Tytuł projektu:

*„Uproszczony system zarządzania Kopalnią Węgla Kamiennego”*

**Kuczu**

# Spis treści

# 1. Wstęp

## 1.1. Cel i opis projektu i schematów

Celem projektu jest zaprojektowanie i zaimplementowanie uproszczonej bazy danych zarządzania Kopalnią Węgla Kamiennego. W projekcie skupiono się na podstawowym zarządzaniu pracownikami oraz na aspektach wydobywczych surowca. Stworzona baza wykorzystuje Microsoft SQL Server 2016.

Baza została podzielona na cztery główne części – schematy.

*Schemat pierwszy **„Employee”*** - „Pracownicy” – pierwsza część bazy odpowiada za proste zarządzanie pracownikami. Znajdują się w nim tabele, funkcje, procedury składowe które pozwalają na przechowywanie danych o pracownikach, nadawane zdeklarowanych stanowisk, miejsca pracy oraz przełożonych. Ponadto w bazie zapisywane są informacje o urlopach oraz przepracowanych godzinach.

*Schemat drugi **„Longwall”*** – „Ściana *(węglowa)*” oraz *schemat trzeci – **„Mineface”*** – „Przodek” - te części bazy pod względem funkcjonalności są zbliżone do siebie. Odpowiadają one za aspekt wydobywczy kopalni, zapewniają to odpowiednie tabele, funkcje, procedury składowe i wyzwalacze. W tej części bazy gromadzone są podstawowe informacje dotyczące w/w wyrobisk, przechowywane są także dzienne raporty z postępów prac i wydobycia oraz dziennik zużytego sprzętu jak i dziennik zamówień brakującego.

Ostatnim *schematem jest **„Sales** (department)”* – „Dział sprzedaży”. Również w jego skład wchodzą odpowiednie tabele, funkcje, procedury składowe i wyzwalacze. W tym schemacie skupiono się na bardzo podstawowej możliwości kupowania węgla poprzez firmy zewnętrzne. Baza pozwala na zapis informacji o kupującym, historii transakcji oraz aktualnym stanie hołdy węglowej.

Rozdziały od drugiego do piątego opisują wraz z kodem stworzone tabele, funkcje, procedury składowane oraz wyzwalacze. W rozdziale szóstym zaprezentowano przykładowe zapytania SQL możliwe do wykonania na bazie. Ostatni rozdział to podsumowanie i wnioski.

## 1.2. Przyjęte ograniczenia projektowe

W związku z tym, że funkcjonowanie kopalni to bardzo złożony wielowarstwowy proces obejmujący wiele działów i dyscyplin zawodowych w projekcie tym zdecydowano się na pewne uproszczenia implementacyjne projektu związanych z logiką działania bazy.

Projekt ten implementuje zarządzanie pracownikami – implementacja głównie z myślą o pionie górniczym, ogranicza się tylko do zapisania podstawowych danych jak m.in. imię, nazwisko, wiek, data urodzenia pesel. Każdy pracownik dostaje, jak w prawdziwym zakładzie górniczym, swój unikalny numer indentyfikacyjny. Baza posiada tabele zapewniające przydzielenie pracownikowi zwierzchnika, przyjęto, że w danym okresie czasu dany pracownik może mieć tylko jednego zwierzchnika. Baza umożliwia również przechowywaniu danych o urlopach pracowników oraz przechowuje godziny oraz daty rozpoczęcia i skończenia pracy.

Kolejną częścią projektu jest implementacja struktur bazy danych umożliwiających zarządzanie dwiema strukturami eksploatacyjnymi kopalni – ścianą eksploatacyjną oraz przodkiem. W projekcie przyjęto, że eksploatowany pokład węgla jest idealnie foremny oraz nie posiada zagrożeń naturalnych. System rozliczania z eksploatacji – składanie raportów z wydobycia, postępu i zużytych środkach jest tylko raz w ciągu dnia –

w domyśle 23:59. Środki/przedmioty jakie można wykorzystać w kopalni są określone w jednej z tabeli – w obrębie kopalni są one nielimitowane. Ponadto podczas składania raportu dotyczącego zużycia danych środków w wyrobisku górniczym po przekroczeniu pewnej minimalnej ilości automatycznie jest składane zamówienie na brakujące przedmioty – przyjęto, że są one natychmiast dostarczane.

Ostatnią zaimplementowaną częścią jest bardzo uproszczony dział sprzedaży wydobytego węgla. Baza posiada dwie tabele – jedna z nich przechowuje podstawowe dane o klientach oraz nadaje im indywidualny numer, druga z kolei to tabela przechowująca dane dotyczące zamówień – wiąże klienta z zamówieniem, nadaje każdemu zamówieniu numer, datę otrzymania i realizacji zamówienia oraz ilość zamówionego węgla jak i cenę jednostkową. Przyjęto, że jeśli kopalnia jest w stanie zrealizować zamówienie od razu po wpłynięciu – tak też się dzieje, w przeciwnym wypadku zamówienia są priorytetowane względem czasu zamówienia. Ostatnim elementem tej części projektu jest tabela przechowująca dane o hołdzie węglowej, w projekcie przyjęto istnienie tylko jednej hołdy. Wartość aktualizowana jest w momencie złożenia dziennego raportu wydobywczego – dodanie wydobytego węgla, lub zrealizowaniu zamówienia – odjęcie konkretnej wartości. Nie prowadzi się bezpośredniej historii aktualizacji hołdy.

## 2. Opis tabel

Jedyną tabelą która nie należy do żadnego schematu jest tabela „Items”:

```
CREATE TABLE Items(
    ItemID INT IDENTITY(1,1),
    ItemName NVARCHAR(150) NOT NULL,
    ItemPrice MONEY CHECK(ItemPrice > 0),
    PRIMARY KEY(ItemID)
)
GO
```

Tabela ta przechowuje informacje o przedmiotach które można użyć na kopalni. Kluczem głównym jest numer indentyfikacyjny przedmiotu.

### 2.1. Employee – Pracownicy

```
CREATE TABLE Employees.Positions(
    PositionID INT,
    PositionName NVARCHAR(50) NOT NULL,
    PRIMARY KEY(PositionID)
)
GO
```

Tabela „Emploees.Positions” służy do przechowywania pozycji – stanowisk na jakich mogą pracować zatrudnieni pracownicy. Kluczem głównym jest „PositionID”.

```
CREATE TABLE Employees.BasicInfo(
    EmployeeID INT IDENTITY(1,1),
    PositionID INT NOT NULL,
    FirstName nvarchar(30) NOT NULL,
    Surname nvarchar(30) NOT NULL,
    Gender NCAHR(1) NOT NULL CHECK(Gender = 'M' OR Gender = 'F'),
    PESEL nvarchar(11) NOT NULL CHECK(LEN(PESEL) = 11),
    TelephoneNumber INT,
    BirthDate DATE NOT NULL,
    HireDate DATE NOT NULL,
    DismissalDate DATE DEFAULT NULL,
    UNIQUE(PESEL),
    PRIMARY KEY(EmployeeID),
    FOREIGN KEY(PositionID) REFERENCES Employees.Positions(PositionID)
)
GO
```

Tabela „Employees.BasicInfo” przechowuje podstawowe informacje o zatrudnionych pracownikach. Każdy pracownik otrzymuje  swój indywidualny numer – „EmployeeID”, jest on kluczem głównym tabeli jest autoinkrementowany. Kluczami obcymi jest „PositionID” który odpowiada numerowi stanowiska w tabeli „Emploees.Positions”. Płeć pracownika określana jest poprzez literke „M” (Male – mężczyzna) lub „F” (Female – kobieta).

```
CREATE TABLE Employees.Vacation(
    EmployeeID INT,
    StartDate DATE,
    EndDate DATE NOT NULL,
    PRIMARY KEY(EmployeeID, StartDate),
    FOREIGN KEY(EmployeeID) REFERENCES Employees.BasicInfo(EmployeeID)
)
GO
```

Tabela „Employees.Vacation” przechowuje informacje na temat urlopów pracowników. Kluczem głównym jest klucz złożony z „EmployeeID” (który jest też kluczem obcym) oraz „StartDate” – dany pracownik nie może wziąć więcej niż jednego urlopu w danym dniu. „EndDate” nie może być puste, założono, że urlop brany będzie na konkretny okres.

```
1.  CREATE TABLE Employees.Superior(
2.      EmployeeID INT,
3.      SuperiorID INT,
4.      StartDate DATE,
5.      PRIMARY KEY(EmployeeID, StartDate),
6.      FOREIGN KEY(EmployeeID) REFERENCES Employees.BasicInfo(EmployeeID),
7.      FOREIGN KEY(SuperiorID) REFERENCES Employees.BasicInfo(EmployeeID)
8.  )
9.  GO
```

Tabela „Employees.Superior” zawiera informacje o przełożonych i dacie kiedy dany przełożony zaczyna zwierzchnictwo. Przyjęto, że dany pracownik może mieć w danym momencie tylko jednego przełożonego. „StartDate” jest również końcem zwierzchnictwa poprzedniego przełożonego, jeśli taki był.

```
1.  CREATE TABLE Employees.WorkHours(
2.      EmployeeID INT,
3.      StartDate SMALLDATETIME,
4.      EndDATE SMALLDATETIME,
5.      PRIMARY KEY(EmployeeID, StartDate),
6.      FOREIGN KEY(EmployeeID) REFERENCES Employees.BasicInfo(EmployeeID)
7.  )
8.  GO
```

Tabela „Employees.WorkHours” zawiera informacje o czasie pracy pracownika. Kluczem głównym jest „EmployeeID” oraz „StartDate”.

```
1.  CREATE TABLE Employees.EmployeeDepartement(
2.      ID INT IDENTITY(1,1),
3.      LongwallID INT,
4.      MineFaceID INT,
5.      EmployeeID INT NOT NULL,
6.      WorkStartDate DATE NOT NULL,
7.      WorkStopDate DATE DEFAULT NULL,
8.      PRIMARY KEY(ID),
9.      FOREIGN KEY(LongwallID) REFERENCES Longwall.BasicInfo(LongwallID),
10.     FOREIGN KEY(MineFaceID) REFERENCES MineFace.BasicInfo(MineFaceID),
11.     FOREIGN KEY(EmployeeID) REFERENCES Employees.BasicInfo(EmployeeID),
12.     CONSTRAINT CK_ValidateDate CHECK(WorkStartDate > WorkStopDate)
13. )
14. GO
```

Tabela „Employees.EmployeeDepartement” zawiera informacje o ścianie bądź przodku oraz okresie w jakim dany pracownik tam pracuje.

## 2.2. **Longwall – Ściana (eksploatacyjna)**

```
1.  CREATE TABLE Longwall.BasicInfo(
2.      LongwallID INT IDENTITY(1,1),
3.      LongwallName NVARCHAR(30) UNIQUE NOT NULL,
4.      LongwallLong INT NOT NULL CHECK(LongwallLong > 0), -- wybieg
5.      LongwallWide INT NOT NULL CHECK(LongwallWide > 0), -- szerokosc
6.      LongwallThick INT NOT NULL CHECK(LongwallThick > 0), -- wysokosc
7.      StartDate DATE NOT NULL,
8.      EndDate DATE DEFAULT NULL,
9.      PRIMARY KEY(LongwallID)
10. )
```

```
GO
```

Tabela „Longwall.BasicInfo” zawiera podstawowe informacje odnośnie ściany eksploatacyjnej, jak unikalny autoinkrementowany numer, unikalną nazwę oraz informacje o długości wybiegu, szerokości, wysokości, dacie rozpoczęcia i zakończenia biegu ściany.

```
CREATE TABLE Longwall.DailyRaport(
    RaportID INT IDENTITY(1,1),
    EmployeeID INT NOT NULL,
    LongwallID INT NOT NULL,
    RaportDate DATE NOT NULL,
    ExcavatedCoalQuantityInKg INT NOT NULL CHECK(ExcavatedCoalQuantityInKg >= 0), -- urobek kg
    WallAdvanceInMeters INT NOT NULL CHECK(WallAdvanceInMeters >= 0), -- postep m
    Comments NVARCHAR(MAX) DEFAULT NULL,
    PRIMARY KEY(RaportID),
    UNIQUE(RaportDate, LongwallID),
    FOREIGN KEY(LongwallID) REFERENCES Longwall.BasicInfo(LongwallID),
    FOREIGN KEY(EmployeeID) REFERENCES Employees.BasicInfo(EmployeeID)
)
GO
```

Tabela „Longwall.DailyRaport” służy do przechowywania informacji o dziennym raporcie z prowadzonych robót eksploatacyjnych. Przechowuje informacje takie jak wydobyty węgiel w kilogramach, postęp ściany oraz komentarz osoby wypełniającej. Przyjęto, że tylko jeden raport na dzień może zostać dodany dla każdej ściany.

```
CREATE TABLE Longwall.ActualWarehouseStatus(
    LongwallID INT NOT NULL,
    ItemID INT NOT NULL,
    AvailableAmount INT NOT NULL,
    MinimumAmount INT NOT NULL CHECK(MinimumAmount > 0),
    PRIMARY KEY(LongwallID, ItemID),
    FOREIGN KEY(LongwallID) REFERENCES Longwall.BasicInfo(LongwallID),
    FOREIGN KEY(ItemID) REFERENCES Items(ItemID)
)
GO
```

Tabela „Longwall.ActualWarehouseStatus” przechowuje informacje takie jak indywidualny numer przedmiotu przypisany do każdej ściany – jakie przedmioty są dostępne. Ich aktualny stan oraz minimalna ilość po której należy zamówić brakujące sztuki.

```
CREATE TABLE Longwall.UsedResources(
    RaportID INT,
    ItemID INT NOT NULL,
    ItemUsedAmount INT NOT NULL CHECK(ItemUsedAmount > 0),
    PRIMARY KEY(RaportID, ItemID),
    FOREIGN KEY(RaportID) REFERENCES Longwall.DailyRaport(RaportID),
    FOREIGN KEY(ItemID) REFERENCES Items(ItemID)
)
GO
```

Tabela „Longwall.UsedResources” zawiera informacje o wykorzystanych przedmiotach w ramach danego dziennego raportu.

```
CREATE TABLE Longwall.ItemsOrder(
    LongwallID INT NOT NULL,
    ItemID INT NOT NULL,
    OrderedItemAmount INT NOT NULL CHECK(OrderedItemAmount > 0),
    OrderDate DATE NOT NULL,
    PRIMARY KEY(LongwallID, ItemID, OrderDate),
    FOREIGN KEY(LongwallID) REFERENCES Longwall.BasicInfo(LongwallID),
```

```
8.      FOREIGN KEY(ItemID) REFERENCES Items(ItemID)
9.  )
10. GO
```

Tabela „Longwall.ItemsOrder” przechowuje informacje o dacie zamówienia przedmiotu, jego ilości. Przyjęto, że dla danej ściany w danym dniu tylko może być jedno zamówienie danego przedmiotu.

## 2.3. **Mineface – Przodek**

```
1.  CREATE TABLE MineFace.BasicInfo(
2.      MineFaceID INT IDENTITY(1,1),
3.      MineFaceName NVARCHAR(30) UNIQUE NOT NULL,
4.      MineFaceLong INT NOT NULL CHECK(MineFaceLong > 0), -- dlugosc
5.      MineFaceThick INT NOT NULL CHECK(MineFaceThick > 0), -- wysokosc
6.      MineFaceSectionalArea INT NOT NULL CHECK(MineFaceSectionalArea > 0), -- pole przekroju
7.      MineFaceTimber INT NOT NULL,
8.      StartDate DATE NOT NULL,
9.      EndDate DATE DEFAULT NULL,
10.     PRIMARY KEY(MineFaceID),
11.     FOREIGN KEY(MineFaceTimber) REFERENCES Items(ItemID)
12. )
13. GO
```

Tabela „MineFace.BasicInfo” przeznaczona jest do przechowywania podstawowych informacji o przodku, takich jak unikalna nazwa, unikalny autoinkrementowany numer oraz jej długości wysokości, polu przekroju i planowanej podziałce obudowy.

```
1.  CREATE TABLE MineFace.DailyRaport(
2.      RaportID INT IDENTITY(1,1),
3.      EmployeeID INT NOT NULL,
4.      MineFaceID INT NOT NULL,
5.      RaportDate DATE NOT NULL,
6.      TimberScaleInMeters INT NOT NULL CHECK(TimberScaleInMeters >= 0), -- podzialka m
7.      ExcavatedCoalQuantityInKg INT NOT NULL CHECK(ExcavatedCoalQuantityInKg >= 0), -- urobek kg
8.      MineFaceAdvanceInMeters INT NOT NULL CHECK(MineFaceAdvanceInMeters >= 0), -- postep m
9.      Comments NVARCHAR(MAX) DEFAULT NULL,
10.     PRIMARY KEY(RaportID),
11.     UNIQUE(RaportDate, MineFaceID),
12.     FOREIGN KEY(MineFaceID) REFERENCES MineFace.BasicInfo(MineFaceID),
13.     FOREIGN KEY(EmployeeID) REFERENCES Employees.BasicInfo(EmployeeID)
14. )
15. GO
```

Tabela „MineFace.DailyRaport”, podobnie jak tabela „Longwall.DailyRaport” służy do przechowywania informacji o dziennych raportach z postępu prac. Dodatkowo zawiera pole na podziałkę obudowy jaka musiała być zastosowana na danej zmianie.

```
1.  CREATE TABLE MineFace.ActualWarehouseStatus(
2.      MineFaceID INT NOT NULL,
3.      ItemID INT NOT NULL,
4.      AvailableAmount INT NOT NULL,
5.      MinimumAmount INT NOT NULL,
6.      PRIMARY KEY(MineFaceID, ItemID),
7.      FOREIGN KEY(MineFaceID) REFERENCES MineFace.BasicInfo(MineFaceID),
8.      FOREIGN KEY(ItemID) REFERENCES Items(ItemID)
9.  )
10. GO
```

Tabela „MineFace.ActualWarehouseStatus” pełni taką samą rolę jak tabela „Longwall.ActualWarehouseStatus”, z tą różnicą, że odnosi się do przodka.

```
CREATE TABLE MineFace.UsedResources(
    RaportID INT,
    ItemID INT NOT NULL,
    ItemUsedAmount INT NOT NULL CHECK(ItemUsedAmount > 0),
    PRIMARY KEY(RaportID, ItemID),
    FOREIGN KEY(RaportID) REFERENCES MineFace.DailyRaport(RaportID),
    FOREIGN KEY(ItemID) REFERENCES Items(ItemID)
)
GO
```

Tabela „MineFace.UsedResources" pełni taką samą rolę jak odpowiadająca jej tabela w schemacie „Longwall".

```
CREATE TABLE MineFace.ItemsOrder(
    MineFaceID INT NOT NULL,
    ItemID INT NOT NULL,
    OrderedItemAmount INT NOT NULL CHECK(OrderedItemAmount > 0),
    OrderDate DATE NOT NULL,
    PRIMARY KEY(MineFaceID, ItemID, OrderDate),
    FOREIGN KEY(MineFaceID) REFERENCES MineFace.BasicInfo(MineFaceID),
    FOREIGN KEY(ItemID) REFERENCES Items(ItemID)
)
```

Tabela „MineFace.ItemsOrder" pełni taką samą rolę jak odpowiadająca jej tabela w schemacie „Longwall".

## 2.4. **Sales (department) – Dział sprzedaży**

```
CREATE TABLE Sales.CoalPileStatus( -- holda
    CoalPileID INT,
    LastModificationDATE DATE,
    CurrentCoalQuantity INT CHECK(CurrentCoalQuantity >= 0),
    PRIMARY KEY(CoalPileID)
)
GO
```

Tabela „Sales.CoalPileStatus" zawiera informacje o aktualnym stanie hołdy. Każde zamówienie i wydobycie węgla aktualizuje status hołdy.

```
CREATE TABLE Sales.Customers(
    CustomerID INT IDENTITY(1,1),
    CompanyName NVARCHAR(30),
    Telephone INT,
    City NVARCHAR(30),
    Adress NVARCHAR(30),
    PostalCode NVARCHAR(30),
    Country NVARCHAR(30),
    PRIMARY KEY(CustomerID)
)
GO
```

Tabela „Sales.Customers" jest to prosta tabela przechowująca podstawowe informacje o klientach kopalni.

```
CREATE TABLE Sales.Orders(
    OrderID INT IDENTITY(1,1),
    OrderDate SMALLDATETIME NOT NULL,
    RecipeDate SMALLDATETIME  DEFAULT NULL,
    CustomerID INT NOT NULL,
    QuantityInKg INT NOT NULL CHECK(QuantityInKg > 0),
    UnitPriceInZl money NOT NULL DEFAULT 100 CHECK(UnitPriceInZl >= 0),
    PRIMARY KEY(OrderID),
    FOREIGN KEY(CustomerID) REFERENCES Sales.Customers(CustomerID)
)
GO
```

Tabela „Sales.Orders” przechowuje informacje o zamówieniach realizowanych bądź zrealizowanych przez kopalnie. Każde zamówienie ma indywidualny numer, datę zamówienia oraz datę zrealizowana zamówienia, jego cenę oraz ilość kupowanego węgla.

# 3. Opis funkcji

Zaprezentowane funkcje w tym rozdziale są funkcjami pełniące role pomocnicze do np. procedur składowanych lub wyzwalaczy, lecz, pod pewnymi warunkami, mogą być użyte autonomicznie.

```
1.  CREATE FUNCTION IsItemWithGivenIdExists(@ItemID INT)
2.  RETURNS BIT
3.  BEGIN
4.      IF EXISTS(SELECT ItemID
5.                  FROM Items
6.                  WHERE ItemID = @ItemID
7.              )
8.      BEGIN
9.          RETURN 1
10.     END
11.
12.     RETURN 0
13. END
14. GO
```

Funkcja „IsItemWithGivenIdExists” nie należy do żadnego schematu, na wejście otrzymuje liczbę – ID przedmiotu, zwraca ‘true’ jeśli przedmiot o podanym ID istnieje, lub ‘false’ jeśli nie.

## 3.1. Employee – Pracownicy

```
1.  CREATE FUNCTION Employees.IsEmployeeWithGivenIDExists(@EmployeeID INT)
2.  RETURNS BIT
3.  BEGIN
4.      IF EXISTS(SELECT FirstName FROM Employees.BasicInfo WHERE EmployeeID = @EmployeeID)
5.      BEGIN
6.          RETURN 1
7.      END
8.
9.      RETURN 0
10. END
11. GO
```

Funkcja „Employees.IsEmployeeWithGivenIDExists” na wejście otrzymuje liczbę – ID pracownika, zwraca ‘true’ jeśli pracownik o podanym ID istnieje, lub ‘false’ jeśli nie.

```
1.  CREATE FUNCTION Employees.IsEmployeeAtVacation(@EmployeeID INT)
2.  RETURNS BIT
3.  BEGIN
4.      DECLARE @EmployeeVacationDate AS DATE
5.
6.      SET @EmployeeVacationDate = (SELECT MAX(EndDate)
7.              FROM Employees.Vacation
8.              WHERE EmployeeID = @EmployeeID)
9.
10.     IF @EmployeeVacationDATE IS NULL OR @EmployeeVacationDate < GETDATE()
11.     BEGIN
12.         RETURN 0
13.     END
14.
15.     RETURN 1
16. END
17. GO
```

Funkcja „Employees.IsEmployeeAtVacation” na wejście otrzymuje liczbę – ID pracownika, zwraca ‘true’ jeśli pracownik w momencie sprawdzania jest na urlopie, ‘false’ w przeciwnym wypadku.

```sql
CREATE FUNCTION Employees.IsEmployeeAtVacationWithGivenPeriod(@EmployeeID INT, @StartDate DATE, @EndDate DATE)
RETURNS BIT
BEGIN
    IF EXISTS((SELECT TOP(1) StartDate
            FROM Employees.Vacation
            WHERE EmployeeID = @EmployeeID AND
                (StartDate BETWEEN @StartDate AND @EndDate) AND
                (EndDate BETWEEN @StartDate AND @EndDate))
            )
    BEGIN
        RETURN 1
    END

    RETURN 0
END
GO
```

Funkcja „Employees.IsEmployeeAtVacationWithGivenPeriod” na wejście otrzymuje ID pracownika i dwie daty, jeśli pracownik o takim ID istnieje i w całym podanym okresie był na urlopie zwraca ‘true’, w przeciwnym razie ‘false’.

```sql
CREATE FUNCTION Employees.IsEmplyeeAtRetired(@EmployeeID INT)
RETURNS BIT
BEGIN
    DECLARE @EmployeeRetireDate AS DATE

    SET @EmployeeRetireDate = (SELECT DismissalDate
            FROM Employees.BasicInfo
            WHERE EmployeeID = @EmployeeID)

    IF @EmployeeRetireDate IS NULL OR @EmployeeRetireDate < GETDATE()
    BEGIN
        RETURN 0
    END

    RETURN 1
END
GO
```

Funkcja „Employees.IsEmplyeeAtRetired” przyjmuje na wejściu ID pracownika, jeśli pracownik istnieje i w momencie sprawdzania jest na emeryturze funkcja zwraca ‘true’, w przeciwnym razie ‘false’.

```sql
CREATE FUNCTION Employees.EmployeeOvertime(@StartDate SMALLDATETIME, @EndDate SMALLDATETIME, @EmployeeID INT)
RETURNS INT
BEGIN
    DECLARE @SumOfHours AS INT

    SET @SumOfHours = (SELECT (SUM(DATEDIFF(hour, StartDate, EndDate) - 8))
            FROM Employees.WorkHours
            WHERE (EmployeeID = @EmployeeID AND
                (StartDate BETWEEN @StartDate AND @EndDate) AND
                (EndDate BETWEEN @StartDate AND @EndDate) AND
                DATEDIFF(hour, StartDate, EndDate) > 8))

    IF @SumOfHours IS NULL
    BEGIN
        RETURN 0
    END

    RETURN @SumOfHours
END
GO
```

Funkcja „Employees.EmployeeOvertime” przyjmuje dwie daty i ID pracownika jeśli pracownik istnieje i ma jakieś nadgodziny (praca ponad 8 godzin podczas jednej zmiany) w podanym okresie to funkcja zwraca sumę przepracowanych nadgodzin w przeciwnym razie 0.

```
1.  CREATE FUNCTION Employees.EmployeeWorkHours(@StartDate SMALLDATETIME, @EndDate SMALLDATETIME, @Empl
    oyeeID INT)
2.  RETURNS INT
3.  BEGIN
4.      DECLARE @SumOfHours AS INT
5.
6.      SET @SumOfHours = (SELECT SUM(DATEDIFF(hour, StartDate, EndDate))
7.              FROM Employees.WorkHours
8.              WHERE (EmployeeID = @EmployeeID AND
9.                  (StartDate BETWEEN @StartDate AND @EndDate) AND
10.                 (EndDate BETWEEN @StartDate AND @EndDate)) )
11.
12.     IF @SumOfHours IS NULL
13.     BEGIN
14.         RETURN 0
15.     END
16.
17.     RETURN @SumOfHours
18. END
19. GO
```

Funkcja „Employees.EmployeeWorkHours” przyjmuje dwie daty i ID pracownika jeśli pracownik istnieje i ma przepracowane godziny w podanym okresie to funkcja zwraca sumę przepracowanych godzin w przeciwnym razie 0.

```
1.  CREATE FUNCTION Employees.EmployeeWorkHoursFromStart(@EmployeeID INT)
2.  RETURNS INT
3.  BEGIN
4.      DECLARE @SumOfHours AS INT
5.
6.      SET @SumOfHours = (SELECT SUM(DATEDIFF(hour, StartDate, EndDate))
7.              FROM Employees.WorkHours
8.              WHERE (EmployeeID = @EmployeeID) )
9.
10.     IF @SumOfHours IS NULL
11.     BEGIN
12.         RETURN 0
13.     END
14.
15.     RETURN @SumOfHours
16. END
17. GO
```

Funkcja „Employees.EmployeeWorkHoursFromStart” przyjmuje ID pracownika jeśli pracownik istnieje i ma przepracowane godziny od początku zatrudnienia to funkcja zwraca sumę przepracowanych godzin w przeciwnym razie 0.

```
1.  CREATE FUNCTION Employees.EmployeeVacationDays(@EmployeeID INT)
2.  RETURNS INT
3.  BEGIN
4.      DECLARE @SumOfDays AS INT
5.
6.      SET @SumOfDays = (SELECT SUM(DATEDIFF(day, StartDate, EndDate))
7.              FROM Employees.Vacation
8.              WHERE EmployeeID = @EmployeeID)
9.
10.     IF @SumOfDays IS NULL
11.     BEGIN
12.         RETURN 0
13.     END
```

```
14.
15.     RETURN @SumOfDays
16. END
17. GO
```

Funkcja „Employees.EmployeeVacationDays” przyjmuje ID pracownika jeśli pracownik istnieje i brał urlop od początku zatrudnienia to funkcja zwraca sumę dni na urlopie, w przeciwnym razie 0.

```
1.  CREATE FUNCTION Employees.IsPeselContainsOnlyDigits(@PESEL NVARCHAR(11))
2.  RETURNS BIT
3.  BEGIN
4.      IF EXISTS(SELECT 1 WHERE @PESEL NOT LIKE '%[^0-9]%')
5.      BEGIN
6.          RETURN 1
7.      END
8.
9.      RETURN 0
10. END
11. GO
```

Funkcja „Employees.IsPeselContainsOnlyDigits” przyjmuje ciąg 11 znaków, jeśli ciąg ten zawiera tylko liczby zwraca ‘true’ w przeciwny wypadku ‘false’.

```
1.  CREATE FUNCTION Employees.IsPeselChecksumValid(@PESEL NVARCHAR(11))
2.  RETURNS BIT
3.  BEGIN
4.      DECLARE @a INT = CAST(SUBSTRING(@PESEL, 1, 1) AS INT)
5.      DECLARE @b INT = CAST(SUBSTRING(@PESEL, 2, 1) AS INT)
6.      DECLARE @c INT = CAST(SUBSTRING(@PESEL, 3, 1) AS INT)
7.      DECLARE @d INT = CAST(SUBSTRING(@PESEL, 4, 1) AS INT)
8.      DECLARE @e INT = CAST(SUBSTRING(@PESEL, 5, 1) AS INT)
9.      DECLARE @f INT = CAST(SUBSTRING(@PESEL, 6, 1) AS INT)
10.     DECLARE @g INT = CAST(SUBSTRING(@PESEL, 7, 1) AS INT)
11.     DECLARE @h INT = CAST(SUBSTRING(@PESEL, 8, 1) AS INT)
12.     DECLARE @i INT = CAST(SUBSTRING(@PESEL, 9, 1) AS INT)
13.     DECLARE @j INT = CAST(SUBSTRING(@PESEL, 10, 1) AS INT)
14.     DECLARE @k INT = CAST(SUBSTRING(@PESEL, 11, 1) AS INT)
15.
16.     DECLARE @checksum INT = @a + 3 * @b + 7 * @c + 9 * @d + @e + 3 * @f + 7 * @g + 9 * @h + @i + 3
    * @j + @k
17.
18.     IF @checksum % 10 = 0
19.     BEGIN
20.         RETURN 1
21.     END
22.
23.     RETURN 0
24. END
25. GO
```

Funkcja „Employees.IsPeselChecksumValid” przyjmuje poprawny(składający się z cyfr) numer pesel i sprawdza czy numer pesel ma poprawną sumę kontrolną. Zwraca ‘true’ jeśli suma kontrolna się zgadza w przeciwnym wypadku ‘false’.

```
1.  CREATE FUNCTION Employees.GetGenderFromPesel(@PESEL NVARCHAR(11))
2.  RETURNS NVARCHAR(1)
3.  BEGIN
4.      IF CAST(SUBSTRING(@PESEL, 10, 1 ) AS INT) % 2 = 0
5.      BEGIN
6.          RETURN 'F'
7.      END
8.
9.      RETURN 'M'
10. END
11. GO
```

Funkcja „Employees.GetGenderFromPesel” przyjmuje poprawny(składający się z cyfr) numer pesel i zwraca „F” jeśli numer reprezentuje kobietę, lub „M” jeśli mężczyznę.

```
CREATE FUNCTION Employees.IsPositionWithGivenIDExists(@PositionID INT)
RETURNS BIT
BEGIN
    IF EXISTS(SELECT PositionID FROM Employees.Positions WHERE PositionID = @PositionID)
    BEGIN
        RETURN 1
    END

    RETURN 0
END
GO
```

Funkcja „Employees.IsPositionWithGivenIDExists” przyjmuje ID pozycji i zwraca ‘true’ jeśli pozycja o danym ID istnieje lub ‘false’ w przeciwnym razie.

```
CREATE FUNCTION Employees.GetLastWorkDepartmentId(@EmployeeID INT)
RETURNS INT
BEGIN
    RETURN (SELECT TOP(1) ID
                FROM Employees.EmployeeDepartement
                WHERE EmployeeID = @EmployeeID AND
                    WorkStopDATE IS NOT NULL
                ORDER BY WorkStartDATE
            )
END
GO
```

Funkcja „Employees.GetLastWorkDepartmentId” przyjmuje ID pracownika, jeśli pracownik istnieje i pracuje w którymś przodku lub ścianie to zwraca jego ID w przeciwnym razie ‘NULL’.

### 3.2. Longwall – Ściana (eksploatacyjna) oraz Mineface – Przodek

Z racji tego, że funkcje wykorzystywane do obsługi ścian i przodków są niemalże identyczne więc zostały opisane wspólnie. Opis będzie ogólny dla wyrobiska, konkretna wartość zwracana lub podawana na wyjście będzie się odnosiła do ściany lub przodka w zależności od użytego schematu.

```
CREATE FUNCTION Longwall.SumOfWallAdvanceInMeters(@LongwallID INT)
RETURNS INT
BEGIN
    RETURN (SELECT SUM(WallAdvanceInMeters)
            FROM Longwall.DailyRaport
            WHERE LongwallID = @LongwallID)
END
GO
```

```
CREATE FUNCTION MineFace.SumOfMineFaceAdvanceInMeters(@MineFaceID INT)
RETURNS INT
BEGIN
    RETURN (SELECT SUM(MineFaceAdvanceInMeters)
            FROM MineFace.DailyRaport
            WHERE MineFaceID = @MineFaceID)
END
GO
```

Powyższe funkcje przyjmują na wejście ID wyrobiska, zwraca sumę postępu wyrobiska od rozpoczęcia eksploatacji.

```sql
CREATE FUNCTION Longwall.DaysOfActivityOfLongwall(@LongwallID INT)
RETURNS INT
BEGIN
    DECLARE @StartDATE AS DATE
    DECLARE @EndDATE AS DATE

    SET @StartDATE = (SELECT StartDATE
            FROM Longwall.BasicInfo
            WHERE LongwallID = @LongwallID)

    SET @EndDATE = (SELECT EndDATE
            FROM Longwall.BasicInfo
            WHERE LongwallID = @LongwallID)

    IF @EndDATE IS NULL
    BEGIN
        SET @EndDATE = GETDATE()
    END

    RETURN DATEDIFF(dayofyear, @StartDATE, @EndDATE)
END
GO
```

```sql
CREATE FUNCTION MineFace.DaysOfActivityOfMineFace(@MineFaceID INT)
RETURNS INT
BEGIN
    DECLARE @StartDATE AS DATE
    DECLARE @EndDATE AS DATE

    SET @StartDATE =(SELECT StartDATE
            FROM MineFace.BasicInfo
            WHERE MineFaceID = @MineFaceID)

    SET @EndDATE = (SELECT EndDATE
            FROM MineFace.BasicInfo
            WHERE MineFaceID = @MineFaceID)

    IF @EndDATE IS NULL
    BEGIN
        SET @EndDATE = GETDATE()
    END

    RETURN DATEDIFF(dayofyear, @StartDATE, @EndDATE)
END
GO
```

Funkcje te zwracają liczbę dni funkcjonowania wyrobiska o podanym ID.

```sql
CREATE FUNCTION Longwall.ActivitiesDaysOfLongwall(@LongwallID INT)
RETURNS INT
BEGIN
    RETURN (SELECT SUM(ExcavatedCoalQuantityInKg)
            FROM Longwall.DailyRaport
            WHERE LongwallID = @LongwallID)
END
GO
```

```sql
CREATE FUNCTION MineFace.SumOfExcavatedCoalQuantityInKg(@MineFaceID INT)
RETURNS INT
BEGIN
    RETURN (SELECT SUM(ExcavatedCoalQuantityInKg)
            FROM MineFace.DailyRaport
            WHERE MineFaceID = @MineFaceID)
END
GO
```

Powyższe funkcje zwracają sumę wydobytego węgla w kg od początku istnienia wyrobiska o podany ID.

```
CREATE FUNCTION Longwall.IsLongwallWithGivenIdExists(@LongwallID INT)
RETURNS BIT
BEGIN
    IF EXISTS(SELECT TOP(1) LongwallName
                FROM Longwall.BasicInfo
                WHERE LongwallID = @LongwallID
        )
    BEGIN
        RETURN 1
    END

    RETURN 0
END
GO
```

```
CREATE FUNCTION MineFace.IsMineFaceWithGivenIdExists(@MinefaceID INT)
RETURNS BIT
BEGIN
    IF EXISTS(SELECT TOP(1) MineFaceName
                FROM MineFace.BasicInfo
                WHERE MineFaceID = @MinefaceID
        )
    BEGIN
        RETURN 1
    END

    RETURN 0
END
GO
```

Funkcje zwracają ‘true’ jeśli wyrobisko o podanym na wejściu ID istnieje, ‘false’ w przeciwnym wypadku.

```
CREATE FUNCTION Longwall.GetLongwallStartDate(@LongwallID INT)
RETURNS DATE
BEGIN
    RETURN (SELECT StartDate FROM Longwall.BasicInfo WHERE LongwallID = @LongwallID)
END
GO
```

```
CREATE FUNCTION Mineface.GetMineFaceStartDate(@MineFaceID INT)
RETURNS DATE
BEGIN
    RETURN (SELECT StartDate FROM MineFace.BasicInfo WHERE MineFaceID = @MineFaceID)
END
GO
```

Funkcja zwraca datę rozpoczęcia eksploatacji wyrobiska o podanym ID.

```
CREATE FUNCTION Longwall.IsLongwallWithGivenDateWorks(@LongwallID INT, @Date DATE)
RETURNS BIT
BEGIN
    IF EXISTS(SELECT StartDate
                FROM Longwall.BasicInfo
                WHERE LongwallID = @LongwallID AND
                    StartDate <= @Date AND
                    (EndDate >= @Date OR EndDate IS NULL)
            )
    BEGIN
        RETURN 1
    END

    RETURN 0
END
GO
```

```
CREATE FUNCTION MineFace.IsMineFaceWithGivenDateWorks(@MinefaceID INT, @Date DATE)
RETURNS BIT
BEGIN
    IF EXISTS(SELECT StartDate
```

```
                FROM MineFace.BasicInfo
                WHERE MineFaceID = @MinefaceID AND
                    StartDate <= @Date AND
                    (EndDate >= @Date OR EndDate IS NULL)
            )
    BEGIN
        RETURN 1
    END

    RETURN 0
END
GO
```

Funkcje zwracają ‘true’ jeśli wyrobisko o podanym ID jest w ruchu do podanej daty w przeciwnym przypadku zwraca ‘false’.

```
CREATE FUNCTION Longwall.IsRaportWithGivenIdExists(@RaportID INT)
RETURNS BIT
BEGIN
    IF EXISTS(SELECT RaportID
                FROM Longwall.DailyRaport
                WHERE RaportID = @RaportID
            )
    BEGIN
        RETURN 1
    END

    RETURN 0
END
GO
```

```
CREATE FUNCTION MineFace.IsRaportWithGivenIdExists(@RaportID INT)
RETURNS BIT
BEGIN
    IF EXISTS(SELECT RaportID
                FROM MineFace.DailyRaport
                WHERE RaportID = @RaportID
            )
    BEGIN
        RETURN 1
    END

    RETURN 0
END
GO
```

Funkcje zwracają ‘true’ jeśli istnieje raport o podanym ID dotyczący danego wyrobiska, w przeciwnym wypadku ‘false’.

```
CREATE FUNCTION Longwall.GetLongwallIdFromRaport(@RaportID INT)
RETURNS INT
BEGIN
    RETURN (SELECT TOP(1) LongwallID FROM Longwall.DailyRaport WHERE RaportID = @RaportID)
END
GO
```

```
CREATE FUNCTION MineFace.GetMineFaceIdFromRaport(@RaportID INT)
RETURNS INT
BEGIN
    RETURN (SELECT TOP(1) MineFaceID FROM MineFace.DailyRaport WHERE RaportID = @RaportID)
END
GO
```

Funkcja zwraca ID wyrobiska do którego należy raport o podanym ID, jeśli raport nie istnieje w ogóle zwraca ‘NULL’.

```sql
CREATE FUNCTION Longwall.IsRaportWithGivenIdLastOne(@RaportID INT)
RETURNS BIT
BEGIN
    DECLARE @LongwallID INT = Longwall.GetLongwallIdFromRaport(@RaportID)

    IF (SELECT TOP(1) RaportID
            FROM Longwall.DailyRaport
            WHERE LongwallID = @LongwallID
            ORDER BY RaportDATE
        ) = @RaportID
    BEGIN
        RETURN 1
    END

    RETURN 0
END
GO
```

```sql
CREATE FUNCTION Mineface.IsRaportWithGivenIdLastOne(@RaportID INT)
RETURNS BIT
BEGIN
    DECLARE @MinefaceID INT = Mineface.GetLongwallIdFromRaport(@RaportID)

    IF (SELECT TOP(1) RaportID
            FROM MineFace.DailyRaport
            WHERE MineFaceID = @MinefaceID
            ORDER BY RaportDATE
        ) = @RaportID
    BEGIN
        RETURN 1
    END

    RETURN 0
END
GO
```

Funkcje zwracają ‘true’ jeśli raport o podanym ID jest najnowszym raportem danego wyrobiska, w przeciwnym wypadku zwracają ‘false’.

```sql
CREATE FUNCTION Longwall.IsItemAvaibleAtLongwallWarehouse(@ItemID INT, @LongwallID INT)
RETURNS BIT
BEGIN
    IF EXISTS(SELECT ItemID
                FROM Longwall.ActualWarehouseStatus
                WHERE ItemID = @ItemID AND
                    LongwallID = @LongwallID
            )
    BEGIN
        RETURN 1
    END

    RETURN 0
END
GO
```

```sql
CREATE FUNCTION MineFace.IsItemAvaibleAtLongwallWarehouse(@ItemID INT, @MinefaceID INT)
RETURNS BIT
BEGIN
    IF EXISTS(SELECT ItemID
                FROM MineFace.ActualWarehouseStatus
                WHERE ItemID = @ItemID AND
                    MineFaceID = @MinefaceID
            )
    BEGIN
        RETURN 1
    END

    RETURN 0
END
GO
```

Funkcje zwracają ‘true’ jeśli przedmiot o podanym ID jest możliwy do użycia w danym wyrobisku, ‘false’ w przeciwnym przypadku.

### 3.3. Sales (department) – Dział sprzedaży

```
1.  CREATE FUNCTION Sales.AllTimeExcavatedCoalQuantityInKg()
2.  RETURNS INT
3.  BEGIN
4.      DECLARE @CoalSumFromLongwalls AS INT
5.      DECLARE @CoalSumFromMineface AS INT
6.
7.      SET @CoalSumFromLongwalls = (SELECT SUM(ExcavatedCoalQuantityInKg)
8.              FROM Longwall.DailyRaport)
9.
10.     SET @CoalSumFromMineface = (SELECT SUM(ExcavatedCoalQuantityInKg)
11.             FROM MineFace.DailyRaport)
12.
13.     IF @CoalSumFromLongwalls IS NULL
14.     BEGIN
15.         SET @CoalSumFromLongwalls = 0
16.     END
17.
18.     IF @CoalSumFromMineface IS NULL
19.     BEGIN
20.         SET @CoalSumFromMineface = 0
21.     END
22.
23.     RETURN (@CoalSumFromLongwalls + @CoalSumFromMineface)
24. END
25. GO
```

Funkcja zwraca sumę wydobytego węgla w kg od początku istnienia kopalni.

```
1.  CREATE FUNCTION Sales.ExcavatedCoalQuantityInKg(@StartDATE DATE, @EndDATE DATE)
2.  RETURNS INT
3.  BEGIN
4.      DECLARE @CoalSumFromLongwalls AS INT
5.      DECLARE @CoalSumFromMineface AS INT
6.      DECLARE @returnSum AS INT
7.
8.      SET @CoalSumFromLongwalls = (SELECT SUM(ExcavatedCoalQuantityInKg)
9.              FROM Longwall.DailyRaport
10.             WHERE (RaportDATE BETWEEN @StartDATE AND @EndDATE))
11.
12.     SET @CoalSumFromMineface = (SELECT SUM(ExcavatedCoalQuantityInKg)
13.             FROM MineFace.DailyRaport
14.             WHERE (RaportDATE BETWEEN @StartDATE AND @EndDATE))
15.
16.     IF @CoalSumFromLongwalls IS NULL
17.     BEGIN
18.         SET @CoalSumFromLongwalls = 0
19.     END
20.
21.     IF @CoalSumFromMineface IS NULL
22.     BEGIN
23.         SET @CoalSumFromMineface = 0
24.     END
25.
26.     SET @returnSum = (@CoalSumFromLongwalls + @CoalSumFromMineface)
27.
28.     RETURN @returnSum
29. END
30. GO
```

Funkcja zwraca sumę wydobytego węgla w kg w podanym okresie.

```sql
CREATE FUNCTION Sales.NumberOfUnrealizedOrders()
RETURNS INT
BEGIN
    RETURN (SELECT COUNT(*) FROM Sales.Orders WHERE RecipeDate IS NULL)
END
GO
```

Funkcja zwraca liczbę niezrealizowanych zamówień.

```sql
CREATE FUNCTION Sales.QuantityAndMoneyOfCoalFromUnrealizedOrders()
RETURNS @returnTable TABLE(
        QuanitityOfCoalInKg INT,
        MoneyFromOrdersInZl MONEY
    )
BEGIN
    INSERT INTO @returnTable
        SELECT SUM(QuantityInKg), SUM(QuantityInKg * UnitPriceInZl) FROM Sales.Orders
        WHERE RecipeDATE IS NULL

    RETURN
END
GO
```

Funkcja zwraca tabelę w sumą węgla oraz pieniędzy z oczekujących zamówień.

```sql
CREATE FUNCTION Sales.GetLastUnrealizedOrderIdAndQuantity()
RETURNS @returnTable TABLE(
        OrderID INT,
        QuanitityOfCoalInKg INT
    )
BEGIN
    DECLARE @OrderID INT
    DECLARE @Quantity INT

    SELECT TOP(1) @OrderID = OrderID, @Quantity = QuantityInKg
                FROM Sales.Orders
                WHERE RecipeDATE IS NULL
                ORDER BY OrderDATE ASC

    IF @OrderID IS NOT NULL
    BEGIN
        INSERT INTO @returnTable VALUES(@OrderID, @Quantity)
    END
    ELSE
    BEGIN
        INSERT INTO @returnTable VALUES(NULL, NULL)
    END

    RETURN
END
GO
```

Funkcja zwraca tabelę z ID zamówienia oraz z ilością zamówionego węgla z najstarszego niezrealizowanego zamówienia.

# 4. Procedury składowane

Celem stworzonych procedur jest dodawanie lub uaktualnienie danych w tabelach poprawnymi danymi – które w części procedur są sprawdzane przed próbą dodania danych. Jeśli któraś dana wejściowa nie spełnia założeń jest rzucany „przyjazny” wyjątek – czyli taki błąd który od razu mówi która dana jest błędna i dlaczego.

Procedury te wykorzystują niektóre funkcje opisane wyżej oraz swoje działanie integrują z niektórymi wyzwalaczami opisanymi w kolejnym rozdziale.

## 4.1. Employee – Pracownicy

```
CREATE PROCEDURE Employees.GetBirthDateFromPesel
    @PESEL NVARCHAR(11),
    @BirthDate DATE OUTPUT
AS
BEGIN
    DECLARE @PeselYearPart INT = CAST(SUBSTRING(@PESEL, 1, 2) AS INT)
    DECLARE @PeselMonthPart INT = CAST(SUBSTRING(@PESEL, 3, 2) AS INT)
    DECLARE @PeselDayPart INT = CAST(SUBSTRING(@PESEL, 5, 2) AS INT)

    IF @PeselDayPart > 31
    BEGIN
        ;THROW 50005, 'Birth day is incorrect!', 1
    END

    DECLARE @Period INT = @PeselMonthPart / 20
    DECLARE @BirthMonth INT = @PeselMonthPart % 20

    IF @BirthMonth > 12 OR @BirthMonth = 0
    BEGIN
        ;THROW 50006, 'Birth month is incorrect!', 1
    END

    DECLARE @BirthYear INT

    IF @Period < 4
    BEGIN
        SET @BirthYear = 1900 + 100 * @Period + @PeselYearPart
    END
    ELSE --@Period == 4
    BEGIN
        SET @BirthYear = 1800 + @PeselYearPart
    END

    SET @BirthDATE =  CAST(
                CAST(@BirthYear AS VARCHAR(4)) +
                RIGHT('0' + CAST(@BirthMonth AS VARCHAR(2)), 2) +
                RIGHT('0' + CAST(@PeselDayPart AS VARCHAR(2)), 2)
            AS DATE)
END
GO
```

Procedura przyjmuje poprawny numer pesel (11 cyfr) i zapisuje do zmiennej BirthDate datę urodzenia. Jeśli pesel będzie wskazywał na dzień większy od 31 to procedura rzuci wyjątek 50005, natomiast jeśli numer miesiąca będzie poza zakresem 0-12 procedura rzuci wyjątek 50006.

```
CREATE PROCEDURE Employees.AddEmployee
    @PositionID INT,
    @FirstName NVARCHAR(30),
    @Surname NVARCHAR(30),
    @PESEL NVARCHAR(11),
    @TelephoneNumber INT,
    @HireDATE DATE
AS
```

```
BEGIN
    IF Employees.IsPositionWithGivenIDExists(@PositionID) = 'false'
    BEGIN
        ;Throw 50007, 'Position with given PositionID doesn''t exists', 1
    END

    IF @FirstName IS NULL OR @FirstName = ''
    BEGIN
        ;THROW 50000, 'Firstname can''t be empty!', 1
    END

    IF @Surname IS NULL OR @Surname = ''
    BEGIN
        ;THROW 50001, 'Surname can''t be empty!', 1
    END

    IF LEN(@PESEL) != 11
    BEGIN
        ;THROW 50002, 'Pesel has to be 11 numnber length!', 1
    END

    IF Employees.IsPeselContainsOnlyDigits(@PESEL) = 'false'
    BEGIN
        ;THROW 50003, 'Pesel can contains only digits!', 1
    END

    IF Employees.IsPeselChecksumValid(@PESEL) = 'false'
    BEGIN
        ;THROW 50004, 'Pesel doesn''t have valid checksum!', 1
    END

    DECLARE @Gender NVARCHAR(1) =  Employees.GetGenderFromPesel(@PESEL)

    DECLARE @BirthDATE DATE

    BEGIN TRY
        EXEC Employees.GetBirthDATEFromPesel @PESEL, @BirthDATE OUTPUT
    END TRY

    BEGIN CATCH
        ;THROW
    END CATCH

    INSERT INTO Employees.BasicInfo
        VALUES(@PositionID, @FirstName, @Surname, @Gender, @PESEL, @TelephoneNumber, @BirthDATE, @H
ireDATE, DEFAULT)
END
GO
```

Procedura wykorzystywana jest do dodawania nowego pracownika. Przyjmuje w pierwszym parametrze ID pozycji na którym pracownik ma zacząć swoją pracę kolejno, imię, nazwisko, nr PESEL, numer telefonu, oraz datę od której zaczyna pracę. Procedura sprawdza poszczególne wartości parametrów podawanych jej na wejściu i rzuca odpowiednie „przyjazne” wyjątki.

```
CREATE PROCEDURE Employees.AddEmployeeVacation
    @EmployeeID INT,
    @StartDATE DATE,
    @EndDATE DATE
AS
BEGIN
    IF @StartDATE < @EndDATE
    BEGIN
        ;THROW 50010, 'Incorrect dates!', 1
    END

    IF Employees.IsEmployeeWithGivenIDExists(@EmployeeID) = 'false'
    BEGIN
        ;THROW 50009, 'Employee with given ID doesn''t exists!', 1
    END
```

```
16.
17.     IF Employees.IsEmployeeAtVacationWithGivenPeriod(@EmployeeID, @StartDATE, @EndDATE) = 'true'
18.     BEGIN
19.         ;THROW 50008, 'Employee already has vacation at this period!', 1
20.     END
21.
22.     INSERT INTO Employees.Vacation
23.         VALUES(@EmployeeID, @StartDATE, @EndDATE)
24. END
25. GO
```

Procedura nadaje pracownikowi urlop. Na wejściu przyjmuje ID pracownika oraz dwie daty – początku i końca urlopu. Procedura sprawdza dane na wejściu oraz to czy pracownikowi można przydzielić urlop – jeśli w aktualnym terminie jest już na urlopie procedura rzuci wyjątek.

```
1.  CREATE PROCEDURE Employees.AddEmployeeDepartement
2.      @EmployeeID INT,
3.      @LongwallID INT,
4.      @MineFaceID INT,
5.      @StartDATE DATE
6.  AS
7.  BEGIN
8.      IF (@LongwallID IS NOT NULL AND @MineFaceID IS NOT NULL) OR
9.          (@LongwallID IS NULL AND @MineFaceID IS NULL)
10.     BEGIN
11.         ;THROW 50020, 'Incorrect department!', 1
12.     END
13.
14.     DECLARE @ID INT = Employees.GetLastWorkDepartmentId(@EmployeeID)
15.
16.     BEGIN TRAN AddEmployeeDepartement
17.         BEGIN TRY
18.             INSERT INTO Employees.EmployeeDepartement VALUES(@LongwallID, @MineFaceID, @EmployeeID,
     @StartDATE, NULL)
19.             UPDATE Employees.EmployeeDepartement SET WorkStopDATE = @StartDATE WHERE ID = @ID
20.         END TRY
21.
22.         BEGIN CATCH
23.             ROLLBACK
24.             ;THROW
25.         END CATCH
26.
27.     COMMIT
28. END
```

Procedura przyjmuje ID pracownika, ID ściany LUB ID przodka do którego pracownik ma być przypisany, oraz datę od której ma zacząć pracować w danym wyrobisku. Procedura sprawdza tylko poprawność danych wejściowych. Resztę walidacji wykonuje baza poprzez sprawdzanie nałożonych ograniczeń na poszczególne tablice. Jeśli któreś ograniczenie nie będzie spełnione procedura cofa zmiany i rzuca wyjątek wygenerowany przez bazę.

### 4.2. Longwall – Ściana (eksploatacyjna) oraz Mineface – Przodek

Z racji tego, że procedurę składowane wykorzystywane do obsługi ścian i przodków są niemalże identyczne więc zostały opisane wspólnie. Opis będzie ogólny dla wyrobiska, konkretna wartość zwracana, uaktualniana, dodawana lub podawana na wyjście będzie się odnosiła do ściany lub przodka w zależności od użytego schematu.

```
1.  CREATE PROCEDURE Longwall.AddLongwall
2.      @LongwallName NVARCHAR(30),
3.      @LongwallLong INT,
4.      @LongwallWide INT,
```

```
    @LongwallThick INT,
    @StartDATE DATE
AS
BEGIN
    IF @LongwallName IS NULL OR @LongwallName = ''
    BEGIN
        ;THROW 50010, 'Longwall name can''t be empty!', 1
    END

    IF @LongwallLong <= 0 OR @LongwallThick <= 0 OR @LongwallWide <= 0
    BEGIN
        ;THROW 50011, 'The dimensions of the longwall must be positive!', 1
    END

    INSERT INTO Longwall.BasicInfo
        VALUES(@LongwallName, @LongwallLong, @LongwallWide, @LongwallThick, @StartDATE, DEFAULT)
END
GO
```

```
CREATE PROCEDURE MineFace.AddMineface
    @MinefaceName NVARCHAR(30),
    @MineFaceLong INT,
    @MineFaceThick INT,
    @MineFaceSectionalArea INT,
    @MineFaceTimber INT,
    @StartDATE DATE
AS
BEGIN
    IF @MinefaceName IS NULL OR @MinefaceName = ''
    BEGIN
        ;THROW 50010, 'Mineface name can''t be empty!', 1
    END

    IF @MineFaceLong <= 0 OR @MineFaceThick <= 0 OR @MineFaceSectionalArea <= 0 OR @MineFaceTimber
<= 0
    BEGIN
        ;THROW 50011, 'The dimensions of the Mineface must be positive!', 1
    END

    INSERT INTO Mineface.BasicInfo
        VALUES(@MinefaceName, @MineFaceLong, @MineFaceThick, @MineFaceSectionalArea, @MineFaceTimbe
r, @StartDATE, DEFAULT)
END
GO
```

Procedury te dodają nowe wyrobisko do bazy. Na wejściu przyjmują nazwę, długość, w przypadku ściany wybieg o szerokość, w przypadku przodka grubość i przekrój oraz podziałkę obudowy, dla oby dwóch wyrobisk podawana jest również data rozpoczęcia ruchu ściany. Gdy nazwa jest pusta lub wymiary są 0 lub ujemne procedura rzuca wyjątek.

```
CREATE PROCEDURE Longwall.CloseLongwall
    @LongwallID INT,
    @LongwallEndDATE DATE
AS
BEGIN
    IF Longwall.IsLongwallWithGivenIdExists(@LongwallID) = 'false'
    BEGIN
        ;THROW 50012, 'Longwall with given id doesn''t exists!', 1
    END

    IF Longwall.GetLongwallStartDATE(@LongwallID) > @LongwallEndDATE
    BEGIN
        ;THROW 50013, 'End date is incorrect!', 1
    END

    UPDATE Longwall.BasicInfo SET EndDATE = @LongwallEndDATE WHERE LongwallID = @LongwallID
END
GO
```

```sql
CREATE PROCEDURE Mineface.CloseMineface
    @MinefaceID INT,
    @MinefaceEndDATE DATE
AS
BEGIN
    IF MineFace.IsMineFaceWithGivenIdExists(@MinefaceID) = 'false'
    BEGIN
        ;THROW 50012, 'Mineface with given id doesn''t exists!', 1
    END

    IF MineFace.GetMineFaceStartDate(@MinefaceID) > @MinefaceEndDATE
    BEGIN
        ;THROW 50013, 'End date is incorrect!', 1
    END

    UPDATE MineFace.BasicInfo SET EndDATE = @MinefaceEndDATE WHERE @MinefaceID = @MinefaceID
END
GO
```

Procedury te przyjmują ID wyrobiska i czas zakończenia biegu wyrobiska. Sprawdzają poprawność wprowadzonych danych, jeśli są błędne to rzucają „przyjazne” wyjątki. Jeśli wszystko się powiedzie to uaktualniana jest wartość pola EndDate odpowiedniego wyrobiska.

```sql
CREATE PROCEDURE Longwall.AddDailyRaport
    @LongwallID INT,
    @EmployeeID INT,
    @ExcavatedCoalQuantityInKg INT,
    @WallAdvanceInMeters INT,
    @Comments VARCHAR(MAX) NULL
AS
BEGIN
    IF Longwall.IsLongwallWithGivenIdExists(@LongwallID) = 'false'
    BEGIN
        ;THROW 50012, 'Longwall with given id doesn''t exists!', 1
    END

    IF Employees.IsEmployeeWithGivenIDExists(@EmployeeID) = 'false'
    BEGIN
        ;THROW 50014, 'Employee with given id doesn''t exists!', 1
    END

    IF @ExcavatedCoalQuantityInKg < 0 OR @WallAdvanceInMeters < 0
    BEGIN
        ;THROW 50015, 'The progression data of the longwall can''t be negative!', 1
    END

    IF Longwall.IsLongwallWithGivenDATEWorks(@LongwallID, GETDATE()) = 'false'
    BEGIN
        ;THROW 50016, 'Longwall doesn''t work!', 1
    END

    INSERT INTO Longwall.DailyRaport
        VALUES(@EmployeeID, @LongwallID, GETDATE(), @ExcavatedCoalQuantityInKg, @WallAdvanceInMeter
s, @Comments)
END
GO
```

```sql
CREATE PROCEDURE Mineface.AddDailyRaport
    @MinefaceID INT,
    @EmployeeID INT,
    @TimberScaleInMeters INT,
    @ExcavatedCoalQuantityInKg INT,
    @MineFaceAdvanceInMeters INT,
    @Comments VARCHAR(MAX) NULL
AS
BEGIN
    IF MineFace.IsMineFaceWithGivenIdExists(@MinefaceID) = 'false'
    BEGIN
        ;THROW 50012, 'Mineface with given id doesn''t exists!', 1
    END
```

```
14.
15.     IF Employees.IsEmployeeWithGivenIDExists(@EmployeeID) = 'false'
16.     BEGIN
17.         ;THROW 50014, 'Employee with given id doesn''t exists!', 1
18.     END
19.
20.     IF @ExcavatedCoalQuantityInKg < 0 OR @MineFaceAdvanceInMeters < 0
21.     BEGIN
22.         ;THROW 50015, 'The progression data of the longwall can''t be negative!', 1
23.     END
24.
25.     IF MineFace.IsMineFaceWithGivenDateWorks(@MinefaceID, GETDATE()) = 'false'
26.     BEGIN
27.         ;THROW 50016, 'Longwall doesn''t work!', 1
28.     END
29.
30.     IF @TimberScaleInMeters <= 0
31.     BEGIN
32.         ;THROW 50017, 'Timber scale has to positive!', 1
33.     END
34.
35.     INSERT INTO Mineface.DailyRaport
36.         VALUES(@EmployeeID, @MinefaceID, GETDATE(), @TimberScaleInMeters, @ExcavatedCoalQuantityInK
    g, @MineFaceAdvanceInMeters, @Comments)
37. END
38. GO
```

Procedury te dodają dzienny raport do bazy. Na wejście dostaje ID wyrobiska, ID pracownika dodającego raport, postęp oraz wydobyty węgiel, ponadto dla przodka podawana jest jeszcze podziałka obudowy. Procedura sprawdza poprawność danych wejściowych, jeśli któraś dana jest niepoprawna to procedura rzuca „przyjazny wyjątek”.

```
1.  CREATE PROCEDURE Longwall.AddUsedResource
2.      @RaportID INT,
3.      @ItemID INT,
4.      @ItemUsedAmount INT
5.  AS
6.  BEGIN
7.      IF @ItemUsedAmount <= 0
8.      BEGIN
9.          ;THROW 50015, 'You cannot add used reosurce with used amount <= 0!', 1
10.     END
11.
12.     IF Longwall.IsRaportWithGivenIdLastOne(@RaportID) = 'false'
13.     BEGIN
14.         ;THROW 50014, 'This raport is not the last one!', 1
15.     END
16.
17.     BEGIN TRAN T1
18.         BEGIN TRY
19.             INSERT INTO Longwall.UsedResources VALUES (@RaportID, @ItemID, @ItemUsedAmount)
20.         END TRY
21.
22.         BEGIN CATCH
23.
24.             ROLLBACK TRAN T1
25.             ;THROW
26.         END CATCH
27.
28.     COMMIT TRAN T1
29. END
30. GO
```

```
1.  CREATE PROCEDURE Mineface.AddUsedResource
2.      @RaportID INT,
3.      @ItemID INT,
4.      @ItemUsedAmount INT
5.  AS
6.  BEGIN
```

```
7.       IF @ItemUsedAmount <= 0
8.     BEGIN
9.         ;THROW 50015, 'You cannot add used reosurce with used amount <= 0!', 1
10.    END
11.
12.    IF MineFace.IsRaportWithGivenIdLastOne(@RaportID) = 'false'
13.    BEGIN
14.        ;THROW 50014, 'This raport is not the last one!', 1
15.    END
16.
17.    BEGIN TRAN T1
18.        BEGIN TRY
19.            INSERT INTO MineFace.UsedResources VALUES (@RaportID, @ItemID, @ItemUsedAmount)
20.        END TRY
21.
22.        BEGIN CATCH
23.
24.            ROLLBACK TRAN T1
25.            ;THROW
26.        END CATCH
27.
28.    COMMIT TRAN T1
29. END
30. GO
```

Procedury te dodają użyte zasoby wykorzystane do pracy skorelowane są z raportem dziennym. Na wejście procedura otrzymuje ID raportu, ID produktu oraz ilość zużytego sprzętu. Procedura ta rzuca przyjazne wyjątki.

### 4.3. **Sales (department) – Dział sprzedaży**

```
1.  CREATE PROCEDURE Sales.AddOrder
2.      @CustomerID INT,
3.      @QuantityInKg INT,
4.      @UnitPriceZl MONEY = 100
5.  AS
6.  BEGIN
7.      BEGIN TRAN T1
8.
9.          BEGIN TRY
10.             INSERT INTO Sales.Orders VALUES (GETDATE(), NULL, @CustomerID, @QuantityInKg, @UnitPric
    eZl)
11.         END TRY
12.
13.         BEGIN CATCH
14.             ROLLBACK TRAN T1
15.             ;THROW
16.         END CATCH
17.
18.     COMMIT TRAN T1
19. END
20. GO
```

Procedura ta dodaje zamówienie, na wejściu otrzymuje ID klienta, ilość zamawianego surowca, oraz wyznaczoną cenę jednostkową. W tej procedurze wykorzystano obsługę błędów zagwarantowaną przez nadane ograniczeń podczas tworzenia tabel.

# 5. Opis wyzwalaczy

Przyjęto, że do bazy w jednym zapytaniu dodawane będą tylko pojedyncze porcje danych – tzn. aktualizujące lub dodające tylko pojedyncze wiersze w tabeli i też w takim z takimi założeniami wyzwalacze zostały stworzone.

## 5.1. Longwall – Ściana (eksploatacyjna) oraz Mineface – Przodek

Z racji tego, że wyzwalacze wykorzystywane do obsługi ścian i przodków są niemalże identyczne więc zostały opisane wspólnie. Opis będzie ogólny dla wyrobiska, konkretna wartość uaktualniana lub dodawana będzie się odnosiła do ściany lub przodka w zależności od użytego schematu.

```
1.  CREATE TRIGGER Longwall.t_UpDateWarhouseStatus
2.      ON Longwall.UsedResources
3.  AFTER INSERT
4.  AS
5.  BEGIN
6.      DECLARE @RaportID INT = (SELECT TOP(1) RaportID FROM INSERTED)
7.      DECLARE @UsedAmount INT = (SELECT TOP(1) ItemUsedAmount FROM INSERTED)
8.      DECLARE @ItemID INT = (SELECT TOP(1) ItemID FROM INSERTED)
9.
10.     DECLARE @LongwallID INT = Longwall.GetLongwallIdFromRaport(@RaportID)
11.
12.     BEGIN TRY
13.         UPDATE Longwall.ActualWarehouseStatus SET AvailableAmount = AvailableAmount - @UsedAmount
14.             WHERE LongwallID = @LongwallID AND ItemID = @ItemID
15.     END TRY
16.
17.     BEGIN CATCH
18.         ;THROW
19.     END CATCH
20. END
21. GO
```

```
1.  CREATE TRIGGER MineFace.t_UpDateWarhouseStatus
2.      ON Mineface.UsedResources
3.  AFTER INSERT
4.  AS
5.  BEGIN
6.      DECLARE @RaportID INT = (SELECT TOP(1) RaportID FROM INSERTED)
7.      DECLARE @UsedAmount INT = (SELECT TOP(1) ItemUsedAmount FROM INSERTED)
8.      DECLARE @ItemID INT = (SELECT TOP(1) ItemID FROM INSERTED)
9.
10.     DECLARE @MinefaceID INT = Mineface.GetMineFaceIdFromRaport(@RaportID)
11.
12.     BEGIN TRY
13.         UPDATE MineFace.ActualWarehouseStatus SET AvailableAmount = AvailableAmount - @UsedAmount
14.             WHERE MineFaceID = @MinefaceID AND ItemID = @ItemID
15.     END TRY
16.
17.     BEGIN CATCH
18.         ;THROW
19.     END CATCH
20. END
21. GO
```

Procedury te wyzwalane są po akcji wstawienia wiersza do tabeli „UsedResources” i powodują uaktualnienie (odjęcie zużytej ilości danego przedmiotu) aktualnego stanu dostępnego przedmiotu.

```
1.  CREATE TRIGGER Longwall.t_OrderItem
2.      ON Longwall.ActualWarehouseStatus
3.  AFTER UPDATE
4.  AS
5.  BEGIN
```

```
    DECLARE @AvaibleAmount INT = (SELECT TOP(1) AvailableAmount FROM INSERTED)
    DECLARE @MinimumAmount INT = (SELECT TOP(1) MinimumAmount FROM INSERTED)
    DECLARE @DeletedAmount INT = (SELECT TOP(1) AvailableAmount FROM DELETED)

    IF @DeletedAmount > @AvaibleAmount
        BEGIN
        BEGIN TRY
            IF @AvaibleAmount < @MinimumAmount
            BEGIN
                DECLARE @LongwallID INT = (SELECT TOP(1) LongwallID FROM INSERTED)
                DECLARE @ItemID INT = (SELECT TOP(1) ItemID FROM INSERTED)

                INSERT INTO Longwall.ItemsOrder VALUES (@LongwallID, @ItemID, (@MinimumAmount - @Av
aibleAmount), GETDATE())
            END
        END TRY

        BEGIN CATCH
            ;THROW
        END CATCH
    END
END
GO
```

```
CREATE TRIGGER Mineface.t_OrderItem
    ON Mineface.ActualWarehouseStatus
AFTER UPDATE
AS
BEGIN
    DECLARE @AvaibleAmount INT = (SELECT TOP(1) AvailableAmount FROM INSERTED)
    DECLARE @MinimumAmount INT = (SELECT TOP(1) MinimumAmount FROM INSERTED)
    DECLARE @DeletedAmount INT = (SELECT TOP(1) AvailableAmount FROM DELETED)

    IF @DeletedAmount > @AvaibleAmount
        BEGIN
        BEGIN TRY
            IF @AvaibleAmount < @MinimumAmount
            BEGIN
                DECLARE @minefaceID INT = (SELECT TOP(1) MineFaceID FROM INSERTED)
                DECLARE @ItemID INT = (SELECT TOP(1) ItemID FROM INSERTED)

                INSERT INTO MineFace.ItemsOrder VALUES (@minefaceID, @ItemID, (@MinimumAmount - @Av
aibleAmount), GETDATE())
            END
        END TRY

        BEGIN CATCH
            ;THROW
        END CATCH
    END
END
GO
```

Procedury te wyzwalane są po aktualizacji tabeli „ActualWarehouseStatus”. Na wstępie sprawdzany jest warunek czy jest to aktualizacja odejmująca czy dodająca ilość dostępnych przedmiotów jeśli nie jest to wyzwalacz kończy swoje działanie. Jeśli natomiast jest to akcja odejmująca ilość, następnie sprawdzany jest warunek czy dostępna ilość przedmiotów jest mniejsza niż dopuszczalne minimum, jeśli jest mniejsza to wykonywane jest zamówienie przedmiotów.

```
CREATE TRIGGER Longwall.t_ItemDelivery
    ON Longwall.ItemsOrder
FOR INSERT
AS
BEGIN
    DECLARE @LongwallID INT = (SELECT TOP(1) LongwallID FROM INSERTED)
    DECLARE @ItemID INT = (SELECT TOP(1) ItemID FROM INSERTED)
    DECLARE @OrderedAmount INT = (SELECT TOP(1) OrderedItemAmount FROM INSERTED)

```

```
    BEGIN TRY
        UPDATE Longwall.ActualWarehouseStatus SET AvailableAmount = AvailableAmount + @OrderedAmoun
t
            WHERE LongwallID = @LongwallID AND ItemID = @ItemID
    END TRY

    BEGIN CATCH
        ;THROW
    END CATCH
END
GO
```

```
CREATE TRIGGER Mineface.t_ItemDelivery
    ON Mineface.ItemsOrder
FOR INSERT
AS
BEGIN
    DECLARE @MineFaceID INT = (SELECT TOP(1) MineFaceID FROM INSERTED)
    DECLARE @ItemID INT = (SELECT TOP(1) ItemID FROM INSERTED)
    DECLARE @OrderedAmount INT = (SELECT TOP(1) OrderedItemAmount FROM INSERTED)

    BEGIN TRY
        UPDATE Mineface.ActualWarehouseStatus SET AvailableAmount = AvailableAmount + @OrderedAmoun
t
            WHERE MineFaceID = @MineFaceID AND ItemID = @ItemID
    END TRY

    BEGIN CATCH
        ;THROW
    END CATCH
END
GO
```

Procedury wyzwalane są po wstawieniu wiersza do tablicy zamówień przedmiotów - „ItemsOrder”. Wyzwalacz ten uaktualnia stan przedmiotów dostępnych które do danego wyrobiska zostały zamówione.

```
CREATE TRIGGER Longwall.t_AddExcavatedCoalAtCoalPile
    ON Longwall.DailyRaport
AFTER INSERT
AS
BEGIN
    DECLARE @AddedCoal INT = (SELECT TOP(1) ExcavatedCoalQuantityInKg FROM INSERTED)

    BEGIN TRY
        UPDATE Sales.CoalPileStatus SET LastModificationDATE = GETDATE(), CurrentCoalQuantity = Cur
rentCoalQuantity + @AddedCoal WHERE CoalPileID = 1
    END TRY

    BEGIN CATCH
        ;Throw
    END CATCH
END
GO
```

```
CREATE TRIGGER Mineface.t_AddExcavatedCoalAtCoalPile
    ON Mineface.DailyRaport
AFTER INSERT
AS
BEGIN
    DECLARE @AddedCoal INT = (SELECT TOP(1) ExcavatedCoalQuantityInKg FROM INSERTED)

    BEGIN TRY
        UPDATE Sales.CoalPileStatus SET LastModificationDATE = GETDATE(), CurrentCoalQuantity = Cur
rentCoalQuantity + @AddedCoal WHERE CoalPileID = 1
    END TRY

    BEGIN CATCH
        ;Throw
    END CATCH
END
```

```
16. GO
```

Procedury te są wywoływane po dodaniu dziennego raportu na tablicy „DailyRaport”. Odpowiadają za dodanie wydobytego węgla na hołdę oraz zmodyfikowanie dany wskazującej na ostatnią jej modyfikację.

## 5.2. Sales (department) – Dział sprzedaży

```
1.  CREATE TRIGGER Sales.t_RealizePendingOrder
2.      ON Sales.CoalPileStatus
3.  AFTER UPDATE
4.  AS
5.  BEGIN
6.      DECLARE @OrderID INT
7.      DECLARE @OrderQuantity INT
8.
9.      SELECT @OrderID = OrderID, @OrderQuantity = QuanitityOfCoalInKg FROM Sales.GetLastUnrealizedOrd
    erIdAndQuantity()
10.
11.     DECLARE @CurrentCoalQuantity INT = (SELECT TOP(1) CurrentCoalQuantity FROM INSERTED)
12.
13.     WHILE @OrderID IS NOT NULL
14.     BEGIN
15.         IF @CurrentCoalQuantity >= @OrderQuantity
16.         BEGIN
17.             BEGIN TRAN T1
18.                 UPDATE Sales.Orders SET RecipeDATE = GETDATE() WHERE OrderID = @OrderID
19.                 UPDATE Sales.CoalPileStatus SET LastModificationDATE = GETDATE(), CurrentCoalQuanti
    ty = CurrentCoalQuantity - @OrderQuantity WHERE CoalPileID = 1
20.             COMMIT TRAN T1
21.
22.             SELECT @OrderID = OrderID, @OrderQuantity = QuanitityOfCoalInKg FROM Sales.GetLastUnrea
    lizedOrderIdAndQuantity()
23.             SET @CurrentCoalQuantity = (SELECT CurrentCoalQuantity FROM Sales.CoalPileStatus)
24.         END
25.         ELSE
26.         BEGIN
27.             SET @OrderID = NULL
28.         END
29.     END
30. END
31. GO
```

Procedura ta wyzwala się po aktualizacji statusu hołdy – tablicy „Sales.CoalPileStatus”, działa on w pętli. Jeśli zostanie uruchomiony to automatycznie sprawdza najstarsze niezrealizowane zamówienie - czy aktualny stan węgla jest w stanie do zrealizowania tego zamówienia, jeśli jest to go realizuje następnie na tej samej zasadzie sprawdzając kolejne zamówienia aż się nie skończą bądź dostępny węgiel będzie niewystarczający.

```
1.  CREATE TRIGGER Sales.t_RealizeOrder
2.      ON Sales.Orders
3.  FOR INSERT
4.  AS
5.  BEGIN
6.      DECLARE @OrderID INT
7.
8.      SELECT @OrderID = OrderID FROM Sales.GetLastUnrealizedOrderIdAndQuantity()
9.
10.     IF @OrderID IS NULL OR @OrderID = (SELECT TOP(1) OrderID FROM INSERTED)
11.     BEGIN
12.         DECLARE @CurrentCoalQuantity INT = (SELECT CurrentCoalQuantity FROM Sales.CoalPileStatus WH
    ERE CoalPileID = 1)
13.         DECLARE @OrderQuantity INT = (SELECT TOP(1) QuantityInKg FROM INSERTED)
14.
15.         IF @CurrentCoalQuantity >= @OrderQuantity
16.         BEGIN
```

```
17.             UPDATE Sales.Orders SET RecipeDATE = GETDATE() WHERE OrderID = @OrderID
18.             UPDATE Sales.CoalPileStatus SET LastModificationDATE = GETDATE(), CurrentCoalQuantity =
    CurrentCoalQuantity - @OrderQuantity WHERE CoalPileID = 1
19.         END
20.     END
21. END
```

Wyzwalacz ten aktywuje się po dodaniu zamówienia – wiersza do tabeli „Sales.Orders”. Sprawdza czy dodane zamówienie jest ostatnim niezrealizowanym jeśli tak to czy aktualny stan hołdy jest w stanie go obsłużyć. Jeśli jest to robione są aktualizacje na hołdzie oraz zamówienie przyjmuje status zrealizowanego.

# 6. Typowe zapytania SQL

Dzięki stworzonym procedurom większość zapytań dodających czy uaktualniających daną informację w bazie można wykonywać za ich pomocą co gwarantuje w niektórych przypadkach „przyjazne” wyjątki oraz nie pozawala dopuścić do złamania przyjętych w fazie projektowej założeń.

Poniżej zaprezentowane zostało kilka przykładowych zapytań dla stworzonej bazy danych.

```
1.  USE KWK
2.  GO
3.
4.  -- Dodanie stanowisk do kopalni
5.  INSERT INTO Employees.Positions VALUES (1, 'Dyrektor'), (2, 'Główny Inżynier'), (3, 'Sztygar Górnic
    zy'), (4, 'Górnik')
6.
7.  -- Wypisanie wszystkich stnowisk
8.  SELECT * FROM Employees.Positions
9.
10. -- Dodanie pracowników
11. EXEC Employees.AddEmployee 1, 'Marian', 'Kowalski', '18060218259', 123456789, '1992-08-09'
12. EXEC Employees.AddEmployee 2, 'Andrzej', 'Kowalik', '41032818230', 951632341, '1993-01-09'
13. EXEC Employees.AddEmployee 3, 'Marcin', 'Kowal', '78120801933', 950000341, '1993-02-09'
14. EXEC Employees.AddEmployee 3, 'Jędrzej', 'Kos', '77060318390', 951632111, '1993-03-09'
15. EXEC Employees.AddEmployee 4, 'Gary', 'D. Lynch', '44033018630', 121632341, '1993-04-09'
16. EXEC Employees.AddEmployee 4, 'Tomasz', 'Kozieł', '33012301153', 121632341, '1993-05-09'
17. EXEC Employees.AddEmployee 4, 'Patryk', 'Tomaszewski', '00291919573', 121632341, '1993-06-09'
18. EXEC Employees.AddEmployee 4, 'Łukasz', 'Kozi', '46101812230', 121632341, '1993-07-09'
19. EXEC Employees.AddEmployee 4, 'Sereniusz', 'Serowy', '63091818833', 121632341, '1993-08-09'
20. EXEC Employees.AddEmployee 4, 'Syriusz', 'Blue', '25081519113', 121632341, '1993-09-09'
21.
22.
23. -- Wypisanie pracowników
24. SELECT * FROM Employees.BasicInfo
25. GO
26.
27. -- Nadanie pracownikowi zwierzchnika
28. INSERT INTO Employees.Superior VALUES(2, 1, '1993-01-09'), (3, 2, '1993-02-09'), (4, 2, '1993-02-
    09'),
29.                                     (5, 3, '1993-02-09'), (6, 3, '1993-02-09'), (7, 3, '1993-02-
    09'),
30.                                     (8, 4, '1993-02-09'), (9, 4, '1993-02-09'), (10, 4, '1993-02-
    09')
31.
32.
33. -- Wypisanie pracowników wraz z ich zwierzchnikami
34. SELECT BI.FirstName, BI.Surname, BI.Gender, BI.PESEL, B.FirstName AS 'Superior Firstname', B.Surnam
    e AS 'Superior Surname'
35.     FROM Employees.BasicInfo AS BI JOIN Employees.Superior AS SUP ON BI.EmployeeID = SUP.EmployeeID

36.         JOIN Employees.BasicInfo B ON SUP.SuperiorID = B.EmployeeID
37.
38. -- Dodanie ścian do kopalni
39. EXEC Longwall.AddLongwall '730 I', 1500, 250, 3, '1994-01-01'
40. EXEC Longwall.AddLongwall '731', 1000, 200, 2, '1994-02-01'
41. EXEC Longwall.AddLongwall '732', 1200, 180, 2, '1994-03-01'
42.
43. -- Zamknięcie ściany
44. EXEC Longwall.CloseLongwall 3, '2000-12-12'
45.
46. -- Dodanie pracownikowi działu w którym pracuje
47. EXEC Employees.AddEmployeeDepartement 2, 1, NULL, '1994-01-02'
48.
49. -- Wypisanie podstawowych informacji o ścianach
50. SELECT * FROM Longwall.BasicInfo
51.
52. -- Dodanie obiektów dostępnych do użycia na koplani
53. INSERT INTO Items VALUES('Kombajn ścianowy', 1000000), ('Kombjan chodnikowy', 500000), ('Noże do ko
    mbajnu', 80), ('Kilof', 50), ('Łopata', 50), ('Gaśnica', 100), ('Obudowa ŁP29', 500)
54.
```

```
-
- Dodanie do ściany możliwych do użycia przedmiotów, z ich ilością startową oraz ilością minimalną

INSERT INTO Longwall.ActualWarehouseStatus VALUES(1, 1, 1, 1), (1, 3, 80, 50), (1, 4, 10, 5), (1, 5, 10, 5)
SELECT * FROM Longwall.ActualWarehouseStatus

-- Utworzenie hołdy
INSERT INTO Sales.CoalPileStatus VALUES(1, GETDATE(), 0)
SELECT * FROM Sales.CoalPileStatus

-- Dodanie raportu dziennego
EXEC Longwall.AddDailyRaport 1, 3, 1000000, 10, 'BRAK'

-- Wypisanie dziennych raportów wraz z nazwą ściany i danymi wpisującego
SELECT DR.RaportID, DR.RaportDate, LW.LongwallName, DR.ExcavatedCoalQuantityInKg, DR.WallAdvanceInMeters, DR.Comments, BI.EmployeeID, BI.FirstName, BI.Surname
    FROM  Longwall.DailyRaport AS DR JOIN Employees.BasicInfo AS BI ON DR.EmployeeID = BI.EmployeeID
        JOIN Longwall.BasicInfo AS LW ON LW.LongwallID = DR.LongwallID

-- Dodanie klienta
INSERT INTO Sales.Customers VALUES('KWK', 13456789, 'Katowice', 'Ulica...', '100-100', 'Polska')
SELECT * FROM Sales.Customers

-- Dodanie zamówień
EXEC Sales.AddOrder 1, 5000, 100
EXEC Sales.AddOrder 1, 1000

-- Wypisanie zamówień oczekujących na realizacje
SELECT C.CompanyName, O.*
    FROM Sales.Orders AS O JOIN Sales.Customers C ON O.CustomerID = C.CustomerID
    WHERE RecipeDate IS NULL
```

# 7. Podsumowanie i wnioski

Zaprezentowana baza dla kopalni węgla kamiennego niestety nie będzie odpowiednia do implementacji dla prawdziwych warunków – jest zbyt uboga w funkcje. Posiada ona podstawowe funkcje pozwalające znaleźć zastosowanie w niewielkich specyficznych „symulacjach" omawianego systemu.

W projekcie przyjęto sporo ograniczeń i kompromisów w porównaniu do implementowanego rzeczywistego systemu. Wiele ograniczeń lub sytuacji nie opisanych w tym projekcie wymaga analizy biznesowej i konsultacji oczekiwań potencjalnego klienta.

Ze strony logicznej nie uwzględniono przypadków które dane można kasować i jakie niesie to za sobą konsekwencje, ponadto nie ustalono co m.in. powinno się dział po wprowadzeniu błędnych danych które system zaakceptuje – np. wprowadzenie złej sumy wydobytego urobku – czy błąd taki po namierzeniu np. jest usuwany i wprowadzany jest nowy poprawny wpis, czy może jest aktualizowany i dodawana jest odpowiednia adnotacja i jak na takie zachowanie powinny reagować skorelowane tablice.

Projekt ten narzucone ma spore ograniczenia dotyczące eksploatacji, nie uwzględnia m.in. zagrożeń naturalnych, nieciągłości pokładu jak i zmiennych warunków górniczo-geologicznych. Składane raporty są przyjęte w systemie 1 raport na 24 godziny co też implikuje gorsze opisanie przebiegu eksploatacji.

Od strony technicznej w projekcie brakuje widoków – po stworzeniu interfejsu graficznego powinny takowe powstać w zależności jakie dane będą potrzebne do wyświetlenia użytkownikowi. Przyjęto także, że obsługa tabel jest jedno wierszowa – nie ma możliwości skorzystania z procedury czy tablicy do której jest podpięty wyzwalacz dodając kilka wierszy, co ma przełożenie na wydajność.