

Piotr Rybarczyk
Nr Indeksu 10117
Nr Grupy Z507
Projekt: [GitHub](#)

Projekt Zaliczeniowy Bazy Danych II

Baza danych dla sieci restauracji

Wywiad z Klientem	3
Schemat Bazy Danych	4
Konfiguracja Globalna	4
Restauracje	5
Pracownicy	6
Zasoby	7
Klienci	8
Zamówienia	9
Założenia Projektowe	10
Polepszenie bezpieczeństwa bazy danych	10
Spójność Danych	11
Raportowa Baza Danych	12
Scenariusze i Rozwiązanie	13
Tworzenie kont i przypisanie ról systemowych	13
Tworzenie kont i przypisanie ról użytkowników	15
Walidacja Danych Zamówień	16
Mechanizm Soft Delete Zamówień	18
Mechanizm Anonimizowania Danych Klienta	20
Widoki bazodanowe	24
Raportowa Baza Danych	26
Podsumowanie i wnioski	29
Skrypt	30

Wywiad z Klientem

Firma naszego klienta przechodzi dynamiczny rozwój i koncentrując się na głównym celu, jakim jest dostarczanie posiłków 24/7/365. Aby sprostać rosnącym wymaganiom rynku i konkurencji, firma zatrudniła zespół programistów, którzy mają za zadanie przygotować nowe warstwy aplikacji i rozwinąć aktualnie istniejące a następnie zająć się ich utrzymaniem. Celem jest umożliwienie różnym członkom zespołu pracy nad projektami niezależnie od siebie. Do nas należy, przygotowanie bazy danych tak, aby dane były bezpieczne, dostępne a sama baza skalowalna.

Firma klienta planuje wprowadzić trzy główne aplikacje:

- **Panel Administracyjny:** Zapewnia możliwość edycji wszystkich danych w systemie, z dostępem ograniczonym do nielicznych uprawnionych użytkowników.
- **Panel Restauracji:** Umożliwia edycję menu, posiłków, produktów, zarządzanie zamówieniami oraz godzinami pracy pracowników.
- **Panel Zamówień:** Służy do zarządzania zamówieniami klientów.

Dodatkowo, planowane jest wprowadzenie niezależnego **panelu raportowego**, który pozwoli na przegląd najważniejszych raportów dotyczących działalności sieci restauracji klienta. Tym zadaniem zajmą się analitycy naszego klienta, ale najpierw musimy im przygotować bazę danych na której mogą pracować.

Dodatkowo, co istotne, **schemat aktualnej baza danych nie może się zmienić**, ani stracić żadnych zawartych w niej danych. Jest już wdrożona na produkcji, i tak ma zostać.

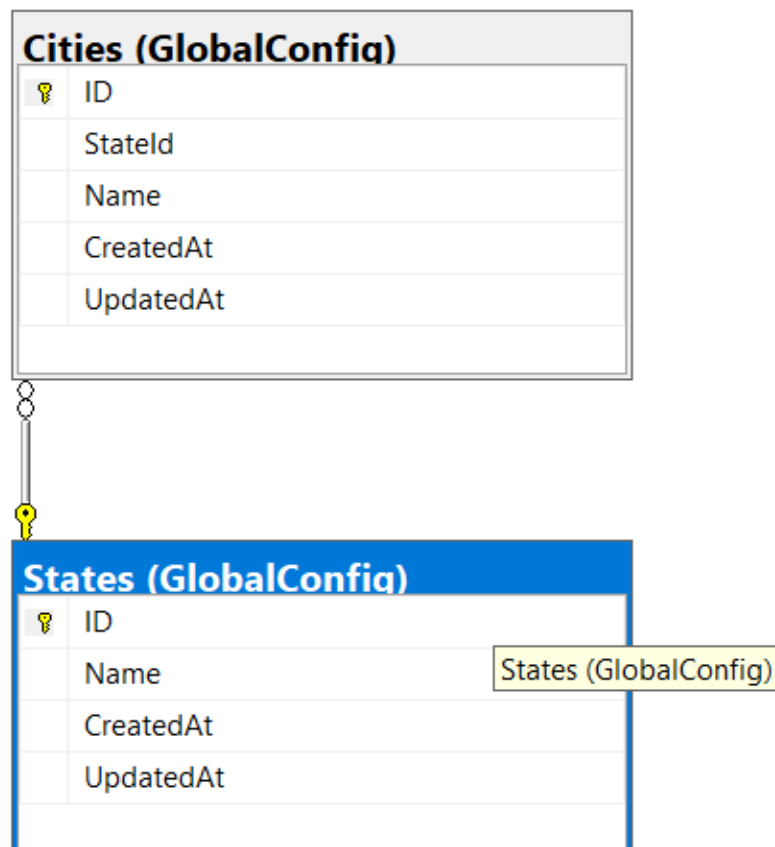
Schemat Bazy Danych

Baza danych klienta została podzielona na wiele podobszarów, schematów, które reprezentują różne części systemu.

W tej części, możemy znaleźć schemat bazy danych, w dniu w którym otrzymujemy ja od klienta.

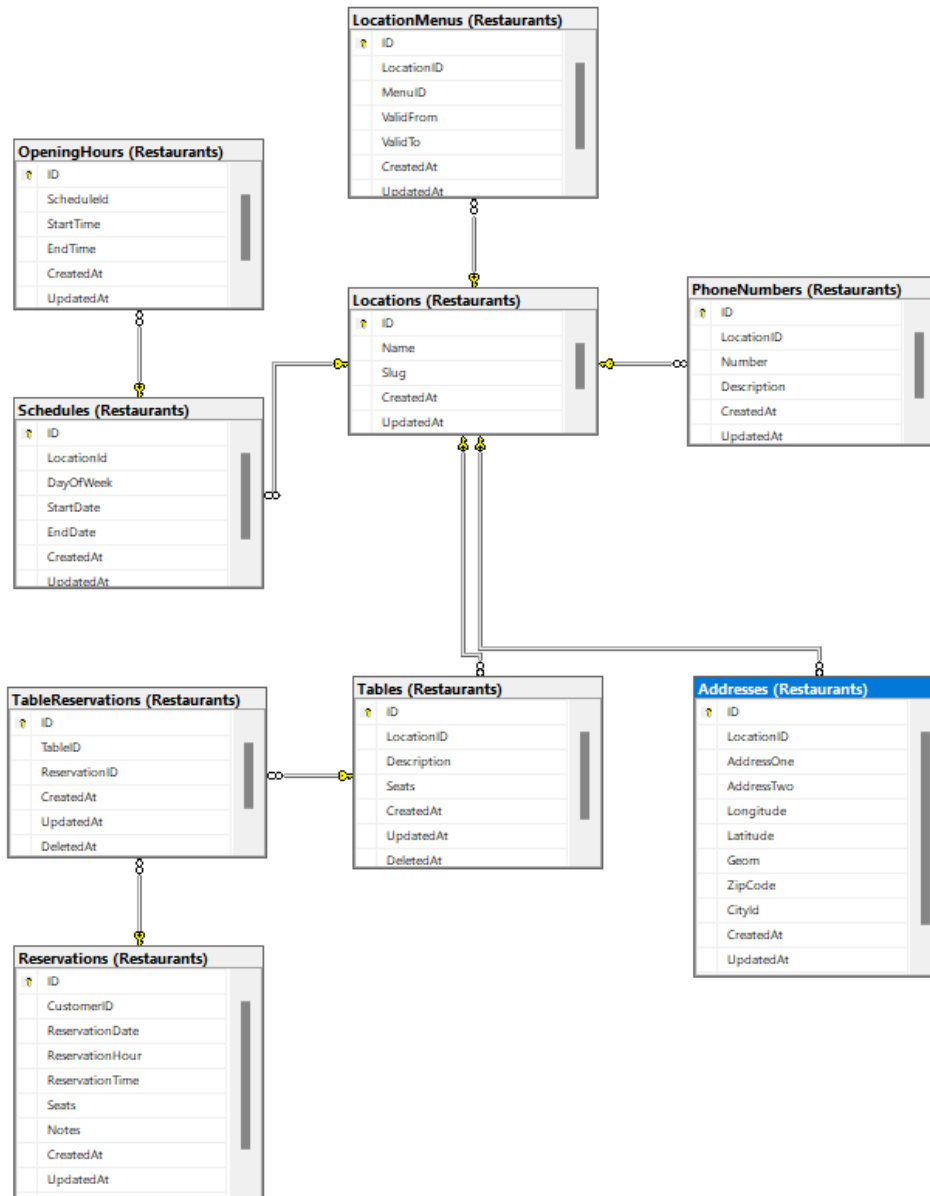
Konfiguracja Globalna

Konfiguracja globalna zawiera dane, które powinny być unikalne nie zależnie od obszaru, w którym beda uzywane. Na dany moment, mamy tutaj dane geolokacyjne



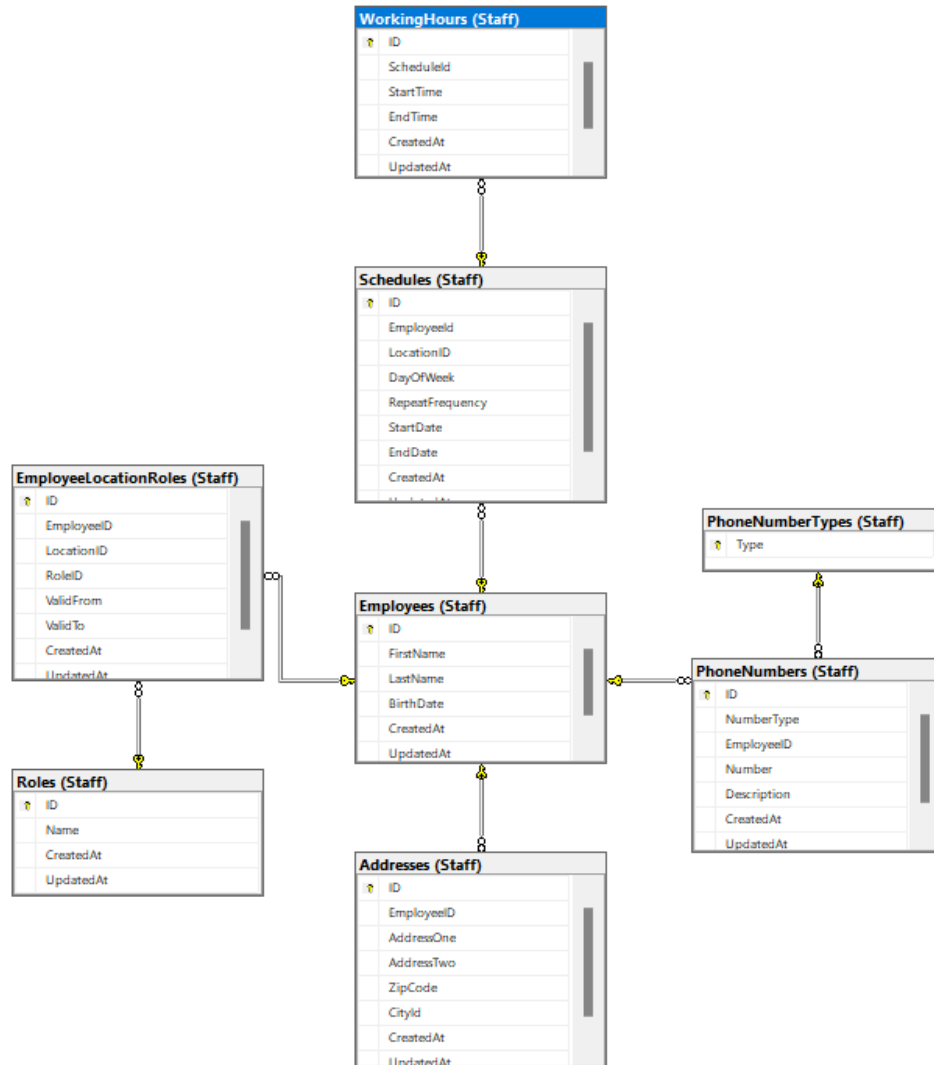
Restauracje

Restauracje zawierają wszystkie podstawowe dane potrzebne do zarządzania restauracjami, takie jak godziny otwarcia, stoliki, dane kontaktowe ale także tabele łączące z innymi obszarami.



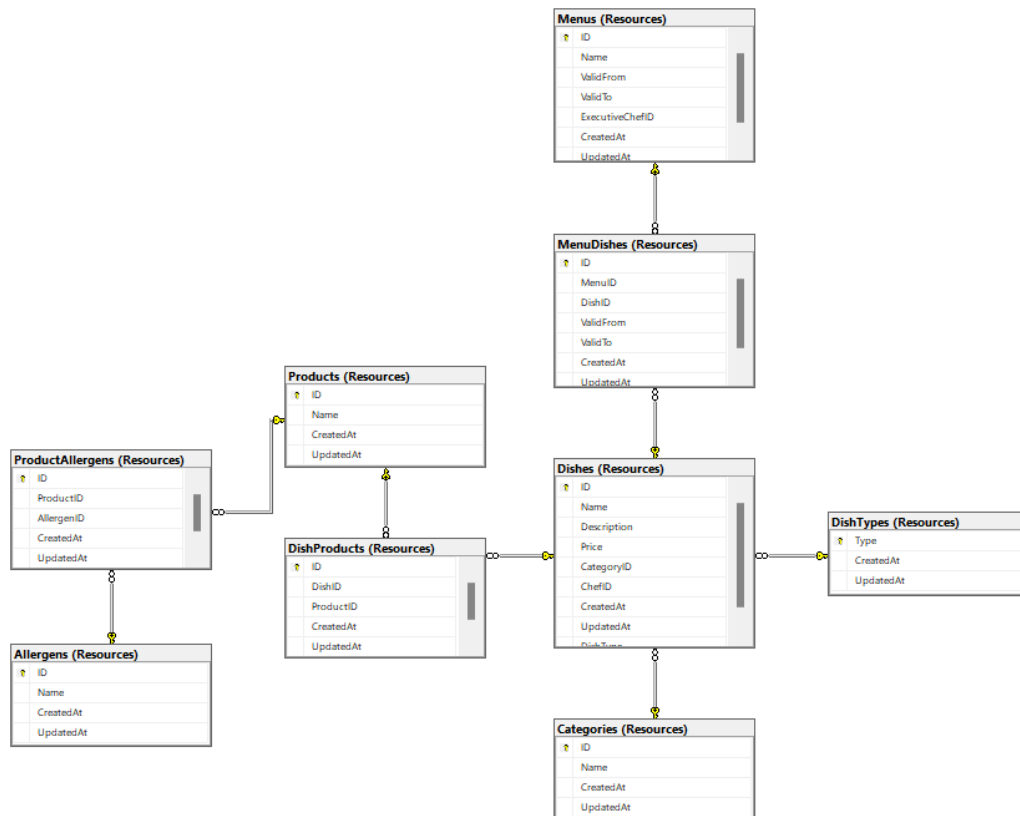
Pracownicy

Pracownicy zawierają wszystkie dane pracowników, ale także ich rolę, godziny pracy czy dane kontaktowe.



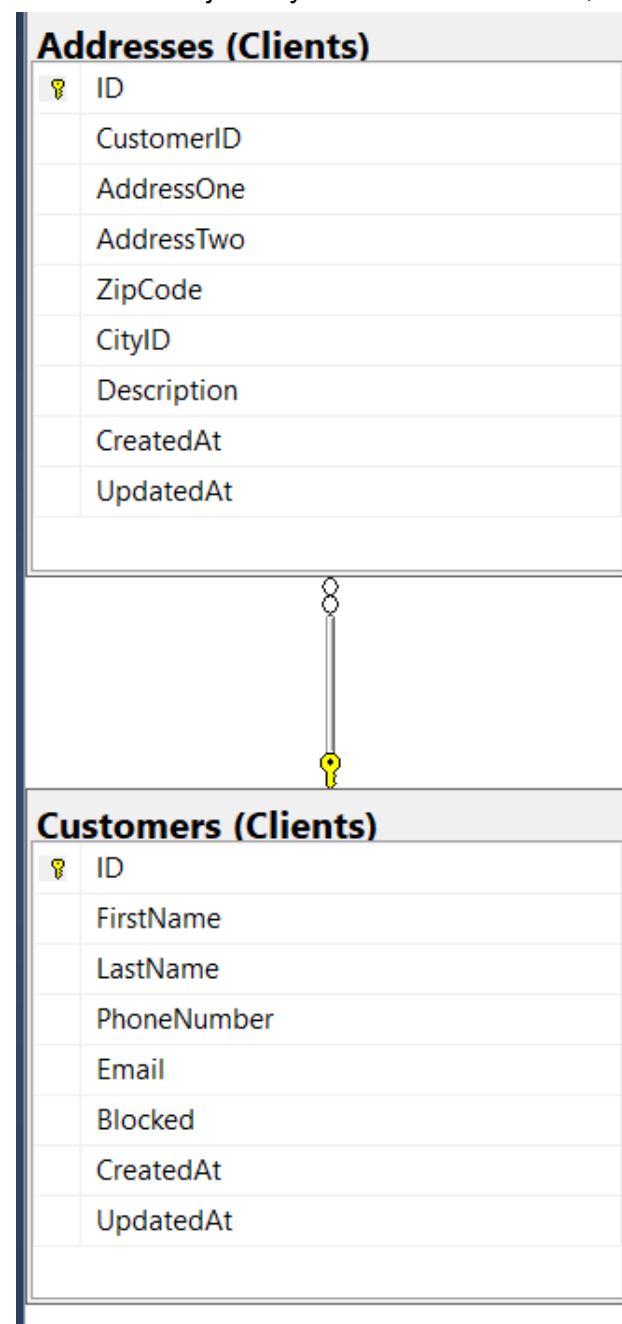
Zasoby

Zasoby zawierają wszystkie dane o dostępnych daniach w restauracji, wymaganych składnikach i zawartych w nich alergenach.



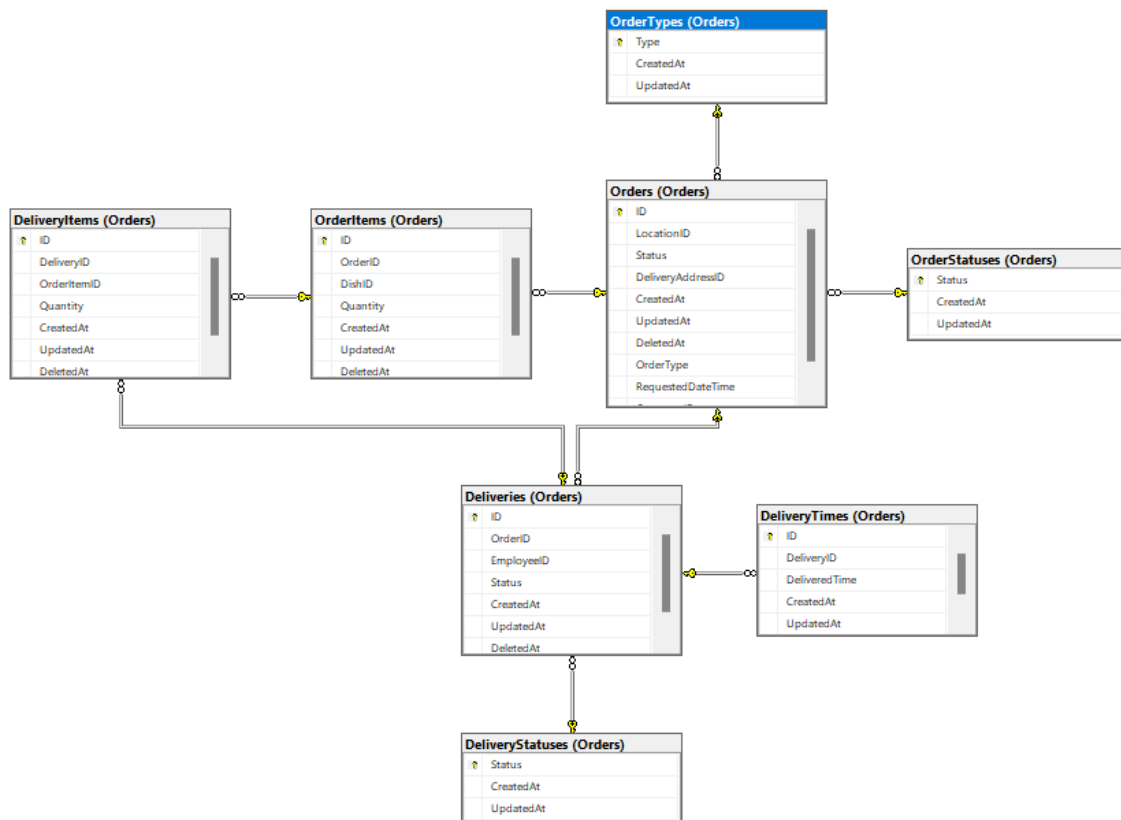
Klienci

Klienci zawierają wszystkie dane o klientach, ich adresy czy numer kontaktowy.



Zamówienia

Zamówienia zawierają historię zamówień, co zostało zamówione i gdzie ale także czy już zostało dostarczone.



Założenia Projektowe

Polepszenie bezpieczeństwa bazy danych

Jednym z głównych celów naszego projektu, jest polepszenie bezpieczeństwa bazy danych. Wiemy, że klient ma zespół deweloperski, który ma przygotować moduły aplikacji odpowiedzialne za różne części firmy. Na tej bazie możemy przygotować zestaw odpowiednich ról i dostępuów.

W uproszczeniu, można założyć, że każda rola może czytać dowolne obiekty z dowolnego obszaru bazy danych, jednak edycja czy też usuwanie danych powinno być ograniczone. Dodatkowo, żadna z aplikacji nie może mieć możliwości zmieniania ustawień same bazy danych.

Sami deweloperzy również powinni mieć możliwość czytania danych w bazie i o bazie ale nie powinni mieć możliwości zmieniania czegokolwiek, z wyłączeniem zespołu odpowiedzialnego za maintenance. .

Na tej zasadzie możemy określić poniższe zestawy uprawnień

Role Aplikacji:

- superuser - specjalnie uprawnienia dla aplikacji do definiowania nowych tabel.
- administrator - uprawnienia dla konta z którego będzie korzystać aplikacja admina. Ma możliwość pracy ze wszystkimi tabelami
- restauracja - uprawnienia dla konta z którego będzie korzystać aplikacja restauracji. Ma możliwość czytania wszystkich danych i edytowania wszystkich danych.
- zamówienia - uprawnienia dla konta z którego będzie korzystać aplikacja zamówień. Może ona czytać wszystkie dane z wyłączeniem pracowników, i edytować tylko dane zamówień i klientów.
- raporty - uprawnienia dla konta modułu raportowego. Ma możliwość czytania wszystkich danych, ale nic nie może edytować.

Role Użytkowników:

- developer - uprawnienia dla dewelopera. Ma możliwość czytania wszystkich danych z wyłączeniem danych osobowych takie jak adresy, imiona i nazwiska
- maintenance - uprawnienia dla wybranych deweloperów. Ma możliwość czytania, edycji i usuwania wszystkich danych.

Spójność Danych

Kolejnym, istotnym dla działania serwisu naszego klienta podpunktem, jest zapewnienie spójności danych w bazie. Oznacza to, że musimy upewnić się czy typ wprowadzanych danych spełnia wymogi biznesowe. A także, że można przeprowadzać operacje opierające się na wielu tabelach jednocześnie, w sposób, który zapewni spójność danych pomiędzy nimi.

Spójność zapisu danych

- Powinniśmy zapewnić, że dane wprowadzane w obszarach takich jak Zamówienia czy Zasoby spełniają logiczne założenia takie jak brak negatywnych wartości w polach Monetarnych czy Ilościowych.
- Niektóre dane nigdy nie powinny być usuwane, oznacza to, że powinniśmy wprowadzić do nich mechanizm, który ukryje je przed wyszukiwaniem.
- Dodatkowo, dane personalne pracowników i klientów powinny być traktowane ze szczególną uwagą ze względu na prawa GDPR, kiedy klient poprosi o usunięcie
- konta musimy upewnić się że dane osobowe zostaną poprawnie usunięte.

Spójność odczytu danych

- W bazie danych znajdują się dziesiątki tabel, każda mającą konkretne znaczenie i przeznaczenie. Niektóre zapytania jednak będą się powtarzać między nimi. W takim przypadku potrzebujemy wprowadzić widoki, których założeniem jest ukrycie logiki przed użytkownikiem i zapewnienie spójnej definicji danych.

Raportowa Baza Danych

Ostatnią częścią projektu, będzie utworzenie oddzielnej bazy raportowej działającej w modelu ELT, tj. Extract Load Transform. Oznacza, to że w pierwszej części naszego projektu musimy zagwarantować przepływ i replikację danych pomiędzy bazami FoodCourt i Food Court Reports.

W procesie ELT, najlepsze praktyki mówią o utworzeniu wielu warstw logicznych w bazie raportowej tj:

- **ING_** - pierwsza warstwa ingestion, gdzie trafiają dane nieobrobione z głównej bazy danych.
- **CLN_** - druga warstwa, cleaned, gdzie dane zostają oczyszczone i wstępnie obrobione.
- **MDL_** - trzecia warstwa, modeled, gdzie dane przybierają ostateczny kształt.

Nasz zespół stanowią bazodanowcy, tak więc do naszych obowiązków należy przygotowanie warstwy **ING_**.

Scenariusze I Rozwiązanie

Tworzenie kont i przypisanie ról systemowych

Wszystkie role, które będą przypisane poszczególnym częściom systemu powinny być tworzone ręcznie. Dodatkowo wszystkie będą miały prefix 'app_'.

APP SUPERUSER

```
/* Tworzenie Loginu */
CREATE LOGIN app_superuser WITH PASSWORD = 'admin123';

USE FoodCourt;

CREATE USER app_superuser FOR LOGIN app_superuser;
EXEC sp_addrolemember 'db_ddladmin', 'app_superuser';
EXEC sp_addrolemember 'db_datawriter', 'app_superuser';
EXEC sp_addrolemember 'db_datareader', 'app_superuser';

/* Testowanie Usera */
EXECUTE AS USER = 'app_superuser';
CREATE TABLE test(id int);
CREATE INDEX test_idx ON test(id)
INSERT INTO test VALUES(1);
SELECT * FROM test;
DROP TABLE test;
REVERT;
```

109 %

Results Messages

	id
1	1

APP ADMIN, RESTAURANTS, REPORTS

```
CREATE LOGIN app_admin WITH PASSWORD = 'admin123';
CREATE LOGIN app_restaurants WITH PASSWORD = 'admin123';
CREATE LOGIN app_reports WITH PASSWORD = 'admin123';

USE FoodCourt;

/* Create Users */
CREATE USER app_admin FOR LOGIN app_admin;
CREATE USER app_restaurants FOR LOGIN app_restaurants;
CREATE USER app_reports FOR LOGIN app_reports;
GO

/* Assign Global Permissions */
/* Global Write */
EXEC sp_addrolemember 'db_datawriter', 'app_admin';
EXEC sp_addrolemember 'db_datawriter', 'app_restaurants';
/* Global Read */
EXEC sp_addrolemember 'db_datareader', 'app_admin';
EXEC sp_addrolemember 'db_datareader', 'app_restaurants';
EXEC sp_addrolemember 'db_datareader', 'app_reports';
GO
```

APP ORDERS

```

/* Create Login */
CREATE LOGIN app_orders WITH PASSWORD = 'admin123';

USE FoodCourt;

/* Create Users */
CREATE USER app_orders FOR LOGIN app_orders;
GO

/* Assign Global Permissions */
EXEC sp_addrolemember 'db_datareader', 'app_orders';
GO

/* Schema Permissions */
CREATE ROLE app_orders_role;
GO

/* app orders role */
GRANT INSERT, UPDATE, DELETE ON SCHEMA::Clients TO app_orders_role;
GRANT INSERT, UPDATE, DELETE ON SCHEMA::Orders TO app_orders_role;
GO

ALTER ROLE app_orders_role ADD MEMBER app_orders;
GO

```

SQLQuery4.sql - LA...R18281\argon (70)) * X 02_create_global_r...R18281\argon (64)) 03_create_custom_r...R18281\argon (56))

```

EXECUTE AS USER = 'app_orders';
SELECT * FROM Resources.Dishes;
REVERT;

```

82 %

Results Messages

	ID	Name	Description	Price	CategoryID	ChefID	CreatedAt	UpdatedAt	DishType
1	156	Cheeseburger	Breaded fried chicken with waffles, and a side of mapl...	78,12	77	392	2023-05-07 15:38:51.2666667	2023-05-07 15:38:51.2666667	Dessert
2	157	Scotch Eggs	Two butter croissants of your choice (plain, almond or ...	17,61	80	374	2023-05-07 15:38:51.2666667	2023-05-07 15:38:51.2666667	Appetizer

SQLQuery4.sql - LA...R18281\argon (70)) * X 02_create_global_r...R18281\argon (64)) 03_create_custom_r...R18281\argon (56))

```

EXECUTE AS USER = 'app_orders';
SELECT * FROM Resources.Dishes;
INSERT INTO Resources.Dishes(Name, Description, Price, CategoryID, ChefID, DishType) VALUES('test', 'test', 100, 77, 392, 'Dessert');
REVERT;

```

82 %

Results Messages

(101 rows affected)
 Msg 229, Level 14, State 5, Line 3
 The INSERT permission was denied on the object 'Dishes', database 'FoodCourt', schema 'Resources'.

Completion time: 2023-12-20T16:28:58.5779817+01:00

Podsumowanie

- Security
 - Users
 - app_admin
 - app_orders
 - app_reports
 - app_restaurants
 - app_superuser
 - dbo
 - guest
 - INFORMATION_SCHEMA
 - sys
 - Roles
 - Database Roles
 - app_orders_role
 - db_accessadmin
 - db_backupoperator
 - db_datareader
 - db_datawriter
 - db_ddladmin
 - db_denydatareader
 - db_denydatawriter
 - db_owner
 - db_securityadmin
 - public

Tworzenie kont i przypisanie ról użytkowników

Wszystkie role, które będą przypisane poszczególnym częściom systemu powinny być utworzone ręcznie i w zadne sposób nie opierać się na rolach serwerowych. Dodatkowo wszystkie będą miały prefix 'user_'.

The screenshot displays three SQL queries executed in SQL Server Enterprise Manager, showing the creation of users and roles, and the execution of queries with permission errors.

Query 1: SQLQuery6.sql - LA...R18281\argon (64))*

```
/* Create user developer */
CREATE LOGIN user_tomek WITH PASSWORD = 'user123' MUST_CHANGE, CHECK_POLICY = ON, CHECK_EXPIRATION = ON;
CREATE USER user_tomek FOR LOGIN user_tomek;
CREATE LOGIN user_anna WITH PASSWORD = 'user123' MUST_CHANGE, CHECK_POLICY = ON, CHECK_EXPIRATION = ON;
CREATE USER user_anna FOR LOGIN user_anna;

CREATE ROLE user_developer_role;
CREATE ROLE user_maintenance_role;

/*Maintenance Role */
EXEC sp_addrolemember 'db_datareader', 'user_anna';
EXEC sp_addrolemember 'db_datawriter', 'user_anna';

/*Developer Role*/
EXEC sp_addrolemember 'db_datareader', 'user_tomek';
DENY SELECT ON Clients.Addresses TO user_developer_role;
DENY SELECT ON Clients.Customers([FirstName]) TO user_developer_role;
DENY SELECT ON Clients.Customers([PhoneNumber]) TO user_developer_role;
DENY SELECT ON Clients.Customers([Email]) TO user_developer_role;
DENY SELECT ON Staff.Addresses TO user_developer_role;
DENY SELECT ON Staff.PhoneNumbers TO user_developer_role;
DENY SELECT ON Staff.Employees([LastName]) TO user_developer_role;
```

Query 2: SQLQuery7.sql - LA...R18281\argon (70))*

```
EXECUTE AS USER = 'user_tomek';
SELECT * FROM Staff.Addresses;
REVERT;
```

Query 3: SQLQuery7.sql - LA...R18281\argon (70))*

```
EXECUTE AS USER = 'user_tomek';
SELECT FirstName FROM Staff.Employees;
REVERT;
```

Messages:

Msg 229, Level 14, State 5, Line 2
The SELECT permission was denied on the object 'Addresses', database 'FoodCourt', schema 'Staff'.
Completion time: 2023-12-20T16:55:27.4407098+01:00

Msg 220, Level 14, State 1, Line 2
The SELECT permission was denied on the column 'LastName' of the object 'Employees', database 'FoodCourt', schema 'Staff'.
Completion time: 2023-12-20T16:56:21.2067352+01:00

Results:

FirstName
1 Brandon
2 Donnie
3 Juanita
4 Elsie
5 Andrea

Walidacja Danych Zamówień

Base Constraints

```
USE FoodCourt
GO
/* Order Schema Validation */
/* Basic Constraints */
ALTER TABLE Orders.OrderItems ADD CONSTRAINT CHK_Quantity_NonNegative CHECK (Quantity >= 0);
ALTER TABLE Orders.DeliveryItems ADD CONSTRAINT CHK_Quantity_NonNegative CHECK (Quantity >= 0);
ALTER TABLE Orders.DeliveryTimes ADD CONSTRAINT CHK_DeliveryTime_NotFuture CHECK (DeliveredTime <= GETDATE());
GO

SQLQuery9.sql - LA...R18281\argon (60)*
SQLQuery8.sql - LA...R18281\argon (64)*
LAPTOP-TV18281.F...Court - Diagram_0*

SELECT * FROM Orders.Deliveries;
INSERT INTO Orders.DeliveryTimes(DeliveryID, DeliveredTime) VALUES(4814, DATETIME(YEAR, 1, GETDATE()));

82 %

Messages
Msg 547, Level 16, State 0, Line 3
The INSERT statement conflicted with the CHECK constraint "CHK_DeliveryTime_NotFuture". The conflict occurred in database "FoodCourt", table "Orders.DeliveryTimes", column "DeliveredTime".
The statement has been terminated.
Completion time: 2023-12-20T19:02:54.2274759+01:00
```

Base Rules

```
USE FoodCourt
GO
database FoodCourt
CREATE
GO
/* Add Title To Tables */
IF NOT EXISTS (SELECT * FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA = 'Staff' AND TABLE_NAME = 'Employees' AND COLUMN_NAME = 'Title')
BEGIN
    ALTER TABLE Staff.Employees
    ADD Title NVARCHAR(5) NULL;
    EXEC sp_bindrule 'TITLE_RULE', 'Staff.Employees.Title';
END
ELSE
BEGIN
    PRINT('Column Title already exists');
END;

IF NOT EXISTS (SELECT * FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA = 'Clients' AND TABLE_NAME = 'Customers' AND COLUMN_NAME = 'Title')
BEGIN
    ALTER TABLE Clients.Customers
    ADD Title NVARCHAR(5) NULL;
    EXEC sp_bindrule 'TITLE_RULE', 'Clients.Customers.Title';
END
ELSE
BEGIN
    PRINT('Column Title already exists');
END;

SELECT * FROM Staff.Employees;
UPDATE Staff.Employees SET Title = 'Mr' WHERE ID = 361;
GO
UPDATE Staff.Employees SET Title = 'Mrs' WHERE ID = 373;
GO

51 %

Results Messages
(170 rows affected)
(1 row affected)
Msg 512, Level 16, State 0, Line 30
A column insert or update conflicts with a rule imposed by a previous CREATE RULE statement. The statement was terminated. The conflict occurred in database "FoodCourt", table "Staff.Employees", column "Title".
The statement has been terminated.
Completion time: 2023-12-20T18:50:13.002949+01:00
```


Check Triggers

The image displays three sequential screenshots of the SQL Server Enterprise Manager interface, specifically the SQL Query window, demonstrating the creation and execution of a trigger.

Top Screenshot: Shows the creation of a trigger named `TRG_CheckQuantity` on the `Orders.DeliveryItems` table. The trigger is configured to fire `AFTER INSERT, UPDATE` and is set to `NOCOUNT ON`. The trigger logic includes a check for existing items in the `Orders.OrderItems` table. If an item exists with a quantity greater than the inserted item's quantity, an error is raised: `RAISERROR('Quantity in DeliveryItem cannot exceed Quantity in OrderItem', 10, 1);` followed by `ROLLBACK TRANSACTION;` and `RETURN;`.

```
CREATE TRIGGER TRG_CheckQuantity ON Orders.DeliveryItems
AFTER INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (
        SELECT 1
        FROM Inserted AS I
        INNER JOIN Orders.OrderItems AS OOI
        ON I.OrderItemID = OOI.ID
        WHERE I.Quantity > OOI.Quantity
    )
    BEGIN
        /* msg, severity, state */
        RAISERROR('Quantity in DeliveryItem cannot exceed Quantity in OrderItem', 10, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END
END
```

Middle Screenshot: Shows the execution of a query that attempts to insert a new item with a quantity of 3, which exceeds the existing item's quantity of 1. The query is: `SELECT * FROM Orders.OrderItems WHERE ID = 26492;` followed by `UPDATE Orders.DeliveryItems SET Quantity = 3 WHERE ID = 26492;`. The execution fails, and the Messages pane displays the following error:

(1 row affected)
Msg 50000, Level 16, State 1, Procedure TRG_CheckQuantity, Line 15 (Batch Start Line 0)
Quantity in DeliveryItem cannot exceed Quantity in OrderItem
Msg 3609, Level 16, State 1, Line 2
The transaction ended in the trigger. The batch has been aborted.
Completion time: 2023-12-28T19:11:41.1414842+01:00

Bottom Screenshot: Shows the execution of a query that successfully inserts a new item with a quantity of 1, which does not exceed the existing item's quantity of 1. The query is: `SELECT * FROM Orders.OrderItems WHERE ID = 26492;` followed by `UPDATE Orders.DeliveryItems SET Quantity = 1 WHERE ID = 26492;`. The execution succeeds, and the Messages pane displays the following message:

(1 row affected)
Completion time: 2023-12-28T19:12:56.2823845+01:00

Mechanizm Soft Delete Zamówień

Pobieranie Powiązań

Funkcja tabelaryczna która zwraca dane o powiązaniach bazując na nazwie tabeli i schemie w której ma operować.

```
--CREATE FUNCTION dbo.GetLinkedTables
(
    @TableName NVARCHAR(128),
    @SchemaName NVARCHAR(128)
)
RETURNS @Result TABLE
(
    TableId NVARCHAR(256),
    SchemaName NVARCHAR(256),
    TableName NVARCHAR(256),
    ParentTableName NVARCHAR(256),
    ColumnName NVARCHAR(256),
    Level INT
)
AS
BEGIN
    WITH RecursiveFKs(ForeignKeyName, ParentObjectId, ReferenceObjectId, ParentColumnName, ReferencedColumnName, Level)
    AS
    (
        SELECT
            FK.name AS ForeignKeyName,
            FK.parent_object_id AS ParentObjectId,
            FK.referenced_object_id AS ReferenceObjectId,
            COL_NAME(FKC.parent_object_id, FKC.parent_column_id) AS ParentColumnName,
            COL_NAME(FKC.referenced_object_id, FKC.referenced_column_id) AS ReferencedColumnName,
            1 AS Level
        FROM
            sys.foreign_keys AS FK
            INNER JOIN sys.foreign_key_columns AS FKC
            ON FK.object_id = FKC.constraint_object_id
        WHERE
            OBJECT_NAME(FK.referenced_object_id) = @TableName
            AND OBJECT_SCHEMA_NAME(FK.parent_object_id) = @SchemaName

        UNION ALL

        SELECT
            FK.name AS ForeignKeyName,
            FK.parent_object_id AS ParentObjectId,
            FK.referenced_object_id AS ReferenceObjectId,
            COL_NAME(FKC.parent_object_id, FKC.parent_column_id) AS ParentColumnName,
            COL_NAME(FKC.referenced_object_id, FKC.referenced_column_id) AS ReferencedColumnName,
            R.Level + 1
        FROM
            sys.foreign_keys AS FK
            INNER JOIN RecursiveFKs AS R
            ON FK.referenced_object_id = R.ParentObjectId
            INNER JOIN sys.foreign_key_columns AS FKC
            ON FK.object_id = FKC.constraint_object_id
        WHERE
            OBJECT_NAME(FK.referenced_object_id) != @TableName
            AND OBJECT_SCHEMA_NAME(FK.parent_object_id) = @SchemaName
    )
    INSERT INTO @Result
    SELECT
        DISTINCT
            ParentObjectId AS TableId,
            OBJECT_SCHEMA_NAME(ParentObjectId) AS SchemaName,
            OBJECT_NAME(ParentObjectId) AS TableName,
            OBJECT_NAME(ReferenceObjectId) AS ParentTableName,
            ParentColumnName AS ColumnName,
            Level
    FROM RecursiveFKs
    ORDER BY Level
    RETURN;
END;
GO
```

Mechanizm Soft Delete

Trigger, który zamiast usuwać Order, dodaje wartość to pola deleted at i uzupełnia kolejne tabele powiązane.

```
CREATE TRIGGER TRG_SoftDeleteRelationships
ON Orders.Orders
INSTEAD OF DELETE
AS
BEGIN
    SET NOCOUNT ON;

    CREATE TABLE #ChangedRows (TableName NVARCHAR(128), ID INT);
    DECLARE @TableSchemaName NVARCHAR(128), @TableName NVARCHAR(128), @ParentTableName NVARCHAR(128), @ColumnName NVARCHAR(128);
    DECLARE @SchemaName NVARCHAR(128), @QUERY NVARCHAR(MAX), @CurrentDateTime DATETIME;

    SET @CurrentDateTime = GETDATE();
    SET @SchemaName = 'Orders';

    UPDATE Orders SET DeletedAt = @CurrentDateTime FROM Orders INNER JOIN deleted ON Orders.ID = deleted.ID;
    INSERT INTO #ChangedRows (TableName, ID) SELECT 'Orders', deleted.ID FROM deleted;

    DECLARE cursor_linkedTables CURSOR FOR
        SELECT SchemaName, TableName, ParentTableName, ColumnName FROM dbo.GetLinkedTables('Orders', @SchemaName);

    OPEN cursor_linkedTables;
    FETCH NEXT FROM cursor_linkedTables INTO @TableSchemaName, @TableName, @ParentTableName, @ColumnName;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        PRINT 'Aktualnie przetwarzana tabela: ' + @TableName;

        SET @QUERY = 'DECLARE @TempTable TABLE (ID INT);' +
            'UPDATE ' + QUOTENAME(@TableSchemaName) + '.' + QUOTENAME(@TableName) +
            ' SET DeletedAt = @CurrentDateTime ' +
            'OUTPUT INSERTED.ID INTO @TempTable ' +
            'WHERE ' + QUOTENAME(@ColumnName) + ' IN (SELECT ID FROM #ChangedRows WHERE TableName = ''' + @ParentTableName + '''); ' +
            'INSERT INTO #ChangedRows (TableName, ID) SELECT ''' + @TableName + ''', ID FROM @TempTable;';

        EXEC sp_executesql @QUERY, N'@CurrentDateTime DATETIME', @CurrentDateTime;
        FETCH NEXT FROM cursor_linkedTables INTO @TableSchemaName, @TableName, @ParentTableName, @ColumnName;
    END

    SELECT * FROM #ChangedRows;
    DROP TABLE #ChangedRows;
    CLOSE cursor_linkedTables;
    DEALLOCATE cursor_linkedTables;
END
```

Sprawdzenie

- Pierwsze okno przed usunięciem
- Drugie okno wybranie wartości
- Trzecie okno po usunięciu

DELETE Orders.Orders WHERE ID = 2006;
GO

62 %

Results Messages

	OrderId	OrderDeletedAt	OrderItemId	OrderItemDeletedAt	OrderDeliveryDeletedAt	OrderDeliveryItemDeletedAt	OrderDeliveryTimeDeletedAt
1	2006	NULL	5960	NULL	NULL	NULL	NULL
2	2006	NULL	5958	NULL	NULL	NULL	NULL
3	2006	NULL	5959	NULL	NULL	NULL	NULL

	TableName	ID
1	Orders	2006
2	OrderItems	5958
3	OrderItems	5959
4	OrderItems	5960

	OrderId	OrderDeletedAt	OrderItemId	OrderItemDeletedAt	OrderDeliveryDeletedAt	OrderDeliveryItemDeletedAt	OrderDeliveryTimeDeletedAt
1	2006	2023-12-23 21:37:48.9766667	5960	2023-12-23 21:37:48.9766667	NULL	NULL	NULL
2	2006	2023-12-23 21:37:48.9766667	5958	2023-12-23 21:37:48.9766667	NULL	NULL	NULL
3	2006	2023-12-23 21:37:48.9766667	5959	2023-12-23 21:37:48.9766667	NULL	NULL	NULL

Mechanizm Anonimizowania Danych Klienta

Dodawanie Pola Deleted At

Dynamiczne dodanie pola deleted_at do wszystkich tabel w schemacie Clients.

```
/* Add Deleted At Field to Clients Tables */
DECLARE @TableName NVARCHAR(256);
DECLARE @ColumnName NVARCHAR(256);
DECLARE @Sql NVARCHAR(MAX);
SET @ColumnName = 'DeletedAt'

-- Kursor do iteracji przez wszystkie tabele w schemacie 'Clients'
DECLARE cursor_table CURSOR FOR
    SELECT t.name
    FROM sys.tables AS t
    INNER JOIN sys.schemas AS s ON t.schema_id = s.schema_id
    WHERE s.name = 'Clients';

OPEN cursor_table;
FETCH NEXT FROM cursor_table INTO @TableName;

WHILE @@FETCH_STATUS = 0
BEGIN
    IF NOT EXISTS (
        SELECT *
        FROM sys.columns AS c
        INNER JOIN sys.tables AS t ON c.object_id = t.object_id
        INNER JOIN sys.schemas AS s ON t.schema_id = s.schema_id
        WHERE s.name = 'Clients'
        AND t.name = @TableName
        AND c.name = @ColumnName
    )
    BEGIN
        PRINT('Processing Table ' + @TableName);
        SET @Sql = N'ALTER TABLE Clients.' + QUOTENAME(@TableName) + N' ADD ' + @ColumnName + ' DATETIME NULL;';
        EXEC sp_executesql @Sql;
    END

    FETCH NEXT FROM cursor_table INTO @TableName;
END

CLOSE cursor_table;
DEALLOCATE cursor_table;
```

Oznaczenia Kolumn Do Anonimizacji

Oznaczanie kolumn do anonimizacji, to powinna być odpowiedzialność DB Admina, lub developera by określić które należy anonimizować, a które nie.

```
CREATE PROCEDURE AddAnoniziationPropertyToColumn
@SchemaName NVARCHAR(128),
@TableName NVARCHAR(128),
@ColumnName NVARCHAR(128)
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (
        SELECT *
        FROM INFORMATION_SCHEMA.COLUMNS
        WHERE TABLE_SCHEMA = @SchemaName
        AND TABLE_NAME = @TableName
        AND COLUMN_NAME = @ColumnName
    )
    BEGIN
        EXEC sp_addextendedproperty
            @name = N'Anonimization',
            @value = N'Yes',
            @level0type = N'Schema', @level0name = @SchemaName,
            @level1type = N'Table', @level1name = @TableName,
            @level2type = N'Column', @level2name = @ColumnName;

        PRINT('Anonimization Property Added');
    END
    ELSE
    BEGIN
        PRINT('Column Does Not Exists');
    END
END;
GO

EXEC AddAnoniziationPropertyToColumn @SchemaName = 'Clients', @TableName = 'Customers', @ColumnName = 'FirstName';
EXEC AddAnoniziationPropertyToColumn @SchemaName = 'Clients', @TableName = 'Customers', @ColumnName = 'LastName';
EXEC AddAnoniziationPropertyToColumn @SchemaName = 'Clients', @TableName = 'Customers', @ColumnName = 'Email';

EXEC AddAnoniziationPropertyToColumn @SchemaName = 'Clients', @TableName = 'Addresses', @ColumnName = 'AddressOne';
EXEC AddAnoniziationPropertyToColumn @SchemaName = 'Clients', @TableName = 'Addresses', @ColumnName = 'AddressTwo';
EXEC AddAnoniziationPropertyToColumn @SchemaName = 'Clients', @TableName = 'Addresses', @ColumnName = 'Description';
```

%

Messages

Anonimization Property Added
Anonimization Property Added
Anonimization Property Added
Anonimization Property Added
Anonimization Property Added
Anonimization Property Added
Anonimization Property Added

Dodawanie Procedury Anonimizującej

Dodanie procedury anonimizującej, która korzystając w powyższych ustawień będzie anonimizować wybrane pola.

```
CREATE PROCEDURE AnonimizeData
    @SchemaName NVARCHAR(128),
    @TableName NVARCHAR(128),
    @RelationColumn NVARCHAR(128),
    @IdValue INT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @ColumnName NVARCHAR(128);
    DECLARE @QUERY NVARCHAR(MAX);

    DECLARE cursor_column CURSOR FOR
    SELECT
        col.name AS ColumnName
    FROM
        sys.extended_properties AS SECT
        INNER JOIN sys.columns AS COL
            ON SECT.major_id = COL.object_id AND SECT.minor_id = COL.column_id
        INNER JOIN sys.tables AS TBL
            ON COL.object_id = TBL.object_id
    WHERE
        TBL.name = @TableName
        AND SCHEMA_NAME(TBL.schema_id) = @SchemaName
        AND SECT.name = 'Anonimization'
        AND SECT.Value = 'Yes';

    OPEN cursor_column;
    FETCH NEXT FROM cursor_column INTO @ColumnName;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        PRINT('Processing Column ' + @ColumnName);
        SET @QUERY = 'UPDATE ' + QUOTENAME(@SchemaName) + '.' + QUOTENAME(@TableName) +
            ' SET ' + QUOTENAME(@ColumnName) + ' = ''Anonimized'' +
            ' WHERE ' + QUOTENAME(@ColumnName) + ' IS NOT NULL AND ' + @RelationColumn + ' = ' + @IdValue;
        EXEC sp_executesql @QUERY;

        FETCH NEXT FROM cursor_column INTO @ColumnName;
    END

    CLOSE cursor_column;
    DEALLOCATE cursor_column;
END;
GO

EXEC AnonimizeData @SchemaName = 'Clients', @TableName = 'Customers', @RelationColumn = 'ID', @IdValue = 2005;
SELECT * FROM Clients.Customers WHERE ID = 2005;
```

Results										
	ID	FirstName	LastName	PhoneNumber	Email	Blocked	CreatedAt	UpdatedAt	Title	DeletedAt
1	2005	Anonimized	Anonimized	(771) 935-5601	Anonimized	0	2023-05-07 17:10:24.7833333	2023-05-07 17:10:24.7833333	NULL	NULL

Dodawanie Triggera Anonimizującego

Dodanie triggera, który po usunięciu rekordu zanonimizuje wybrane pola

```
--CREATE TRIGGER TRG_AnonimizeAfterDelete
ON Clients.Customers
INSTEAD OF DELETE
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @ParentTableName NVARCHAR(256), @TableSchemaName NVARCHAR(256), @TableName NVARCHAR(256), @ColumnName NVARCHAR(256), @IdValue INT;
    DECLARE @CurrentDateTime DATETIME;
    DECLARE @RelationColumn NVARCHAR(256);
    DECLARE @QUERY NVARCHAR(MAX);

    SET @CurrentDateTime = GETDATE();

    /* Fetch Customer ID */
    DECLARE CustomerCursor CURSOR FOR SELECT ID FROM deleted;
    OPEN CustomerCursor;
    FETCH NEXT FROM CustomerCursor INTO @IdValue;
    CLOSE CustomerCursor;
    DEALLOCATE CustomerCursor;

    /* Soft Delete And Anonimize Customer */
    UPDATE Clients.Customers SET DeletedAt = @CurrentDateTime FROM Clients.Customers WHERE ID = @IdValue;
    EXEC AnonimizeData @SchemaName = 'Clients', @TableName = 'Customers', @RelationColumn = 'ID', @IdValue = @IdValue;

    /* Soft Delete And Anonimize Other Tables */
    DECLARE cursorLinkedTables CURSOR FOR
        SELECT SchemaName, TableName FROM dbo.GetLinkedTables('Customers', 'Clients');

    OPEN cursorLinkedTables;
    FETCH NEXT FROM cursorLinkedTables INTO @TableSchemaName, @TableName;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        PRINT 'Processing Table: ' + @TableName;

        SET @QUERY = 'UPDATE ' + QUOTENAME(@TableSchemaName) + '.' + QUOTENAME(@TableName) + ' SET DeletedAt = @CurrentDateTime ' +
            'WHERE CustomerID = ' + CAST(@IdValue AS NVARCHAR(18)) + ' ';
        EXEC sp_executesql @QUERY, N'@CurrentDateTime DATETIME', @CurrentDateTime;

        EXEC AnonimizeData @SchemaName = @TableSchemaName, @TableName = @TableName, @RelationColumn = 'CustomerID', @IdValue = @IdValue;
        FETCH NEXT FROM cursorLinkedTables INTO @TableSchemaName, @TableName;
    END

    CLOSE cursorLinkedTables;
    DEALLOCATE cursorLinkedTables;
END
```

46 %

Messages

Processing Table: FirstNames
Processing Table: LastNames
Processing Table: Email
Processing Table: Addresses
Processing Table: AddressOne
Processing Table: AddressTwo
Processing Table: AddressThree
(1 row affected)

Completion Time: 2023-10-22T17:28:40.4201289+01:00

Widoki bazodanowe

Widok Aktywnych Pracowników

Widok ten zwraca pracowników, którzy aktualnie pracują w danej lokacji.

```
USE FoodCourt
GO

CREATE VIEW Restaurants.ActiveEmployees AS
SELECT
    SE.ID AS EmployeeId
    , SE.FirstName + ' ' + SE.LastName AS EmployeeFullName
    , SR.Name AS Role
    , SELR.ValidTo AS WorkingUntill
    , SELR.LocationId AS LocationId
FROM Staff.Employees AS SE
INNER JOIN Staff.EmployeeLocationRoles AS SELR
    ON SE.Id = SELR.EmployeeId
INNER JOIN Staff.Roles AS SR
    ON SELR.RoleId = SR.Id
WHERE
    SELR.ValidFrom <= CAST(GETDATE() AS Date) AND SELR.ValidTo >= CAST(GETDATE() AS Date);

/***** Script for SelectTopNRows command from SSMS *****/
SELECT TOP (1000) [EmployeeId]
    ,[EmployeeFullName]
    ,[Role]
    ,[WorkingUntill]
    ,[LocationId]
FROM [FoodCourt].[Restaurants].[ActiveEmployees]
```

31 %

Results Messages

	EmployeeId	EmployeeFullName	Role	WorkingUntill	LocationId
1	364	Elsie Goldner	Delivery Driver	2024-03-22	141
2	364	Elsie Goldner	Cook	2024-02-21	141
3	365	Andrea Bailey	General Manager	2024-01-08	133

Widok Szczegółowy Lokacji

Widok ten zwraca aktualne dane adresowe lokacji.

```
USE FoodCourt
GO

CREATE VIEW Restaurants.LocationDetails AS
WITH RestaurantPhoneNumbers AS (
    SELECT
        LocationID
        , Number
        , row_number() over (partition by LocationID order by CreatedAt desc) as RowNum
    FROM
        Restaurants.PhoneNumbers
)
SELECT
    RL.ID AS LocationId
    , RL.Name AS LocationName
    , RA.AddressOne + ' ' + RA.AddressTwo AS LocationAddress
    , RA.ZipCode AS LocationZipCode
    , RA.Geom AS AddressGeom
FROM
    Restaurants.Locations AS RL
INNER JOIN
    Restaurants.Addresses AS RA
    ON RL.Id = RA.LocationId AND RA.IsCurrent = 1
INNER JOIN
    RestaurantPhoneNumbers AS RPN
    ON RPN.LocationID = RL.ID AND RPN.RowNum = 1;
```


/***** Script for SelectTopNRows command from SSMS *****/					
SELECT TOP (1000) [LocationId]					
, [LocationName]					
, [LocationAddress]					
, [LocationZipCode]					
, [AddressGeom]					
FROM [FoodCourt].[Restaurants].[LocationDetails]					

81 %					
Results Spatial results Messages					
	LocationId	LocationName	LocationAddress	LocationZipCode	AddressGeom
1	125	Location 0	733 Nadene Dale Suite 464	83890	0xE6100000010C35F0A31AF6D33DC06B990CC7F3934940
2	126	Location 1	576 Ashely Pike Suite 440	48344	0xE6100000010C77BD3445808D4F401477BCC96FF75CC0
3	127	Location 2	9055 Vincenzo Loaf Apt. 487	54924	0xE6100000010C0C94145800274940F1457BBC90BA56C0
4	128	Location 3	470 Debroah Club Apt. 648	69137	0xE6100000010C09DD257156C42B404A0B9755D80438C0
5	129	Location 4	864 Bettyann Lake Apt. 608	84519	0xE6100000010C7C5F5CAAD2E24540A704C4245C8C5040
6	130	Location 5	1306 Hudson Falls Suite 274	72960	0xE6100000010C0B47904AB13B2340A6EF350447EB61C0
7	131	Location 6	315 Chi Extension Apt. 170	44613	0xE6100000010C53910A630B6255C0F8544E7B4A1C5E40

Raportowa Baza Danych

Raportowa baza danych powinna być utrzymywana w formacie ELT, to znaczy że do warstwy ingestion powinny trafiać wszystkie dane z głównej bazy danych. Jako, że rekordy są soft-deletowane, nie musimy przejmować się usuwaniem danych z INGESTION.

Utworzenie bazy danych i schematu ingestion.

```
/* Create FoodCourtReports Database */
IF NOT EXISTS (SELECT * FROM sys.databases WHERE name = N'FoodCourtReports')
BEGIN
    CREATE DATABASE FoodCourtReports;
END
ELSE
BEGIN
    PRINT('Database FoodCourtReports already exists')
END;
GO

/*Create ingestion schema */
USE FoodCourtReports
GO

IF NOT EXISTS (SELECT * FROM sys.schemas WHERE name = N'INGESTION')
BEGIN
    EXEC('CREATE SCHEMA INGESTION;');
END
ELSE
BEGIN
    PRINT('Schema INGESTION already exists')
END;
GO
```

Utworzenie mirrora tabeli w schemacie Ingestion

Mirror, lub kopia tabeli powinna odzwierciedlać tabelę główną. Możemy się jeszcze w przyszłości pokusić o dodanie podobnej anonimizacji jak w procesie usuwania.

The screenshot displays the SQL Server Enterprise Manager interface on the left, showing the 'FoodCourtReports' database and its 'Tables' folder. The main window shows a SQL query editor with a script to create a mirror table. The script includes a function 'CreateMirrorTable' that takes a source schema and table name as input. It checks if the target table exists in the 'INGESTION' schema and creates it if it does not. The script then uses 'EXEC' to call the 'CreateMirrorTable' function with the source schema 'Orders' and table name 'Orders'.

```
DECLARE @CHECK_QUERY NVARCHAR(MAX);
DECLARE @CREATE_QUERY NVARCHAR(MAX);
DECLARE @targetSchema NVARCHAR(256);
SET @targetSchema = N'INGESTION';

SET @CHECK_QUERY = N'SELECT @tableExists =
CASE
    WHEN EXISTS (SELECT * FROM sys.tables WHERE name = @tableName AND SCHEMA_NAME(schema_id) = @targetSchema) THEN 1
    ELSE 0
END;

DECLARE @tableExists BIT;
EXEC sp_executesql @CHECK_QUERY, N'@tableName NVARCHAR(128), @targetSchema NVARCHAR(256), @tableExists BIT OUTPUT', @tableName, @targetSchema, @tableExists OUTPUT;

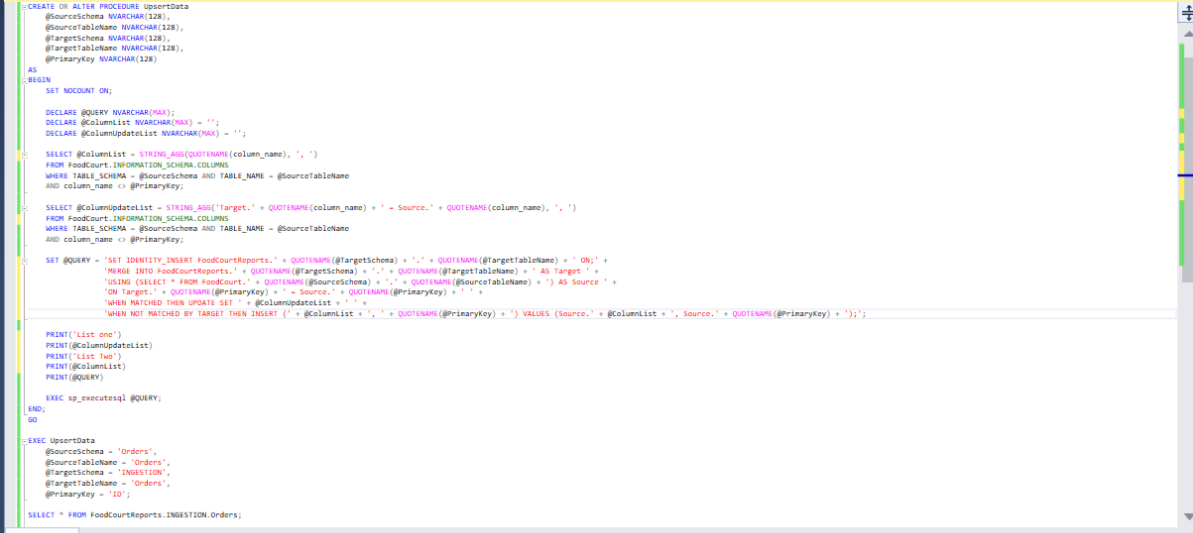
IF @tableExists = 0
BEGIN
    SET @CREATE_QUERY = N'SELECT * INTO FoodCourtReports.' + QUOTENAME(@targetSchema) + '.' + QUOTENAME(@tableName) +
    N' FROM FoodCourt.' + QUOTENAME(@sourceSchema) + '.' + QUOTENAME(@tableName) + ' WHERE 1 = 0;';
    PRINT(@CREATE_QUERY);
    EXEC sp_executesql @CREATE_QUERY;
END
ELSE
BEGIN
    PRINT('Table ' + @tableName + ' already exists');
END
GO

EXEC CreateMirrorTable
@SourceSchema = 'Orders',
@TableName = 'Orders';
```

The Messages pane at the bottom shows the execution progress, indicating that the table 'Orders' has been created in the 'INGESTION' schema.

Przeniesienie danych do Ingestion

W przypadku ELT, możemy przenosić dane 1-1. W dzisiejszych czasach pamięć jest tańsza niż koszty pracy. ELT zajmuje 2x więcej miejsca ale daje nam możliwość naprawiania danych bez sięgania do źródła.



```
--CREATE OR ALTER PROCEDURE UpsertData
--(
--    @SourceSchema NVARCHAR(128),
--    @SourceTableName NVARCHAR(128),
--    @TargetSchema NVARCHAR(128),
--    @TargetTableName NVARCHAR(128),
--    @PrimaryKey NVARCHAR(128)
--)
AS
--BEGIN
--    SET NOCOUNT ON;

--    DECLARE @Query NVARCHAR(MAX);
--    DECLARE @ColumnList NVARCHAR(MAX) = '';
--    DECLARE @ColumnUpdateList NVARCHAR(MAX) = '';

--    SELECT @ColumnList = STRING_AGG(QUOTENAME(column_name), ', ')
--    FROM FoodCourt.INFORMATION_SCHEMA.COLUMNS
--    WHERE TABLE_SCHEMA = @SourceSchema AND TABLE_NAME = @SourceTableName
--    AND column_name <> @PrimaryKey;

--    SELECT @ColumnUpdateList = STRING_AGG('Target.' + QUOTENAME(column_name) + ' = Source.' + QUOTENAME(column_name), ', ')
--    FROM FoodCourt.INFORMATION_SCHEMA.COLUMNS
--    WHERE TABLE_SCHEMA = @SourceSchema AND TABLE_NAME = @SourceTableName
--    AND column_name <> @PrimaryKey;

--    SET @Query = 'SET IDENTITY_INSERT FoodCourtReports, ' + QUOTENAME(@TargetSchema) + '.' + QUOTENAME(@TargetTableName) + ' ON; ' +
--    'MERGE INTO FoodCourtReports, ' + QUOTENAME(@TargetSchema) + '.' + QUOTENAME(@TargetTableName) + ' AS Target ' +
--    'USING (SELECT * FROM FoodCourt.' + QUOTENAME(@SourceSchema) + '.' + QUOTENAME(@SourceTableName) + ') AS Source ' +
--    'ON Target.' + QUOTENAME(@PrimaryKey) + ' = Source.' + QUOTENAME(@PrimaryKey) + ' ' +
--    'WHEN MATCHED THEN UPDATE SET ' + @ColumnUpdateList + ' ' +
--    'WHEN NOT MATCHED BY TARGET THEN INSERT (' + @ColumnList + ', ' + QUOTENAME(@PrimaryKey) + ') VALUES (Source.' + @ColumnList + ', Source.' + QUOTENAME(@PrimaryKey) + ');';

--    PRINT('List one')
--    PRINT(@ColumnUpdateList)
--    PRINT('List two')
--    PRINT(@ColumnList)
--    PRINT(@Query)

--    EXEC sp_executesql @Query;
--END;
--GO

--EXEC UpsertData
--(
--    @SourceSchema = 'Orders',
--    @SourceTableName = 'Orders',
--    @TargetSchema = 'INGESTION',
--    @TargetTableName = 'Orders',
--    @PrimaryKey = 'ID'
--);

SELECT * FROM FoodCourtReports.INGESTION.Orders;
```

ID	LocationID	Status	DeliveryAddressID	CreatedAt	UpdatedAt	DeletedAt	OrderType	RequestedDateTime	CustomerID
1	2006	137	Pending	2023-05-07 17:10:25.3766667	2023-05-07 17:10:25.3766667	NULL	DineIn	2023-05-07 14:40:19.537	NULL
2	2007	130	Ready	2023-05-07 17:10:25.3833333	2023-05-07 17:10:25.3833333	NULL	Pickup	2023-05-05 23:51:27.023	NULL
3	2008	134	Preparing	2023-05-07 17:10:25.3866667	2023-05-07 17:10:25.3866667	NULL	Pickup	2023-05-03 06:53:09.867	NULL
4	2009	142	Pending	2023-05-07 17:10:25.3866667	2023-05-07 17:10:25.3866667	NULL	DineIn	2023-05-05 14:41:30.037	NULL
5	2010	127	Preparing	2023-05-07 17:10:25.3900000	2023-05-07 17:10:25.3900000	NULL	DineIn	2023-05-02 22:41:33.103	NULL
6	2011	131	Ready	2023-05-07 17:10:25.3900000	2023-05-07 17:10:25.3900000	NULL	DineIn	2023-05-06 13:15:55.870	NULL
7	2012	127	Completed	2023-05-07 17:10:25.3933333	2023-05-07 17:10:25.3933333	NULL	Pickup	2023-05-05 18:45:26.923	NULL
8	2013	126	Pending	2023-05-07 17:10:25.3933333	2023-05-07 17:10:25.3933333	NULL	Pickup	2023-05-03 12:21:10.503	NULL
9	2014	144	Pending	2023-05-07 17:10:25.3966667	2023-05-07 17:10:25.3966667	NULL	DineIn	2023-05-05 18:43:50.037	NULL
10	2015	130	Pending	2023-05-07 17:10:25.4000000	2023-05-07 17:10:25.4000000	NULL	Pickup	2023-05-03 12:21:10.503	2023-05-03 12:21:10.503

Automatyzacja procesu

Procesy takie powinny być wykonywane automatycznie, bez ingerencji człowieka.



```
--DECLARE @JobID BINARY(16);
--DECLARE @JobName NVARCHAR(128);
--SET @JobName = @JobName;

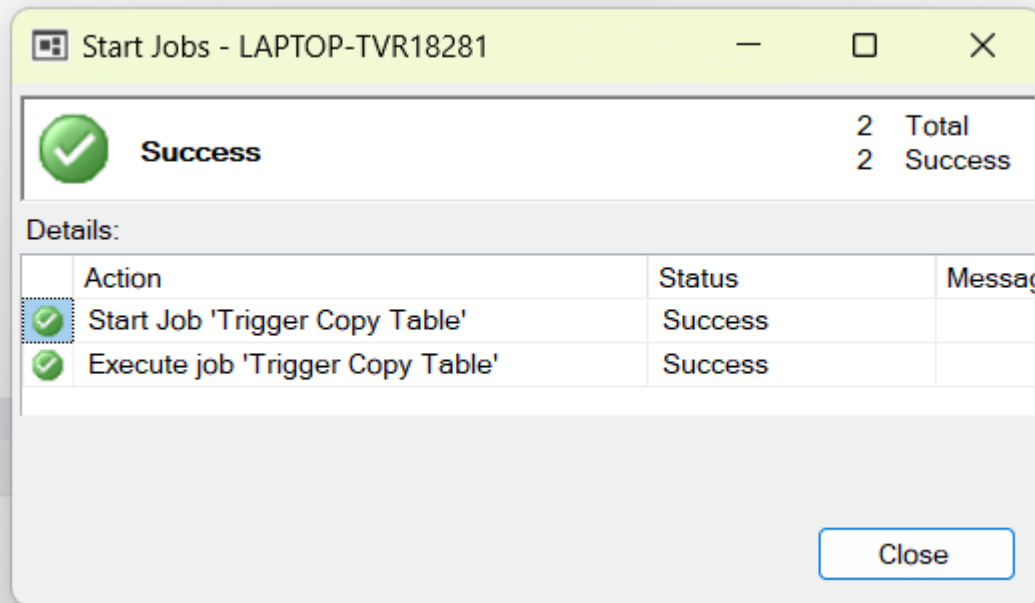
--EXEC msdb.dbo.sp_add_job
--(
--    @job_name = N'Trigger Copy Table',
--    @enabled = 1,
--    @notify_level_eventlog = 0,
--    @notify_level_email = 0,
--    @notify_level_netsend = 0,
--    @notify_level_page = 0,
--    @delete_level = 0,
--    @description = N'Trigger CreateMirrorTable every 24 h.',
--    @category_name = N'[Uncategorized (Local)]',
--    @owner_login_name = N'sa',
--    @job_id = @JobID OUTPUT;
--);

--EXEC msdb.dbo.sp_add_jobstep
--(
--    @job_id = @JobID,
--    @step_name = N'Trigger Copy Table',
--    @subsystem = N'TSQL',
--    @command = N'EXEC FoodCourtReports.dbo.CreateMirrorTable
--    (
--        @SourceSchema = ''Orders'',
--        @TableName = ''Orders''
--    );',
--    @cmdexec_success_code = 0,
--    @on_success_action = 1,
--    @on_fail_action = 2,
--    @retry_attempts = 0,
--    @retry_interval = 0,
--    @database_name = N'FoodCourtReports',
--    @flags = 0;
--);

--DECLARE @ScheduleID INT;
--EXEC msdb.dbo.sp_add_jobschedule
--(
--    @job_id = @JobID,
--    @name = N'Every 24h',
--    @enabled = 1,
--    @freq_type = 4,
--    @freq_interval = 1,
--    @freq_subday_type = 1,
--    @freq_subday_interval = 0,
--    @freq_relative_interval = 0,
--    @freq_recurrence_factor = 1,
--    @active_start_date = 20231226,
--    @active_end_date = 99991231,
--    @active_start_time = 010000,
--    @active_end_time = 235959,
--    @schedule_id = @ScheduleID OUTPUT;
--);

--EXEC msdb.dbo.sp_attach_schedule
--(
--    @job_id = @JobID,
--    @schedule_id = @ScheduleID;
--);

SELECT @JobID = job_id FROM msdb.dbo.sysjobs WHERE (name = @JobName);
EXEC msdb.dbo.sp_add_jobserver @job_id = @JobID, @server_name = N'local';
```



- FoodCourt
- FoodCourtReports
 - Database Diagrams
 - Tables
 - System Tables
 - FileTables
 - External Tables
 - Graph Tables
 - INGESTION.Orders

Podsumowanie i wnioski

Do przygotowania projektu, wykorzystałem poprzedni projekt w którym już zawarte są klucze obce, widoki i schematy bazy danych. Projekt ten załączam jako drugi plik.

W powyższym ćwiczeniu, wiele operacji powinno się wykonać wielokrotnie dla wielu tabel. W skrypcie zamieszczam jednak tylko pojedyncze wykonanie każdego z nich, by nie tworzyć sztuki dla sztuki i dziesiątek kartek do przejrzenia.

Bezpieczeństwo dostępu do danych, jest w dzisiejszych czasach jest jednym z najważniejszych obowiązków inżyniera, ustawienie wielu różnych ról, loginów i kont może nie tylko nam pomóc chronić dane, ale i wysledzić co poszło nie tak w procesie w przypadku jakiegoś wycieku lub innej formy ataku.

Jeśli chodzi o sekcje pierwszą, z regułami dotyczącymi danych, Rule są ciekawą lecz wydaje mi się trochę przestarzałą opcją. Zdecydowanie wole opierać się na Constraintach lub Enum'ach dla prostej walidacji danych i na Triggerach dla bardziej skomplikowanej logiki biznesowej.

Skrypty są ciekawym i potężnym rozwiązaniem, nie wiem jednak czy pokusiłbym się o szerokie ich stosowanie w komercyjnym projekcie, a jeśli już to na pewno tylko do najważniejszych dla biznesu operacjach, takich jak anonimizacja danych, gdzie błąd na poziomie kodu może kosztować firmę wysokie odszkodowania.

I ostatnie, czyli aplikacja bazodanowa do drugiej bazy danych. Nie znam na tyle możliwości MS SQL, żeby stwierdzić czy rozwiązanie oparte na takich procedurach jest optymalne, ale w prawdziwym życiu wolałbym jednak skorzystać z Kafki jako message broker'a lub SISS w środowisku Microsoft. Ludzie tworzący te rozwiązania zjedli na tym zęby tam gdzie mi rosną dopiero mleczaki. Tym jednak, było to ciekawe zadanie.

Skrypt

Z racji wielkości plików generujących bazę pliki są załączone oddzielnie i dostępne na [GitHubie](#).