POLITECHNIKA ŚWIĘTOKRZYSKA W KIELCACH		
Bazy danych 2		
Daniel Iwaniec **Artur Kałuża**	**Projekt**	**Grupa 311A**

1. Wstęp

Tematem projektu jest **hurtownia danych** dla składu materiałów budowlanych. Zakres zagadnień obejmuje stworzenie struktury hurtowni danych, przygotowanie danych oraz ich import i analizę.

2. Struktura

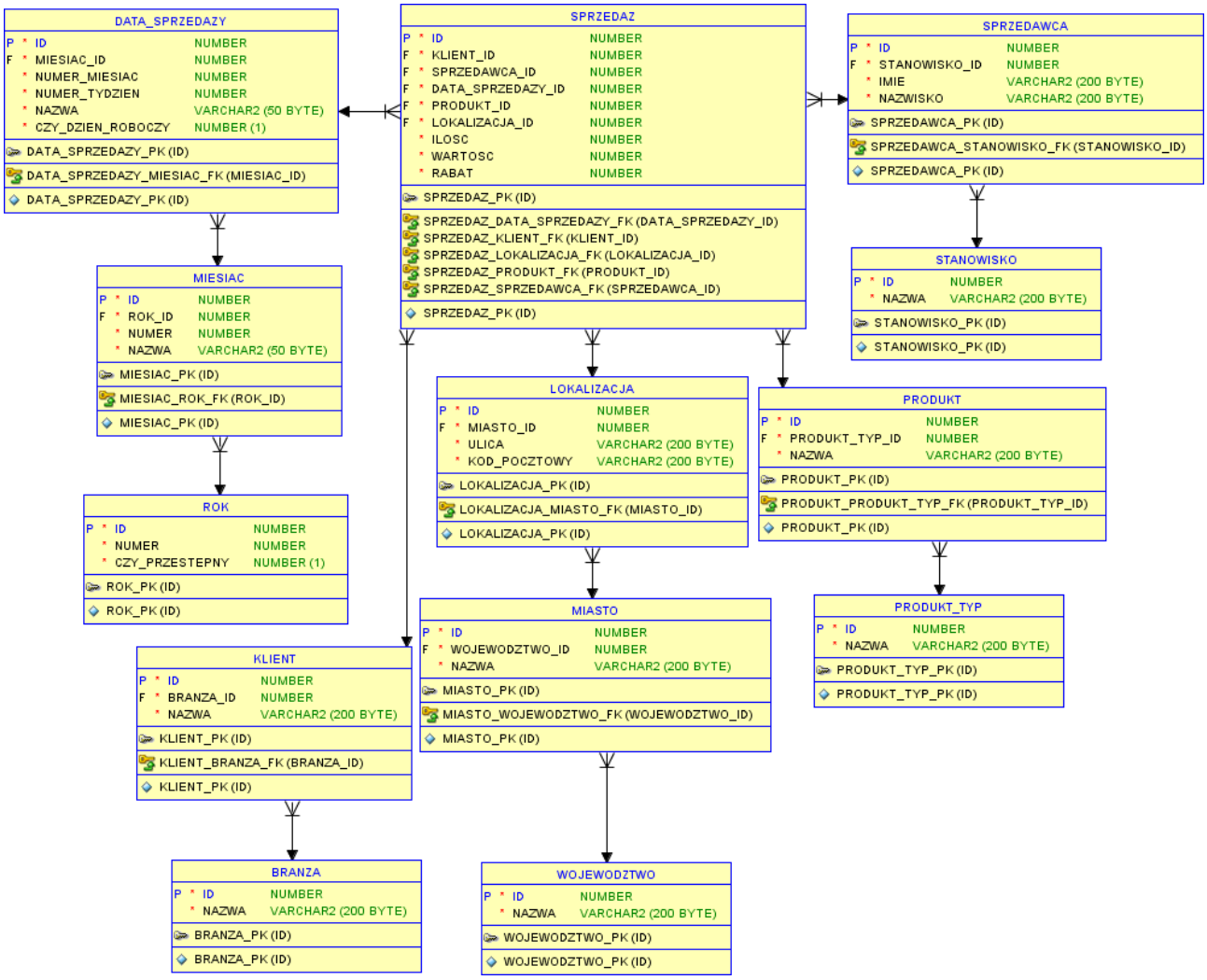


Diagram został wygenerowany w notacji bazującej na notacji **kruczej stopki** (ang. *crow's foot*) za pomocą programu Oracle SQL Data Modeler.

Struktura została stworzona zgodnie z konwencją **schematu płatka śniegu** (ang. *snowflake schema*). Składa się z **tabeli faktów** (sprzedaz) oraz 5 znormalizowanych **wymiarów** (produkt, sprzedawca, klient, data_sprzedazy, lokalizacja).

Wszystkie tabele (*encje*) spełniają wymogi pierwszej postaci normalnej (**1NF**), ponieważ każdy wiersz posiada klucz klucz główny (ang. *primary key* lub *PK*) identyfikujący jendoznacznie dane, które są atomowe i nie powielają się.

Wszystkie tabele spełniają również wymogi drugiej postaci normalnej (**2NF**), ponieważ dane nie zależące funkcyjnie od klucza głównego zostały przeniesione do osobnych encji i powiązane za pomocą kluczy obcych (ang. *foregin key* lub *FK*).

Wszystkie tabele spełniają również wymogi trzeciej postaci normalnej (**3NF**), ponieważ nie istnieją atrybuty wtórne nie zależące od klucza głównego (np. brak kolumny z wartością sprzedaży bez odliczonego rabatu).

```
CREATE OR REPLACE PROCEDURE DROP_ALL IS
  CURSOR table_cursor IS
    SELECT object_name FROM user_objects WHERE object_type = 'TABLE';

  BEGIN
    FOR table_item IN table_cursor LOOP
      EXECUTE IMMEDIATE ('DROP TABLE ' || table_item.object_name || ' CASCADE CONSTRAINTS');
    END LOOP;

    EXCEPTION WHEN OTHERS THEN ROLLBACK;
  END;
/
EXECUTE DROP_ALL;
DROP PROCEDURE DROP_ALL;

CREATE TABLE branza (
  id    INTEGER       NOT NULL,
  nazwa VARCHAR2(200) NOT NULL,
  CONSTRAINT branza_pk PRIMARY KEY (id)
);

CREATE TABLE klient (
  id        INTEGER       NOT NULL,
  branza_id INTEGER       NOT NULL,
  nazwa     VARCHAR2(200) NOT NULL,
  CONSTRAINT klient_pk        PRIMARY KEY (id),
  CONSTRAINT klient_branza_fk FOREIGN KEY (branza_id) REFERENCES branza (id)
);

CREATE TABLE stanowisko (
  id    INTEGER       NOT NULL,
  nazwa VARCHAR2(200) NOT NULL,
  CONSTRAINT stanowisko_pk PRIMARY KEY (id)
);

CREATE TABLE sprzedawca (
  id            INTEGER       NOT NULL,
  stanowisko_id INTEGER       NOT NULL,
  imie          VARCHAR2(200) NOT NULL,
  nazwisko      VARCHAR2(200) NOT NULL,
  CONSTRAINT sprzedawca_pk            PRIMARY KEY (id),
  CONSTRAINT sprzedawca_stanowisko_fk FOREIGN KEY (stanowisko_id) REFERENCES stanowisko (id)
);

CREATE TABLE rok (
  id             INTEGER      NOT NULL,
  numer          INTEGER      NOT NULL,
  czy_przestepny NUMBER(1)    NOT NULL,
  CONSTRAINT rok_pk PRIMARY KEY (id)
);

CREATE TABLE miesiac (
  id     INTEGER      NOT NULL,
  rok_id INTEGER      NOT NULL,
  numer  INTEGER      NOT NULL,
  nazwa  VARCHAR2(50) NOT NULL,
  CONSTRAINT miesiac_pk     PRIMARY KEY (id),
  CONSTRAINT miesiac_rok_fk FOREIGN KEY (rok_id) REFERENCES rok (id)
);

CREATE TABLE data_sprzedazy (
  id                INTEGER      NOT NULL,
  miesiac_id        INTEGER      NOT NULL,
  numer_miesiac     INTEGER      NOT NULL,
  numer_tydzien     INTEGER      NOT NULL,
  nazwa             VARCHAR2(50) NOT NULL,
  czy_dzien_roboczy NUMBER(1)    NOT NULL,
  CONSTRAINT data_sprzedazy_pk         PRIMARY KEY (id),
  CONSTRAINT data_sprzedazy_miesiac_fk FOREIGN KEY (miesiac_id) REFERENCES miesiac (id)
);

CREATE TABLE produkt_typ (
  id    INTEGER       NOT NULL,
  nazwa VARCHAR2(200) NOT NULL,
  CONSTRAINT produkt_typ_pk PRIMARY KEY (id)
);
```

```
CREATE TABLE produkt (
  id             INTEGER       NOT NULL,
  produkt_typ_id INTEGER       NOT NULL,
  nazwa          VARCHAR2(200) NOT NULL,
  CONSTRAINT produkt_pk             PRIMARY KEY (id),
  CONSTRAINT produkt_produkt_typ_fk FOREIGN KEY (produkt_typ_id) REFERENCES produkt_typ (id)
);

CREATE TABLE wojewodztwo (
  id    INTEGER       NOT NULL,
  nazwa VARCHAR2(200) NOT NULL,
  CONSTRAINT wojewodztwo_pk PRIMARY KEY (id)
);

CREATE TABLE miasto (
  id             INTEGER       NOT NULL,
  wojewodztwo_id INTEGER       NOT NULL,
  nazwa          VARCHAR2(200) NOT NULL,
  CONSTRAINT miasto_pk             PRIMARY KEY (id),
  CONSTRAINT miasto_wojewodztwo_fk FOREIGN KEY (wojewodztwo_id) REFERENCES wojewodztwo (id)
);

CREATE TABLE lokalizacja (
  id           INTEGER       NOT NULL,
  miasto_id    INTEGER       NOT NULL,
  ulica        VARCHAR2(200) NOT NULL,
  kod_pocztowy VARCHAR2(200) NOT NULL,
  CONSTRAINT lokalizacja_pk        PRIMARY KEY (id),
  CONSTRAINT lokalizacja_miasto_fk FOREIGN KEY (miasto_id) REFERENCES miasto (id)
);

CREATE TABLE sprzedaz (
  id                INTEGER           NOT NULL,
  klient_id         INTEGER           NOT NULL,
  sprzedawca_id     INTEGER           NOT NULL,
  data_sprzedazy_id INTEGER           NOT NULL,
  produkt_id        INTEGER           NOT NULL,
  lokalizacja_id    INTEGER           NOT NULL,
  ilosc             INTEGER           NOT NULL,
  wartosc           INTEGER           NOT NULL,
  rabat             INTEGER DEFAULT 0 NOT NULL,
  CONSTRAINT sprzedaz_pk                PRIMARY KEY (id),
  CONSTRAINT sprzedaz_klient_fk         FOREIGN KEY (klient_id)         REFERENCES klient (id),
  CONSTRAINT sprzedaz_sprzedawca_fk     FOREIGN KEY (sprzedawca_id)     REFERENCES sprzedawca (id),
  CONSTRAINT sprzedaz_data_sprzedazy_fk FOREIGN KEY (data_sprzedazy_id) REFERENCES data_sprzedazy (id),
  CONSTRAINT sprzedaz_produkt_fk        FOREIGN KEY (produkt_id)        REFERENCES produkt (id),
  CONSTRAINT sprzedaz_lokalizacja_fk    FOREIGN KEY (lokalizacja_id)    REFERENCES lokalizacja (id)
);
```

Procedura **DROP_ALL** pobiera z predefiniowanej tabeli **user_object** wszystkie utworzone tabele do kursora, które następnie za pomocą pętli są kolejno usuwane. Wykonywanie **DDL** wewnątrz procedury wymaga użycia **EXECUTE IMMEDIATE**. Na końcu procedura jest usuwana. Powodem użycia tej procedury jest możliwość jej poprawnego wykonania w przypadku gdy tabele nie istnieją.

Wszystkie zdefiniowane tabele posiadają jawnie zdefiniowane *klucze główne* oraz *klucze obce* (jeżeli te są potrzebne). Wszystkie zdefiniowane kolumny nie mogą być puste (**NOT NULL**). Wartości liczbowe są zdefiniowane jako **INTEGER**, tekstowe jako **VARCHAR2**, a logiczne jako **NUMER(1)** (gdzie 0 oznacza fałsz, a 1 prawdę).

3. Import danych

```
Import danych za pomocą SQL*Loader'a do tabeli BRANZA

SQL*Loader: Release 11.2.0.2.0 - Production on Pn Gru 29 13:39:14 2014

Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.

Commit point reached - logical record count 4
Commit point reached - logical record count 5
```

```
Import danych za pomocą SQL*Loader'a do tabeli KLIENT

SQL*Loader: Release 11.2.0.2.0 - Production on Pn Gru 29 13:39:14 2014

Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.

Commit point reached - logical record count 64
Commit point reached - logical record count 99
Commit point reached - logical record count 100
```

```
Import danych za pomocą SQL*Loader'a do tabeli ROK

SQL*Loader: Release 11.2.0.2.0 - Production on Pn Gru 29 13:39:14 2014

Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.

Commit point reached - logical record count 4
Commit point reached - logical record count 5
```

```
Import danych za pomocą SQL*Loader'a do tabeli MIESIAC

SQL*Loader: Release 11.2.0.2.0 - Production on Pn Gru 29 13:39:14 2014

Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.

Commit point reached - logical record count 59
Commit point reached - logical record count 60
```

```
Import danych za pomocą SQL*Loader'a do tabeli DATA_SPRZEDAZY

SQL*Loader: Release 11.2.0.2.0 - Production on Pn Gru 29 13:39:14 2014

Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.

Commit point reached - logical record count 64
Commit point reached - logical record count 128
Commit point reached - logical record count 192
Commit point reached - logical record count 256
Commit point reached - logical record count 320
Commit point reached - logical record count 384
Commit point reached - logical record count 448
Commit point reached - logical record count 512
Commit point reached - logical record count 576
Commit point reached - logical record count 640
Commit point reached - logical record count 704
Commit point reached - logical record count 768
Commit point reached - logical record count 832
Commit point reached - logical record count 896
Commit point reached - logical record count 960
Commit point reached - logical record count 1024
Commit point reached - logical record count 1088
Commit point reached - logical record count 1152
Commit point reached - logical record count 1216
Commit point reached - logical record count 1280
Commit point reached - logical record count 1344
Commit point reached - logical record count 1408
Commit point reached - logical record count 1472
Commit point reached - logical record count 1536
Commit point reached - logical record count 1600
Commit point reached - logical record count 1664
Commit point reached - logical record count 1728
Commit point reached - logical record count 1792
Commit point reached - logical record count 1825
Commit point reached - logical record count 1826
```

```
Import danych za pomocą SQL*Loader'a do tabeli STANOWISKO

SQL*Loader: Release 11.2.0.2.0 - Production on Pn Gru 29 13:39:14 2014

Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.

Commit point reached - logical record count 4
Commit point reached - logical record count 5
```

```
Import danych za pomocą SQL*Loader'a do tabeli SPRZEDAWCA

SQL*Loader: Release 11.2.0.2.0 - Production on Pn Gru 29 13:39:14 2014

Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.

Commit point reached - logical record count 49
Commit point reached - logical record count 50
```

```
 Import danych za pomocą SQL*Loader'a do tabeli PRODUKT_TYP

SQL*Loader: Release 11.2.0.2.0 - Production on Pn Gru 29 13:39:14 2014

Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.

Commit point reached - logical record count 23
Commit point reached - logical record count 24
 Import danych za pomocą SQL*Loader'a do tabeli PRODUKT

SQL*Loader: Release 11.2.0.2.0 - Production on Pn Gru 29 13:39:14 2014

Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.

Commit point reached - logical record count 64
Commit point reached - logical record count 128
Commit point reached - logical record count 192
Commit point reached - logical record count 256
Commit point reached - logical record count 320
Commit point reached - logical record count 384
Commit point reached - logical record count 448
Commit point reached - logical record count 512
Commit point reached - logical record count 576
Commit point reached - logical record count 640
Commit point reached - logical record count 704
Commit point reached - logical record count 768
Commit point reached - logical record count 832
Commit point reached - logical record count 896
Commit point reached - logical record count 960
Commit point reached - logical record count 1024
Commit point reached - logical record count 1088
Commit point reached - logical record count 1152
Commit point reached - logical record count 1189
Commit point reached - logical record count 1190
 Import danych za pomocą SQL*Loader'a do tabeli WOJEWODZTWO

SQL*Loader: Release 11.2.0.2.0 - Production on Pn Gru 29 13:53:20 2014

Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.

Commit point reached - logical record count 15
Commit point reached - logical record count 16
 Import danych za pomocą SQL*Loader'a do tabeli MIASTO

SQL*Loader: Release 11.2.0.2.0 - Production on Pn Gru 29 13:53:20 2014

Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.

Commit point reached - logical record count 31
Commit point reached - logical record count 32
 Import danych za pomocą SQL*Loader'a do tabeli LOKALIZACJA

SQL*Loader: Release 11.2.0.2.0 - Production on Pn Gru 29 13:53:20 2014

Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.

Commit point reached - logical record count 63
Commit point reached - logical record count 64
```

```
Import danych za pomocą SQL*Loader'a do tabeli SPRZEDAZ

SQL*Loader: Release 11.2.0.2.0 - Production on Pn Gru 29 13:39:14 2014

Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.

Commit point reached - logical record count 64
Commit point reached - logical record count 128
Commit point reached - logical record count 192
Commit point reached - logical record count 256
Commit point reached - logical record count 320
Commit point reached - logical record count 384
Commit point reached - logical record count 448
Commit point reached - logical record count 512
Commit point reached - logical record count 576
Commit point reached - logical record count 640
Commit point reached - logical record count 704
Commit point reached - logical record count 768
Commit point reached - logical record count 832
Commit point reached - logical record count 896
Commit point reached - logical record count 960
Commit point reached - logical record count 1024
Commit point reached - logical record count 1088
Commit point reached - logical record count 1152
Commit point reached - logical record count 1216
Commit point reached - logical record count 1280
Commit point reached - logical record count 1344
Commit point reached - logical record count 1408
Commit point reached - logical record count 1472
Commit point reached - logical record count 1536
Commit point reached - logical record count 1600
Commit point reached - logical record count 1664
Commit point reached - logical record count 1728
Commit point reached - logical record count 1792
Commit point reached - logical record count 1856
Commit point reached - logical record count 1920
Commit point reached - logical record count 1984
Commit point reached - logical record count 2048
Commit point reached - logical record count 2112
Commit point reached - logical record count 2176
Commit point reached - logical record count 2240
Commit point reached - logical record count 2304
Commit point reached - logical record count 2368
Commit point reached - logical record count 2432
Commit point reached - logical record count 2496
Commit point reached - logical record count 2560
Commit point reached - logical record count 2624
Commit point reached - logical record count 2688
Commit point reached - logical record count 2752
Commit point reached - logical record count 2816
Commit point reached - logical record count 2880
Commit point reached - logical record count 2944
Commit point reached - logical record count 3008
Commit point reached - logical record count 3072
Commit point reached - logical record count 3136
Commit point reached - logical record count 3200
Commit point reached - logical record count 3264
Commit point reached - logical record count 3328
Commit point reached - logical record count 3392
Commit point reached - logical record count 3456
Commit point reached - logical record count 3520
Commit point reached - logical record count 3584
Commit point reached - logical record count 3648
Commit point reached - logical record count 3712
Commit point reached - logical record count 3776
Commit point reached - logical record count 3840
Commit point reached - logical record count 3904
Commit point reached - logical record count 3968
Commit point reached - logical record count 4032
Commit point reached - logical record count 4096
Commit point reached - logical record count 4160
Commit point reached - logical record count 4224
Commit point reached - logical record count 4288
Commit point reached - logical record count 4352
Commit point reached - logical record count 4416
Commit point reached - logical record count 4480
Commit point reached - logical record count 4544
Commit point reached - logical record count 4608
Commit point reached - logical record count 4672
Commit point reached - logical record count 4736
Commit point reached - logical record count 4800
Commit point reached - logical record count 4864
Commit point reached - logical record count 4928
Commit point reached - logical record count 4992
Commit point reached - logical record count 4999
Commit point reached - logical record count 5000
```

```
@echo off
chcp 65001>nul

set /P login=Podaj login || set login=
if [%login%] == [] goto :error

set /P password=Podaj hasło || set password=
if [%password%] == [] goto :error

echo.
echo.
echo %ESC%[30;48;5;118m Import danych za pomocą SQL*Loader'a do tabeli ROK %ESC%[0m
sqlldr userid=%login%/%password% control=rok.ctl log=rok.log

echo.
echo.
echo %ESC%[30;48;5;118m Import danych za pomocą SQL*Loader'a do tabeli MIESIAC %ESC%[0m
sqlldr userid=%login%/%password% control=miesiac.ctl log=miesiac.log

echo.
echo.
echo %ESC%[30;48;5;118m Import danych za pomocą SQL*Loader'a do tabeli DATA_SPRZEDAZY %ESC%[0m
sqlldr userid=%login%/%password% control=data_sprzedazy.ctl log=data_sprzedazy.log

echo.
echo.
echo %ESC%[30;48;5;118m Import danych za pomocą SQL*Loader'a do tabeli BRANZA %ESC%[0m
sqlldr userid=%login%/%password% control=branza.ctl log=branza.log

echo.
echo.
echo %ESC%[30;48;5;118m Import danych za pomocą SQL*Loader'a do tabeli KLIENT %ESC%[0m
sqlldr userid=%login%/%password% control=klient.ctl log=klient.log

echo.
echo.
echo %ESC%[30;48;5;118m Import danych za pomocą SQL*Loader'a do tabeli STANOWISKO %ESC%[0m
sqlldr userid=%login%/%password% control=stanowisko.ctl log=stanowisko.log

echo.
echo.
echo %ESC%[30;48;5;118m Import danych za pomocą SQL*Loader'a do tabeli SPRZEDAWCA %ESC%[0m
sqlldr userid=%login%/%password% control=sprzedawca.ctl log=sprzedawca.log

echo.
echo.
echo %ESC%[30;48;5;118m Import danych za pomocą SQL*Loader'a do tabeli PRODUKT_TYP %ESC%[0m
sqlldr userid=%login%/%password% control=produkt_typ.ctl log=produkt_typ.log

echo.
echo.
echo %ESC%[30;48;5;118m Import danych za pomocą SQL*Loader'a do tabeli PRODUKT %ESC%[0m
sqlldr userid=%login%/%password% control=produkt.ctl log=produkt.log

echo.
echo.
echo %ESC%[30;48;5;118m Import danych za pomocą SQL*Loader'a do tabeli WOJEWODZTWO %ESC%[0m
sqlldr userid=%login%/%password% control=wojewodztwo.ctl log=wojewodztwo.log

echo.
echo.
echo %ESC%[30;48;5;118m Import danych za pomocą SQL*Loader'a do tabeli MIASTO %ESC%[0m
sqlldr userid=%login%/%password% control=miasto.ctl log=miasto.log

echo.
echo.
echo %ESC%[30;48;5;118m Import danych za pomocą SQL*Loader'a do tabeli LOKALIZACJA %ESC%[0m
sqlldr userid=%login%/%password% control=lokalizacja.ctl log=lokalizacja.log

echo.
echo.
echo %ESC%[30;48;5;118m Import danych za pomocą SQL*Loader'a do tabeli SPRZEDAZ %ESC%[0m
sqlldr userid=%login%/%password% control=sprzedaz.ctl log=sprzedaz.log

goto:eof
:error
echo %ESC%[30;48;5;196m Musisz podać dane dostępowe %ESC%[0m
```

Dane zostały przygotowane w formie plików **CSV** (pełne pliki zawarte w katalogu /dane). Źródłem danych były skrypty **PHP** bezpośrednio generujące dane oraz parsujące katalogi oraz sklepy interetowe. Wszystkim plikom **CSV** utworzone zostały odpowiadające pliki **CTL**. (pełne pliki zawarte w katalogu /dane). Dane zostały zaimportowane za pomocą **SQL*Loader'a** (pliki logów zawarte w katalogu /dane). Automatyczny import został zrealizowany za pomocą pliku wsadowego **BAT**. Plik wsadowy pobiera od użytkownika login oraz hasło potrzebne do utworzenia połączenia z bazą danych. Następnie wszystkie tabele są wypełniane danymi. Dodatkowo plik zmienia stronę kodową na **UTF-8** (aby widoczne były polskie znaki) oraz używa kolorowania **ANSICON**.

4. Zapytania

Wszystkie wyniki zapytań są umieszczone w folderze **/wyniki_zapytan**.

```
SELECT lokalizacja_id, produkt_id, SUM(ilosc), SUM(wartosc), SUM(rabat) FROM sprzedaz
GROUP BY rollup(lokalizacja_id, produkt_id);
```

Zapytanie pobiera z tabeli faktów lokalizację, produkt oraz sumuje wszystkie miary. Sumowanie jest wykonywane dla konkretnej lokalizacji i produktu, konkretnej lokalizacji oraz dla wszystkich faktów.

```
SELECT s.klient_id, s.produkt_id, pt.id AS "PRODUKT_TYP_ID", SUM(s.ilosc), SUM(s.wartosc), SUM(s.rabat) FROM
sprzedaz s
JOIN produkt p ON s.produkt_id = p.id
JOIN produkt_typ pt ON p.produkt_typ_id = pt.id
GROUP BY rollup(s.klient_id, s.produkt_id, pt.id);
```

Zapytanie pobiera z tabeli faktów klienta, produkt oraz typ produktu (za pomocą klauzuli JOIN) oraz sumuje wszystkie miary. Sumowanie jest wykonywane dla konkretnego klienta, produktu oraz typu produktu a następnie dla konkretnego klienta oraz produktu, następnie dla konkretnego klienta, a na końcu dla wszystkich faktów.

```
SELECT b.id AS "BRANZA_ID", s.klient_id, s.produkt_id, SUM(s.ilosc), SUM(s.wartosc), SUM(s.rabat) FROM
sprzedaz s
JOIN klient k ON s.klient_id = k.id
JOIN branza b ON k.branza_id = b.id
GROUP BY cube(b.id, s.klient_id, s.produkt_id);
```

Zapytanie pobiera z tabeli faktów branżę (za pomocą klauzuli JOIN), klienta, produkt oraz sumuje wszystkie miary. Sumowanie jest wykonywane dla:

- branży, klienta oraz produktu,
- branży oraz klienta,
- branży oraz produktu,
- klienta oraz produktu,
- samej branży,
- samego klienta,
- samego produktu,
- wszystkich faktów.

```
SELECT * FROM
  (
    SELECT lokalizacja_id, produkt_id, SUM(ilosc), SUM(wartosc), SUM(rabat) FROM sprzedaz
    GROUP BY cube(lokalizacja_id, produkt_id)
    ORDER BY lokalizacja_id ASC, produkt_id ASC
  )
WHERE lokalizacja_id IS NOT NULL AND produkt_id IS NOT NULL
UNION ALL
SELECT * FROM
  (
    SELECT lokalizacja_id, produkt_id, SUM(ilosc), SUM(wartosc), SUM(rabat) FROM sprzedaz
    GROUP BY cube(lokalizacja_id, produkt_id)
    ORDER BY lokalizacja_id ASC
  )
WHERE lokalizacja_id IS NOT NULL AND produkt_id IS NULL
```

```
UNION ALL
SELECT * FROM
  (
    SELECT lokalizacja_id, produkt_id, SUM(ilosc), SUM(wartosc), SUM(rabat) FROM sprzedaz
    GROUP BY cube(lokalizacja_id, produkt_id)
    ORDER BY produkt_id ASC
  )
WHERE lokalizacja_id IS NULL AND produkt_id IS NOT NULL
UNION ALL
SELECT * FROM
  (
    SELECT lokalizacja_id, produkt_id, SUM(ilosc), SUM(wartosc), SUM(rabat) FROM sprzedaz
    GROUP BY cube(lokalizacja_id, produkt_id)
  )
WHERE lokalizacja_id IS NULL AND produkt_id IS NULL;
```

To zapytanie pobiera lokalizację, produkt oraz sumuje wszystkie miary. Sumowanie jest wykonywane dla lokalizacji oraz produktu, następnie dla samej lokalizacji, samego produktu, a na końcu dla wszystkich faktów. Zapytanie jest wykonane 4 razy jako podzapytanie z którego wybierane są kolejne grupowania, które są złączone za pomocą unii. Dzięki temu kolejne grupowania są uporządkowane w inny niż domyślny sposób.

```
SELECT lokalizacja_id, produkt_id, SUM(ilosc), SUM(wartosc), SUM(rabat) FROM sprzedaz
GROUP BY grouping sets ((lokalizacja_id, produkt_id), (lokalizacja_id), (produkt_id), ());
```

Zapytanie pobiera z tabeli faktów lokalizację, produkt oraz sumuje wszystkie miary. Sumowanie jest wykonywane dla konkretnej lokalizacji i produktu, konkretnej lokalizacji, konkretnego produktu oraz dla wszystkich faktów. Grupowanie zadziała tak samo jak za pomocą cube(lokalizacja_id, produkt_id).

```
SELECT st.id AS "STANOWISKO_ID", s.sprzedawca_id, s.produkt_id, SUM(s.ilosc), SUM(s.wartosc), SUM(s.rabat)
FROM sprzedaz s
JOIN sprzedawca sp ON s.sprzedawca_id = sp.id
JOIN stanowisko st ON sp.stanowisko_id = st.id
GROUP BY grouping sets ((st.id, s.sprzedawca_id, s.produkt_id), (st.id), (s.produkt_id));
```

Zapytanie pobiera stanowisko (za pomocą klauzuli JOIN), sprzedawcę, produkt oraz sumuje wszystkie miary. Sumowanie jest wykonywane dla:

- stawnowiska, sprzedawcy oraz produktu,
- samego stanowiska,
- samego produktu.

Takich grupowań nie da się utworzyć za pomocą rollup() lub cube().

```
SELECT s.sprzedawca_id, s.produkt_id, s.wartosc, SUM(s.wartosc) OVER (PARTITION BY s.sprzedawca_id) AS
"WARTOSC_SPRZEDAWCA"
FROM sprzedaz s
ORDER BY s.sprzedawca_id ASC, s.produkt_id ASC;
```

Zapytanie pobiera sprzedawcę, produkt, wartość pojedynczej sprzedaży oraz całkowitą wartość sprzedaży dla każdego sprzedawcy z tabeli faktów.

```
SELECT b.id AS "BRANZA_ID", s.produkt_id, s.wartosc,
SUM(s.wartosc) OVER (PARTITION BY b.id) AS "WARTOSC_BRANZA",
CONCAT(ROUND(100 * (SUM(s.wartosc) OVER (PARTITION BY b.id) / SUM(s.wartosc) OVER ()), 2), '%') AS "UDZIAL
%",
SUM(s.wartosc) OVER () AS "WARTOSC_CALOSC"
FROM sprzedaz s
JOIN klient k ON s.klient_id = k.id
JOIN branza b ON k.branza_id = b.id
ORDER BY b.id ASC, s.produkt_id ASC;
```

Zapytanie pobiera z tabeli faktów branżę klienta, produkt, wartość sprzedaży dla konkretnej sprzedaży, udział procentowy branży w sprzedaży oraz całkowitą wartość sprzedaży. Udziały procentowe po zaokrągleniu są bardzo podobne, ponieważ skrypt losujący był stworzony tak, żeby równomiernie rozłożyć watości.

```
SELECT r.numer as "ROK", m.numer as "MIESIAC", s.wartosc,
SUM(s.wartosc) over (PARTITION BY r.id ORDER BY m.id RANGE BETWEEN CURRENT ROW AND CURRENT ROW) AS
"SUMA_MIESIAC",
SUM(s.wartosc) over (PARTITION BY r.id ORDER BY m.id RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS
"SUMA_ROK"
FROM sprzedaz s
JOIN data_sprzedazy ds ON s.data_sprzedazy_id = ds.id
JOIN miesiac m ON ds.miesiac_id = m.id
JOIN rok r ON m.rok_id = r.id
ORDER BY r.numer, m.numer;
```

Zapytanie pobiera rok, miesiąc oraz wartość dla każdego faktu sprzedaży. Dodatkowo wyświetlana jest sumaryczna wartość dla miesiąca oraz od początku roku za pomocą pratiton by ... order by.

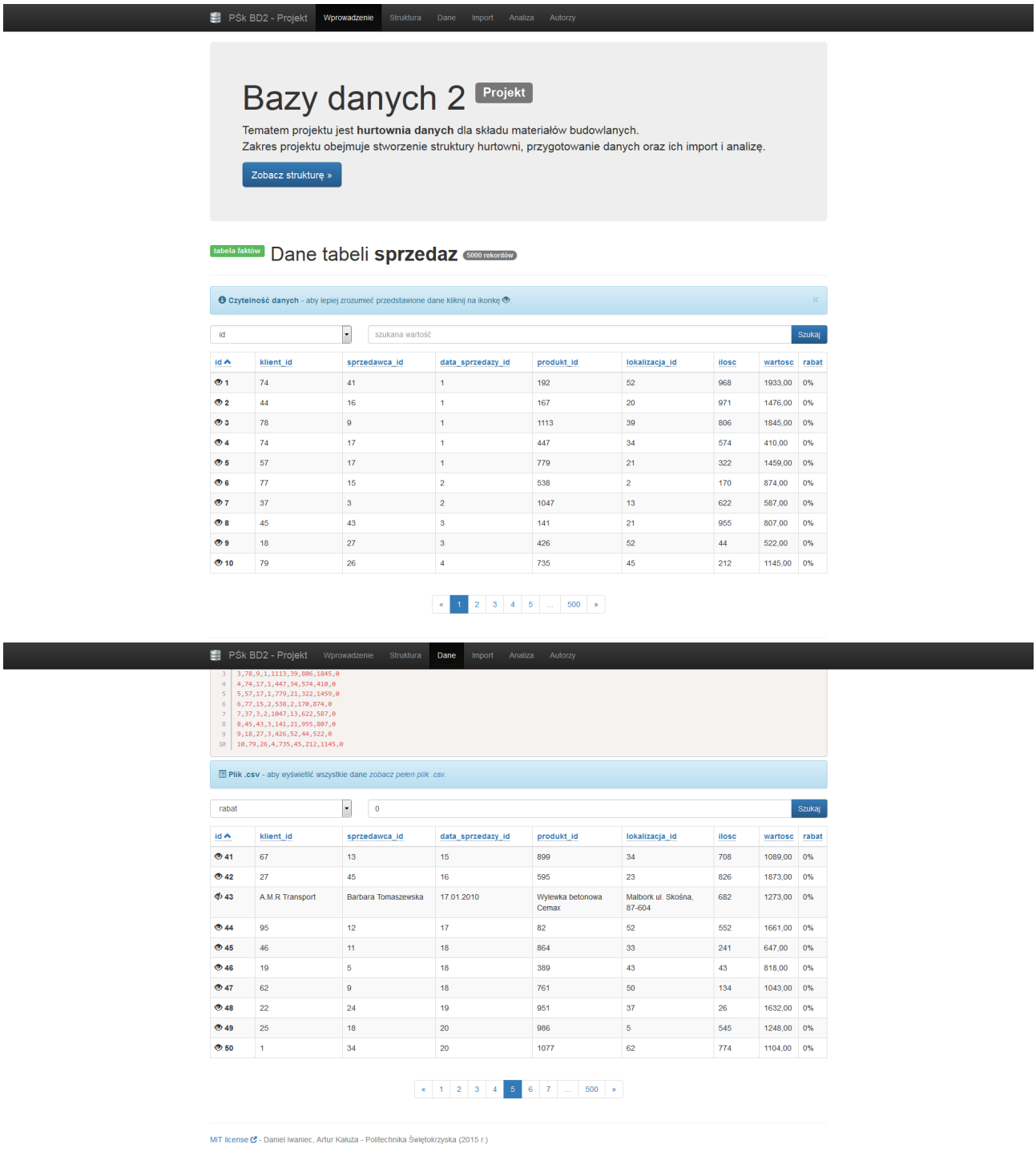
```
SELECT s.lokalizacja_id, s.produkt_id, s.wartosc,
RANK() over (PARTITION BY s.lokalizacja_id ORDER BY s.wartosc DESC) AS "RANKING",
DENSE_RANK() over (PARTITION BY s.lokalizacja_id ORDER BY s.wartosc DESC) AS "DENSE_RANKING",
ROW_NUMBER() over (PARTITION BY s.lokalizacja_id ORDER BY s.wartosc DESC) AS "ROW_NUMBER"
FROM sprzedaz s
ORDER BY s.lokalizacja_id, "DENSE_RANKING";
```

Zapytanie pobiera lokalizację, produkt oraz wartość każdego faktu sprzedaży. Dodatkowo wyświetlany ranking wartości sprzedaży w danej lokalizacji oraz numer porządkowy każdego wiersza. Wyniki są posortowane rosnąco według lokalizacji oraz miejsca w rankingu.

```
SELECT LOKALIZACJA_ID AS "lokalizacja", PRODUKT_ID AS "produkt", SUM_ILOSC, SUM_WARTOSC, SUM_RABAT FROM
  (
    SELECT lokalizacja_id, produkt_id, SUM(ilosc) AS "SUM_ILOSC", SUM(wartosc) AS "SUM_WARTOSC", SUM(rabat)
AS "SUM_RABAT" FROM sprzedaz
    GROUP BY lokalizacja_id, produkt_id
    ORDER BY lokalizacja_id ASC, produkt_id ASC
  )
UNION ALL
SELECT LOKALIZACJA_ID AS "lokalizacja", NULL AS "produkt", SUM_ILOSC, SUM_WARTOSC, SUM_RABAT FROM
  (
    SELECT lokalizacja_id, SUM(ilosc) AS "SUM_ILOSC", SUM(wartosc) AS "SUM_WARTOSC", SUM(rabat) AS
"SUM_RABAT" FROM sprzedaz
    GROUP BY lokalizacja_id
    ORDER BY lokalizacja_id ASC
  )
UNION ALL
SELECT NULL AS "lokalizacja", PRODUKT_ID AS "produkt", SUM_ILOSC, SUM_WARTOSC, SUM_RABAT FROM
  (
    SELECT produkt_id, SUM(ilosc) AS "SUM_ILOSC", SUM(wartosc) AS "SUM_WARTOSC", SUM(rabat) AS "SUM_RABAT"
FROM sprzedaz
    GROUP BY produkt_id
    ORDER BY produkt_id ASC
  )
UNION ALL
SELECT NULL AS "lokalizacja", NULL AS "produkt", SUM_ILOSC, SUM_WARTOSC, SUM_RABAT FROM
  (
    SELECT SUM(ilosc) AS "SUM_ILOSC", SUM(wartosc) AS "SUM_WARTOSC", SUM(rabat) AS "SUM_RABAT" FROM sprzedaz
  )
;
```

Zapytanie pobiera lokalizację, produkt oraz sumę wszystkich miar. Sumowanie jest wykonywane dla lokalizacji i produktu, samej lokalizacji, samego produktu oraz dla wszystkich faktów. Podazapytania złączone są za pomocą unii. Zapytanie działa jak samo jak zapytanie z GROUP BY CUBE(lokalizacja_id, produkt_id), ale bez użycia cube.

4. GUI napisane w PHP



Do hurtowni danych została napisana aplikacja w PHP, która pozwala przeglądać dane. Zrzuty ekranu z apliakcji znajdują się w folderze /grafiki/php, a sama aplikacja w /php.

Repozytorium aplikacji: https://github.com/Ghutix/BD2/tree/master/php.

Ocena aplikacji wg **Scrutinizer** *(continuous inspection)* - 9.24*:* https://scrutinizer-ci.com/g/Ghutix/BD2/?branch=master.