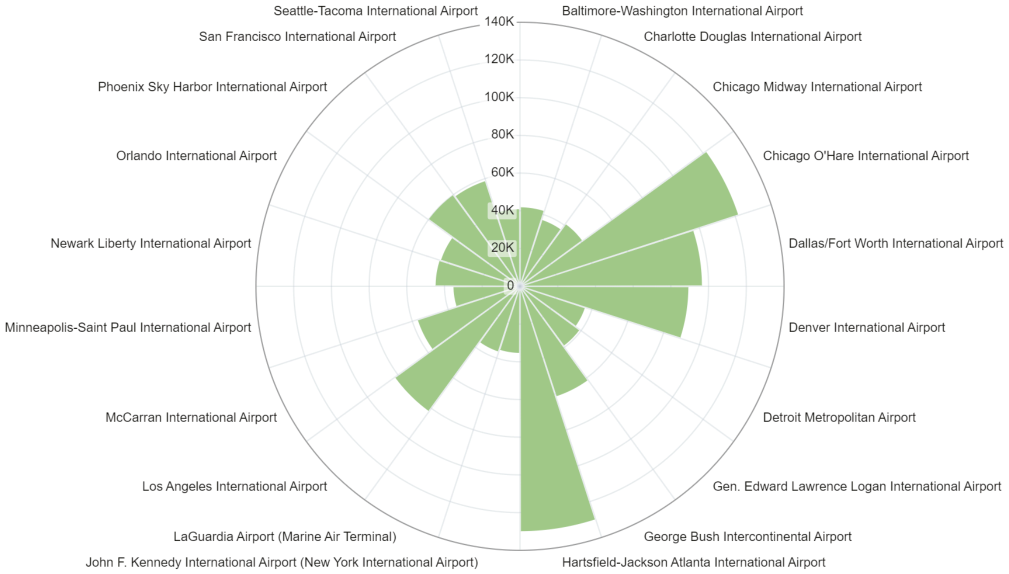
Visualisierung

STATE, DELAYED_FLIGHTS



DELAYED_FLIGHTS by AIRPORT

Top 20 DELAYED_FLIGHTS



EXPLAIN PLAN und Indices

```
In [13]: query2_2 = """
CREATE INDEX idx_delays_flight_id ON Delays(flight_id);

CREATE INDEX idx_flight_origin ON Flight(
    origin_state,
    origin_airport,
    origin_city,
    origin_latitude,
    origin_longitude
);

CREATE INDEX idx_flight_id ON Flight(flight_id);

CREATE INDEX idx_departure_delay ON Delays(departure_delay);
```

```
"""


df2_3 = explain_plan_query(query2_1)
print(df2_3.to_string())
```

```
                                                                                 PLAN_TABL
E_OUTPUT
0                                                                        Plan hash value: 5
78984892
1
2   -------------------------------------------------------------------------------
--------
3   | Id  | Operation            | Name   | Rows  | Bytes |TempSpc| Cost (%CPU)| Ti
me     |
4   -------------------------------------------------------------------------------
--------
5   |   0 | SELECT STATEMENT     |        |  3507 |   267K|       | 33622   (3)| 0
0:00:02 |
6   |*  1 |  FILTER              |        |       |       |       |            |
|
7   |   2 |   HASH GROUP BY      |        |  3507 |   267K|       | 33622   (3)| 0
0:00:02 |
8   |*  3 |    HASH JOIN         |        |  5264K|   391M|    38M| 33225   (2)| 0
0:00:02 |
9   |   4 |     TABLE ACCESS FULL| FLIGHT |   493K|    32M|       |  3322   (1)| 0
0:00:01 |
10  |   5 |     TABLE ACCESS FULL| DELAYS |  5332K|    45M|       | 22630   (2)| 0
0:00:01 |
11  -------------------------------------------------------------------------------
--------
12
13                                    Predicate Information (identified by operat
ion id):
14                                    -------------------------------------------
--------
15
16                                                                1 - filter(COUNT
(*)>100)
17                                                        3 - access("F"."FLIGHT_ID"="D"."FLI
GHT_ID")
```

3.Analyse:Weather

3.1 10 Tage meisten Stornierungen wegen Wetter

SQL

In [14]:

```
# Query 1: Top 10 days with most weather-related cancellations
query3_1 = """
SELECT
    f.year,
    f.month,
    f.day,
    COUNT(*) as WEATHER_CANCELLATIONS,
    ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*)
                              FROM Delays
                              WHERE cancelled = 1
                              AND cancellation_reason = 'B'), 2) as PERCENTAGE_OF_T
FROM Delays f
WHERE f.cancelled = 1
AND f.cancellation_reason = 'B'
GROUP BY f.year, f.month, f.day
"""
```

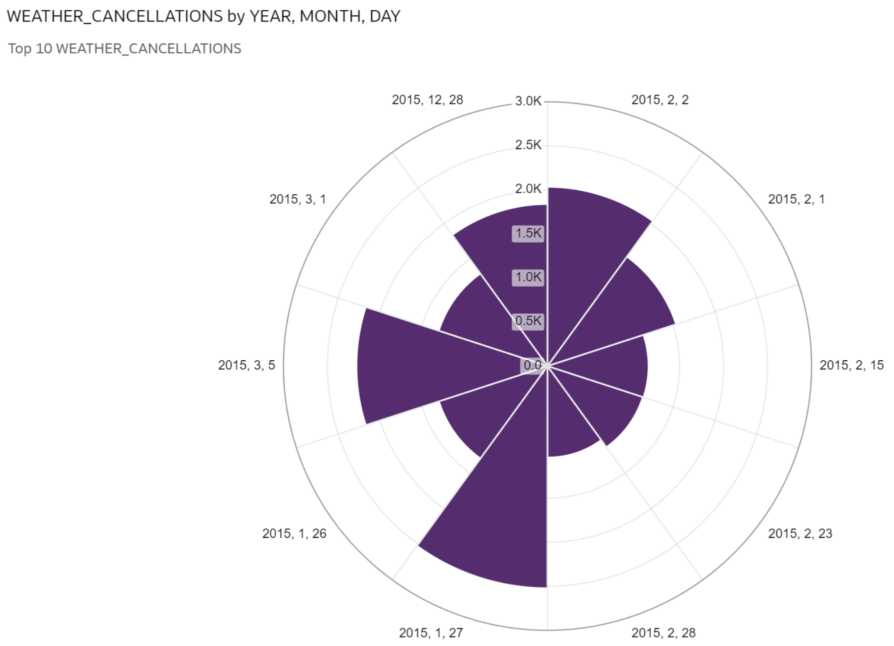
```
df3_1 = run_query(query3_1)
df3_1
```

Out[14]:

	YEAR	MONTH	DAY	WEATHER_CANCELLATIONS	PERCENTAGE_OF_TOTAL
0	2015	1	9	136	0.28
1	2015	1	14	81	0.17
2	2015	1	15	27	0.06
3	2015	1	16	19	0.04
4	2015	2	6	135	0.28
...	...	...	...	...	...
326	2015	11	30	30	0.06
327	2015	12	8	34	0.07
328	2015	12	11	14	0.03
329	2015	12	10	22	0.05
330	2015	12	18	72	0.15

331 rows × 5 columns

Visualisierung



EXPLAIN PLAN und Indices

In [15]:

```
query3_2 = """
y-- Index for filtering conditions
CREATE INDEX idx_delays_cancel ON Delays(cancelled, cancellation_reason);

-- Composite index for grouping columns
CREATE INDEX idx_delays_date ON Delays(year, month, day);

-- Combined index covering all conditions and grouping columns
CREATE INDEX idx_delays_cancel_date ON Delays(cancelled, cancellation_reason, year,
"""
```

```python
df3_3 = explain_plan_query(query3_1)
print(df3_3.to_string())
```

```
                                                                PLAN_TABLE_OUTPUT
0                                                      Plan hash value: 1046980958
1                                                                                 
2   -----------------------------------------------------------------------------
3    | Id  | Operation          | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
4   -----------------------------------------------------------------------------
5    |   0 | SELECT STATEMENT   |        |   331 |  4965 | 45240   (2)| 00:00:02 |
6    |   1 |  SORT AGGREGATE    |        |     1 |     5 |            |          |
7    |*  2 |   TABLE ACCESS FULL| DELAYS | 47874 |   233K| 22618   (2)| 00:00:01 |
8    |   3 |  HASH GROUP BY     |        |   331 |  4965 | 45240   (2)| 00:00:02 |
9    |*  4 |   TABLE ACCESS FULL| DELAYS | 47874 |   701K| 22618   (2)| 00:00:01 |
10  -----------------------------------------------------------------------------
11                                                                                
12                            Predicate Information (identified by operation id):
13                            ---------------------------------------------------
14                                                                                
15                        2 - filter("CANCELLED"=1 AND "CANCELLATION_REASON"='B')
16                4 - filter("F"."CANCELLED"=1 AND "F"."CANCELLATION_REASON"='B')
17                                                                                
18                                                                           Note
19                                                                          -----
20                          - dynamic statistics used: dynamic sampling (level=2)
21                                 - 1 Sql Plan Directive used for this statement
```

3.2 An welchen Flughafen

SQL

In [16]:

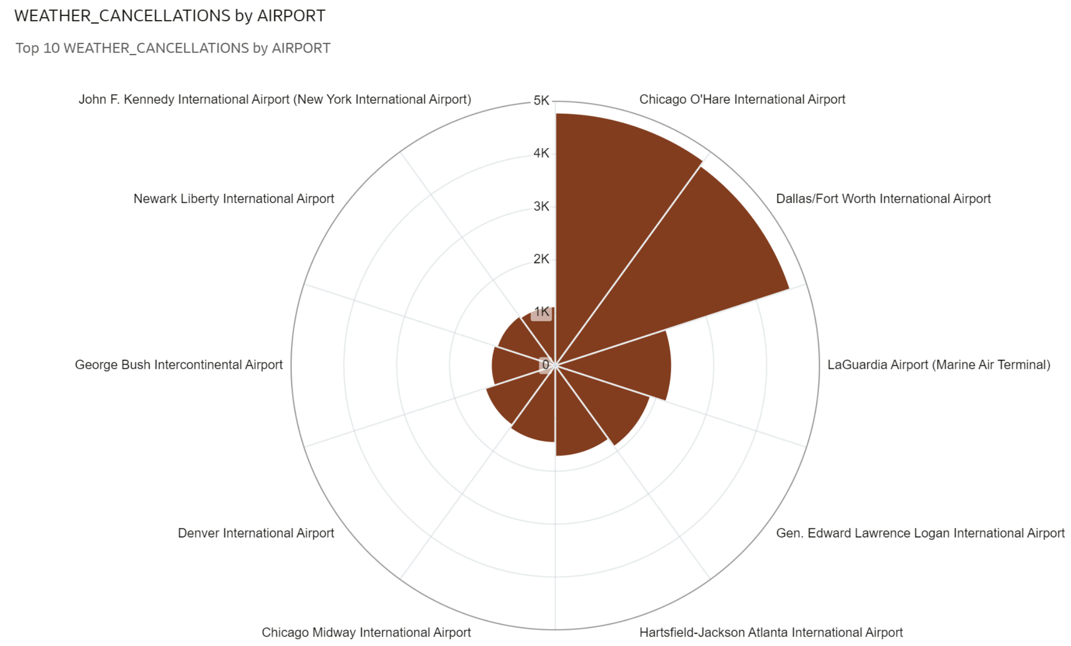
```python
query3_4 = """
SELECT
    f.origin_state as STATE,
    f.origin_airport as AIRPORT,
    COUNT(*) as TOTAL_CANCELLATIONS,
    SUM(CASE WHEN d.cancellation_reason = 'B' THEN 1 ELSE 0 END) as WEATHER_CANCELL
    ROUND(
        (SUM(CASE WHEN d.cancellation_reason = 'B' THEN 1 ELSE 0 END) * 100.0) /
        NULLIF(COUNT(*), 0),
        2
    ) as CANCELLATION_PERCENTAGE,
    COUNT(DISTINCT TRUNC(f.scheduled_departure)) as AFFECTED_DAYS
FROM Flight f
JOIN Delays d ON f.flight_id = d.flight_id
WHERE d.cancelled = 1
GROUP BY
    f.origin_state,
    f.origin_airport
HAVING SUM(CASE WHEN d.cancellation_reason = 'B' THEN 1 ELSE 0 END) > 0
ORDER BY WEATHER_CANCELLATIONS DESC
"""
df3_4 = run_query(query3_4)
df3_4
```

Out[16]:

	STATE	AIRPORT	TOTAL_CANCELLATIONS	WEATHER_CANCELLATIONS	CANCELLATIO
0	IL	Chicago O'Hare International Airport	8547	4769	
1	TX	Dallas/Fort Worth International Airport	6254	4664	
2	NY	LaGuardia Airport (Marine Air Terminal)	4531	2191	
3	MA	Gen. Edward Lawrence Logan International Airport	2654	1882	
4	GA	Hartsfield-Jackson Atlanta International Airport	2557	1707	
...	...	...	...	...	
306	UT	Canyonlands Field	1	1	
307	NY	Plattsburgh International Airport	2	1	
308	HI	Lihue Airport	44	1	
309	NY	Ithaca Tompkins Regional Airport	4	1	
310	ME	Bangor International Airport	3	1	

311 rows × 6 columns

Visualisierung



EXPLAIN PLAN und Indices

In [17]:

```
query3_5 = """
-- 1. Index for JOIN operations
CREATE INDEX idx_delays_flight_id ON Delays(flight_id);
CREATE INDEX idx_flight_id ON Flight(flight_id);

-- 2. Composite index for filtering and analysis on Delays table
CREATE INDEX idx_delays_cancel ON Delays(
    cancelled,
    cancellation_reason,
    flight_id
);

-- 3. Composite index for grouping columns on Flight table
CREATE INDEX idx_flight_origin ON Flight(
    origin_state,
    origin_airport,
    scheduled_departure
);
"""


df3_5 = explain_plan_query(query3_4)
print(df3_5.to_string())
```

```
                                                                               PLAN
_TABLE_OUTPUT
0                                                                    Plan hash valu
e: 2411417394
1
2   -------------------------------------------------------------------------------
-------------
3   | Id  | Operation                | Name     | Rows  | Bytes |TempSpc| Cost (%CP
U)| Time     |
4   -------------------------------------------------------------------------------
-------------
5   |   0 | SELECT STATEMENT         |          |   615 | 57195 |       | 26620
(2)| 00:00:02 |
6   |   1 |  SORT ORDER BY           |          |   615 | 57195 |       | 26620
(2)| 00:00:02 |
7   |*  2 |   FILTER                 |          |       |       |       |
|          |
8   |   3 |    HASH GROUP BY         |          |   615 | 57195 |       | 26620
(2)| 00:00:02 |
9   |   4 |     VIEW                 | VW_DAG_0 | 23655 |  2148K|       | 26616
(2)| 00:00:02 |
10  |   5 |      HASH GROUP BY       |          | 23655 |  1409K|  6552K| 26616
(2)| 00:00:02 |
11  |*  6 |       HASH JOIN          |          | 87429 |  5208K|       | 25929
(2)| 00:00:02 |
12  |*  7 |        TABLE ACCESS FULL| DELAYS   | 87430 |   853K|       | 22602
(2)| 00:00:01 |
13  |   8 |        TABLE ACCESS FULL| FLIGHT   |   493K|    24M|       |  3322
(1)| 00:00:01 |
14  -------------------------------------------------------------------------------
-------------
15
16                                          Predicate Information (identified by o
peration id):
17                                          ---------------------------------------
-------------
18
19                                                                   2 - filter(SUM
("ITEM_4")>0)
20                                                     6 - access("F"."FLIGHT_I
D"="D"."FLIGHT_ID")
21                                                                 7 - filter
("D"."CANCELLED"=1)
22
23
Note
24
-----
25                                         - dynamic statistics used: dynamic sampl
ing (level=2)
26                                                - 1 Sql Plan Directive used for t
his statement
```

4. Beste Zeit

4.1 Wochentage

SQL

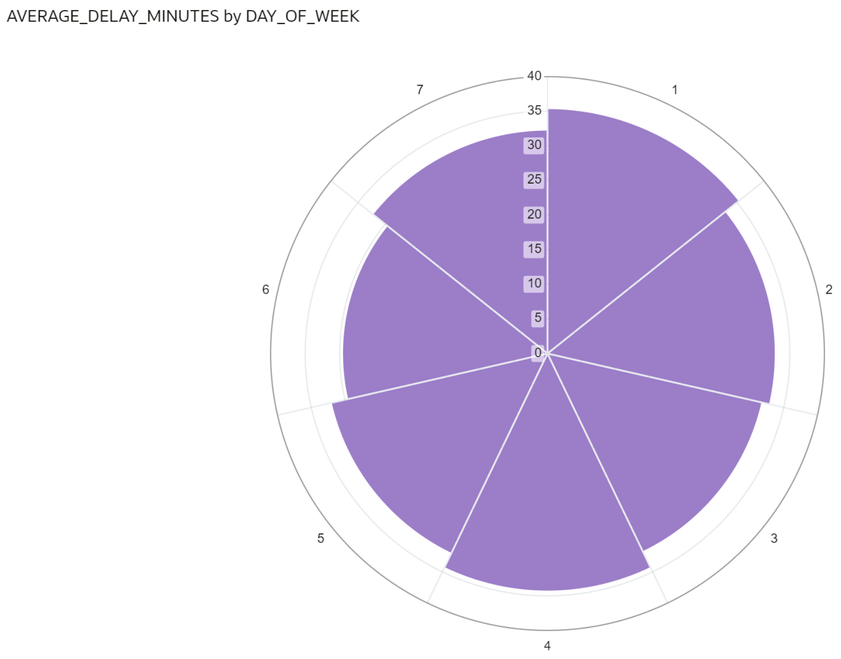
In [18]:

```
query4_1 = """
SELECT
    t.DAY_OF_WEEK AS DAY_OF_WEEK,
    ROUND(AVG(f.DEPARTURE_DELAY +
        NVL(f.TAXI_OUT, 0) +
        NVL(f.SECURITY_DELAY, 0) +
        NVL(f.AIRLINE_DELAY, 0) +
        NVL(f.WEATHER_DELAY, 0) +
        NVL(f.AIR_SYSTEM_DELAY, 0)
    ), 2) AS AVERAGE_DELAY_MINUTES,
    COUNT(*) AS TOTAL_FLIGHTS
FROM DELAYS f
JOIN TIME t ON
    f.DAY = t.DAY AND
    f.MONTH = t.MONTH AND
    f.YEAR = t.YEAR
WHERE f.CANCELLED = 0
GROUP BY t.DAY_OF_WEEK
ORDER BY AVERAGE_DELAY_MINUTES ASC
"""
df4_1 = run_query(query4_1)
df4_1
```

Out[18]:

	DAY_OF_WEEK	AVERAGE_DELAY_MINUTES	TOTAL_FLIGHTS
0	6	29.53	629444
1	3	31.69	780519
2	5	31.97	772027
3	7	32.21	741681
4	2	32.82	765893
5	4	34.24	777598
6	1	35.31	778322

Visualisierung



EXPLAIN PLAN und Indices

In [19]:

```
query4_2 = """
-- 1. Composite index for the JOIN conditions on Delays table
CREATE INDEX idx_delays_date ON Delays(
    year,
    month,
    day,
    cancelled
);

-- 2. Composite index for the JOIN conditions on Time table
CREATE INDEX idx_time_date ON Time(
    year,
    month,
    day,
    day_of_week
);

-- 3. Index for delay columns (optional, for covering index)
CREATE INDEX idx_delays_all_delays ON Delays(
    departure_delay,
    taxi_out,
    security_delay,
    airline_delay,
    weather_delay,
    air_system_delay
);
"""


df4_3 = explain_plan_query(query4_1)
print(df4_3.to_string())
```

```
                                                                  PLAN_TABLE_OUTPUT
0                                                       Plan hash value: 2274342594
1   
2   -------------------------------------------------------------------------------
3   | Id  | Operation            | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
4   -------------------------------------------------------------------------------
5   |   0 | SELECT STATEMENT     |        |     7 |   287 | 23483   (5)| 00:00:01 |
6   |   1 |  SORT ORDER BY       |        |     7 |   287 | 23483   (5)| 00:00:01 |
7   |   2 |   HASH GROUP BY      |        |     7 |   287 | 23483   (5)| 00:00:01 |
8   |*  3 |    HASH JOIN         |        |  5245K|   205M| 22691   (2)| 00:00:01 |
9   |   4 |     TABLE ACCESS FULL| TIME   |   365 |  4745 |     3   (0)| 00:00:01 |
10  |*  5 |     TABLE ACCESS FULL| DELAYS |  5245K|   140M| 22649   (2)| 00:00:01 |
11  -------------------------------------------------------------------------------
12  
13                              Predicate Information (identified by operation id):
14                              ---------------------------------------------------
15  
16                 3 - access("F"."DAY"="T"."DAY" AND "F"."MONTH"="T"."MONTH" AND 
17                                                           "F"."YEAR"="T"."YEAR")
18                                                    5 - filter("F"."CANCELLED"=0)
```

4.2 Sationale Unterschiede

SQL

In [33]:

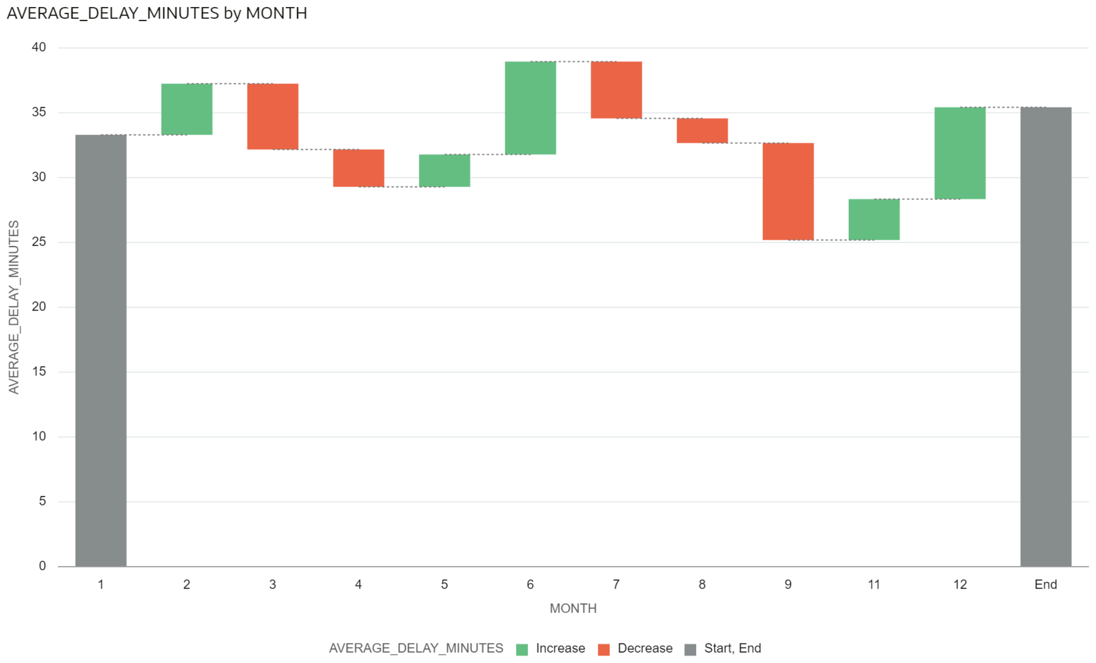
```
query4_4 = """
SELECT
    t.MONTH AS MONTH,
    ROUND(AVG(
        NVL(f.DEPARTURE_DELAY, 0) +
        NVL(f.TAXI_OUT, 0) +
        NVL(f.SECURITY_DELAY, 0) +
        NVL(f.AIRLINE_DELAY, 0) +
        NVL(f.WEATHER_DELAY, 0) +
        NVL(f.AIR_SYSTEM_DELAY, 0)
    ), 2) AS AVERAGE_DELAY_MINUTES,
    COUNT(*) AS TOTAL_FLIGHTS
FROM DELAYS f
```

```
JOIN TIME t ON
    f.DAY = t.DAY AND
    f.MONTH = t.MONTH AND
    f.YEAR = t.YEAR
WHERE f.CANCELLED = 0
GROUP BY t.MONTH
ORDER BY t.MONTH ASC
"""
df4_4 = run_query(query4_4)
df4_4
```

Out[33]:

	MONTH	AVERAGE_DELAY_MINUTES	TOTAL_FLIGHTS
0	1	33.30	457986
1	2	37.24	408674
2	3	32.17	493310
3	4	29.29	480631
4	5	31.78	491299
5	6	38.95	494777
6	7	34.57	515912
7	8	32.67	505484
8	9	25.19	462871
9	11	28.34	463373
10	12	35.42	471167

Visualisierung



EXPLAIN PLAN und Indices

In [21]:

```
query4_5 = """
-- 1. Composite index for the JOIN conditions on Delays table
CREATE INDEX idx_delays_date_cancel ON Delays(
    year,
    month,
    day,
    cancelled
);

-- 2. Composite index for the JOIN conditions on Time table
CREATE INDEX idx_time_date_month ON Time(
    year,
    month,
    day
);

-- 3. Index for delay columns
CREATE INDEX idx_delays_all_delays ON Delays(
    departure_delay,
    taxi_out,
    security_delay,
    airline_delay,
    weather_delay,
    air_system_delay
);
"""


df4_5 = explain_plan_query(query4_4)
print(df4_5.to_string())
```

```
                                                                     PLAN_TABLE_OUT
PUT
0                                                         Plan hash value: 1954953
516
1
2   -------------------------------------------------------------------------------
---
3   | Id  | Operation              | Name     | Rows  | Bytes | Cost (%CPU)| Time     
|
4   -------------------------------------------------------------------------------
---
5   |   0 | SELECT STATEMENT       |          |    11 |   649 | 23046   (4)| 00:00:0
1 |
6   |   1 |  SORT GROUP BY         |          |    11 |   649 | 23046   (4)| 00:00:0
1 |
7   |   2 |   NESTED LOOPS         |          |   334 | 19706 | 23045   (4)| 00:00:0
1 |
8   |   3 |    VIEW                | VW_GBC_6 |   334 | 16366 | 23045   (4)| 00:00:0
1 |
9   |   4 |     HASH GROUP BY      |          |   334 |  9352 | 23045   (4)| 00:00:0
1 |
10  |*  5 |      TABLE ACCESS FULL| DELAYS   |  5245K|   140M| 22649   (2)| 00:00:0
1 |
11  |*  6 |    INDEX UNIQUE SCAN   | PK_TIME  |     1 |    10 |     0   (0)| 00:00:0
1 |
12  -------------------------------------------------------------------------------
---
13
14                                 Predicate Information (identified by operation i
d):
15                                 ------------------------------------------------
---
16
17                                                       5 - filter("F"."CANCELLE  
D"=0)
18                         6 - access("ITEM_3"="T"."DAY" AND "ITEM_2"="T"."MONTH" A
ND
19                                                                "ITEM_1"="T"."YEA
R")
20
21                                                                                N
ote
22                                                                               --
---
23                               - dynamic statistics used: dynamic sampling (level
=2)
```

5. Analyse: Fluglinien

5.1 Fluglinien mit meisten Flügen

SQL

In [23]:

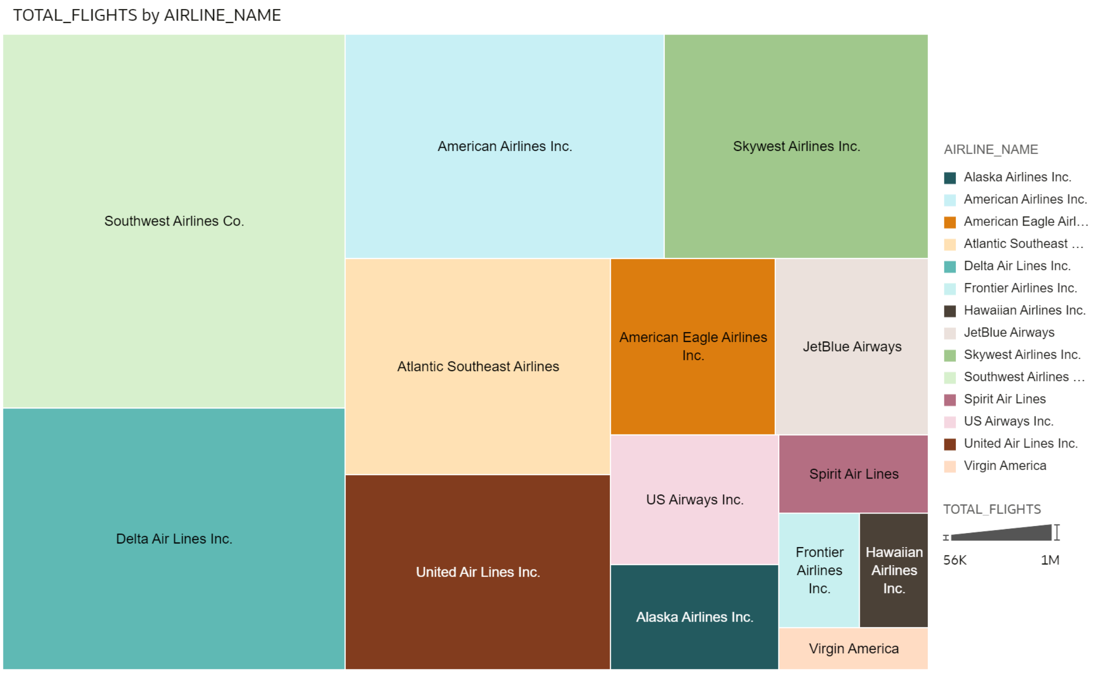
```
query5_1 ="""
SELECT
    a.AIRLINE AS AIRLINE_NAME,
    COUNT(*) AS TOTAL_FLIGHTS
FROM DELAYS f
JOIN AIRLINE a ON f.AIRLINE_IATA = a.IATA_CODE
WHERE f.CANCELLED = 0
GROUP BY a.AIRLINE
ORDER BY TOTAL_FLIGHTS DESC
"""
```

```
df5_1 = run_query(query5_1)
df5_1
```

Out[23]:

	AIRLINE_NAME	TOTAL_FLIGHTS
0	Southwest Airlines Co.	1141618
1	Delta Air Lines Inc.	796522
2	American Airlines Inc.	638500
3	Skywest Airlines Inc.	529826
4	Atlantic Southeast Airlines	511495
5	United Air Lines Inc.	463384
6	American Eagle Airlines Inc.	257898
7	JetBlue Airways	240989
8	US Airways Inc.	194648
9	Alaska Airlines Inc.	157418
10	Spirit Air Lines	105228
11	Frontier Airlines Inc.	82159
12	Hawaiian Airlines Inc.	69871
13	Virgin America	55928

Visualisierung



EXPLAIN PLAN und Indices

In [24]:
```
query5_2 = """
-- 1. Index for the JOIN condition on Delays table
CREATE INDEX idx_delays_airline ON Delays(
    airline_iata,
    cancelled
);
```

```
-- 2. Index for airline lookup
CREATE INDEX idx_airline_iata ON Airline(
    iata_code,
    airline
);
"""


df5_3 = explain_plan_query(query5_1)
print(df5_3.to_string())
```

```
                                                                   PLAN_TABLE_OUTPU
T
0                                                         Plan hash value: 79856489
2
1
2   -------------------------------------------------------------------------------
-
3   | Id  | Operation             | Name     | Rows  | Bytes | Cost (%CPU)| Time
|
4   -------------------------------------------------------------------------------
-
5   |   0 | SELECT STATEMENT     |          |    14 |   434 | 23483   (5)| 00:00:01
|
6   |   1 |  SORT ORDER BY       |          |    14 |   434 | 23483   (5)| 00:00:01
|
7   |   2 |   HASH GROUP BY      |          |    14 |   434 | 23483   (5)| 00:00:01
|
8   |*  3 |    HASH JOIN         |          |  5245K|   155M| 22691   (2)| 00:00:01
|
9   |   4 |     TABLE ACCESS FULL| AIRLINE |    14 |   350 |     3   (0)| 00:00:01
|
10  |*  5 |     TABLE ACCESS FULL| DELAYS  |  5245K|    30M| 22649   (2)| 00:00:01
|
11  -------------------------------------------------------------------------------
-
12
13                               Predicate Information (identified by operation i
d):
14                               --------------------------------------------------
-
15
16                                    3 - access("F"."AIRLINE_IATA"="A"."IATA_COD
E")
17                                                     5 - filter("F"."CANCELLED"=
0)
```

5.2 Fluglinien mit höchste Verspätungen

SQL

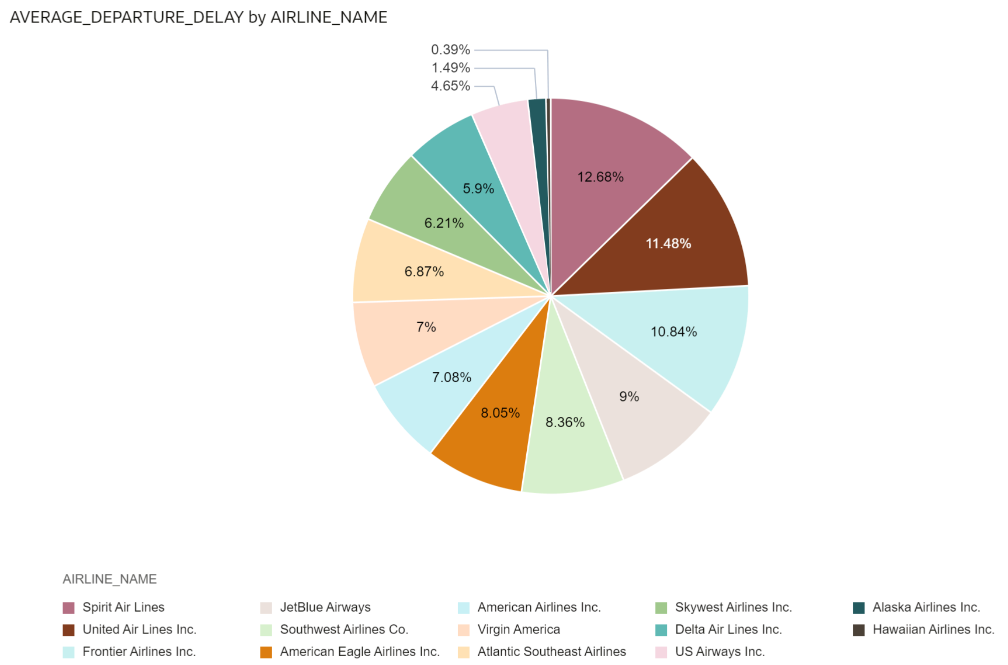
In [25]:

```
query5_4 = """
SELECT
    a.AIRLINE AS AIRLINE_NAME,
    ROUND(AVG(f.DEPARTURE_DELAY), 2) AS AVERAGE_DEPARTURE_DELAY
FROM DELAYS f
JOIN AIRLINE a ON f.AIRLINE_IATA = a.IATA_CODE
WHERE f.CANCELLED = 0
  AND f.DEPARTURE_DELAY IS NOT NULL
GROUP BY a.AIRLINE
ORDER BY AVERAGE_DEPARTURE_DELAY DESC
"""
df5_4 = run_query(query5_4)
df5_4
```

Out[25]:

	AIRLINE_NAME	AVERAGE_DEPARTURE_DELAY
0	Spirit Air Lines	16.65
1	United Air Lines Inc.	15.07
2	Frontier Airlines Inc.	14.23
3	JetBlue Airways	11.81
4	Southwest Airlines Co.	10.98
5	American Eagle Airlines Inc.	10.57
6	American Airlines Inc.	9.29
7	Virgin America	9.19
8	Atlantic Southeast Airlines	9.02
9	Skywest Airlines Inc.	8.15
10	Delta Air Lines Inc.	7.75
11	US Airways Inc.	6.11
12	Alaska Airlines Inc.	1.95
13	Hawaiian Airlines Inc.	0.51

Visualisierung



EXPLAIN PLAN und Indices

In [26]:

```
query5_5 = """
-- 1. Composite index for Delays table
CREATE INDEX idx_delays_airline_delay ON Delays(
    airline_iata,
    cancelled,
    departure_delay
);

-- 2. Index for Airline table
CREATE INDEX idx_airline_iata ON Airline(
    iata_code,
    airline
);
"""


df5_5 = explain_plan_query(query5_4)
print(df5_5.to_string())
```

```
                                                                           PLAN_TABLE_OUTPU
T
0                                                                  Plan hash value: 79856489
2
1
2   -------------------------------------------------------------------------------
-
3   | Id  | Operation            | Name     | Rows  | Bytes | Cost (%CPU)| Time
|
4   -------------------------------------------------------------------------------
-
5   |   0 | SELECT STATEMENT     |          |    14 |   490 | 23476   (5)| 00:00:01
|
6   |   1 |  SORT ORDER BY       |          |    14 |   490 | 23476   (5)| 00:00:01
|
7   |   2 |   HASH GROUP BY      |          |    14 |   490 | 23476   (5)| 00:00:01
|
8   |*  3 |    HASH JOIN         |          | 5163K|   172M| 22697   (2)| 00:00:01
|
9   |   4 |     TABLE ACCESS FULL| AIRLINE  |    14 |   350 |     3   (0)| 00:00:01
|
10  |*  5 |     TABLE ACCESS FULL| DELAYS   | 5163K|    49M| 22656   (2)| 00:00:01
|
11  -------------------------------------------------------------------------------
-
12
13                              Predicate Information (identified by operation i
d):
14                              --------------------------------------------------
-
15
16                                   3 - access("F"."AIRLINE_IATA"="A"."IATA_COD
E")
17                 5 - filter("F"."DEPARTURE_DELAY" IS NOT NULL AND "F"."CANCELLED"=
0)
```

6.Beliebte Flüge

6.1 Beliebteste Flüge

SQL

In [27]:

```
query6_1 = """
SELECT
    f.destination_city AS CITY_NAME,
    f.destination_state AS STATE,
    COUNT(*) AS INCOMING_FLIGHTS
FROM FLIGHT f
JOIN DELAYS d ON f.flight_id = d.flight_id
WHERE d.CANCELLED = 0
GROUP BY
    f.destination_city,
    f.destination_state
ORDER BY INCOMING_FLIGHTS DESC
"""
```

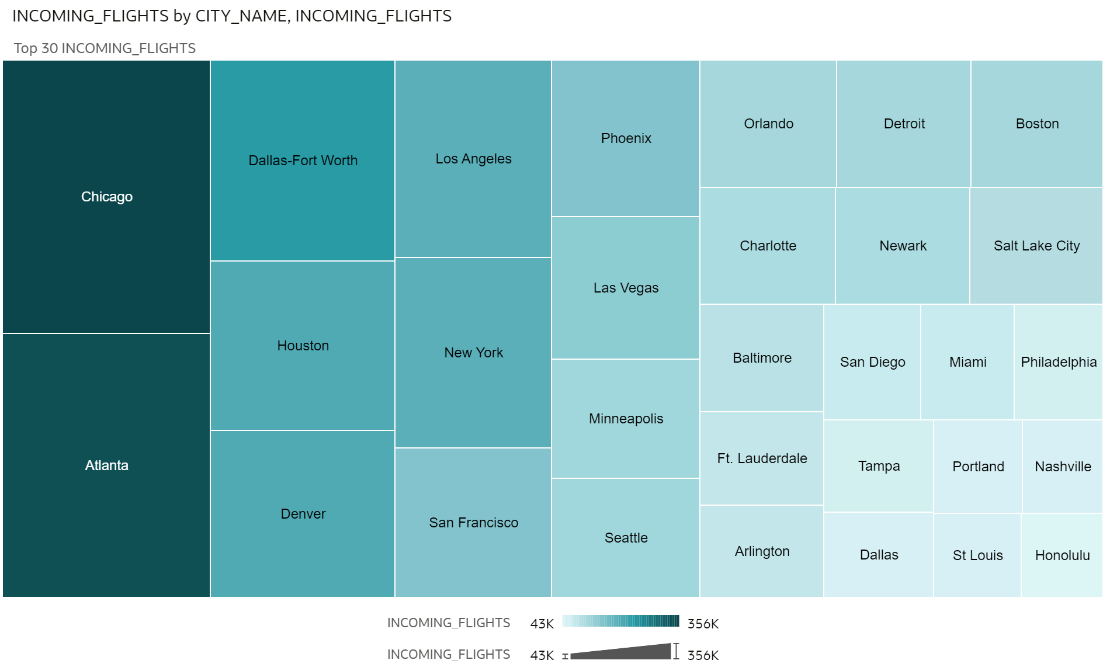
```
df6_1 = run_query(query6_1)
df6_1
```

Out[27]:

	CITY_NAME	STATE	INCOMING_FLIGHTS
0	Chicago	IL	355664
1	Atlanta	GA	344189
2	Dallas-Fort Worth	TX	232833
3	Houston	TX	195516
4	Denver	CO	193700
...	...	...	...
313	St Cloud	MN	77
314	Dillingham	AK	77
315	Gustavus	AK	76
316	King Salmon	AK	63
317	Ithaca	NY	31

318 rows × 3 columns

Visulaisierung



EXPLAIN PLAN und Indices

In [28]:

```
query6_2 = """
-- 1. Index for JOIN operations
CREATE INDEX idx_delays_flight_id ON Delays(
    flight_id,
    cancelled
);

CREATE INDEX idx_flight_id ON Flight(flight_id);

-- 2. Composite index for destination grouping
```

```
CREATE INDEX idx_flight_destination ON Flight(
    destination_city,
    destination_state
);
"""


df6_3 = explain_plan_query(query6_1)
print(df6_3.to_string())
```

```
                                                                           PLAN_TABL
E_OUTPUT
0                                                                 Plan hash value: 6
73129677
1
2   --------------------------------------------------------------------------------
--------
3   | Id  | Operation            | Name   | Rows  | Bytes |TempSpc| Cost (%CPU)| Ti
me     |
4   --------------------------------------------------------------------------------
--------
5   |   0 | SELECT STATEMENT     |        |   318 |  8268 |       | 32498   (4)| 0
0:00:02 |
6   |   1 |  SORT ORDER BY       |        |   318 |  8268 |       | 32498   (4)| 0
0:00:02 |
7   |   2 |   HASH GROUP BY      |        |   318 |  8268 |       | 32498   (4)| 0
0:00:02 |
8   |*  3 |    HASH JOIN         |        |  5177K|   128M|    14M| 31717   (2)| 0
0:00:02 |
9   |   4 |     TABLE ACCESS FULL| FLIGHT |   493K|  8678K|       |  3322   (1)| 0
0:00:01 |
10  |*  5 |     TABLE ACCESS FULL| DELAYS |  5245K|    40M|       | 22649   (2)| 0
0:00:01 |
11  --------------------------------------------------------------------------------
--------
12
13                                      Predicate Information (identified by operat
ion id):
14                                      --------------------------------------------
--------
15
16                                               3 - access("F"."FLIGHT_ID"="D"."FLI
GHT_ID")
17                                                             5 - filter("D"."CANCE
LLED"=0)
```

6.2 Beliebsteste Abflughafen

SQL

In [30]:

```
query6_4 = """
SELECT
    f.ORIGIN_AIRPORT AS AIRPORT_NAME,
    f.ORIGIN_CITY AS CITY,
    f.ORIGIN_STATE AS STATE,
    COUNT(*) AS OUTBOUND_FLIGHTS
FROM FLIGHT f, DELAYS d
WHERE d.FLIGHT_ID = f.FLIGHT_ID
    AND d.CANCELLED = 0
GROUP BY
    f.ORIGIN_AIRPORT,
    f.ORIGIN_CITY,
    f.ORIGIN_STATE
ORDER BY OUTBOUND_FLIGHTS DESC
"""
```

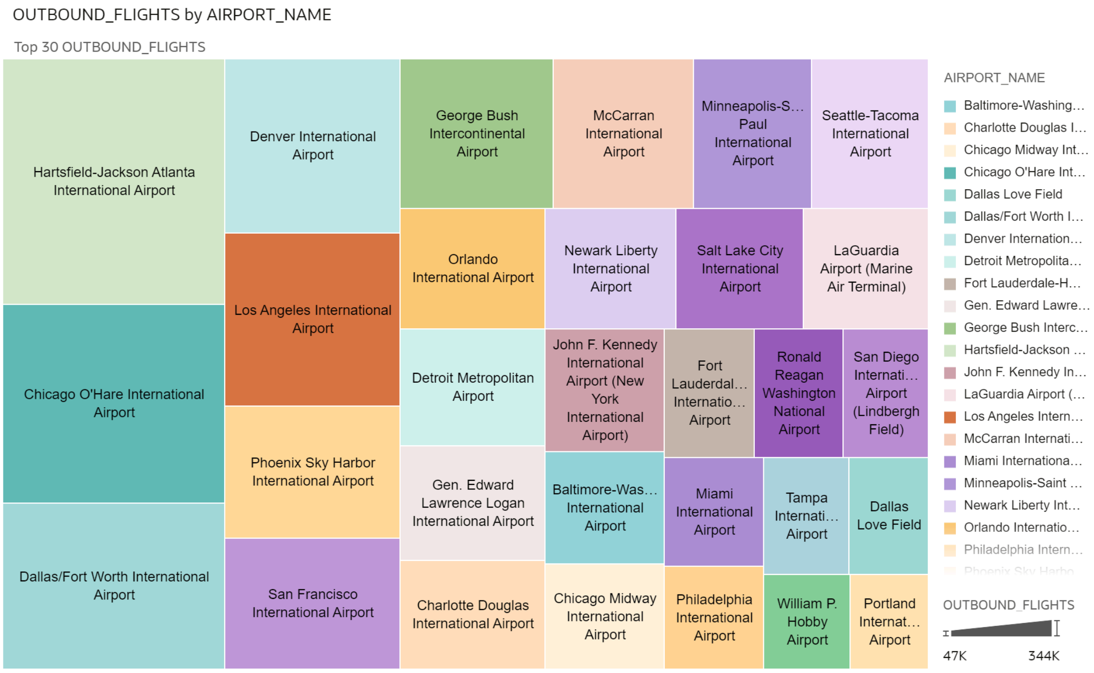
```
df6_4 = run_query(query6_4)
df6_4
```

Out[30]:

	AIRPORT_NAME	CITY	STATE	OUTBOUND_FLIGHTS
0	Hartsfield-Jackson Atlanta International Airport	Atlanta	GA	344279
1	Chicago O'Hare International Airport	Chicago	IL	277336
2	Dallas/Fort Worth International Airport	Dallas-Fort Worth	TX	233297
3	Denver International Airport	Denver	CO	193932
4	Los Angeles International Airport	Los Angeles	CA	192509
...	...	...	...	...
317	St. Cloud Regional Airport	St Cloud	MN	78
318	Dillingham Airport	Dillingham	AK	77
319	Gustavus Airport	Gustavus	AK	76
320	King Salmon Airport	King Salmon	AK	63
321	Ithaca Tompkins Regional Airport	Ithaca	NY	30

322 rows × 4 columns

Visualisierung



OUTBOUND_FLIGHTS by AIRPORT_NAME

Top 30 OUTBOUND_FLIGHTS

EXPLAIN PLAN und Indices

In [31]:

```
query6_5 = """
-- 1. Index for JOIN operations
CREATE INDEX idx_delays_flight_cancel ON Delays(
    flight_id,
    cancelled
);

CREATE INDEX idx_flight_id ON Flight(flight_id);
```

```
-- 2. Composite index for origin grouping
CREATE INDEX idx_flight_origin ON Flight(
    origin_airport,
    origin_city,
    origin_state
);
"""


df6_5 = explain_plan_query(query6_4)
print(df6_5.to_string())
```

```
                                                                          PLAN_TABL
E_OUTPUT
0                                                                Plan hash value: 6
73129677
1
2   -------------------------------------------------------------------------------
--------
3   | Id  | Operation            | Name   | Rows  | Bytes |TempSpc| Cost (%CPU)| Ti
me     |
4   -------------------------------------------------------------------------------
--------
5   |   0 | SELECT STATEMENT     |        | 12296 |   768K|       | 33386   (4)| 0
0:00:02 |
6   |   1 |  SORT ORDER BY       |        | 12296 |   768K|       | 33386   (4)| 0
0:00:02 |
7   |   2 |   HASH GROUP BY      |        | 12296 |   768K|       | 33386   (4)| 0
0:00:02 |
8   |*  3 |    HASH JOIN         |        |  5177K|   316M|    32M| 32605   (2)| 0
0:00:02 |
9   |   4 |     TABLE ACCESS FULL| FLIGHT |   493K|    26M|       |  3322   (1)| 0
0:00:01 |
10  |*  5 |     TABLE ACCESS FULL| DELAYS |  5245K|    40M|       | 22649   (2)| 0
0:00:01 |
11  -------------------------------------------------------------------------------
--------
12
13                                      Predicate Information (identified by operat
ion id):
14                                      -------------------------------------------
--------
15
16                                              3 - access("D"."FLIGHT_ID"="F"."FLI
GHT_ID")
17                                                            5 - filter("D"."CANCE
LLED"=0)
```