

Hurtownie danych – Projekt

Proces tworzenia hurtowni danych powinien być poprzedzony zrozumieniem „potrzeb biznesu” oraz rzeczywistości (dziedziny problemowej) reprezentowanej przez dostępne zasoby danych. Realizacja poniższego zadania ma uzmysłwić występujące problemy w określonym (wybranym) wycinku rzeczywistości, a następnie umożliwić zidentyfikowanie (określenie) potrzeb, celu i możliwości analiz biznesowych, by wspierać procesy decyzyjne (podejmowanie właściwych decyzji biznesowych).

Projekt końcowy powinien zawierać przynajmniej jedną kostkę Analysis Services, dotyczącą danych wybranych i przetworzonych przez studenta przy użyciu Integration Services. Utworzona kostka powinna:

- zawierać przynajmniej 5 wymiarów, w tym co najmniej dwa o strukturze hierarchicznej (np. czas, miejsce, itp)
- posiadać co najmniej 3 miary, w tym min. jedną nieaddytywną
- odpowiadająca jej tabela faktów powinna posiadać co najmniej 10000 rekordów.

Projekt – etap I

Propozycja tematu

1. Proszę przygotować zakres realizacji projektu zgodnie z poniższą specyfikacją oraz przedyskutować propozycję projektu z osobą prowadzącą zajęcia. Poczynione uzgodnienia zarejestrować w formie wniosków.

Zakres opracowania projektu HD

1.1. Tytuł projektu

Analiza Wypadków Lotniczych 1982-2023 - Narodowa Rada Bezpieczeństwa Transportu.

Aviation Accident Database & Synopses, up to 2023 - National Transportation Safety Board.

1.2. Charakterystyka dziedziny problemowej

Dziedzina problemowa obejmuje wypadki lotnicze z lat 1982-2023 i zawiera informacje takie jak data, kraj, lokalizacja, szczegóły samolotu, linii lotniczej, celu lotu lub fazy, w których wypadki miały miejsce, a także informacje o ofiarach.

1.3. Krótki opis obszaru analizy

Analiza będzie skupiać się na zbadaniu zależności pomiędzy częstotliwością wypadków, a rodzajem linii lotniczej, miejscem wypadku, warunkami pogodowymi, czy producentem lub wyposażeniem samolotów.

1.4. Problemy i potrzeby

Projekt ma na celu zbadanie zależności między częstotliwością wypadków a różnymi czynnikami, takimi jak warunki pogodowe, producent samolotów, faza lotu itp. Urząd lotnictwa cywilnego potrzebuje tych danych do statystyk i kategoryzacji wypadków czy podziału tychże statystyk ze względu na państwa lub same stany w przypadku US.

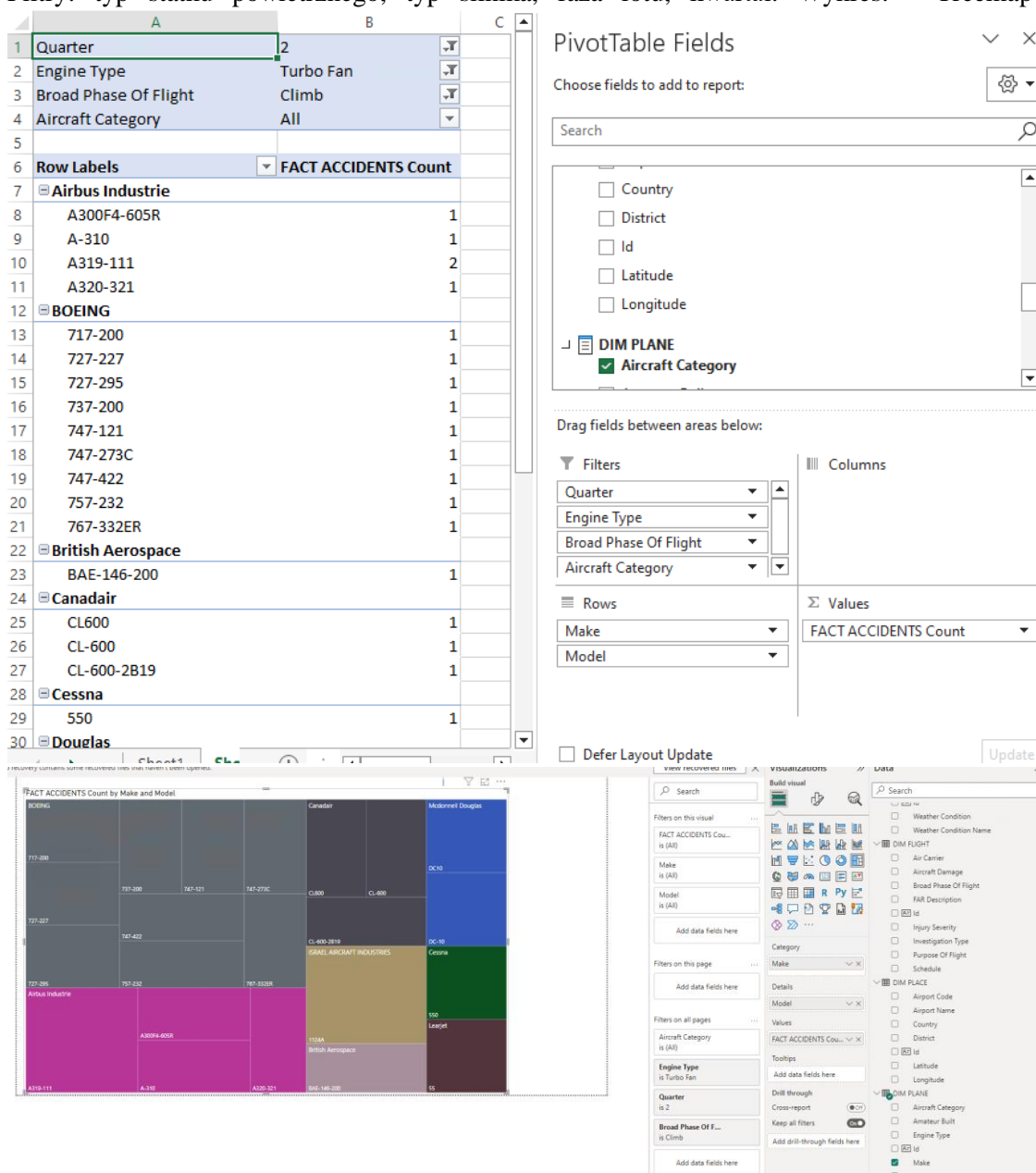
1.5. Cel przedsięwzięcia

1.5.1. Oczekiwania

Określenie związku między warunkami lotu a częstotliwością wypadków oraz liczbą i zakresem uszkodzeń samolotu lub uszczerbkiem na zdrowiu ludzi podczas wypadku.

1.5.2. Zakres analizy – badane aspekty (min. 10 wielowymiarowych zestawień, które zostaną utworzone po wdrożeniu kostki) (Zredukowano do 4)

1. Analiza badająca liczbę wypadków w zależności od typu „samolotu”, producenta, modelu, liczby silników i typu silnika, a także o fazę lotu. Taka analiza pozwoliłaby na identyfikację szczególnie niebezpiecznych modeli lub producentów „samolotów”, które są bardziej podatne na wypadki. Wskaźniki: liczba wypadków. Wymiary: według DIM_PLANE (marka, model). Filtry: typ statku powietrznego, typ silnika, faza lotu, kwartał. Wykres: - Treemap



2. Analiza wypadków lotniczych w zależności od miejsca zdarzenia. Taka analiza pozwoliłaby na identyfikację obszarów, w których dochodzi najczęściej do wypadków lotniczych i pozwoliłaby na podjęcie odpowiednich działań w celu poprawy bezpieczeństwa lotów w tych obszarach. Dla Stanów Zjednoczonych jest znaczna ilość danych, która pozwoli dokładną analizę w tym obszarze. Wskaźniki: suma poszkodowanych: Wymiary: według hierarchii wymiaru DIM_PLACE. Filtry: faza lotu, przewoźnik, miesiąc. Wykres - Mapa geograficzna.

	A	B	C	D
1	Air Carrier	All		
2	Broad Phase Of Flight	Approach		
3	Month In Words	December		
4				
5	Row Labels	Total Of Injured		
6	ABERDEEN	2		
7	ADDISON	0		
8	ADELANTO	1		
9	AGANA	0		
10	ALAKANUK	0		
11	ALBERMARLE	2		
12	ALBUQUERQUE	2		
13	Alcoa	2		
14	Allen	0		
15	Alpine	1		
16	AMADO	1		
17	Anaheim Hills	2		
18	ANCHORAGE	2		
19	ANTIGO	2		
20	APACHE JUNCTION	3		
21	APEX	2		
22	ARLINGTON	2		
23	Armonk	1		
24	ASPEN	1		
25	Atlanta	4		
26	AUBURN	0		
27	AURORA	3		
28	AUSTELL	1		
29	Austin	2		
30	AXTON	1		

PivotTable Fields

Choose fields to add to report:

Search

FACT ACCIDENTS

- ☐ FACT ACCIDENTS Count
- ☐ Mortality
- ☐ Total Fatal Injuries
- ☐ Total Minor Injuries
- ☒ Total Of Injured
- ☐ Total Serious Injuries
- ☐ Total Uninjured

Drag fields between areas below:

Filters

- Air Carrier
- Broad Phase Of Flight
- Month In Words

Rows

- District

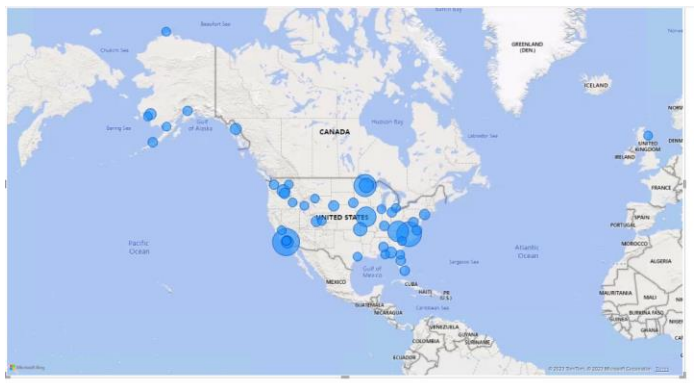
Columns

Σ Values

- Total Of Injured

☐ Defer Layout Update

Update



Search

Filters on this visual

District is (All)

Total Of Injured is (All)

Add data fields here

Filters on this page

Add data fields here

Filters on all pages

Air Carrier is not

Month is 10, 11, or 12

Broad Phase Of Flight is Approach

Add data fields here

Build visual

Legend

Add data fields here

Latitude

Add data fields here

Longitude

Add data fields here

Bubble size

Total Of Injured

Tooltips

Add data fields here

Drill through

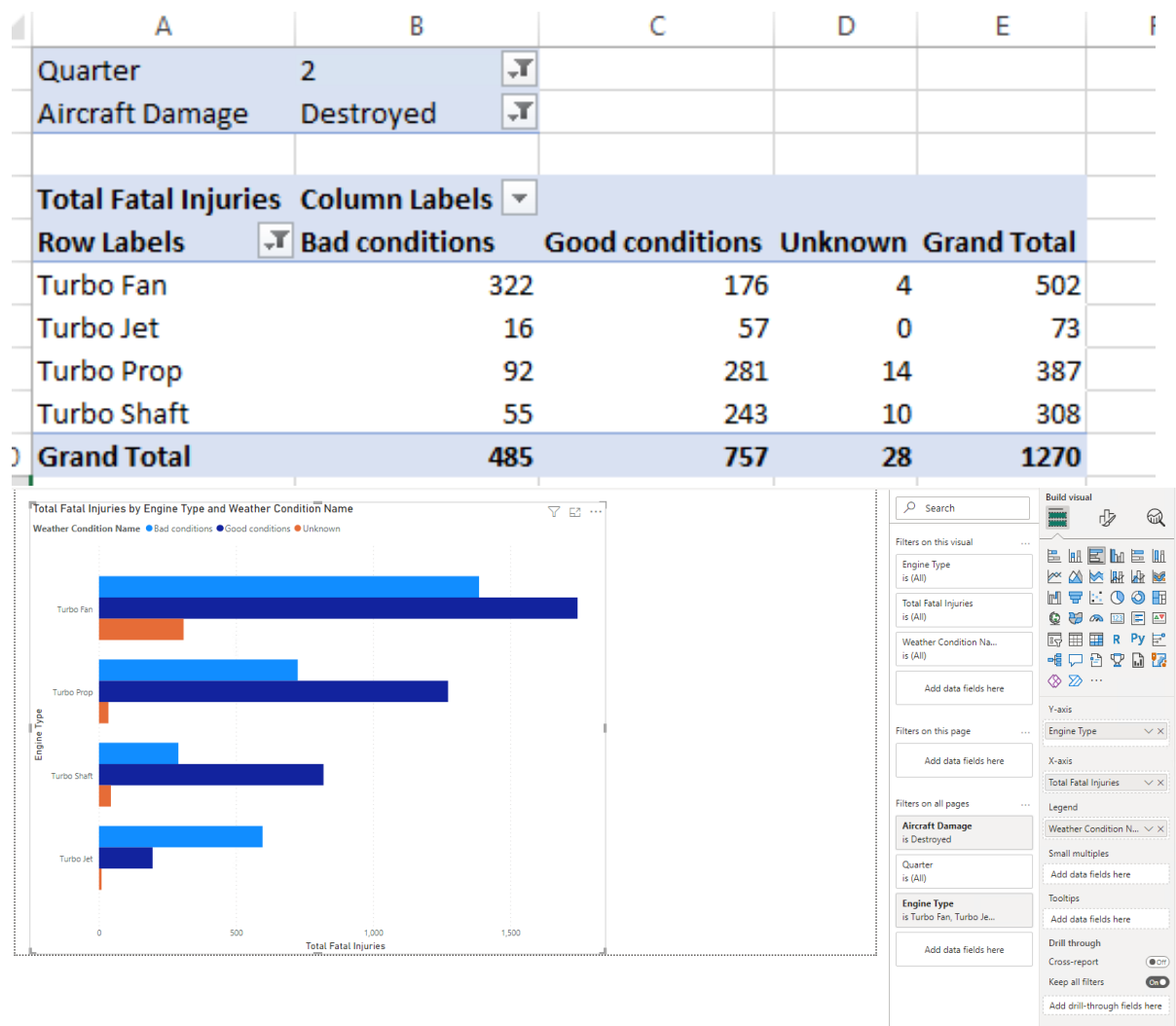
Cross-report

Search

FACT ACCIDENTS

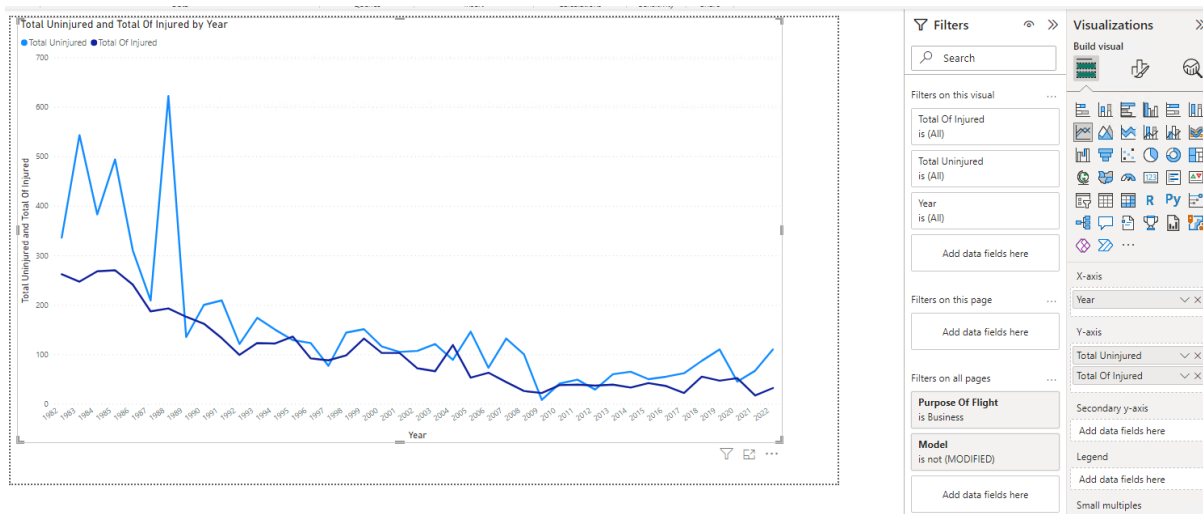
☐ Total Uninjured
 ☒ DIM CONDITIONS
 ☐ Id
 ☐ Weather Cond...
 ☐ Weather Cond...
 ☐ DIM FLIGHT
 ☐ Air Carrier
 ☐ Aircraft Damage
 ☐ Broad Phase Of...
 ☐ FAR Description
 ☐ Id
 ☐ Injury Severity
 ☐ Investigation T...
 ☐ Purpose Of Flig...
 ☐ Schedule
 ☒ DIM PLACE
 ☐ Airport Code
 ☐ Airport Name
 ☐ Country
 ☐ District
 ☐ Id
 ☐ Latitude
 ☐ Longitude
 ☒ DIM PLANE
 ☐ Aircraft Category
 ☐ Amateur Built
 ☐ Engine Type

3. Analiza zależności wypadków lotniczych od warunków pogodowych oraz od rodzaju. Taka analiza pozwoliłaby na zidentyfikowanie sytuacji, w których warunki pogodowe wpływają na zwiększenie ryzyka wypadków lotniczych dla konkretnych samolotów posiadających dany typ silników i pozwoliłaby na podejmowanie odpowiednich działań zapobiegawczych. Wskaźniki: suma wszystkich zmarłych. Wymiary: stan pogody, rodzaje silników. Filtry: rodzaj silników, zniszczenia samolotu, kwartał. Wykres: Clustered bar chart



4. Analiza badającą liczbę osób poszkodowanych oraz liczbę osób nieposzkodowanych w zależności od przewoźnika lotniczego, celu lotu, roku. Taka analiza pozwoliłaby na zidentyfikowanie w czasie, przewoźników lub typów lotów w danym roku, które są bardziej narażone na wypadki lotnicze i pozwoliłaby na podejmowanie odpowiednich działań, takich jak zwiększenie nadzoru nad tymi przewoźnikami. Wymiary: hierarchia czasu, Wskaźniki: suma poszkodowanych, suma nieposzkodowanych. Filtry: model samolotu, cel lotu. Wykres: Line chart

Purpose Of Flight	Business		
Model	(Multiple Items)		
Row Labels	Total Uninjured	Total Of Injured	
1982	336	262	
1983	543	247	
1984	383	268	
1985	494	270	
1986	310	241	
1987	209	187	
1988	622	193	
1989	135	176	
1990	200	162	
1991	209	133	
1992	121	99	
1993	174	123	
1994	150	122	
1995	129	136	
1996	123	92	
1997	77	88	
1998	144	98	
1999	151	132	
2000	116	103	
2001	105	103	
2002	107	72	
2003	121	66	
2004	89	119	
2005	146	53	
2006	73	63	



1.6. Źródła danych (lokalizacja, format, dostępność)

Wstępna analiza źródeł danych

Dane są dostępne na stronie kaggle:

<https://www.kaggle.com/datasets/khsamaha/aviation-accident-database-synopses>

Do pobrania są dostępne dwa pliki AviationData.csv oraz USState_Codes.csv.

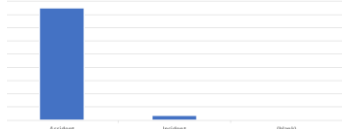

Dane pochodzą z NTSB – National Transportation Safety Board:

<https://www.nts.gov/>

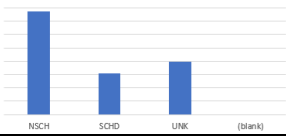


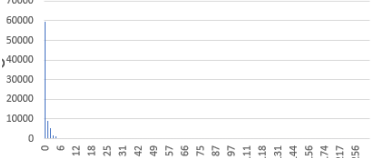


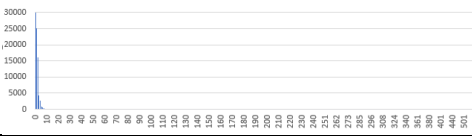
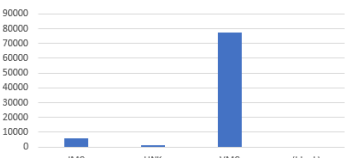
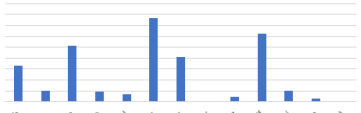
Lp.	Plik	Typ	Liczba rekordów	Rozmiar [MB]	Opis
1.	AviationData.csv	csv	88889	21.4 MB	Tabela zawierająca informacje o wypadkach, samolotach, czasie, miejscu i osobach rannych/zabitych
2.	USState_Codes.csv	csv	62	0,0009 MB	Tabela zawierająca poszczególne stany Stanów Zjednoczonych/regiony i ich kody

2. Profilowanie danych

2.1. Analiza danych

Plik: Aviation_Data.csv				
Lp.	Atrybut	Typ danych	Zakres wartości	Uwagi – ocena jakości danych
1.	Event.Id	nvarchar(50)		Indeks zdarzenia. NULL = 0.0%
2.	Investigation.Type	nvarchar(20)	{ Accident, Incident }	Typ zdarzenia. NULL = 0.0 % 
3.	Accident.Number	nvarchar(20)		Numer zdarzenia. NULL = 0.0 %
4.	Event.Date	DATE	24/10/1948 do 29/12/2022	Data zdarzenia. NULL = 0.0 %
5.	Location	nvarchar(100)		Lokalizacja, w której wypadek miał miejsce. NULL=0.06% 
6.	Country	nvarchar(50)		Kraj, w którym wypadek miał miejsce. NULL=0.25%
7.	Latitude	nvarchar(20)		Wartość podawana w postaci dwóch notacji DMS, DD (wymaga preprocessingu). NULL = 61.33%
8.	Longitude	nvarchar(20)		Wartość podawana w postaci dwóch notacji DMS, DD (wymaga preprocessingu). NULL = 61.33%
9.	Airport.Code	nvarchar(10)		Kod lotniska, w obrębie którego miał miejsce wypadek. NULL=43.17%

10.	Airport.Name	nvarchar(100)		Nazwa lotniska, w obrębie którego miał miejsce wypadek. NULL=40.49%
11.	Injury.Severity	nvarchar(20)		Informacja na temat liczby zmarłych pasażerów. NULL=1.12%
12.	Aircraft.Damage	nvarchar(15)		Informacja na temat powagi obrażeń statku lotniczego. NULL= 3.59%
13.	Aircraft.Category	nvarchar(30)		Rodzaj statku. NULL=63.68%
14.	Registration.Number	nvarchar(15)		Numer rejestracyjny statku. NULL=1.48%
15.	Make	nvarchar(50)		Producent samolotu. NULL=0.7%
16.	Model	nvarchar(50)		Model samolotu, NULL=0.1%
17.	Amateur.Built	nvarchar(3)	{ Yes, No, NULL }	Informacja na temat budowy samolotu. NULL=0.11% Yes ~ 10%, No ~ 90%
18.	Number.ofEngines	int	{ NULL, 0, 1, 2, 3, 4, 6, 8 }	Liczba silników. NULL= 6.84%
19.	Engine.Type	nvarchar(30)		Rodzaj silników. NULL = 7.96%

20.	FAR.Description	nvarchar(200)		Opis genezy lotu. NULL = 63.97%
21.	Schedule	nvarchar(10)	{NULL, UNK, SCHD, NSCH}	Kod harmonogramu lotu. NULL = 85.85% 
22.	Purpose.of.flight	nvarchar(30)		Cel lotu. NULL = 6.97% 
23.	Air.carrier	nvarchar(100)		Linia lotnicza. NULL = 81.27% 
24.	Total.Fatal.Injuries	int	0 - 349	Liczba zmarłych. NULL = 12.83% 
25.	Total.Serious.Injuries	int	0 - 161	Liczba poważnie rannych. NULL = 14.07% 
26.	Total.Minor.Injuries	int	0 - 380	Liczba lekko rannych. NULL = 13.42% 
27.	Total.Uninjured	int	0 - 699	Suma niezranionych, NULL = 6.65% 
28.	Weather.Condition	nvarchar(5)	{NULL, VMC, UNK, IMC}	Kod warunków pogodowych, NULL = 5.05% 
29.	Broad.phase.of.flight	nvarchar(20)		Faza lotu, w której miał miejsce wypadek, NULL = 30.56% 

30.	Report.Status	nvarchar(MAX)		Bardzo różne opisy powodów wypadków. NULL = 7.18%
31.	Publication.Date	nvarchar(10)	1982-01-01 do 2022-10-31	Data publikacji wypadku. Format daty zapisany w postaci ciągu znaków „YYYY-MM-DD” – wymaga pre processingu. NULL = 15.49%

Plik: USState_Codes.csv				
Lp.	Atrybut	Typ danych	Zakres wartości	Uwagi – ocena jakości danych
1.	US_State	nvarchar(20)		Kod stanu dotyczący Stanów Zjednoczonych.
2.	Abbreviation	nvarchar(2)		Rozwinięcie stanu.

2.2. Ocena przydatności danych w pliku do tworzenia hurtowni danych

Lp.	Plik	Ocena jakości danych
1.	Aviation_Data.csv	Niezbędne do utworzenia hurtowni, obejmuje lata 1948-2022. Dość duża ilość pól jest niekompletna. Głównie obejmuje region Stanów Zjednoczonych. Wymagane ujednolicenie danych w, niektórych kolumnach.
2.	USState_Codes.csv	Opcjonalne – zawiera jedynie pełne nazwy regionów lub stanów

2.3. Definicja typów encji/klas (wraz z własnościami) oraz związków pomiędzy nimi

Encje:

1. Time
 - Event.Date
2. Place
 - Country
 - Location
 - Latitude
 - Longitude
 - Airport.Code
 - Airport.Name
3. Accident
 - Accident.Number
 - Investigation.Type
 - Injury.Severity
 - Aircraft.damage
 - FAR.Description
 - Schedule

- Purpose.of.flight
 - Air.carrier
 - Broad.phase.of.flight
 - Total.Fatal.Injuries
 - Total.Serious.Injuries
 - Total.Minor.Injuries
 - Total.Uninjured
4. Plane
- Make
 - Model
 - Amateur.Built
 - Number.of.Engines
 - Engine.Type
 - Aircraft.Category
5. Conditions
- Weather.Condition

Związki:

Wyznacza(ACCIDENT(0,N) : TIME(1,1))

Encja Accident musi dotyczyć tylko jednej encji Time.

Encja Time może dotyczyć wielu encji Accident.

Określa(ACCIDENT(0,N) : PLACE(1,1))

Encja Accident musi dotyczyć tylko jednej encji Place.

Encja Place może dotyczyć wielu encji Accident.

Wskazuje(ACCIDENT(1,N) : PLANE(1,1))

Encja Accident musi dotyczyć tylko jednej encji Plane.

Encja Plane może dotyczyć wielu encji Accident.

Opisuje(ACCIDENT(0,N) : CONDITIONS(1,1))

Encja Accident musi dotyczyć tylko jednej encji Conditions.

Encja Conditions może dotyczyć wielu encji Accident.

2.4. Propozycja wymiarów, hierarchii, miar (w tym nieaddytywnych)

DIM_TIME:

Id	int	PK, NOT NULL
Year	int	NOT NULL
Quarter	int	NOT NULL

Month	int	NOT NULL
Month In Words	nvarchar(10)	NOT NULL
Day	int	NOT NULL
Day In Words	nvarchar(10)	NOT NULL

DIM_PLACE:

Id	int	PK, NOT NULL
Country	nvarchar(50)	NULL
District	nvarchar(100)	NULL
Region_Name	nvarchar(100)	NULL
Latitude	decimal(18,6)	NULL
Longitude	decimal(18,6)	NULL
Airport_Code	nvarchar(10)	NULL
Airport_Name	nvarchar(100)	NULL

DIM_FLIGHT:

Id	int	PK, NOT NULL
Investigation_Type	nvarchar(20)	NULL
Injury_Severity	nvarchar(20)	NULL
Aircraft_damage	nvarchar(15)	NULL
FAR_Description	nvarchar(200)	NULL
Schedule	nvarchar(10)	NULL
Purpose_of_flight	nvarchar(30)	NULL
Air_carrier	nvarchar(100)	NULL
Broad_phase_of_flight	nvarchar(20)	NULL

DIM_PLANE:

Id	int	PK, NOT NULL
Make	nvarchar(50)	NULL
Model	nvarchar(50)	NULL
Amateur_Built	nvarchar(3)	NULL
Number_of_Engines	int	NULL
Engine_Type	nvarchar(30)	NULL
Aircraft_Category	nvarchar(30)	NULL

DIM_CONDITIONS:

Id	int	PK, NOT NULL
Weather_Condition	nvarchar(5)	NOT NULL
Weather_Condition_Name	nvarchar(30)	NOT NULL

FACT_ACCIDENTS:

Id	int	PK, NOT NULL
Flight_Id	int	NOT NULL
Time_Id	int	NOT NULL
Place_Id	int	NOT NULL
Plane_Id	int	NOT NULL

Weather_Conditions_Id	int	NOT NULL
Total_Fatal_Injuries	int	NULL
Total_Serious.Injuries	int	NULL
Total_Minor.Injuries	int	NULL
Total_Uninjured	int	NULL
TotalOfInjured	int	NULL
Mortality	decimal(18,6)	NULL

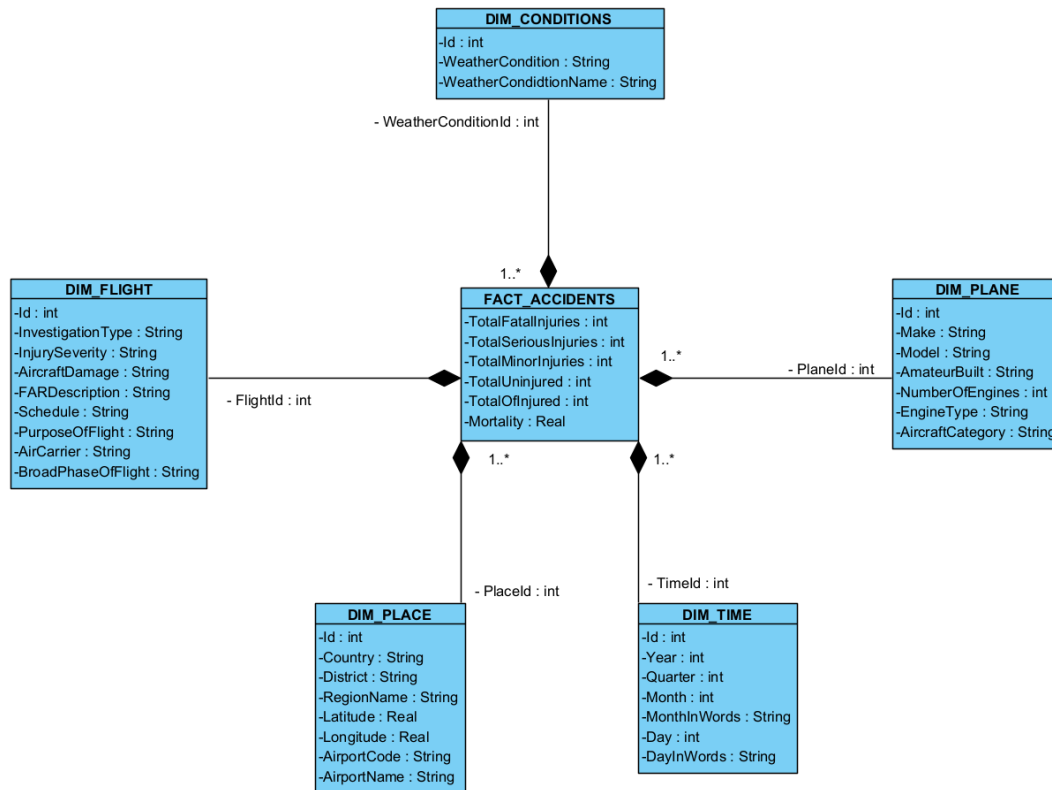
Hierarchie:

DIM_TIME: Year, Quarter, Month In Words, Day, Day In Words.

DIM_PLACE: Country, Region_Name, District

DIM_PLANE: Aircraft_Category, Engine_Type, Number_Of_Engines, Make, Model

2.5. Diagram klas – model danych utworzony na podstawie danych zgromadzonych w plikach



3. Utworzyć bazę danych zgodnie z zaproponowanym konceptualnym modelem danych (p. 2.3. i 2.4.)

```

CREATE TABLE DIM_TIME
(
    Id INT PRIMARY KEY,
    "Year" INT NOT NULL,
    "Quarter" INT NOT NULL,
    "Month" INT NOT NULL,
    "Month In Words" NVARCHAR(10) NOT NULL,
    "Day" INT NOT NULL,
    "Day In Words" NVARCHAR(10) NOT NULL
);
  
```

```

CREATE TABLE DIM_PLACE
(
    Id INT IDENTITY(1,1) PRIMARY KEY,
    Country NVARCHAR(50) NULL,
    District NVARCHAR(100) NULL,
    Region_Name NVARCHAR(100) NULL,
    Latitude DECIMAL(18,6) NULL,
    Longitude DECIMAL(18,6) NULL,
    Airport_Code NVARCHAR(10) NULL,
    Airport_Name NVARCHAR(100) NULL
);
  
```

```

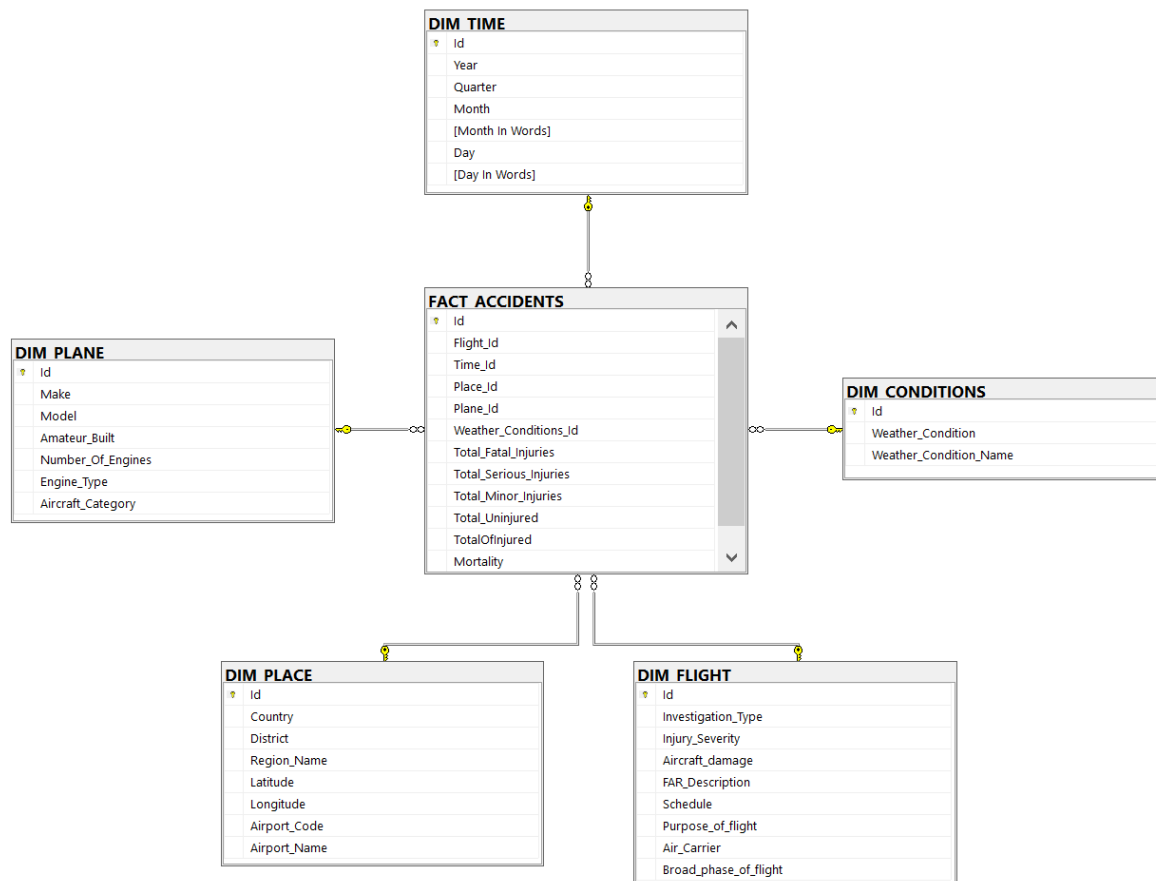
CREATE TABLE DIM_CONDITIONS
(
  
```

```
        Id INT IDENTITY(1,1) PRIMARY KEY,
        Weather_Condition NVARCHAR(5) NULL,
        Weather_Condition_Name NVARCHAR(30) NOT NULL
    );
CREATE TABLE DIM_PLANE
(
    Id INT IDENTITY(1,1) PRIMARY KEY,
    Make NVARCHAR(50) NULL,
    Model NVARCHAR(50) NULL,
    Amateur_Built NVARCHAR(3) NULL,
    Number_Of_Engines INT NULL,
    Engine_Type NVARCHAR(30),
    Aircraft_Category NVARCHAR(30)
);

CREATE TABLE DIM_FLIGHT
(
    Id INT IDENTITY(1,1) PRIMARY KEY,
    Investigation_Type NVARCHAR(20) NULL,
    Injury_Severity NVARCHAR(20) NULL,
    Aircraft_damage NVARCHAR(15) NULL,
    FAR_Description NVARCHAR(200) NULL,
    Schedule NVARCHAR(10) NULL,
    Purpose_of_flight NVARCHAR(30) NULL,
    Air_Carrier NVARCHAR(100) NULL,
    Broad_phase_of_flight NVARCHAR(20) NULL
);

CREATE TABLE FACT_ACCIDENTS
(
    Id INT IDENTITY(1,1) PRIMARY KEY,
    Flight_Id INT NOT NULL,
    Time_Id INT NOT NULL,
    Place_Id INT NOT NULL,
    Plane_Id INT NOT NULL,
    Weather_Conditions_Id INT NOT NULL,
    Total_Fatal_Injuries INT NULL,
    Total_Serious_Injuries INT NULL,
    Total_Minor_Injuries INT NULL,
    Total_Uninjured INT NULL,
    TotalOfInjured INT NULL,
    Mortality DECIMAL(18,6) NULL
);

ALTER TABLE FACT_ACCIDENTS
    ADD CONSTRAINT CONDITIONS_FOREIGN_KEY FOREIGN KEY(Weather_Conditions_Id)
REFERENCES DIM_CONDITIONS(Id),
    CONSTRAINT ACCIDENT_FOREIGN_KEY FOREIGN KEY(Flight_Id) REFERENCES
DIM_FLIGHT(Id),
    CONSTRAINT PLACE_FOREIGN_KEY FOREIGN KEY(Place_Id) REFERENCES
DIM_PLACE(Id),
    CONSTRAINT PLANE_FOREIGN_KEY FOREIGN KEY(Plane_Id) REFERENCES
DIM_PLANE(Id),
    CONSTRAINT EVENT_DATE_FOREIGN_KEY FOREIGN KEY(Time_Id) REFERENCES
DIM_TIME(Id);
```



Wnioski:

Pierwszy etap projektu okazał się być zaskakująco wymagający, być może dlatego, iż jest to najważniejszy etap projektowania hurtowni danych, który już na samym początku określa cel, problem, potrzebę dogłębnej analizy danych.

Wybrany przez nas zbiór danych, zaczerpnięty z rządowej strony, przedstawiał się jako godny zaufania i będący na najwyższym poziomie zestaw danych. Jednakże po przyjrzeniu się wartościom atrybutów, okazało się, że mamy do czynienia z dużą niekompletnością danych.

Po zdefiniowaniu typów encji oraz związków, doszliśmy do wniosku, iż rozsądnym krokiem będzie rozdzielenie encji Accident na wymiar DIM_FLIGHT oraz na tabelę faktów FACT_ACCIDENTS, ponieważ pozwoli to nam na dodatkowe filtrowanie po uzyskanych wynikach.

Analizując poszczególne atrybuty natknęliśmy się na potrzebę przetworzenia niektórych wartości atrybutów. Zauważyliśmy niespójności co do formatu zapisanych wartości dla szerokości i długości geograficznych, niektóre zapisane w formacie DMS – Degrees Minutes Seconds, inne w DD – Decimal Degrees. Dodatkowo zauważyliśmy, inny format zapisu daty w atrybucie dotyczącym publikacji wypadku. W celu późniejszego wytworzenia hierarchii ze względu na lokalizację wypadków, podzieliliśmy atrybut Location na nazwę okręgu oraz kod jego stanu, ponieważ zauważyliśmy, że kod danego stanu określa większy obszar niż nazwa okręgu.

Projekt – etap II

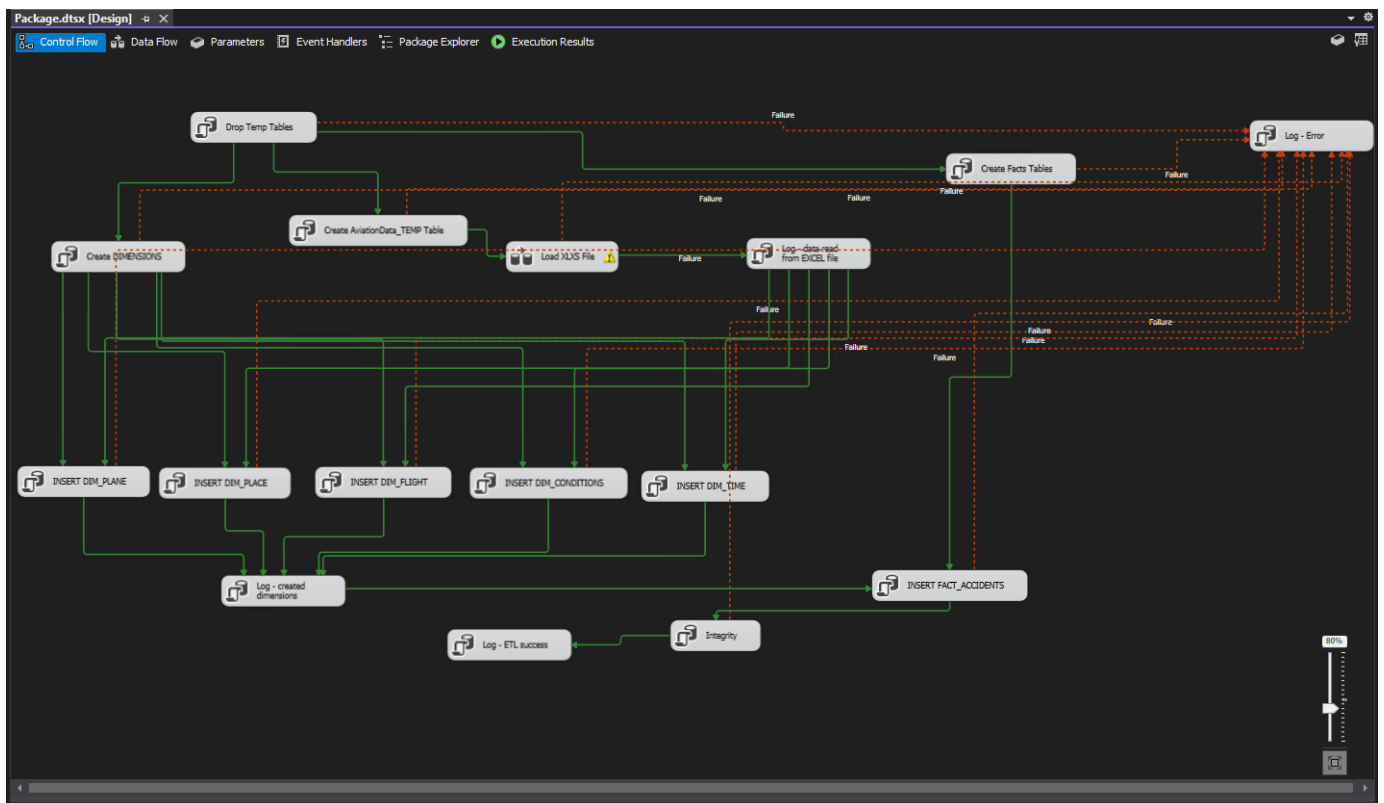
Proces ETL

1. Utworzone tabele w poprzednim punkcie wypełnić danymi zgodnie z ustalonymi założeniami projektowymi wykorzystując zapytania SQL lub inne narzędzia dostępne w Integration Services.

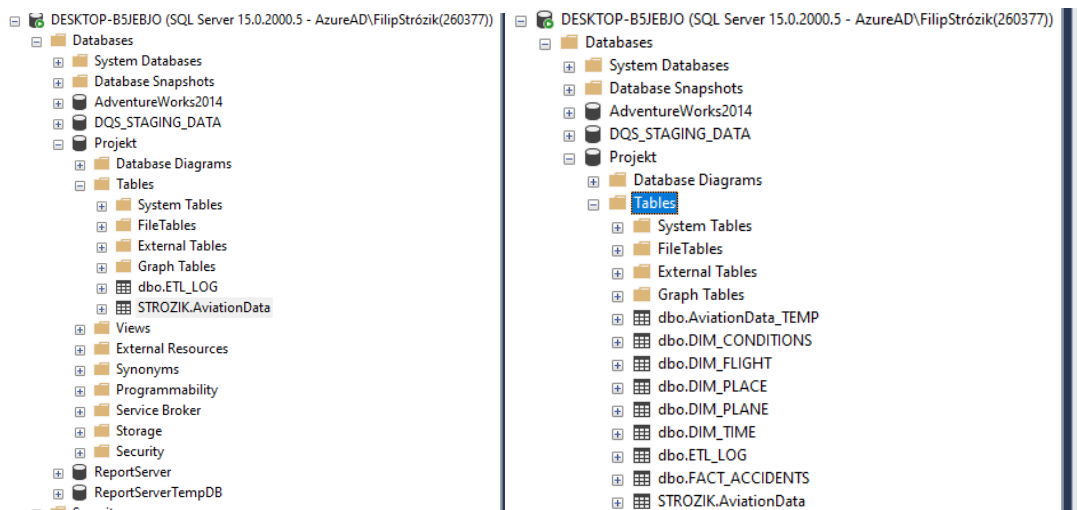
Przy ocenie będą brane następujące elementy pakietu(ów):

- właściwa struktura procesu ETL (odpowiednie rozbiecie procesu ETL na zadania/pakiety, dobrze dobrane nazwy poszczególnych zadań, wprowadzona automatyzacja, obsługa błędów, itp.)
- stabilność i prawidłowe, bezbłędne wykonanie
- złożoność przeprowadzonych operacji. Przykładowo, jeżeli dane źródłowe już są w pełni zdenormalizowane proszę nie spodziewać się maksymalnej liczby punktów za ten element
- dokumentacja powinna zawierać krótki opis dotyczący każdego zadania, które pozwoli zorientować się, jaki jest jego cel (np. zadanie Z kopiuje dane z tabeli X i Y do tabeli T dokonując denormalizacji) oraz mapę logiczną procesu ETL.

Control Flow:



Widok bazy danych przed i po uruchomieniu procesu ETL:



Uruchomienie Pakietu:

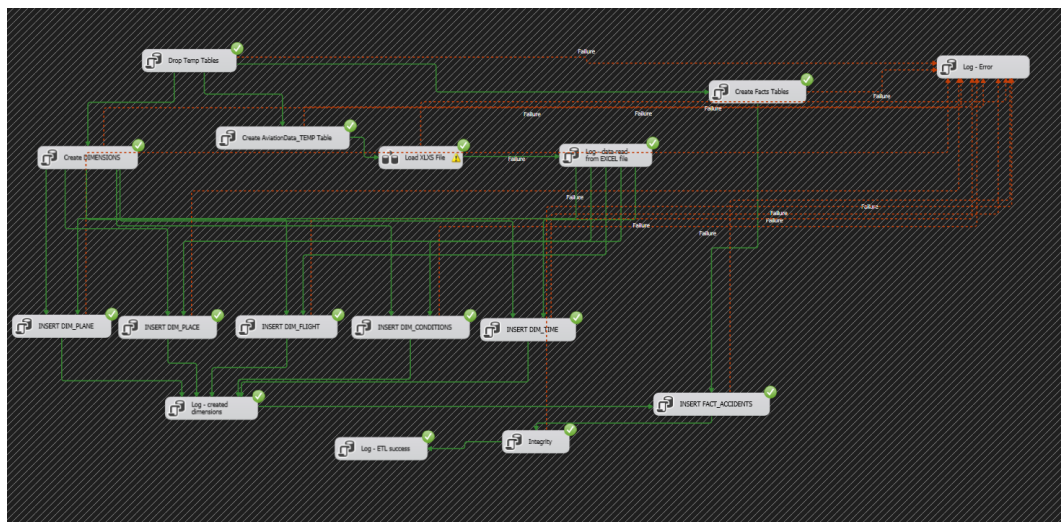


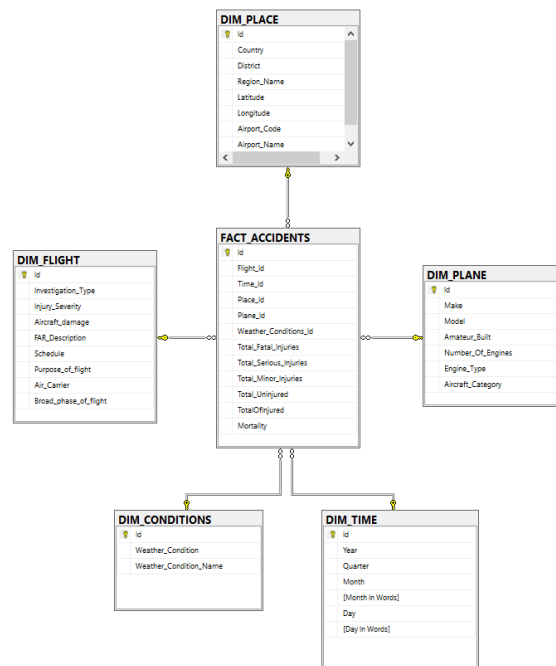
Tabela Faktów:

id	Flight_id	Time_id	Place_id	Plane_id	Weather_Conditions_id	Total_Fatal_Injuries	Total_Serious_Injuries	Total_Minor_Injuries	Total_Uninjured	TotalInjured	Mortality
79700	18136	19860815	58937	22795	1	2	0	0	0	2	1.000000
79701	18136	19860816	21435	5945	1	2	0	0	0	2	1.000000
79702	18136	19860819	46331	20877	1	2	0	0	0	2	1.000000
79703	18136	19860820	55944	151	1	2	0	0	0	2	1.000000
79704	18136	19860830	56256	1964	1	2	3	0	0	5	0.400000
79705	18136	19860901	27095	19340	1	2	2	0	0	4	0.500000
79706	18136	19860902	15063	15806	1	2	0	0	0	2	1.000000
79707	18136	19860912	13176	8165	1	2	0	0	0	2	1.000000
79708	18136	19860920	44973	8181	1	2	0	0	0	2	1.000000
79709	18136	19860930	53123	8165	1	2	0	0	0	2	1.000000
79710	18136	19861021	23742	14542	1	2	0	0	0	2	1.000000
79711	18136	19861025	30145	4743	1	2	0	0	0	2	1.000000
79712	18136	19861102	22703	7247	4	2	0	0	0	2	1.000000
79713	18136	19861117	18046	19340	4	2	0	0	0	2	1.000000
79714	18136	19861230	33493	12438	1	2	0	0	0	2	1.000000
79715	18136	19861230	50441	7871	1	2	0	0	0	2	1.000000
79716	18136	19870101	42852	12429	1	2	0	0	0	2	1.000000
79717	18136	19870116	41078	13496	4	2	0	0	0	2	1.000000
79718	18136	19870125	22213	18407	1	2	0	0	0	2	1.000000
79719	18136	19870127	19758	13460	4	2	0	0	0	2	1.000000
79720	18136	19870201	13917	1971	1	2	0	0	0	2	1.000000
79721	18136	19870222	48106	1893	1	2	0	0	0	2	1.000000
79722	18136	19870315	8102	1983	1	2	0	0	0	2	1.000000
79723	18136	19870320	38881	21764	1	2	2	0	0	4	0.500000
79724	18136	19870409	36648	19340	1	2	2	0	0	4	0.500000
79725	18136	19870425	59574	21788	1	2	0	0	0	2	1.000000
79726	18136	19870509	18647	2355	1	2	0	0	0	2	1.000000
79727	18136	19870509	57447	19618	1	2	0	0	0	2	1.000000
79728	18136	19870510	15061	2376	1	2	0	0	0	2	1.000000
79729	18136	19870510	16028	19798	1	2	0	0	0	2	1.000000
79730	18136	19870516	53059	8420	1	2	0	0	0	2	1.000000
79731	18136	19870624	60224	20523	1	2	0	0	0	2	1.000000
79732	18136	19870626	36155	4500	1	2	0	0	0	2	1.000000
79733	18136	19870705	54567	9869	1	2	0	0	0	2	1.000000
79734	18136	19870712	54243	13216	1	2	0	0	0	2	1.000000

Query executed successfully.

DESKTOP-B5JEBJO (SQL Server 15.0.2000.5 - AzureAD\FilipStrózik(260377)) | Projekt | 00:00:00 | 81,275 rows

Diagram wypełnionych tabel:



Opisy węzłów ETL:

Przygotowanie tabeli danych SQL:

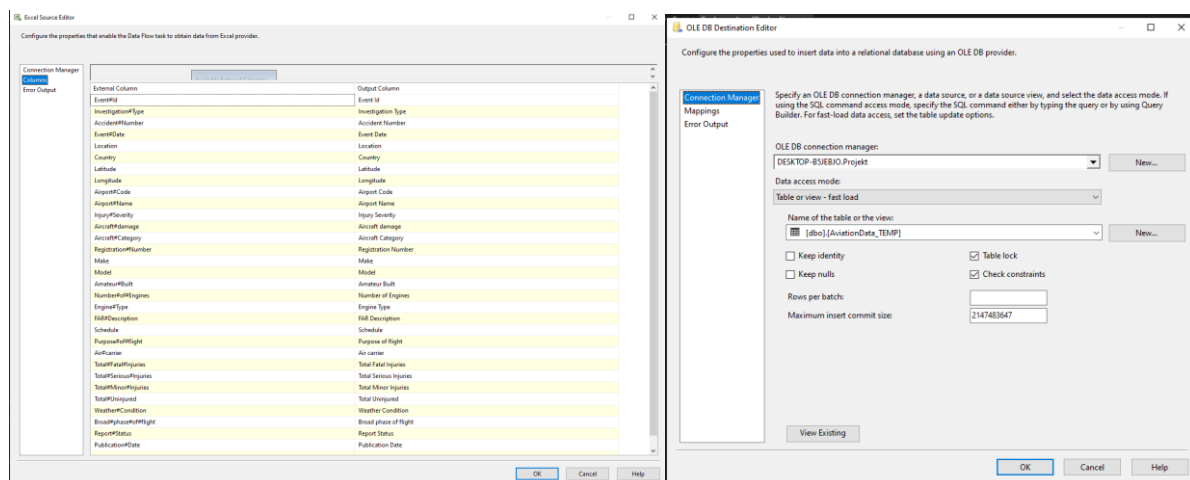
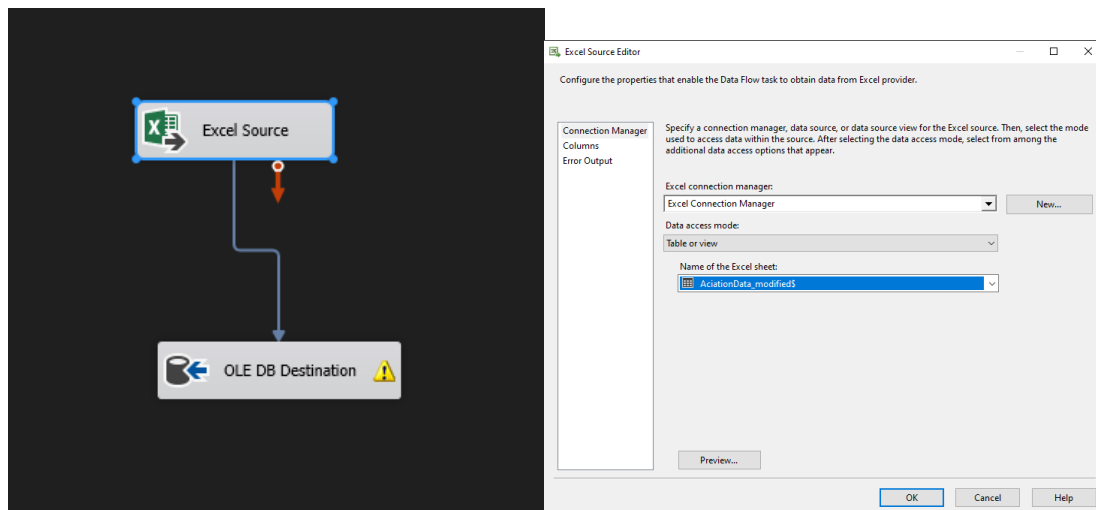
```

-- Drop Temp Tables-----
DROP TABLE IF EXISTS [AviationData_TEMP];

-- Create AviationData_TEMP Table-----
CREATE TABLE [AviationData_TEMP](
    [Event_Id] [nvarchar](50) NOT NULL,
    [Investigation_Type] [nvarchar](20) NOT NULL,
    [Accident_Number] [nvarchar](20) NOT NULL,
    [Event_Date] [date] NOT NULL,
    [Location] [nvarchar](100) NULL,
    [Country] [nvarchar](50) NULL,
    [Latitude] [decimal](18, 6) NULL,
    [Longitude] [decimal](18, 6) NULL,
    [Airport_Code] [nvarchar](10) NULL,
    [Airport_Name] [nvarchar](100) NULL,
    [Injury_Severity] [nvarchar](20) NULL,
    [Aircraft_damage] [nvarchar](15) NULL,
    [Aircraft_Category] [nvarchar](30) NULL,
    [Registration_Number] [nvarchar](15) NULL,
    [Make] [nvarchar](50) NULL,
    [Model] [nvarchar](50) NULL,
    [Amateur_Built] [nvarchar](3) NULL,
    [Number_of_Engines] [int] NULL,
    [Engine_Type] [nvarchar](30) NULL,
    [FAR_Description] [nvarchar](200) NULL,
    [Schedule] [nvarchar](10) NULL,
    [Purpose_of_flight] [nvarchar](30) NULL,
    [Air_carrier] [nvarchar](100) NULL,
    [Total_Fatal_Injuries] [int] NULL,
    [Total_Serious_Injuries] [int] NULL,
    [Total_Minor_Injuries] [int] NULL,
    [Total_Uninjured] [int] NULL,
    [Weather_Condition] [nvarchar](5) NULL,
    [Broad phase of flight] [nvarchar](MAX) NULL,
    [Report_Status] [nvarchar](max) NULL,
    [Publication_Date] [date] NULL
);
  
```

Pierwsze polecenie usunie tabelę AviationData_TEMP, jeżeli taka istnieje. Drugie natomiast utworzy tabelę o tej samej nazwie. Struktura tej tabeli próbuje jak najlepiej odwzorować “surową” tabelę, którą jest skoroszyt w pliku .xlsx. Zdecydowaliśmy się na zapisanie pliku .csv do .xlsx, ponieważ mimo wielu starań próba delimitowania kolumn nie przynosiła porządnego skutku. Ze względu na to, że kolumna Location, zawiera w swoich wartościach znak ‘,’; niemożliwym jest użycie oferowanych przez Integration Services sposobów rozdzielania kolumn w pliku csv. MS Excel poradził sobie z tym zadaniem bezproblemowo.

Załadowanie danych z arkusza:



Załadowanie odbywa się przy pomocy węzła Excel Source, w którym wybieramy odpowiednio plik i skoroszyt. Musimy zadbać o poprawne zmapowanie kolumn, szczególnie w przypadku innych nazw. Wybieramy też tabelę, do której mają być wpisane dane.

Wybieramy naszą tymczasową tabelę AviationData_TEMP.

Wykorzystany kod SQL:

```

-- Create DIMENSIONS-----
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[dbo].[DIM_TIME]') AND type in (N'U'))
CREATE TABLE DIM_TIME
(
    Id INT PRIMARY KEY,
    "Year" INT NOT NULL,
    "Quarter" INT NOT NULL,
    "Month" INT NOT NULL,
    "Month In Words" NVARCHAR(10) NOT NULL,
    "Day" INT NOT NULL,
    "Day In Words" NVARCHAR(10) NOT NULL
);

IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[dbo].[DIM_PLACE]') AND type in (N'U'))
CREATE TABLE DIM_PLACE
(
    Id INT IDENTITY(1,1) PRIMARY KEY,
    Country NVARCHAR(50) NULL,
    District NVARCHAR(100) NULL,
    Region Name NVARCHAR(100) NULL,
    Latitude DECIMAL(18,6) NULL,
    Longitude DECIMAL(18,6) NULL,
    Airport_Code NVARCHAR(10) NULL,
    Airport_Name NVARCHAR(100) NULL
);

IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[dbo].[DIM_CONDITIONS]') AND type in (N'U'))
CREATE TABLE DIM_CONDITIONS
(
    Id INT IDENTITY(1,1) PRIMARY KEY,
    Weather_Condition NVARCHAR(5) NULL,
    Weather_Condition Name NVARCHAR(30) NOT NULL
);

IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[dbo].[DIM_PLANE]') AND type in (N'U'))
CREATE TABLE DIM_PLANE
(
    Id INT IDENTITY(1,1) PRIMARY KEY,
    Make NVARCHAR(50) NULL,
    Model NVARCHAR(50) NULL,
    Amateur Built NVARCHAR(3) NULL,
    Number_Of_Engines INT NULL,
    Engine_Type NVARCHAR(30),
    Aircraft_Category NVARCHAR(30)
);

IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[dbo].[DIM_FLIGHT]') AND type in (N'U'))
CREATE TABLE DIM_FLIGHT
(
    Id INT IDENTITY(1,1) PRIMARY KEY,
    Investigation Type NVARCHAR(20) NULL,
    Injury_Severity NVARCHAR(20) NULL,
    Aircraft_damage NVARCHAR(15) NULL,
    FAR_Description NVARCHAR(200) NULL,
    Schedule NVARCHAR(10) NULL,
    Purpose_of_flight NVARCHAR(30) NULL,
    Air_Carrier NVARCHAR(100) NULL,
    Broad_phase_of_flight NVARCHAR(20) NULL
);

-- Create Facts Tables -----
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[dbo].[FACT_ACCIDENTS]') AND type in (N'U'))
CREATE TABLE FACT_ACCIDENTS
(
    Id INT IDENTITY(1,1) PRIMARY KEY,
    Flight_Id INT NOT NULL,
    Time_Id INT NOT NULL,
    Place_Id INT NOT NULL,
    Plane_Id INT NOT NULL,
    Weather_Conditions_Id INT NOT NULL,
    Total_Fatal_Injuries INT NULL,
    Total_Serious_Injuries INT NULL,
    Total_Minor_Injuries INT NULL,
    Total_Uninjured INT NULL,
    TotalOfInjured INT NULL,
    Mortality DECIMAL(18,6) NULL
);

```

Powyższy kod SQL tworzy pięć różnych tabel wymiarowych (DIMENSIONS) oraz tabelę faktów w bazie danych. Każda tabela reprezentuje różne aspekty danych i będzie służyć do przechowywania informacji w kontekście hurtowni danych. Struktura wymiarów jak i tabeli faktów jest identyczna co do tabel z poprzedniego etapu projektu. Kod SQL sprawdza również, czy tabele już istnieją w bazie danych, a jeśli nie, to tworzy je. Warunek IF NOT EXISTS jest wykorzystywany, aby uniknąć błędów podczas ponownego uruchamiania skryptu tworzenia tabel, gdy tabele już istnieją.

```

-- INSERT DIM_PLANE -----
INSERT INTO DIM_PLANE (
    Make, Model, Amateur_Built, Number_Of_Engines,
    Engine_Type, Aircraft_Category
)
    SELECT
        COALESCE(Make, 'Unknown') AS Make,
        COALESCE(Model, 'Unknown') AS Model,
        COALESCE(Amateur_Built, 'Unk') AS Amateur_Built,
        ISNULL(Number_Of_Engines, 0) AS Number_Of_Engines,
        COALESCE(Engine_Type, 'Unknown') AS Engine_Type,
        COALESCE(Aircraft_Category, 'Unknown') AS Aircraft_Category
    FROM
        (
            SELECT
                DISTINCT Make,
                Model,
                Amateur_Built,
                Number_Of_Engines,
                Engine_Type,
                Aircraft_Category
            FROM
                AviationData_TEMP
        ) A
    EXCEPT
SELECT
    Make, Model, Amateur_Built, Number_Of_Engines, Engine_Type, Aircraft_Category
FROM DIM_PLANE;

-- INSERT DIM_PLACE -----
INSERT INTO DIM_PLACE (Country, District, Region_Name, Latitude, Longitude, Airport_Code, Airport_Name)
SELECT DISTINCT
    COALESCE(Country, 'Unknown') AS Country,
    CASE
        WHEN CHARINDEX(',', REVERSE(Location)) > 0
        THEN LEFT(Location, LEN(Location) - CHARINDEX(',', REVERSE(Location)))
        ELSE NULL
    END AS District,
    CASE
        WHEN Location IS NULL THEN NULL
        ELSE SUBSTRING(Location, LEN(Location) - CHARINDEX(',', REVERSE(Location)) + 3, LEN(Location))
    END AS Region_Name,
    Latitude,
    Longitude,
    COALESCE(Airport_Code, 'Unknown') AS Airport_Code,
    COALESCE(Airport_Name, 'Unknown') AS Airport_Name
FROM AviationData_TEMP
EXCEPT
SELECT Country, District, Region_Name, Latitude, Longitude, Airport_Code, Airport_Name
FROM DIM_PLACE;

-- INSERT DIM_FLIGHT -----
INSERT INTO DIM_FLIGHT (Investigation_Type, Injury_Severity, Aircraft_Damage, FAR_Description, Schedule, Purpose_of_Flight,
    Air_Carrier, Broad_Phase_Of_Flight)
SELECT DISTINCT
    COALESCE(Investigation_Type, 'Unknown'),
    COALESCE(Injury_Severity, 'Unknown'),
    COALESCE(Aircraft_Damage, 'Unknown'),
    COALESCE(FAR_Description, 'Unknown'),
    COALESCE(Schedule, 'Unknown'),
    COALESCE(Purpose_of_flight, 'Unknown'),
    COALESCE(Air_carrier, 'Unknown'),
    COALESCE(Broad_phase_of_flight, 'Unknown')
FROM AviationData_TEMP
EXCEPT
SELECT
    Investigation_Type,
    Injury_Severity,
    Aircraft_Damage,
    FAR_Description,
    Schedule,
    Purpose_of_Flight,
    Air_Carrier,
    Broad_Phase_Of_Flight
FROM DIM_FLIGHT;

-- INSERT DIM_CONDITIONS -----
INSERT INTO DIM_CONDITIONS (Weather_Condition, Weather_Condition_Name)
    SELECT
        Weather_Condition AS Weather_Condition_Code,
        CASE
            WHEN Weather_Condition = 'VMC' THEN 'Good conditions'
            WHEN Weather_Condition = 'UNK' OR Weather_Condition = '' OR Weather_Condition IS NULL THEN 'Unknown'
            WHEN Weather_Condition = 'IMC' THEN 'Bad conditions'
        END AS Weather_Condition_Name
    FROM
        (
            SELECT DISTINCT
                Weather_Condition
            FROM
                AviationData_TEMP
        ) A
    EXCEPT
SELECT Weather_Condition, Weather_Condition_Name
FROM DIM_CONDITIONS;

```

```

-- INSERT DIM TIME -----
DECLARE @D INT;
SET @D = (SELECT TOP 1 DATEPART(YYYY, Event_Date) * 10000 + DATEPART(MM, Event_Date) * 100 + DATEPART(DD, Event_Date) FROM
AviationData_TEMP ORDER BY 1);

DECLARE @COUNTER DATE;
SET @COUNTER = CONVERT(date, CAST(@D AS nvarchar));

DECLARE @END INT;
SET @END = (SELECT TOP 1 DATEPART(YYYY, Event_Date) * 10000 + DATEPART(MM, Event_Date) * 100 + DATEPART(DD, Event_Date) FROM
AviationData_TEMP ORDER BY 1 DESC);

WHILE (@D <= @END)
BEGIN
    IF NOT EXISTS (
        SELECT 1
        FROM DIM_TIME
        WHERE Id = @D
    )
    BEGIN
        INSERT INTO DIM_TIME
        VALUES (
            @D,
            YEAR(@COUNTER),
            DATEPART(QQ, @COUNTER),
            MONTH(@COUNTER),
            DATENAME(MONTH, @COUNTER),
            DAY(@COUNTER),
            DATENAME(WEEKDAY, @COUNTER)
        );
    END;

    SET @COUNTER = DATEADD(DAY, 1, @COUNTER);
    SET @D = CAST(CONVERT(varchar(8), @COUNTER, 112) AS INT);
END;

-- INSERT FACT ACCIDENTS -----
INSERT INTO FACT_ACCIDENTS (Flight_Id, Time_Id, Place_Id, Plane_Id, Weather_Conditions_Id, Total_Fatal_Injuries,
Total_Serious_Injuries, Total_Minor_Injuries, Total_Uninjured, TotalOfInjured, Mortality)
SELECT
    DIM_FLIGHT.Id AS Flight_Id,
    DIM_TIME.Id AS Time_Id,
    DIM_PLACE.Id AS Place_Id,
    DIM_PLANE.Id AS Plane_Id,
    DIM_CONDITIONS.Id AS Weather_Conditions_Id,
    ISNULL(AviationData_TEMP.Total_Fatal_Injuries, 0) AS Total_Fatal_Injuries,
    ISNULL(AviationData_TEMP.Total_Serious_Injuries, 0) AS Total_Serious_Injuries,
    ISNULL(AviationData_TEMP.Total_Minor_Injuries, 0) AS Total_Minor_Injuries,
    ISNULL(AviationData_TEMP.Total_Uninjured, 0) AS Total_Uninjured,
    ISNULL(AviationData_TEMP.Total_Fatal_Injuries, 0) + ISNULL(AviationData_TEMP.Total_Serious_Injuries, 0) +
    ISNULL(AviationData_TEMP.Total_Minor_Injuries, 0) AS TotalOfInjured,
    CASE
        WHEN AviationData_TEMP.Total_Fatal_Injuries IS NULL THEN 0
        WHEN AviationData_TEMP.Total_Uninjured + AviationData_TEMP.Total_Serious_Injuries +
        AviationData_TEMP.Total_Minor_Injuries = 0 THEN 1
        ELSE AviationData_TEMP.Total_Fatal_Injuries * 1.0 / (AviationData_TEMP.Total_Uninjured +
        AviationData_TEMP.Total_Serious_Injuries + AviationData_TEMP.Total_Minor_Injuries)
    END AS Mortality
FROM AviationData_TEMP
JOIN DIM_FLIGHT ON
    CONCAT(
        COALESCE(AviationData_TEMP.Investigation_Type, 'Unknown'),
        COALESCE(AviationData_TEMP.Injury_Severity, 'Unknown'),
        COALESCE(AviationData_TEMP.Aircraft_Damage, 'Unknown'),
        COALESCE(AviationData_TEMP.FAR_Description, 'Unknown'),
        COALESCE(AviationData_TEMP.Schedule, 'Unknown'),
        COALESCE(AviationData_TEMP.Purpose_Of_Flight, 'Unknown'),
        COALESCE(AviationData_TEMP.Air_Carrier, 'Unknown'),
        COALESCE(AviationData_TEMP.Broad_Phase_Of_Flight, 'Unknown')
    ) =
    CONCAT(
        DIM_FLIGHT.Investigation_Type,
        DIM_FLIGHT.Injury_Severity,
        DIM_FLIGHT.Aircraft_Damage,
        DIM_FLIGHT.FAR_Description,
        DIM_FLIGHT.Schedule,
        DIM_FLIGHT.Purpose_Of_Flight,
        DIM_FLIGHT.Air_Carrier,
        DIM_FLIGHT.Broad_Phase_Of_Flight
    )
JOIN DIM_TIME ON DIM_TIME.Id = DATEPART(YYYY, Event_Date) * 10000 +
DATEPART(MM, Event_Date) * 100 + DATEPART(DD, Event_Date)
JOIN DIM_PLANE ON
    CONCAT(
        DIM_PLANE.Make,
        DIM_PLANE.Model,
        DIM_PLANE.Amateur_Built,
        CAST(DIM_PLANE.Number_Of_Engines AS nvarchar(2)),
        DIM_PLANE.Engine_Type,
        DIM_PLANE.Aircraft_Category
    ) =
    CONCAT(
        COALESCE(AviationData_TEMP.Make, 'Unknown'),
        COALESCE(AviationData_TEMP.Model, 'Unknown'),
        COALESCE(AviationData_TEMP.Amateur_Built, 'Unk'),
        CAST(AviationData_TEMP.Number_Of_Engines AS nvarchar(2)),
        COALESCE(AviationData_TEMP.Engine_Type, 'Unknown'),
        COALESCE(AviationData_TEMP.Aircraft_Category, 'Unknown')
    )
JOIN DIM_CONDITIONS ON DIM_CONDITIONS.Weather_Condition = COALESCE(AviationData_TEMP.Weather_Condition,
'Unknown')

```

```

        JOIN DIM_PLACE ON
        CONCAT(
            DIM_PLACE.Country,
            ISNULL(CONCAT_WS(' ', DIM_PLACE.District, DIM_PLACE.Region_Name), ''),
            DIM_PLACE.Latitude,
            DIM_PLACE.Longitude,
            DIM_PLACE.Airport_Code,
            DIM_PLACE.Airport_Name)
        =
        CONCAT(
            COALESCE(AviationData_TEMP.Country, 'Unknown'),
            AviationData_TEMP.Location,
            AviationData_TEMP.Latitude,
            AviationData_TEMP.Longitude,
            COALESCE(AviationData_TEMP.Airport_Code, 'Unknown'),
            COALESCE(AviationData_TEMP.Airport_Name, 'Unknown')
        )
    EXCEPT
SELECT Flight_Id, Time_Id, Place_Id, Plane_Id, Weather_Conditions_Id, Total_Fatal_Injuries, Total_Serious_Injuries,
Total_Minor_Injuries, Total_Uninjured, TotalOfInjured, Mortality
FROM dbo.FACT_ACCIDENTS;

```

Odpowiednio INSERT DIM PLANE:

SQL pełni funkcję wstawiania danych do tabeli wymiarów. Użyto funkcji COALESCE, która służy do zamiany wartości NULL na inną wartość, w tym przypadku 'Unknown'. ISNULL sprawdza, czy wartość w kolumnie Number_Of_Engines jest NULL, jeśli tak, zamienia ją na 0. DISTINCT pobiera unikalne wartości kolumn z AviationData_TEMP. EXCEPT wykonuje różnicę zbiorów pomiędzy wynikiem podzapytania A, a danymi już obecnymi w tabeli DIM_PLANE. W ten sposób zostaną wybrane tylko te rekordy, które nie istnieją jeszcze w tabeli DIM_PLANE.

Odpowiednio INSERT DIM PLACE:

Tutaj też mamy do czynienia z użyciem DISTINCT, COALESCE oraz EXCEPT, zatem działanie jest analogiczne jak powyżej, ale z kolumny Location staramy się wyodrębnić informację o dystrykcie i nazwie regionu w celu zrealizowania hierarchii podczas budowania przyszłej kostki danych.

```

CASE
    WHEN CHARINDEX(' ', REVERSE(Location)) > 0
    THEN LEFT(Location, LEN(Location) - CHARINDEX(' ',
REVERSE(Location)))
    ELSE NULL
END AS District,
CHARINDEX(' ', REVERSE(Location))

```

Ta część kodu znajduje pozycję ostatniego wystąpienia przecinka w odwróconym łańcuchu znaków "Location". Metoda REVERSE odwraca łańcuch znaków, a CHARINDEX znajduje pozycję przecinka.

```
LEN(Location) - CHARINDEX(' ', REVERSE(Location))
```

Odejmuje pozycję przecinka od całkowitej długości łańcucha "Location", aby otrzymać liczbę znaków przed ostatnim przecinkiem.

```
LEFT(Location, LEN(Location) - CHARINDEX(' ', REVERSE(Location)))
```

Metoda LEFT pobiera określoną liczbę znaków z lewej strony łańcucha "Location", która jest obliczana jako długość łańcucha minus pozycja ostatniego przecinka. To daje nam fragment łańcucha przed ostatnim przecinkiem, który reprezentuje dystrykt.

```

CASE
    WHEN Location IS NULL THEN NULL
    ELSE SUBSTRING(Location, LEN(Location) - CHARINDEX(' ',
REVERSE(Location)) + 3, LEN(Location))
END AS Region_Name,
SUBSTRING(Location, LEN(Location) - CHARINDEX(' ', REVERSE(Location)) + 3,
LEN(Location))

```

Metoda SUBSTRING pobiera podłańcuch z łańcucha "Location" na podstawie określonego zakresu. Wyrażenie

```
LEN(Location) - CHARINDEX(' ', REVERSE(Location)) + 3
```

oblicza pozycję początkową, która jest trzy znaki po ostatnim przecinku w odwróconym łańcuchu "Location". Następnie, LEN(Location) określa liczbę znaków do pobrania, aby otrzymać fragment łańcucha po ostatnim przecinku, który reprezentuje nazwę regionu.

```
WHEN Location IS NULL THEN NULL
```

Ta linia sprawdza, czy wartość w kolumnie "Location" jest NULL. Jeśli tak, to przypisuje wartość NULL dla kolumny "Region_Name".

Odpowiednio INSERT DIM FLIGHT:

W tym zapytaniu SQL, DISTINCT, EXCEPT oraz COALESCE zostały użyte analogicznie do powyższych zapytań, zapytanie wprowadza nowe unikalne rekordy do tabeli.

Odpowiednio INSERT DIM CONDIDITIONS:

W tym kodzie SQL należy zwrócić uwagę na wykorzystaniu instrukcji CASE aby przypisać nowe wartości do kolumny Weather_Condition_Name na podstawie wartości w kolumnie Weather_Condition. Na przykład, jeśli Weather_Condition ma wartość 'VMC', przypisze "Good conditions", a jeśli jest 'IMC', przypisze "Bad conditions". W przypadku wartości 'UNK' lub pustych wartości, przypisze "Unknown". W ten sposób kod przekształca dane w sposób bardziej opisowy i łatwiejszy do zrozumienia w kontekście warunków pogodowych.

Odpowiednio INSERT DIM TIME:

```
DECLARE @D
```

Deklaruje zmienną @D jako liczbę całkowitą, która będzie przechowywać wartość daty w formacie RRRRMMDD.

```
SET @D
```

Przypisuje do zmiennej @D wartość daty, pobraną z kolumny Event_Date z tabeli AviationData_TEMP. Wartość ta jest obliczana jako suma roku, miesiąca i dnia przemnożonych przez odpowiednie mnożniki. Instrukcja SELECT TOP 1 zwraca najmniejszą wartość daty (Event_Date) z tabeli AviationData_TEMP, sortując wyniki w porządku rosnącym.

```
DECLARE @COUNTER
```

Deklaruje zmienną @COUNTER jako typ daty DATE. Przypisuje jej wartość @D, przekonwertowaną na datę.

```
SET @END
```

Przypisuje do zmiennej @END największą wartość daty (Event_Date) z tabeli AviationData_TEMP, sortując wyniki w porządku malejącym.

```
WHILE (@D <= @END)
```

Wykonuje się dopóki wartość zmiennej @D jest mniejsza lub równa wartości zmiennej @END. Sprawdza, czy istnieje wiersz w tabeli DIM_TIME o Id równym @D. Jeśli nie istnieje, wykonuje instrukcję INSERT, która wstawia nowy wiersz do tabeli DIM_TIME. Wartości kolumn wstawianego wiersza są pobierane z daty przechowywanej w zmiennej @COUNTER, włączając rok, kwartał, miesiąc, miesiąc jako nazwę tekstową, dzień oraz dzień jako nazwę tekstową. Po wstawieniu wiersza, wartość zmiennej @COUNTER jest zwiększana o 1 dzień za pomocą instrukcji SET i DATEADD. Wartość zmiennej @D jest aktualizowana, przekształcając wartość zmiennej @COUNTER na liczbę całkowitą w formacie RRRRMMDD.

Odpowiednio INSERT FACT ACCIDENTS:

Kod SQL ma za zadanie wpisać wartości do tabeli faktów. Wykorzystuje funkcję CONCAT, COALESCE i CAST do porównywania wartości w tabeli tymczasowej (AviationData_TEMP) z wartościami w tabelach wymiarowych w celu znalezienia odpowiadających rekordów, dzięki czemu możliwe jest złączenie z wymiarami na podstawie tylko tych wartości. Poprawne złączenie umożliwia przypisanie referencji do rekordów pochodzących z wymiarów. Wykonuje obliczenia i transformacje danych, takie jak zsumowanie liczby poszkodowanych, obliczenie wskaźnika śmiertelności (Mortality) na podstawie liczby obrażeń w stosunku do ogólnej liczby poszkodowanych.

```
-- Integrity -----
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id = OBJECT_ID(N'[dbo].[CONDITIONS_FOREIGN_KEY]') AND
parent_object_id = OBJECT_ID(N'[dbo].[FACT_ACCIDENTS]'))
ALTER TABLE FACT_ACCIDENTS
    ADD CONSTRAINT CONDITIONS_FOREIGN_KEY FOREIGN KEY(Weather_Conditions_Id) REFERENCES DIM_CONDITIONS(Id);

IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id = OBJECT_ID(N'[dbo].[ACCIDENT_FOREIGN_KEY]') AND
parent_object_id = OBJECT_ID(N'[dbo].[FACT_ACCIDENTS]'))
ALTER TABLE FACT_ACCIDENTS
    ADD CONSTRAINT ACCIDENT_FOREIGN_KEY FOREIGN KEY(Flight_Id) REFERENCES DIM_FLIGHT(Id);

IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id = OBJECT_ID(N'[dbo].[PLACE_FOREIGN_KEY]') AND parent_object_id
= OBJECT_ID(N'[dbo].[FACT_ACCIDENTS]'))
ALTER TABLE FACT_ACCIDENTS
    ADD CONSTRAINT PLACE_FOREIGN_KEY FOREIGN KEY(Place_Id) REFERENCES DIM_PLACE(Id);

IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id = OBJECT_ID(N'[dbo].[PLANE_FOREIGN_KEY]') AND parent_object_id
= OBJECT_ID(N'[dbo].[FACT_ACCIDENTS]'))
ALTER TABLE FACT_ACCIDENTS
    ADD CONSTRAINT PLANE_FOREIGN_KEY FOREIGN KEY(Plane_Id) REFERENCES DIM_PLANE(Id);

IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id = OBJECT_ID(N'[dbo].[EVENT_DATE_FOREIGN_KEY]') AND
parent_object_id = OBJECT_ID(N'[dbo].[FACT_ACCIDENTS]'))
ALTER TABLE FACT_ACCIDENTS
    ADD CONSTRAINT EVENT_DATE_FOREIGN_KEY FOREIGN KEY(Time_Id) REFERENCES DIM_TIME(Id);

-- Log - Error -----
INSERT INTO ETL_LOG VALUES(?, GETDATE(),1, 'error');

-- Log - data read from EXCEL file -----
INSERT INTO ETL_LOG VALUES(?, GETDATE(),0, 'data read from Excel ');

-- Log - created dimensions -----
INSERT INTO ETL_LOG VALUES(?, GETDATE(),0, 'created dimensions1 ');

-- Log - ETL success -----
INSERT INTO ETL_LOG VALUES(?, GETDATE(),0, 'success');
```

Odpowiednio Integrity:

Powyższy kod SQL zawiera wiele zapytań ALTER TABLE, które dodają ograniczenia klucza obcego do tabeli FACT_ACCIDENTS, tylko i wyłącznie jeżeli dane ograniczenie nie istnieje.

Odpowiednio Log:

Wpisuje do dziennika zdarzeń wartości z odpowiednim opisem a parametr to System:ServerExecutionID.

Target				Source			Transformation
Table Name	Column Name	Data Type	Table type	Table Name	Column Name	Data Type	
DIM_FLIGHT	Id	int	Dimension				Identity
DIM_FLIGHT	Investigation_Type	nvarchar	Dimension	AviationData_TEMP	Investigation_Type	nvarchar	COALESCE(Investigation_Type, 'Unknown'),
DIM_FLIGHT	Injury_Severity	nvarchar	Dimension	AviationData_TEMP	Injury_Severity	nvarchar	COALESCE(Injury_Severity, 'Unknown'),
DIM_FLIGHT	Aircraft_damage	nvarchar	Dimension	AviationData_TEMP	Aircraft_damage	nvarchar	COALESCE(Aircraft_damage, 'Unknown'),
DIM_FLIGHT	FAR_Description	nvarchar	Dimension	AviationData_TEMP	FAR_Description	nvarchar	COALESCE(FAR_Description, 'Unknown'),
DIM_FLIGHT	Schedule	nvarchar	Dimension	AviationData_TEMP	Schedule	nvarchar	COALESCE(Schedule, 'Unknown'),
DIM_FLIGHT	Purpose_of_flight	nvarchar	Dimension	AviationData_TEMP	Purpose_of_flight	nvarchar	COALESCE(Purpose_of_flight, 'Unknown'),
DIM_FLIGHT	Air_Carrier	nvarchar	Dimension	AviationData_TEMP	Air_Carrier	nvarchar	COALESCE(Air_carrier, 'Unknown'),
DIM_FLIGHT	Broad_phase_of_flight	nvarchar	Dimension	AviationData_TEMP	Broad_phase_of_flight	nvarchar	COALESCE(Broad_phase_of_flight, 'Unknown')
DIM_CONDITIONS	Id	int	Dimension				Identity
DIM_CONDITIONS	Weather_Condition	nvarchar	Dimension	AviationData_TEMP	Weather_Condition	nvarchar	---
DIM_CONDITIONS	Weather_Condition_Name	nvarchar	Dimension	AviationData_TEMP	Weather_Condition	nvarchar	WHEN Weather_Conditions = 'VMC' THEN 'Good conditions' WHEN Weather_Conditions = 'UNK' OR Weather_Conditions=' ' OR Weather_condition IS NULL THEN 'Unknown conditions' WHEN Weather_Conditions = 'IMC' THEN 'Bad conditions'
DIM_PLACE	Id	int	Dimension				Identity
DIM_PLACE	Country	nvarchar	Dimension	AviationData_TEMP	Country	nvarchar	COALESCE(Country, 'Unknown')
DIM_PLACE	District	nvarchar	Dimension	AviationData_TEMP	Location	nvarchar	WHEN CHARINDEX(',', REVERSE(Location)) > 0 THEN LEFT(Location, LEN(Location) - CHARINDEX(',', REVERSE(Location))) ELSE NULL END
DIM_PLACE	Region_Name	nvarchar	Dimension	AviationData_TEMP	Location	nvarchar	WHEN Location IS NULL THEN NULL ELSE SUBSTRING(Location, LEN(Location) - CHARINDEX(',', REVERSE(Location)) + 3, LEN(Location))
DIM_PLACE	Latitude	decimal	Dimension	AviationData_TEMP	Latitude	decimal	---
DIM_PLACE	Longitude	decimal	Dimension	AviationData_TEMP	Longitude	decimal	---
DIM_PLACE	Airport_Code	nvarchar	Dimension	AviationData_TEMP	Airport_Code	nvarchar	COALESCE(Airport_Code, 'Unknown')
DIM_PLACE	Airport_Name	nvarchar	Dimension	AviationData_TEMP	Airport_Name	nvarchar	COALESCE(Airport_Name, 'Unknown')
DIM_PLANE	Id	int	Dimension				Identity
DIM_PLANE	Make	nvarchar	Dimension	AviationData_TEMP	Make	nvarchar	COALESCE(Make, 'Unknown')
DIM_PLANE	Model	nvarchar	Dimension	AviationData_TEMP	Model	nvarchar	COALESCE(Model, 'Unknown')
DIM_PLANE	Amateur_Built	nvarchar	Dimension	AviationData_TEMP	Amateur_Built	nvarchar	COALESCE(Amateur_Built, 'Unk')
DIM_PLANE	Number_Of_Engines	int	Dimension	AviationData_TEMP	Number_Of_Engines	int	ISNULL(Number_Of_Engines,0)
DIM_PLANE	Engine_Type	nvarchar	Dimension	AviationData_TEMP	Engine_Type	nvarchar	COALESCE(Engine_Type, 'Unknown')
DIM_PLANE	Aircraft_Category	nvarchar	Dimension	AviationData_TEMP	Aircraft_Category	nvarchar	COALESCE(Aircraft_Category, 'Unknown')
DIM_TIME	Id	int	Dimension	AviationData_TEMP	Event_Date	date	DATEPART(YYYY, Event_Date) * 10000 + DATEPART(MM, Event_Date) * 100 + DATEPART(DD, Event_Date)
DIM_TIME	Year	int	Dimension	AviationData_TEMP	Event_Date	date	YEAR(Event_Date)

DIM_TIME	Quarter	int	Dimension	AviationData_TEMP	Event_Date	date	DATEPART(QQ, Event_Date)
DIM_TIME	Month	int	Dimension	AviationData_TEMP	Event_Date	date	MONTH(Event_Date)
DIM_TIME	Month In Words	nvarchar	Dimension	AviationData_TEMP	Event_Date	date	DATENAME(MONTH, Event_Date)
DIM_TIME	Day	int	Dimension	AviationData_TEMP	Event_Date	date	DAY(Event_Date)
DIM_TIME	Day In Words	nvarchar	Dimension	AviationData_TEMP	Event_Date	date	DATENAME(WEEKDAY, Event_Date),
FACT_ACCIDENTS	Id	int	Fact				Identity
FACT_ACCIDENTS	Flight_Id	Int	Fact	DIM_FLIGHT	Id	int	---
FACT_ACCIDENTS	Time_Id	int	Fact	DIM_TIME	Id	int	---
FACT_ACCIDENTS	Place_Id	int	Fact	DIM_PLACE	Id	int	---
FACT_ACCIDENTS	Plane_Id	int	Fact	DIM_PLANE	Id	int	---
FACT_ACCIDENTS	Weather_Conditions_Id	int	Fact	DIM_CONDITIONS	Id	int	---
FACT_ACCIDENTS	Total_Fatal_Injuries	int	Fact	AviationData_TEMP	Total_Fatal_Injuries	int	ISNULL(AviationData_TEMP.Total_Fatal_Injuries, 0)
FACT_ACCIDENTS	Total_Serious_Injuries	int	Fact	AviationData_TEMP	Total_Serious_Injuries	int	ISNULL(AviationData_TEMP.Total_Serious_Injuries, 0)
FACT_ACCIDENTS	Total_Minor_Injuries	int	Fact	AviationData_TEMP	Total_Minor_Injuries	int	ISNULL(AviationData_TEMP.Total_Minor_Injuries, 0)
FACT_ACCIDENTS	Total_Uninjured	int	Fact	AviationData_TEMP	Total_Uninjured	int	ISNULL(AviationData_TEMP.Total_Uninjured, 0)
FACT_ACCIDENTS	TotalOfInjured	int	Fact	AviationData_TEMP	Total_Fatal_Injuries, Total_Serious_Injuries, Total_Minor_Injuries	int	ISNULL(AviationData_TEMP.Total_Fatal_Injuries, 0) + ISNULL(AviationData_TEMP.Total_Serious_Injuries, 0) + ISNULL(AviationData_TEMP.Total_Minor_Injuries, 0)
FACT_ACCIDENTS	Mortality	decimal	Fact	AviationData_TEMP	Total_Uninjured, Total_Fatal_Injuries, Total_Serious_Injuries, Total_Minor_Injuries	int	WHEN AviationData_TEMP.Total_Fatal_Injuries IS NULL THEN 0 WHEN AviationData_TEMP.Total_Uninjured + AviationData_TEMP.Total_Serious_Injuries + AviationData_TEMP.Total_Minor_Injuries = 0 THEN 1 ELSE AviationData_TEMP.Total_Fatal_Injuries * 1.0 / (AviationData_TEMP.Total_Uninjured + AviationData_TEMP.Total_Serious_Injuries + AviationData_TEMP.Total_Minor_Injuries + AviationData_TEMP.Total_Fatal_Injuries)

Wnioski:

Proces ETL został przez nas odpowiednio podzielony na zadania oraz data flow, co poskutkowało przejrzystością i kontrolą nad poszczególnymi etapami. Nazwy zadań zostały dobrze dobrane, odzwierciedlają przede wszystkim cel. Obsługa błędów również została uwzględniona, co oznacza, że ewentualne problemy są identyfikowane i rejestrowane w dzienniku zdarzeń.

Całość pakietu działa stabilnie, bez poważnych błędów lub przerw w wykonywaniu zadań. Można podkreślić optymalizację polegającą na asynchronicznym wykonywaniu podprocesów niezależnych od siebie. W momencie zadania, które dotyczy wyników asynchronicznych zadań, odpowiednio oczekuje się na poprawne wykonanie każdego z nich. Dane są wypełniane zgodnie z ustalonymi założeniami, co potwierdza prawidłowość działania procesu.

Proces ETL nie obsługuje usuwania tabel wymiarowych w zamian za to wpisuje nowe rekordy. Przetrzymywana jest kopia tymczasowa źródłowych danych w ramach całego procesu ETL. Przeprowadzone operacje są dostosowane do potrzeb projektu i uwzględniają specyfikę danych źródłowych i docelowych.

Dokumentacja procesu ETL zawiera szczegółowe informacje o każdym zadaniu, opisując jego cel. Dzięki temu łatwo jest zrozumieć, jakie transformacje danych są wykonywane.

Mapa logiczna procesu ETL przedstawia w formie tabeli poszczególne transformacje, ze źródła do celu, co ułatwia analizę i zrozumienie.

Zbiór danych w postaci tabeli z dużą liczbą kolumn, jest trudny do rozłożenia na wymiary, ponieważ, tylko porównując wszystkie wartości dotyczące danego wymiaru jesteśmy w stanie określić, czy dane zdarzenie dotyczy tego samego obiektu w wymiarze. Wymagało to dodania klucza sztucznego.

Podsumowując, proces ETL został zaprojektowany i wykonany zgodnie z założeniami projektowymi. Posiada odpowiednią strukturę, jest stabilny i wykonuje się bezbłędnie. Operacje są dostosowane do potrzeb projektu hurtowni danych. Dokumentacja procesu jest kompletna i umożliwia łatwe zrozumienie poszczególnych zadań oraz ogólnej logiki procesu ETL.

Projekt – etap III

Kostka:

1. Przygotować projekt kostki, edytować wymiary, dodać miary kalkulowane. Przygotować zestawienia z p. 1.5.2. oraz pokazać inne ciekawe zależności w analizowanych danych (analiza w głąb, a nie tylko tabele przestawne).

Przy ocenie będą brane następujące elementy kostki:

- prawidłowa struktura kostki – model kostki powinien analitykowi na intuicyjne i łatwe korzystanie z danych
- miary kalkulowane
- dokumentacja, która powinna zawierać krótki opis wszystkich wymiarów, wszystkich ich atrybutów oraz wszystkich miar

Wnioski:

Projekt – etap IV

Prezentacja

Prezentacja powinna zawierać 4-8 slajdów (trwać ok. 8 minut) i wyjaśniać jakie dane są przedmiotem analizy. Prezentacja powinna być zakończona, krótką demonstracją, która pokaże najciekawsze związki między danymi znajdującymi się w kostce.

Uwaga. Projekt będzie ostatecznie zaliczony po złożeniu pisemnego sprawozdania zawierającego opisy poszczególnych etapów pracy.